Statistische Anwendungen 2
Einführung in die Datenanalyse und Statistik mit R

Prof. Dr. Ulrich Wellisch
Wirtschaftsmathematik-Aktuarwissenschaften
Hochschule Rosenheim

WS 2015/2016

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Anwendungsgebiete von R

R ist eine vollständige, interaktive Entwicklungsumgebung zur grafischen und statistischen Datenanalyse.

Mithilfe von R können die folgenden, typischen Aufgabenstellungen in einem Datenanalyse-Prozess effizient bearbeitet werden:

1. Datenmanagement und Datenmanipulation
2. Berechnungen in Vektor-, Matrix- und Arraystrukturen
3. Deskriptive und explorative Datenanalyse
4. Erzeugen von Grafiken und Erstellen von Berichten
5. Induktive statistische Methoden
6. Verfahren aus der Künstlichen Intelligenz, maschinellen Lernen und dem Data Mining
7. Objektorientierte Programmiersprache (Interpretersprache)

Insbesondere der Programmiersprachen-Charakter unterscheidet R von anderen, oft sehr Oberflächen-orientierten, Analysewerkzeugen und ermöglicht so eine große Flexibilität bei der Datenanalyse (z.B. können eigene Analyseverfahren als Funktionen erstellt werden). In R Programme können Teilprogramme, die mit anderen Programmiersprachen (z.B. C) erstellt sind, leicht eingebunden werden. Dies kann z.B. zur Optimierung

der Laufzeiten von Programmen angewandt werden, da R als Interpretersprache hier Nachteile besitzt.

Die in der Grundversion von R geringe Benutzerführung mittels GUI erscheinen dem R-Einsteiger im Vergleich zu stark Oberflächen(Menü)-orientierten Analysewerkzeugen zunächst eher nachteilig und mühevoll. Nach einer gewissen Einarbeitungszeit und Vertrautheit mit dem R-System können allerdings dann die Vorteile des flexiblen Programmiersprache-Systems ausgenutzt werden.

Ferner gibt es verschiedene erweiterte R-Editoren (z.B. R-WinEdt mit dem Paket **RWinEdt**, Ligges (2003) oder **Tinn-R**, Tinn-R Development Team (2006)) und auch erweiterte, mehr menügesteuerte R-GUI (z.B. R Commander mit dem Paket **Rcmdr**, Fox (2005) oder **RStudio**, *www.rstudio.com*), mit denen das Grundsystem ergänzt werden kann. Mit der GUI-Erweiterung R Commander steht dem Anwender ein menügesteuertes statistisches Analysesystem (vergleichbar mit SPSS oder SAS-Insight) zur Verfügung. Das RStudio gibt es in einer Desktop- und einer Server-Version. Die erweiterten Editoren sind vor allem bei der Bearbeitung größerer Programmierprojekte zu empfehlen.

1.2 Historie, Installation und Dokumentation von R

In den 1990-er Jahren erstellten R. Ihaka und R. Gentleman mit R eine Open Source Umsetzung der S-Programmiersprache, die in den 1980-er Jahren bei AT&T von R. Becker, J. Chambers und A. Wilks als hochspezielle Software zur effizienten Datenanalyse entwickelt wurde. Die Sprache S bildet auch die Basis des kommerziellen Datenanalysesystems S-PLUS. Daher kann man R als die wissenschaftlich orientierte und kostenfreie Version des professionellen und kommerziellen Werkzeugs S-PLUS betrachten.

Seit 1997 ist die Weiterentwicklung von R in dem *R Core Team* gebündelt, in dem Entwickler und statistisch, methodische Forscher aus der ganzen Welt zusammenarbeiten. Neben der Erweiterung der Methodenvielfalt wird hier insbesondere auch an der Programm-Performance und Stabilität gearbeitet.
Die breite (wissenschaftliche) Anwendung von R, die im R Core Team gebündelte technische und methodische Weiterentwicklung und die offene Diskussion von Stärken und Schwächen der Software sind Vorteile eines Open Source Konzepts im Vergleich zu kommerziellen Anbietern.

Durch die enge Verwandschaft von R und S sind R- bzw. S-Programme im Wesentlichen zueinander kompatibel und S- bzw. S-PLUS-Literatur (Technische Handbücher und auch methodische Literatur) ist auch für den R-Anwender verwendbar.

Der Quellcode (selbstentpackende Software) der Grundversion von R kann über die internationale Projekt-Homepage

http://cran.r-project.org

bzw. die deutschsprachige Projekt-Hompage

http://r-project.de

auf den eigenen Rechner geladen werden. Auf den Seiten findet man auch grundlegende Informationen zu R, Standard-Dokumentationen, Handbücher, Demos und Zusatzpakete (sogenannte *packages* s.u.) mit erweiterten Funktionalitäten. Hier werden auch neue R-Versionen zur Verfügung gestellt.
Im R-Umfeld sind nicht nur der Quellcode und alle entwickelten Zusatzpakete kostenlos erhältlich, sondern man findet auch eine Vielzahl von R-Dokumentationen (technische und methodische) frei zugänglich im Netz (entweder über die R-Projekt-Seiten oder auf anderen Seiten von Hochschulen, R-Anwendern etc.). Die erhältlichen Dokumentationen und Artikel sind für unterschiedlichsten Zielgruppen verfasst. So findet man technisch orientierte, theoretisch methodische und sehr angewandte methodische Literatur. Teilweise richten sich die Veröffentlichung an R- oder auch Statistik-Experten, teilweise findet man auch Literatur für den R- und/oder Statistik-Anfänger. Das Suchen nach frei zugänglicher R-Literatur mit einer Suchmaschine lohnt sich auf jeden Fall. Darüber hinaus gibt es natürlich auch käuflich zu erwerbende Literatur zu dem Themenkomplex Datenanalyse und Statistik mit R.

Nach der Installation von R stehen dem Anwender einige Datensätze z.B. für Übungen zur Verfügung. Eine Übersicht dieser Datensätze erhält man mit `data()`. Durch `data(Daten-Name)` aktiviert man den Datensatz `Daten-Name` für die aktuelle R-Sizung.

1.3 Zusatzpakete (packages)

Das Konzept der Zusatzpakete, auch packages genannt, ist eine grundlegende Vorgehensweise bei R. In dem Grundsystem von R sind die üblichen Techniken zum Datenmanagement, Grafik-Erzeugung und statistischen Analyseverfahren implementiert (im sogenannten `base`-Paket). Für einen Standardanwender sind die Funktionalitäten des Grundsystems in der Regel ausreichend. Zusätzlich gibt es noch eine Vielzahl von Zusatzpaketen, die man bei Bedarf (über die R-Projekt-Seiten) installieren kann. Darin finden sich erweiterte Methoden sowohl für das Datenmanagement (z.B. Zugriff auf Datenbanken) als auch zur Datenanalyse (z.B. semiparametrische Regressionsmodelle oder Entscheidungsbaumverfahren).

Die Installation von Zusatzpaketen kann über das R-Menü `Pakete` (`Packages`) in der R-Windows-GUI (`RGui`) über den Menüpunkt `Pakete -> Installiere Paket(e)...`

erfolgen, wobei für den Zugriff auf die R-Projekt-Seiten natürlich ein Internetzugang vorhanden sein muß. Um ein bestimmtes Paket (und damit die darin enthaltenen R-Funktionen) während einer R-Sitzung dann zur Verfügung zu haben, muß innerhalb der R-Sitzung einmalig der Befehl `library(Paket-Name)` abgesetzt werden. Alternativ kann man wieder über die `RGui` den Menüpunkt `Pakete -> Lade Paket...` verwenden. Nach der Bereitstellung eines Pakets erhält man über `help(package = Paket-Name)` eine Beschreibung des Pakets und seiner Funktionen.

Insbesondere im Bereich moderner statistischer Verfahren findet man hier anwendbare Funktionen, die in der kommerziellen Standardsoftware zur Datenanalyse nicht oder nur selten unterstützt werden.

Kapitel 2

Die Entwicklungsumgebung und Programmiersprache R

Wenn R unter Windows auf dem Rechner installiert ist, kann R entweder über `Programme -> R -> Version` oder, falls vorhanden, direkt über die entsprechende R-Ikone gestartet werden. Durch die Eingabe von `q()` in der R Console kann die R Umgebung wieder verlassen werden.

In der R-Syntax wird das `#`-Zeichen zur Definition von Kommentarzeilen in einem Code verwendet.

2.1 R Hilfesystem

Für den Umgang mit den verschiedenen R-Befehlen bzw. R-Funktionen ist das Hilfesystem, genauer die Hilfefunktion `help()` (oder `?()`), sehr hilfreich.
Durch `help(Funktionsname)` erhält man eine Erklärung der Syntax und der Wirkungsweise von Funktionsargumenten und Parameter einer R-Funktion. Weiter werden typische Anwendungsbeispiele der Funktionen gegeben.

2.2 Objekte

R ist eine objektorientierte Interpreter-Sprache und eine vektorbasierte Sprache.

In R (wie in jeder objektorientierten Sprache) sind praktisch alle in der Programmierung auftretenden Größen *Objekte*. So sind z.B. Datensätze, Matrizen, Grafiken,

statistische Modelle oder auch Funktionen Objekte. Für die Klassen von Objekten gibt es dann im Allgemeinen spezielle *Methoden* von anwendbaren Funktionen.

Jedes Objekt besitzt eine Klasse (abzufragen mit `class()`), eine Länge (`length()`) und einen *Datentyp* (oder *Modus*), vgl. unten. Ein Objekt kann noch weitere *Attribute* besitzen, die über `attributes()` abgefragt und über `attr()` gesetzt werden können.

Die Funktion `str()` liefert die Struktur eines Objektes, d.h. u.a. den Datentyp, die Attribute und auch eine geeignete Darstellung der im Objekt enthaltenen Daten.

R ist eine vektorbasierte Sprache, d.h. jedes Objekt wird R-intern durch einen Vektor repräsentiert.

2.3 Anwendung von R Funktionen

Die Anwendung von R-Funktionen bilden einen wesentlichen Teil des Arbeitens mit der R-Umgebung bei Datenanalysen. Bei R-Funktionen gibt es verpflichtende Argumente, die die Funktionen zur korrekten Arbeitsweise benötigen. Dies kann z.B. der Datensatz oder allgemein ein R-Objekt sein, auf den die Funktion angewendet werden soll. Darüber hinaus besitzen viele R-Funktionen optionale Argumente oder Parameter, mit denen man die Funktionen in ihrer Funktionsweise verändern kann. Werden diese optionalen Parameter nicht aktiv gesetzt, dann arbeitet R mit voreingestellten Werten der Parameter (sogenannte defaults). Die Argumente und Parameter werden typischerweise in runden Klammern nach dem Funktionsnamen angegeben, d.h. man erhält eine Syntax vom Typ

`funktionsname(Argument1=,Parameter1=,...)`.

Ohne eine konkrete Zuweisung vom Typ `Parameter1=...` ist jeweils genau eine Stelle in den runden Klammern einer R-Funktion für einen bestimmten Parameter bzw. Argument reserviert. Z.B. könnte die erste Stelle im Argument-/Parametervektor einer Funktion für das `Argument1` reserviert sein und die zweite Stelle die Stelle für den `Parameter1` sein. So würden z.B. folgende beide Funktionsaufrufe

```
funktionsname(Argument1 = daten, Parameter1=42)
```

und

```
funktionsname(daten, 42)
```

identisch sein.

Aus Gründen der Übersichtlichkeit und Lesbarkeit ist eine Angabe der Parameter- bzw. Argumentnamen (vgl. den ersten Aufruf oben) zu empfehlen. Der zweite Aufruf besitzt als Kurzversion natürlich den Vorteil, dass er weniger Eingabeaufwand erfordert.

Viele R-Funktionen sind *generisch*, d.h. für verschiedene Objekttypen, die einer Funktion als Argument übergeben werden, besitzt die Funktion auch unterschiedliche Funktionalitäten. So liefert der elementare Grafik-Befehl `plot()` angewandt auf einen Daten-Vektor und auf ein Objekt vom Typ *model* völlig unterschiedliche Grafiken.

Funktionen und Missing Values:

Ein *missing value* (d.h. ein fehlender Wert) in einer Datenstruktur (z.B. einem Datenvektor) wird in R durch die Abkürzung `NA` (Not Available) dargestellt.

R-Funktionen haben (meist) eine spezielle, vorbestimmte Reaktion auf das Auftreten von missing values in den Funktionsargumenten. Oft kann über einen Parameter die Vorgehensweise der Funktion beim Auftreten von missing values festgelegt werden.

Beispiel:

```
> x <- c(1,1,NA,5,3) # Definition eines Datenvektors x
> x
[1]  1  1 NA  5  3
> length(x) # L"ange des Vektors x
[1] 5
> y <- c(2,2,3,4,6)
> x+y
[1]  3  3 NA  9  9
> mean(x) # arithmetisches Mittel von x
[1] NA
```

Bemerkung: Ab dem Zeichen `#` wird der Zeileninhalt als Kommentar interpretiert.

2.4 Interaktives Arbeiten mit der R Console

Die einfachste Anwendung von R ist die Verwendung der interaktiven R-Umgebung im Befehlsmodus, sozusagen als Taschenrechner mit sehr großen Funktionalitäten. In der R Console zeigt das Bereitschaftszeichen `>` an, dass eine Eingabe möglich ist. Eingaben werden mit `return` abgeschlossen und zur Ausführung freigegeben.

Falls Befehlseingaben über mehr als eine Zeile erfolgen sollen, verwendet man in den weiteren Zeilen als erstes Zeichen jeweils das `+` Zeichen. Mehrere Befehle in einer Zeile werden durch das `;` Zeichen getrennt.

Für Dezimalzahlen ist der Dezimalpunkt zu verwenden, z.B. `3.14`. Für Addition, Subtraktion, Multiplikation und Division verwendet man die gebräuchlichen Operatoren: `+,-,*` und /. Als Potenzoperator steht `^` zur Verfügung. Innerhalb von Formeln können die runden Klammersymbole `( )` in gewohnter arithmetischer Weise verwendet werden.

Beispiel:

`> 0.25*2+3` ergibt nach `return` die Ausgabe in der R Console
```
[1] 3.5
```

`> 0.25*(2+3)` ergibt nach `return` die Ausgabe in der R Console
```
[1] 1.25
```

Dem Anwender stehen eine Vielzahl von mathematischen Funktionen und Operationen zur Verfügung. In der folgenden Tabelle ist eine Auswahl von grundlegenden Funktionen und Operationen zusammengefasst. Zur genaueren Erläuterung der Funktionen (insbesondere für die Erklärung der Argumente und Parameter der Funktionen) und für Beispielaufrufe der Funktionen sei auf die `help()`-Funktion verwiesen.

R-Funktion	Bedeutung
%/%	ganzzahlige Division
%%	Modulo Division
`round()`	kaufmännisches Runden
`floor()`	Abrunden
`ceiling()`	Aufrunden
`sqrt()`	Quadratwurzel
`cos()`	Cosinus
`sin()`	Sinus
`tan()`	Tangens
`cot()`	Cotangens
`exp()`	Exponentialfunktion
`log()`	natürlicher Logarithmus
`logb()`	Logarithmus zu einer beliebigen Basis
`choose()`	Binomialkoeffizient
`factorial()`	Fakultät
`abs()`	Absolutbetrag

Die Zahlen π und e sind über `pi` und `exp(1)` verfügbar.

Beispiel:

```
> round(exp(1),2)
[1] 2.72
```

`Inf` und `-Inf` stehen für ∞ bzw. $-\infty$. Der Wert `NaN` beschreibt eine nicht definierte Größe und `NULL` die leere Menge.

Die Anwendung von R ist natürlich nicht auf diese Taschenrechner-Funktion der R Console beschränkt. Der nächste Schritt in der effizienten Anwendung von R ist die Verwendung von Variablen, in denen entsprechende Größen gespeichert werden können. Im Hinblick auf die Objektorientierung von R sind solche Größen ganz allgemein Objekte wie (Daten-)Vektoren, Matrizen, Listen, statistische Modelle, Grafiken usw. Hierbei spielt insbesondere der Vektorbegriffe eine zentrale Rolle, da Daten zu verschiedenen Beobachtungen prinzipiell in einem Vektor erfasst werden und viele R-Funktionen auf Vektoren angewandt werden können bzw. ganz spezielle Vektorfunktionen zur Verfügung stehen. Für erweiterte Datenstrukturen wird das Vektorkonzept dann auf Matrizen bzw. Listen (Datentabellen, data frames) angehoben und es werden erweiterte Funktionen betrachtet.

Ausgehend von der reinen Taschenrechnerfunktion der R Console gelangt man zu folgenden, erweiterten Arbeitsweisen mit R.

1. Verwendung des Variablensystems zur Speicherung von Datenstrukturen bzw. allgemein von Objekten
2. Bearbeitung von Datenstrukturen (z.B. Selektionen aus Datentabellen)
3. Anwendung von Grafik-Funktionen
4. Anwendung von Statistik-Funktionen
5. Programmierung von eigenen Funktionen

2.5 Arbeiten mit Skriptfiles

Bei größeren R-Anwendungen und längeren Porgrammcodes ist das alleinige Arbeiten mit der R-Console nicht angemessen.

Eine fortgeschrittene Arbeitsweise ist die Befehleingabe in einem Skriptfile, aus dem die Befehle oder auch Teil-Befehle in die R-Console zur Ausführung kopiert werden können. Die Skript-Files können als Text-Files gespeichert werden. Mit einer systematischen Ablage von Sktiptfiles können Datenanalysen lückenlos dokumentiert werden und es ist jederzeit möglich, Analyseprozesse zu wiederholen.

Die Verwendung von Skriptfiles (Laden, Speichern, etc.) wird in der R-Console über die Menüleiste gesteuert.

Eine Alterantive bzw. Ergänzung zu dem Arbeiten mit Skriptfiles mithilfe der R-Console ist die Befehlseingabe über spezielle Editoren, wie z.B. R-WinEdt oder Tinn-R, oder die Verwendung einer GUI wie RStudio. RStudio unterstützt den Anwender durch die grafische Gestaltung und Zusatzfunktionalitäten in der Organisation, Verwaltung

und Durchführung von Programmierprojekten. Vorteile solcher Editoren bzw. GUI sind z.B. Unterstützung bei Programmtests, automtische Code-Ergänzung, Code-Korrektur, Hilfe bei der package-Verwaltung, Optimierung des Datenmanagements, Unterstützung einer Projektoganisation usw. Insbesondere bei umfangreichen Programmierprojekten mit R werden solche Editoren bzw. GUI verwendet.

R-Skripte (Programme) können auch als `.R`-Datei abgespeichert (über Menü) und mittels dem Befehl

```
source(Pfad/Filename.R)
```

direkt in das System eingelesen und ausgeführt werden.

2.6 Workspace und Speicherung

Alle während einer R Sitzung vom Anwender erzeugten Objekte (Daten, Funktionen etc.) werden temporär im sogenannten Workspace unter dem Arbeitsverzeichnis abgelegt. Das Arbeitsverzeichnis von R ist das Verzeichnis des Rechners, aus dem R aufgerufen wurde. Mittels `getwd()` kann der Pfad des Arbeitsverzeichnis abgefragt und durch `setwd()` geändert werden.

Pfadangaben: Das Zeichen \ bildet in R ein Sonderzeichen. Deshalb muß für Pfadangaben unter dem Windows-Betriebssystem anstelle des Windows-üblichen backslash \ entweder ein slash / oder ein doppeltes backslash-Zeichen \\ verwendet werden.

Beim Beenden von R (mit `q()`) frägt das System, ob der aktuelle Workspace gespeichert werden soll. Wählt man die Speicherung, wird der Workspace (d.h. z.B. die erzeugten Datentabellen) in der Datei `.Rdata` und die zuletzt in die Konsole eingegebenen Befehle (sogenannte history) in die Datei `.Rhistory` im Arbeitsverzeichnis abgelegt. Bei einem erneuten Start von R wird der aktuell im Arbeitsverzeichnis gespeicherte Workspace hochgeladen. Alternativ kann das Speichern des Workspace mit dem Befehl `save.image()` und das Laden mittels `load()` durchgeführt werden.

Die Funktion `ls()` zeigt alle Objekte des aktuellen Workspace an. Mit `rm()` können Objekte aus dem Workspace gelöscht werden.

2.7 Variablen, Datentypen und Datenstrukturen

2.7.1 Zuweisung von Variablen

Ein Objekt (z.B. Datenvektor) wird einer Variablen mittels dem Zuweisungsoperator

`<-`

(alternativ auch mit `=`) zugewiesen und damit in der Variablen mit einem gewählten Variablennamen gespeichert. Auf den Inhalt der Variablen kann dann durch Verwendung des Variablennamens immer wieder zugegriffen werden.

Beispiel:

```
> Radius <- 3
> 2*Radius*pi
[1] 18.84956
> Radius
[1] 3
```

Durch den Befehl `Radius <- 3` wird in die Variable mit Namen `Radius` der Zahlenwert 3 geschrieben. Auf diesen Wert kann nun durch Eingabe von `Radius` zugegriffen werden und z.B. in weiteren Berechnungen verwendet werden.

Die Ausgabe des Inhalts einer Variablen `Var` erfolgt durch

`> Var` oder `print(Var)`.

Die Variable und deren Inhalt bleibt die gesamte R-Sitzung im sogenannten Workspace gespeichert. Beim Verlassen von R durch `q()` erhält man die Option, diesen Workspace zu sichern und kann dann in folgenden Sitzungen wieder auf die Variable zugreifen. Durch den Befehl `rm()` kann man eine Variable während der Sitzung löschen. Mit der Eingabe der Funktion `ls()` erhält man alle während der aktuellen R-Sitzung im Workspace gespeicherten Variablen bzw. allgemeiner aller gespeicherten Objekte.

Bei der Namensgebung von Variablen bzw. allgemein von Objekten müssen bestimmte Konventionen beachtet werden (das erste Zeichen muss ein Buchstabe sein und außer dem Punkt (`.`) dürfen keine weiteren Sonderzeichen verwendet werden) und man beachte, dass in R Groß - und Kleinschreibung unterschieden wird.

Aus Gründen der Übersichtlichkeit haben sich typische Namensgebungen vom Typ `d.wetter2011` als nützlich erwiesen. So kann etwa aus dem Namen `d.wetter2011` abgelesen werden, dass es sich um Daten (Präfix: d.) bzgl. dem Wetter aus dem Jahr

2011 handelt. Man beachte, dass dies aber keine geforderte Syntax ist, sondern die Namensgebung vielmehr innerhalb der Regeln völlig frei ist. Bei größeren Datenanalyse-Projekten erleichtert eine gewisse Namensgebungs-Konvention die Übersichtlichkeit und Transparenz allerdings erheblich.

In Variablen können nicht nur numerische Werte, sondern z.B. auch Werte vom character-Typ (Zeichenketten, String-Typ) geschrieben werden. Dabei ist zu beachten, dass Zeichen(ketten) stets in Hochkomma `" "` zu schreiben sind.

Beispiel:

```
> Name <- "Gauss"
> Name
[1] "Gauss"
```

2.7.2 Datentypen

In R werden folgende Klassen atomarer (nicht weiter unterteilbare) Datentypen unterschieden.

Datentyp	Beschreibung	Beispiel
`Null`	leere Menge	`NULL`
`logical`	logische Werte	`TRUE` oder `FALSE`
`numeric`	relle Zahlen	`42.17`
`complex`	komplexe Zahlen	`5+2i`
`character`	Zeichenfolgen	`"Wort"`

Wenn man den Sonderfall des Datentyps `Null` vernachlässigt, erkennt man, dass die Tabelle der Datentypen hierarchisch (von unten nach oben) aufgebaut ist. D.h. ein Datentyp in der Tabelle kann jeweils durch die Datentypen, die sich in der Tablle in den Zeilen darunter befinden, repräsentiert werden. Ein Datentyp kann aber nicht durch Datentypen, die sich in Zeilen über dem betreffeden Datentyp befinden dargestellt werden.

Der Datentyp eines Objekts kann durch die Funktion `mode()` abgefragt werden.

Beim Zusammenführen von Datenstrukturen, die jeweils einen unterschiedlichen Datentyp beinhalten, wird für die erstellte gemeinsame Datenstruktur jeweils der Datentyp vewendet, der in der Tabelle am weitesten unten steht.

Beispiel:

Wird ein numerischer Vektor (d.h. ein Vektor mit numerischen Komponenten) und ein character-Vektor (d.h. ein Vektor mit character-Komponenten) verbunden, erhält man

einen character-Vektor.

Mittels der R-Funktion `str()` kann die Datenstruktur (inklusive des Datentyps) eines Objekts angezeigt werden.

Mit den Funktionen

`as.DATENTYP()`,

wobei `DATENTYP` durch einen R-Datentyp (z.B. `character`) zu ersetzen ist, kann ein Objekt in den nach `DATENTYP` gewählten Datentyp transformiert werden.

Beispiel:

```
> Var1 <- 120
> Var1
[1] 120
> Var2 <- as.character(Var1)
> Var2
[1] "120"
```

Mit den Funktionen `is.DATENTYP()` können Objekte auf den entsprechenden Datentyp getestet werden.

R-interner Speichermodus

Objekte des Datentyps `numeric` können in R als `integer` für Ganzzahlen und als `double` für reele Zahlen in doppelter Maschinengenauigkeit abgespeichert werden. Der Speichermodus kann durch `typeof()` ermittelt werden.

Entsprechend den Funktionen zu Datentypen können auch die Funktionen

`is.SPEICHERMODUS()` und `as.SPEICHERMODUS`

verwendet werden.

Der nicht-atomare Datentyp Faktor

Mittels der R-Funktion `factor()` kann ein Objekt (z.B. ein Datenvektor) als Ausprägungen eines Faktors, d.h. eines qualitativen Merkmals, generiert werden. Die verschiedenen Ausprägungen des Faktors werden mit Stufen (`Levels`) bezeichnet. Die Faktor-Eigenschaft des Datenvektor-Objekts wird dann bei der Anwendung verschiedener grafischer und statistischer R-Funktionen automatisch entsprechend berücksichtigt.

2.7.3 Datenvektoren

In R können Vektoren von einem beliebigen Datentyp sein, allerdings müssen alle Komponenten des Vektors den gleichen Datentyp besitzen. Die Darstellung von n Beobachtungen $x_1, x_2, \ldots, x_n$ eines Merkmals M in einem $n-$dimensionalen (Daten-)Vektor $(x_1, x_2, \ldots, x_n)$ ist eine fundamentale Vorgehensweise innerhalb von R.

Erzeugen eines Datenvektors

Mittels der Funktion `var1 <- c(`$x_1, x_2, \ldots, x_n$`)` kann man einer Variablen mit Namen `var1` einen Datenvektor mit den Vektorkomponenten (= Beobachtungen) $x_1, x_2, \ldots, x_n$ zuweisen. Für die praktische Anwendung in der Datenanalyse sind vor allem die Fälle, dass die Beobachtungen $x_1, x_2, \ldots, x_n$, numerisch sind, Zeichenketten darstellen oder Komponenten vom Faktor-Typ sind, relevant.

Beispiel:

```
> daten <- c(1,4,5.5,2,100)
> daten
[1] 1.0 4.0 5.5 2.0 100.0
```

Durch den Befehl `daten <- c(1,4,5.5,2,100)` erzeugt man also die Variable (oder allgemein das Objekt) mit Namen `daten`, in der der entsprechende Datenvektor gespeichert ist.
Ganz analog kann man einen Datenvektor mit Zeichenketten-Komponenten erzeugen, wobei man für jede Komponente die Schreibweise mit Hochkomma beachten muss.

Beispiel:

```
> stadt <- c("New York","Berlin","Tokio")
> stadt
[1] "New York" "Berlin" "Tokio"
```

Mithife der `c()`-Funktion (in Anlehnung an *combine* oder *concatenate*) können mehrere Vektoren zu einem Vektor kombiniert werden.

Beispiel:

```
> x <- c(1,2,3)
> y <- c(9,8,7)
> z <- c(x,y)
> z
[1] 1 2 3 9 8 7
```

oder

```
> x <- c(1,2,3)
> y <- c("A","B")
> z <- c(x,y)
> z
[1] "1" "2" "3" "A" "B"
```

Bemerkung: Es ist zu beachten, dass jeweils alle Komponenten eines Vektors immer vom selben Datentyp sein müssen.

Vektorfunktionen:

Die in 2.1 angegebenen Operatoren für die Grundrechenarten und elementaren Funktionen (z.B. `sin()`) können auch auf Datenvektoren angewendet werden. In der Regel finden die Berechnungen bei einem Vektor dann immer komponentenweise statt.

Beispiel:

```
> vec1 <- c(1,2,5,5,1)
> vec2 <- c(0,3,3,1,1)
> erg <- vec1 + vec2
> erg
[1] 1 5 8 6 2
```

```
> round(exp(vec2),1)
[1] 1.0 20.1 20.1 2.7 2.7
```

Zusätzlich zu diesen Funktionen gibt es eine Vielzahl von R-Funktionen, die speziell für Datenvektoren als Funktionsargument entwickelt sind, vgl. folgende Tabelle. Für die genauere Erläuterung der Syntax und für Beispiele sei auf die R-Hilfefunktion verwiesen.

R-Funktion	Bedeutung
`sum()`	Summe der Komponenten
`prod()`	Produkt der Komponenten
`t()`	Transponieren des Vektors
`cumsum()`	Kumulierte Komponentensumme
`cumprod()`	Kumuliertes Komponentenprodukt
`length()`	Länge des Vektors
`min()`	Minimum der Komponenten
`max()`	Maximum der Komponenten
`sort()`	Sortiert den Vektor aufsteigend
`sort(x,decreasing=TRUE)`	Sortiert den Vektor `x` absteigend
`order()`	Sortierindex
`rank()`	Ränge der Komponenten
`diff()`	Vektor der Komponentendifferenzen
`rep()`	Wiederholung von Komponenten
`seq()`	Indexvektor

Beispiele:

```
> x <- c(1,1,2,5)
> y <- c(2,2,1,0)
> sum(x)
[1] 9
> sum(x)/length(x)
[1] 2.25
> x-y
[1] -1 -1  1  5
> rank(x)
[1] 1.5 1.5 3.0 4.0
> max(x)
[1] 5
> min(x)
[1] 1
> min(x,y)
[1] 0
###

> z <- rep(x="A",times=5)
> z
[1] "A" "A" "A" "A" "A"
###
```

```
> s <- seq(from=-5, to=25, by=5)
> s
[1] -5  0  5 10 15 20 25
```

Indizierung und Sequenzen:

Man kann auf einzelne Komponenten eines Datenvektors `x` zugreifen, indem man die Nummer der Komponente (= Indizierung) bzw. den Vektor der Nummern der betreffenden Komponenten in eckigen Klammern `[ ]` dem Vektornamen nachstellt.

Beispiel:

```
> xvec <- c(1,1,4,5,2,10)
> xvec[3]
[1] 4
> xvec[c(1,3,6)]
[1] 1 4 10

> yvec <- c("A","B","D")
> yvec[2]
[1] "B"
```

Ein innerhalb `[ ]` dem Index(vektor) vorangestelltes − Zeichen bewirkt einen Ausschluss der entsprechenden Elemente.

Steht eine Indizierung auf der linken Seite einer Zuweisung, können die entsprechenden Elemente eines Vektors ersetzt bzw. gesetzt werden.

Beispiel:

```
> x <- c(1,2,3,4,5)

> x <- c(10,20,30,40,50)
> x[-c(1,5)]
[1] 20 30 40

> x[c(2,3)] <- c(99,100)
> x
[1]  10  99 100  40  50
```

Mithilfe der `[ ]`-Syntax können auch Vektorelemente über eine logische Bedingung selektiert werden.

Beispiel:

```
> x <- c(1,1,1,5,6,7)
> x[x > 2]
[1] 5 6 7
```

Mittels dem Operator `:` kann man Sequenzen bilden und erhält so z.B. schnell einen Vektor aufeinanderfolgender natürlicher Zahlen. So ergibt der Befehl `x <- 1:5` den Vektor `x` mit den Komponenten 1, 2, 3, 4, 5.
Mit der R-Funktion `rep(x,n)` erzeugt man einen Vektor, dessen Komponenten aus der `n`-fachen Wiederholung des Vektors `x` bestehen.

Beispiel:

```
> x <- rep(2:4,3)
> x
[1] 2 3 4 2 3 4 2 3 4
```

Spezielle Funktionen für character-Objekte:

In der folgenden Tabelle sind einige elementare Operationen für charakter-Objekte, z.B. Vektoren mit character-Komponenten, aufgeführt.

R-Funktion	Bedeutung
`paste()`	Zusammensetzten von Zeichenketten
`strsplit()`	Umkehrung von `paste()`
`substring()`	Erstellen eines Vektors von Teil-Zeichenketten
`nchar()`	Längen der im Argument auftretenden Zeichenketten

Beispiel:

```
> a <- paste("Vorname","Nachnahme")
> a
[1] "Vorname Nachnahme"
> b <- paste("Vorname","Nachnahme",sep="+")
> b
[1] "Vorname+Nachnahme"
> c <- substring(text=b,first=1,last=7)
> c
[1] "Vorname"
> nchar(c)
[1] 7
```

Zur Text-Ausgabe in der R-Console werden vor allem die Funktionen

`print()` und `cat()`

verwendet. Die Funktion `cat()` konvertiert alle durch Kommata getrennt angegebenen Funktionsargumente in Zeichenketten, hängt diese unter Verwendung des im Parameter `sep` (default ist das Leerzeichen) spezifizierten Trenn-Zeichens zusammen und gibt das Ergebnis aus.

Beispiel:

```
> x <- 500
> cat("Die Zahl ist:", x, "\n")
#
# "\n" als letztes Argument bewirkt einen Zeilenumbruch nach cat()
#
Die Zahl ist: 500
```

2.7.4 Funktionen für Objekte vom Datentyp: Faktor

Ein numerischer oder character Vektor `x` wird durch `factor(x)` oder `as.factor(x)` in einen Datenvektor vom Typ `factor` umgewandelt. Die Faktorstufen (*Levels*) entsprechen methodisch den Ausprägungsklassen eines nominalen oder ordinalen Merkmals.

Mit der Funktion `levels` können neue Bezeichnungen der Faktorstufen definiert werden, zusätzliche Stufen eingeführt werden und Faktorstufen zusammengelegt werden. Solche Arbeitsschritte nennt man *Recodierung* des Faktor-Objekts.

Beispiel:

```
> x <- c(1,1,1,2,1,3,3,4)
> x
[1] 1 1 1 2 1 3 3 4

> y <- as.factor(x)
> y
[1] 1 1 1 2 1 3 3 4
Levels: 1 2 3 4

> levels(y) <- c("sehr gut","gut","befriedigend","ausreichend")
> y
[1] sehr gut     sehr gut     sehr gut     gut          sehr gut
```

```
[6] befriedigend befriedigend ausreichend
Levels: sehr gut gut befriedigend ausreichend
```

Mithilfe der `levels()`-Funktion können die Faktorstufen (Levels) gruppiert und zusammengelegt werden.

Beispiel:

```
> x <- c("A","A","B","C","A","C","B","A")
> x <- as.factor(x)
> x
[1] A A B C A C B A
Levels: A B C
> y <- x
> y
[1] A A B C A C B A
Levels: A B C
> levels(y) <- c("A","nicht-A","nicht-A")
#Beachte: identische Komponentenanzahl bei (A,B,C) und
#("A","nicht-A","nicht-A") - Zuordnung in Reihenfolge
> y
[1] A       A       nicht-A nicht-A A       nicht-A nicht-A A
Levels: A nicht-A
```

Die Umbenennungen und Zusammenlegungen (Recodierung) von Faktorstufen wird durch eine Listen-Zuweisung innerhalb der Funktion `levels` wesentlich anwenderfreundlicher, da bei diesem Vorgehen die Reihenfolge der auftretenden Stufen innerhalb des Datenvektors nicht beachtet werden muss.

Beispiel:

```
> x <- c("2","1","1","4","3","2")
> x <- factor(x)
> x
[1] 2 1 1 4 3 2
Levels: 1 2 3 4
> levels(x) <- list(A="1",B="2",C="3",C="4")
> x
[1] B A A C C B
Levels: A B C
```

Mit dem Befehl `cut()` kann eine Klassenbildung bei numerischen Merkmalen durchgeführt werden. Dabei wird ein numerischer Vektor in einen Vektor des Faktor-Datentyps transformiert.

Beispiel:

```
> x <- c(1,1,1,2,2,3,3,4,4,5)
> y <- cut(x,breaks=c(0,2,4,6),labels = c("top","mid","low"),right=TRUE)
> y
 [1] top top top top top mid mid mid mid low
Levels: top mid low
```

Der logische Parameter `right=TRUE` (default) gibt an, dass zur Klasseneinteilung links-offene und rechts-abgeschlossene Intervalle verwendet werden. Die Intervallgrenzen werden im Parameter-Vektor `breaks` definiert.

Häufigkeitstabellen:

Die Funktion `table()` erstellt die Tabelle der absoluten Häufigkeiten jedes Levels in einem Faktor-Vektor. Bei zwei Argumenten erfolgt die Ausgabe der Kontingenztafel aller Levelkombinationen. Bei mehr als zwei Argumenten werden die Häufigkeiten aller Levelkombinationen durch mehrere Kontingenztafeln angegeben. Die Funktion `table()` kann auch ganz analog auf numerische und character Datenvektoren angewendet werden.

Beispiel:

```
> y <- factor(c("m","m","m","w","m","w","w","w"))
> x <- factor(c("A","A","B","C","A","C","B","A"))

> table(x)
x
A B C
4 2 2

> table(x,y)
   y
x   m w
  A 3 1
  B 1 1
  C 0 2
```

Mit der Funktion `prop.table()` können relative Häufigkeitstabellen erzeugt werden. Die Funktion wird dazu auf Häufigkeitsvektoren angewendet, d.h. man wird i.A. vor der Funktion `prop.table()` zunächst die Funktion `table()` anwenden. Mittels `prop.table()` lassen sich auch Zeilen- und Spaltenhäufigkeiten von Kontingenztabellen berechnen, s.u.

Beispiel:

```
> x <- c(1,1,1,4,5,5,7,7,7,7)
> table(x)
x
1 4 5 7
3 1 2 4
> prop.table(table(x))
x
  1   4   5   7
0.3 0.1 0.2 0.4
```

Faktorengruppierte Funktionsauswertung mit `tapply()`:

Prinzipielle Syntax: `tapply(Vektor1,Vektor2,Funktion)`

Funktionsweise: Mehrfache Anwendung von `Funktion` auf das Argument `Vektor1` gruppiert nach den Faktor-Levels von `Vektor2`.

Beispiel:

```
> y <- factor(c("m","m","m","w","m","w","w","w"))
> x <- c(1,1,1,5,2,6,7,8)
> table(x,y)
   y
x   m w
  1 3 0
  2 1 0
  5 0 1
  6 0 1
  7 0 1
  8 0 1
> tapply(x,y,mean)
   m    w
1.25 6.50
```

2.7.5 Datenmatrizen

Das Konzept eines Datenvektors wird auf Datenmatrizen erweitert. Analog zu den Datenvektoren können die Elemente einer Datenmatrix immer nur einem Datentyp angehören.

Erzeugen einer Datenmatrix:

Mit dem Befehl `matrix(data,nrow=n,ncol=m,byrow=F)` kann eine $n \times m-$Datenmatrix (n Zeilen und m Spalten) definiert werden. Dabei bezeichnet `data` den Vektor der Elemente der Matrix, `n` die Anzahl der Zeilen, `m` die Anzahl der Spalten und der Parameter `byrow=F` (der die Default-Einstellug ist und auch weggelassen werden kann) gibt an, dass der Datenvektor `daten` spaltenweise in die Matrix geschrieben wird. Für einen zeilenweisen Eintrag des Datenvektors in die Matrix muss man `byrow=TRUE` setzten.

Beispiel:

```
> A <- matrix(c(1,2,3,4,5,6),2,3)
> A
      [,1]  [,2]  [,3]
 [1,]    1     3     5
 [2,]    2     4     6
```

Matrizen können auch durch Aneinanderreihen von Spalten (`cbind`) oder Zeilen (`rbind`) erzeugt werden.

Beispiel:

```
> A <- cbind(c(1,2),c(3,4),c(5,6))
> A
      [,1]  [,2]  [,3]
 [1,]    1     3     5
 [2,]    2     4     6
```

Indizierung:

Um auf das Element a_{ij} ($i-$te Zeile und $j-$te Spalte) einer Matrix A zuzugreifen, verwendet man den Befehl `A[i,j]`. Man erhält die gesamte $i-$te Zeile (das ist wieder ein Vektor) durch `A[i,]` bzw. die gesamte $j-$te Spalte (auch das ist wieder ein Vektor) der Matrix A durch `A[,j]`.

Beispiel:

```
> A <- cbind(c(1,2,3),c(4,5,6))
> A
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

> S1 <- A[,1]
> S1
[1] 1 2 3
```

Matrixfunktionen:

Die Grundrechenarten und viele Standardfunktionen sind bei Matrizen komponentenweise definiert. Darüber hinaus gibt es spezielle Funktionen, die für Matrix-Objekte konzipiert sind. In der folgenden Tabelle sind einige Beispiele solcher Funktionen angegeben. Als Argument der Funktionen wird dabei jeweils eine Matrix betrachtet. Für eine genauere Betrachtung und Auflistung von Matrixfunktionen sei an dieser Stelle an weiterführende Literatur bzw. die Hilfefunktion verwiesen. Innerhalb von R sind viele aus der linearen Algebra bekannten Matrix-Berechnungen (z.B. Matrizenmultiplikation, Inversenberechnung, Eigenwerte usw.) durch Funktionen unterstützt.

R-Funktion	Bedeutung
$\%*\%$	Matrizenmultiplikation
`diag()`	Abfragen und Setzen der Hauptdiagonalen einer Matrix
`det()`	Determinante
`cbind()`, `rbind()`	Matrizen spalten- bzw. zeilenweise zusammenfügen
`t()`	Transponieren einer Matrix
`dim()`	Anzahl Spalten und Zeilen
`qr()`	QR-Zerlegung
`dimnames()`	Zeilen- und Spaltennamen
`crossprod()`	effiziente Berechnung des Matrixkreuzprodukts $A^T B$
`solve()`	u.a. Invertieren einer Matrix
`eigen()`	Eigenwerte und -vektoren
`sum()`	Summe aller Matrixelemente
`rowSums()`	Vektor der Zeilensummen
`colSums()`	Vektor der Spaltensummen
`nrow()`	Anzahl der Zeilen
`ncol()`	Anzahl der Spalten

Beispiel:

```
> A <- cbind(c(1,2,3),c(4,5,6))
> A
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> S1 <- A[,1]
> S1
[1] 1 2 3

> B <- A%*%t(A)
> B
     [,1] [,2] [,3]
[1,]   17   22   27
```

```
[2,]   22   29   36
[3,]   27   36   45
> det(B)
[1] 0

> C <- cbind(c(B[1,1],B[2,1]),c(B[1,2],B[2,2]))
> C
     [,1] [,2]
[1,]   17   22
[2,]   22   29
> det(C)
[1] 9

# Zeilen- und Spaltenbezeichnungen mit der Funktion dimnames

# Matrix definieren

> daten <- matrix(c(30,40,25,56,30,46),nrow=2)
> daten
     [,1] [,2] [,3]
[1,]   30   25   30
[2,]   40   56   46

> dim(daten)
[1] 2 3

# Zeilen- und Spaltenbezeichnung mittels Listen-Struktur

> dimnames(daten) <- list(c("A","B"),c("u","v","w"))
> daten
   u  v  w
A 30 25 30
B 40 56 46
```

2.7.6 Datentabellen (Data frame)

In der praktischen Datenanalyse betrachtet man oft Datensätze, in denen pro Merkmalsträger (Beobachtung) mehr als ein Merkmal (Variable) beobachtet wurden (multivariate Stichproben). Für diesen Fall bilden *Datentabellen* (*data frame* = Daten Rahmen) die wichtigste Datenstruktur in R.

Eine Datentabelle (Datenrahmen) stellt eine Erweiterung einer Datenmatrix dar, indem die Spalten eines data frame aus verschiedenen Datentypen (in der Anwendung meist numerisch und Zeichenketten bzw. Datentyp: Faktor) bestehen können. Damit

können in einem Datenobjekt pro Merkmalsträger sowohl metrische Merkmale (numerischer Datentyp) als auch nominale bzw. ordinale Merkmale (Zeichenketten) erfasst werden. Datentabellen sind in R ein Objekt vom Typ: `Liste`. Datentabellen bilden den für die Anwendung wichtigsten Spezialfall der noch allgemeineren, übergeordneten Datenstruktur Liste. Listen sind die flexibelste Datenstruktur in R und können als Elemente Objekte unterschiedlicher Datenstruktur enthalten.

Erzeugen einer Datentabelle:

Die Erzeugung einer Datentabelle erfolgt mit der Funktion `data.frame()` und soll anhand eines konkreten Beispiels erläutert werden.

Man nehme an, dass man eine Datentabelle namens `Daten1` erzeugen möchte, in der von 5 Personen jeweils das Geschlecht (Ausprägungen: m und w) und das Gewicht in [kg] erfasst sind. Der entsprechende R-Befehl lautet:

```
> Daten1 <- data.frame(Geschlecht=c("m","w","w","m","m"),Gewicht=c(70,62,60,80,91))
```

Durch die Eingabe
`>Daten1`
gibt R in der Console das Datentabellen-Objekt aus:

```
  Geschlecht Gewicht
1          m      70
2          w      62
3          w      60
4          m      80
5          m      91
```

Die erste Spalte der Datentabelle wird automatisch erzeugt und gibt eine Indizierung der Beobachtungen (hier im Beispiel: 5 Merkmalsträger = Personen) an. In der zweiten und dritten Spalte befinden sich die Ausprägungen der beiden Merkmale (Variablen) Geschlecht und Gewicht. Jede Zeile entspricht einer Beobachtung und fasst alle Merkmals-Ausprägungen jeweils einer Person zusammen. Bei data frames werden Spalten vom character-Typ automatisch als Faktor deklariert.

Neue Spalten in einem data frame ergänzen und erzeugen:

Einer Datentabelle `daten` kann einer weiterer Datenvektor `vector` oder auch eine weitere Datentabelle `daten2` mittels

```
daten <- data.frame(daten,vector)
```

bzw.

```
daten <- data.frame(daten,daten2)
```

(spaltenweise) zugefügt werden. Dabei müssen die Zeilenanzahlen der Objekte übereinstimmen. Alternativ und kürzer kann einer Datentabelle `daten` eine weitere Spalte, d.h. ein Datenvektor-Objekt `Vektor` mit Namen `Spalte-neu`, durch

```
daten$Spalte-neu <- Vektor
```

zugefügt werden.

Aus einer vorhandenen Spalte (Variable) einer Datentabelle `daten` kann eine neue Spalte gebildet und an die Datentabelle angehängt werden.

Beispiel:

```
> Geschlecht <- c("m","m","w","m","w")
> Alter <- c(20,20,30,25,50)
> daten <- data.frame(Alter,Geschlecht)
> daten
  Alter Geschlecht
1    20          m
2    20          m
3    30          w
4    25          m
5    50          w

> Wohnort <- c("R","M","M","R","B")
> daten1 <- daten
> daten1$Ort <- Wohnort
> daten1
  Alter Geschlecht Ort
1    20          m   R
2    20          m   M
3    30          w   M
4    25          m   R
5    50          w   B

> daten$Alter.Monate <- daten$Alter*12
> daten
  Alter Geschlecht Alter.Monate
1    20          m          240
2    20          m          240
3    30          w          360
4    25          m          300
5    50          w          600
```

Bezeichnung der Spalten (Variablen):

Mittels der Funktion `names()` können die Bezeichnungen (Namen) der Spalten einer Datentabelle eingetragen bzw. geändert werden. Diese Funktion kann übrigens auch ganz analog für Datenvektoren angewandt werden. Bei Datenmatrizen kann man mithilfe der Funktion `dimnames()` den Zeilen und Spalten Namen zuordnen. Die Funktionen `colnames()` und `rownames()` geben die Spalten- bzw. Zeilennamen des Objekts zurück.

Beispiel:

```
> Daten1 <- data.frame(Geschlecht=c("m","w","w","m","m"),Gewicht=c(70,62,60,80,91))
> names(Daten1) <- c("gender","weight")
> Daten1
   gender  weight
1       m      70
2       w      62
3       w      60
4       m      80
5       m      91
```

Zusammenfassung einer Datentabelle:

Mithilfe der R-Funktion `summary()` wird eine Zusammenfassung einer Datentabelle erzeugt.

Beispiel:

```
> Daten.Schaden <- data.frame(Schaden=c(150,200,300,0,15),
+ Geschlecht=c("m","m","w","w","m"))
> summary(Daten.Schaden)
    Schaden    Geschlecht
 Min.   :  0   m:3
 1st Qu.: 15   w:2
 Median :150
 Mean   :133
 3rd Qu.:200
 Max.   :300
```

Bemerkung: Das `+` Zeichen in der zweiten Zeile des Beispiels ermöglicht in R einen Befehl auch über mehr als eine Eingabezeile zu schreiben.

Bei Datentabellen mit vielen Zeilen können die Funktionen `head()` und `tail()` hilfreich sein. Mit den beiden Funktionen kann man die ersten bzw. letzten Zeilen eines dataframes (oder Vektors) anzeigen lassen.

Beispiel:

```
> x <- c(1,2,1,3,4,5,6,7,8,9)
> head(x,2)
[1] 1 2
```

2.7.7 Arrays und Listen

Arrays (Felder) sind eine Erweiterung der zwei-dimensionalen Datenstruktur einer Datenmatrix auf beliebig große Dimensionen. Zur Erzeugung eines arrays verwendet man den R-Befehl `array()`.

Listen sind durch eine rekursive Struktur definiert und werden vor allem für fortgeschrittene Programmierprojekte verwendet. Listen können Elemente mit unterschiedlicher Datenstruktur enthalten und bilden eine sehr flexibel einsetzbare Datenstruktur. Die Festlegung einer Liste erfolgt mit der Funktion `list()`.

Dem Anwender begegnen Listen beim Auslesen von Einzelinformationen aus den Ausgabe-Objekten von R-Funktionen, die oft in Listen-Struktur angelegt sind. So kann z.B. auf Details von Signifikanztests oder Ergebnissen einer statistischen Modellbildung zugegriffen werden. Dazu verwendet man die übliche Syntax

```
[ ] oder auch $
```

Die Komponenten der Listen können entweder durch numerische Indizes oder über ihre Namen, also die bezeichnende Zeichenkette (character) ausgewählt werden. Die Index-Nummer oder den Namen einer interessierenden Komponente einer Ausgabe-Liste findet man am schnell über die R-`help`-Funktion angewandt auf die entsprechende R-Funktion. Die Komponenten der Ausgabe-Liste werden in der R-Hilfe unter `Value` aufgeführt.

Beispiel:

```
> x <- rnorm(n=100,mean=0,sd=5)

> t.test(x=x,mu=1)

        One Sample t-test

data:  x
t = -2.0333, df = 99, p-value = 0.0447
```

```
alternative hypothesis: true mean is not equal to 1
95 percent confidence interval:
 -1.0975203  0.9743948
sample estimates:
  mean of x
-0.06156278

>  mode(t.test(x=x,mu=1))
[1] "list"

> KI <- t.test(x=x,mu=1)[4]
> KI$conf.int
[1] -1.0975203  0.9743948
attr(,"conf.level")
[1] 0.95

> ki <- t.test(x=x,mu=1)$conf.int
> ki
[1] -1.0975203  0.9743948
attr(,"conf.level")
[1] 0.95

> mode(ki)
[1] "numeric"

> LS <- ki[1]
> RS <- ki[2]
> Breite <- RS - LS

> Breite
[1] 2.071915
> LS
[1] -1.097520
> RS
[1] 0.9743948
```

2.7.8 Selektive und gruppierte vektorwertige Anwendung von Funktionen

Die in der folgenden Tabelle aufgeführten Funktionen ermöglichen eine Anwendung von Funktionen auf Teilbereiche eines Datensatzes, z.B. einer Datentabelle. Diese Funktionen ermöglichen eine performante, vektorwertige Programmierung von fortgeschrittenen Projekten.

R-Funktion	Bedeutung
`merge()`	Zusammenführen von Datentabellen
`by()`	Anwenden einer Funktion auf Teilmengen einer Datentabelle
`apply()`	Vektorwertige Anwendungen von Funktionen auf Matrizen und Arrays
`lapply()`	Elementweise Anwendung einer Funktion auf Listen, Datentabellen und Vektoren
`sapply()`	analog `lapply()` mit simplifizierten Ausgabeobjekt
`tapply()`	Nach Faktor gruppierte Anwendung einer Funktion auf einen Vektor mit tabellarischer Ausgabe

2.8 Selektionen

Bei der Analyse eines Datensatzes, der in einer Datentabelle, d.h. einem data frame, vorliegt, ist man oft an Berechnungen für einzelne Spalten (d.h. Variablen) oder einzelne Zeilen (d.h. Beobachtungen) interessiert. So könnte man etwa im obigen Beispiel nach dem Durchschnittsgewicht aller Männer fragen. Um eine derartige Auswahl aus einer Datentabelle zu realisieren stehen in R verschiedene Funktionalitäten zur Verfügung. Es sei darauf hingewiesen, dass die Umsetzung von (insbesondere bedingten) Selektionen ganz allgemein eine Grundanforderung an Datenanalyse- (bzw. allgemeiner: an Datenbank- oder auch sogenannten Business Intelligence-) Systeme ist.

Zur Auswahl einer Spalte oder Zeile kann man die von Datenmatrizen bekannte Syntax mittels `[ ]` verwenden.

`Daten1[3,]` ergibt den Vektor, der aus der dritten Zeile (d.h. die dritte Beobachtung) der Datentabelle `Daten1` besteht.

`Daten1[,"Gewicht"]` ergibt den Vektor, der aus der Spalte der Datentabelle `daten1` mit dem Namen `Gewicht` besteht.

Das Ausschneiden einer Spalte aus einer Datentabelle kann man auch kürzer durch z.B. `Daten1$Gewicht` erreichen. Die allgemeine Syntax lautet also hier:

```
Name der Datentabelle $ Spaltenname
```

und bewirkt ein Herausschneiden der Spalte `Spaltenname` aus der Datentabelle.

Durch Anwendung des Befehls `attach(Name der Datentabelle)` wird dem System mitgeteilt, dass sich jeder folgende Aufruf eines Spaltennamens auf die im `attach`-Befehl genannte Datentabelle bezieht. D.h. z.B. nach dem Befehl `attach(Daten1)` verkürzt sich die Syntax zum Herausschneiden der Spalte `Gewicht` nochmals auf die Eingabe von `Gewicht`. Über den Befehl `detach(Name der Datentabelle)` wird die direkte Zugriffsmöglichkeit wieder aufgehoben.

Die ausgeschnittenen Spalten können über eine Zuweisung wieder in einem Datenvektor abgelegt werden.

Beispiel:
```
> v.Gewicht <- Daten1$Gewicht
> v.Gewicht
[1] 70 62 60 80 91
```

Bedingte Selektionen:

Unter einer bedingten Selektion versteht man z.B. die Fragestellung, dass man aus obiger Datentabelle alle Beobachtungen selektieren möchte, bei denen das Geschlecht weiblich ist oder bei denen das Gewicht größer als 65 [kg] ist.

Um solche Bedingungen formulieren zu können benötigt man in einer Programmier- bzw. Abfragesprache **Vergleichsoperatoren**

R-Operator	Bedeutung
==	gleich
!=	ungleich
<	kleiner
<=	kleiner oder gleich
>	größer
>=	größer oder gleich

und **logische Operatoren**

R-Operator	Bedeutung
&	logisches und
|	logisches oder
!	Negation (nicht)
xor	exklusives oder

Selektionen bei Datenvektoren:

Mittels der Vergleichsoperatoren, logischen Operatoren und der `[ ]`-Syntax können Komponenten eines Vektors bzgl. Bedingungen selektiert werden.

Beispiel:
```
> a <- c(1,2,3,4,5,6,7,8,9,10)
> b <- a[a >= 8]
> c <- a[a > 5 & a < 8]
> a
[1] 1 2 3 4 5 6 7 8 9 10
> b
```

```
[1] 8 9 10
> c
[1] 6 7
```

Selektion in Datentabellen durch Bilden von Teilmengen:

Mit dem Befehl

```
subset(Name der Datentabelle, Bedingung)
```

können aus einer Datentabelle diejenigen Beobachtungen (Zeilen) extrahiert werden, die die angegebene Bedingung erfüllen. Die Bedingung bezieht sich dabei auf Variablen (Spalten) der Datentabelle und wird mittels der logischen und/oder Vergleichs-Operatoren formuliert.

Beispiel:

```
> Daten2 <- data.frame(Geschlecht=
+ c("m","w","w","m","m"),Gewicht=c(70,62,60,80,91), Alter=c(20,25,30,24,35))
> Daten2
   Geschlecht  Gewicht  Alter
 1          m       70     20
 2          w       62     25
 3          w       60     30
 4          m       80     24
 5          m       91     35

Daten3 <- subset(Daten2,Geschlecht=="m" & Alter < 28)
> Daten3
   Geschlecht  Gewicht  Alter
 1          m       70     20
 4          m       80     24
```

Innerhalb der `subset()`-Funktion kann durch den optionalen Parameter `select` durch

```
subset(Name der Datentabelle, Bedingung, select=Spaltenname1:Spaltenname2:...)
```

festgelegt werden, welche Spalten (Variablen) der ursprünglichen Datentabelle in der selektierten Datentabelle weiter vorhanden sein sollen.

Beispiel:

```
Daten4 <- subset(Daten2,Geschlecht=="m" & Alter < 28,select=Gewicht:Alter)
> Daten4
   Gewicht  Alter
 1       70     20
 4       80     24
```

2.9 Eingabe und Einlesen von Daten

Neben den oben eingeführten Techniken zur Erzeugung von Datenvektoren oder Datentabellen gibt es einige Hilfsmittel zur bequemeren Eingabe von Datensätzen. In der Anwendung liegen die Daten oft schon außerhalb von R (z.B. als Excel-Datei, Datenbank-Format oder ASCII-Datei) vor und werden dann in R eingelesen.

2.9.1 Hilfsmittel bei der Dateneingabe

Mittels dem Befehl `scan()` kann man innerhalb der R Console zeilenweise einen numerischen Datenvektor eingeben. Um etwa in den Vektor `a` die Werte 1, 3 und 5 einzutragen würde man den Befehl

```
> a <- scan()
```

absetzen, dann die Komponenten zeilenweise mit `return` als Eingabebestätigung eingeben und mit einer Leereingabe mit Abschluß `return` die Eingabe beenden.

Ein Hilsmittel zur Dateneingabe in der Form einer Datentabelle stellt der **data editor** dar. Der data editor (Daten-Editor) ist eine grafische Oberfläche, die der aus der Tabellenkalkulation (z.B. MS Excel) bekannten Tabellenstruktur entspricht. Um mit dem data editor z.B. eine Datentabelle mit Namen `Daten` zu erzeugen, muss man zunächst dem Objekt `Daten` den data frame-Typ zuweisen. Dies erfolgt mit der Eingabe

```
> Daten <- data-frame()
```

Im zweiten Schritt kann man nun über das Menü `Edit\Data editor...` in der RGui den data editor starten. R meldet sich dann mit dem Frage-Fenster `Name of data frame or matrix`, in das man den Datentabellen-Namen, z.B. `Daten`, eingibt. Danach können die Daten in die angebotene Tabelle eingegeben werden (Bemerkung: Charakter-Werte müssen nicht mit Hochkomma eingegeben werden und der Datentyp Faktor wird bei der Eingabe von Zeichenketten automatisch generiert). Nach dem schließen der Eingabetabelle steht nun die Datentabelle (data frame), z.B. mit Namen `Daten` zur Verfügung.

Durch Anklicken des Spalten(Variablen)-Namens kann man den Typ der Variablen (des Datenvektors) festlegen.

Alternativ kann man die Datentabelle `Daten` über

```
Daten <- edit(data.frame())
```

anlegen und den Daten-Editor öffnen. Ist die Datentabelle `Daten` bereits definiert kann über den Befehl

```
fix(Daten)
```

der Daten-Editor gestartet werden. Im Editor durchgeführte Änderungen werden bei Schließen des Editors automatisch im Data Frame übernommen.

2.9.2 Einlesen von Daten aus externen Dateien nach R

Wenn die zu betrachtenden Daten bereits in einer Datei außerhalb des R-Systems vorliegen, möchte man die Daten natürlich nicht noch einmal eingeben, sondern die Fremd-Datei in R einlesen. Diese Fragestellung ist im allgemeinen Datenanalyse- bzw. Datenbanken-Prozess ein ganz zentrales Problem (in der Datenbankterminologie spricht man z.B. von ETL, d.h. Extraction, Transformation und Loading) und kann hier nur kurz angesprochen werden.

Für diese Problemstellung gibt es sehr mächtige, spezielle packages, die einen Datentransfer zwischen R und anderen Systemen ermöglichen. Für die Anbindung von R an die Datenbanksysteme MySQL, Access oder Oracle können z.B. die packages `RMySQL`, `RODBC` oder `ROracle` verwendet werden. Mit dem Paket `RODBC` kann auch eine komfortable Verbindung zu MS-Excel als Datenbank realisiert werden. Mithilfe des Pakets `foreign` können Binärdateien verschiedener Statistik-Systeme (z.B. SAS und SPSS) in R eingelesen werden. Für genauere Ausführungen zu dieser Thematik sei auf die Dokumentationen der entsprechenden Pakete und auf andere Veröffentlichungen (z.B. R-Manual R Data Import/Export) verwiesen.

Eine zusätzliche Dimension bei der Frage des Datenimport- und Export aus Systemen außerhalb von R ist die Tatsache, dass die anderen Systeme (Datenbanken etc.) oft auch noch auf anderen physischen Rechnern mit unterschiedlichen Betriebssystemen installiert sind. Eine typische Situation in der Finanzwirtschaft ist, dass die interessierenden Daten orginär auf einem Datenbanksystem eines Großrechners (sogenannter Host mit entsprechendem Host-Betriebssystem) liegen und man die Daten zur Analyse mit R auf einen PC mit Windows-Betriebssystem transformieren möchte.

Grundsätzlich ist (bei nicht zu großem Dateiumfang) durch die Transformation einer externen Datei, die in einem bestimmten Format (z.B. Datenbankformat) vorliegt, in eine Text-Datei (ASCII-Datei) der Austausch von Dateien zwischen verschiedenen Software-Systemen (und auch Hardware-Systemen) möglich. Diese in der Praxis oft verwendete Möglichkeit des Datentransfers zu R soll im Folgenden genauer dargestellt werden.

Dazu geht man nun davon aus, dass die Daten als ASCII-Datei (Text-Datei) auf dem gleichen Rechner wie R gespeichert vorliegen. Z.B. könnte auf dem Laufwerk `c:` eines

PC mit Windowsbetriebssystem im Ordner `Daten` die Datei `Personendaten.txt` gespeichert sein. Eine MS-Excel-Tabelle kann z.B. über das Excel-Menü `Datei\Speichern unter...` im Text-Format gespeichert werden, wenn man bei der Speicherung als Dateityp ein Text-Format (z.B. `Text (Tabs getrennt)`) auswählt.
Die abgespeicherte Datei im Text-Format kann mit einem Text-Editor geöffnet werden und hat z.B. folgendes Aussehen

```
Geschlecht  Alter
m           20
w           24
w           30
m           28
m           22
```

Das Einlesen der Text-Datei erfolgt nun mithilfe der R-Funktion `read.table()`. Hinsichtlich der vielen Möglichkeiten der Anpassung der Funktion `read.table()` sei wieder auf die R-Hilfe verwiesen. Die grundsätzliche Funktionsweise von `read.table()`wird anhand des folgenden Beispiels klar.

Beispiel:

```
> PD <- read.table("c:\\Daten\\Personendaten.txt",header=TRUE)
```

Nach diesem Befehl liegt die Datei, nun unter dem Namen `PD`, als R-Datei, genauer als Datentabelle (data frame) in R vor. Bei der Eingabe des Pfads der ursprünglichen Text-Datei ist zu beachten, dass doppelte backslash-Zeichen \\ zu verwenden sind. Der Zusatzparameter `header=True` gibt an, dass die erste Zeile der Datei als Variablen(Spalten)-Namen verwendet werden soll. Die Variablennamen werden dann automatisch in die R-Datentabelle übernommen. Tabellenspalten mit Zeichenketten-Einträgen werden in der R-Datentabelle automatisiert als Faktor-Typ deklariert.

Es können auch Daten aus der Zwischenablage in R eingelesen werden. Dies kann z.B. beim Einlesen von Datensätzen aus dem Internet hilfreich sein. Dazu wird die Datei im Text-Format einfach markiert und in die Zwischenablage kopiert. Das Einlesen in R erfolgt dann durch `read.table("clipboard")`.

Bei der `read.table()`-Funktion wird als default-Einstellung davon ausgegangen, dass die einzelnen Werte der in R einzulesenden Datei durch ein oder mehrere Leerzeichen bzw. durch Tabulatoren oder Leerzeilen getrennt sind. Diese Trennzeichen werden z.B. beim Abspeichern von MS-Excel-Tabellen im Text-Dateienformat erzeugt. Ein anderes in der Anwendung oft vorkommendes Dateiformat zum Transport und Einlesen von Dateien über verschiedene Software-Systeme hinweg ist das sogenannte `csv`-Format (comma separated value Format). Hier werden als Trennzeichen der Daten Kommata oder auch Strichpunkte verwendet. Die Dateien besitzen dann einen Namen mit Endung `.csv`. Das R-System stellt für dieses Dateiformat die Funktion `read.csv()` zur Verfügung (vgl. R-Hilfe).

In der folgenden Tabelle sind die grundlegenden Parameter der `read.table()`-Funktion zusammengefasst.

Parameter	Funktionalität
`header`	Logischer Wert, ob die erste Zeile Spaltennamen darstellt
`sep`	Spaltentrennzeichen in der externen Datei
`dec`	Dezimaltrennzeichen in der externen Datei
`quote`	Anführungszeichen in der externen Datei
`na.strings`	Vektor der Zeichen für fehlende Werte in der externen Datei
`colClasses`	Vektor der Datentypen der Spalten in der externen Datei

2.10 Speichern und Ausgabe von Datentabellen

Neben der direkten Anbindung von R an eine Datenbank oder eine Tabellenkalkulation kann für die Ausgabe einer Datentabelle als Tetx-File die R-Funktion `write.table()` verwendet werden. Diese Funktion bildet das entsprechnende Gegenstück zur oben eingeführten Funktion `read.table()` und besitzt ganz ähnliche Parameter. Mehr Details zur Funktion `write.table()` findet man wieder, wenn man die R-Hilfe `help(write.table)` aufruft.

2.11 Fehlende Werte und Ausreißer

2.11.1 Fehlende Werte (missing values)

In der praktischen Datenanalyse muß darauf geachtet werden, wie man mit fehlenden Werten (missing values) adäquat umgeht. Dies betrifft schon das Einlesen von Datensätzen, aber auch die weitere Analyse von Daten, z.B. die Berechnung von statistischen Maßzahlen.

Man spricht von fehlenden Werten, wenn in in einem Datensatz bei einer Beobachtung für eine bestimmte Variable kein Eintrag vorhanden ist. D.h. also, wenn in einer Datentabelle nicht alle Zeilen besetzt sind.

So könnten z.B. bei einer Befragung von Kunden (= Beobachtungen) einige Kunden zu ihrem Einkommen (= Variable) keine Antwort gegeben haben. Ein anderes Beispiel für fehlende Werte sind Vertagsdaten einer Versicherung, in denen nicht für alle Merkmale Werte vorhanden sind.

Ein fehlender Wert wird innerhalb von R mit `NA (Not Available)` bezeichnet und in R-Funktionen (meist) speziell berücksichtigt. Die Art der Berücksichtigung kann

oft über einen entsprechenden Funktionsparameter variiert werden. Das Setzen von `na.rm = TRUE` bewirkt etwa einen Ausschluss der missing values bei der Auswertung bestimmter Funktionen.

Beispiel:

```
> daten <- c(1,3,2,5,NA,1,NA,5)
> daten
[1]  1  3  2  5 NA  1 NA  5
> is.na(daten)
[1] FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE
> daten1 <- na.omit(daten)
> daten1
[1] 1 3 2 5 1 5
attr(,"na.action")
[1] 5 7
attr(,"class")
[1] "omit"
> sum(daten1)
[1] 17
> sum(daten)
[1] NA
```

In dem Beispiel wurde ein Datenvektor erzeugt, bei dem die fünfte und siebte Beobachtung jeweils ein fehlender Wert (`NA`) ist. Die Funktion `is.na()` liefert einen logischen Vektor mit FALSE bzw. TRUE für nicht-missing bzw. missing. Mit der Funktion `na.omit()` können missing values aus einem Datenvektor ausgeschlossen werden.

Bei Einlesen einer Datei in R mit der Funktion `read.table()` kann man die in der externen Datei verwendete Bezeichnung für fehlende Werte als Parameter `na.string` angeben.

Der Umgang von R-Funktionen mit fehlenden Daten ist unterschiedlich.
Arithmetische Operationen liefern als Ergebnis `NA`, falls ein Argument bei der Berechnung `NA` ist. Viele grafische Funktionen ignorieren fehlende Werte. Bei grundlegenden Funktionen, die Vektor- oder Matrixkomponenten zusammenfassen (z.B. `sum()`), gib es meist einen Funktions-Parameter `na.rm (NA remove)`, den man auf `TRUE` setzen kann bzw. die default-Einstellung `na.rm=TRUE` ist, und somit fehlende Werte aus den Berechnungen ausgeschlossen werden. Bei statistisch, methodisch höher entwickelten Funktionen (z.B. `t.test()`) gibt es einen analogen Parameter `na.action`, der auf `na.omit` (was auch die default-Einstellung ist) gesetzt bewirkt, dass fehlende Werte nicht in die Berechnungen einbezogen werden. Über den Parameter `na.action` können weitere (methodisch tieferliegende) Vorgehnsweisen zur Behandlung von fehlenden Werten (z.B. Ersetzen der fehlenden Werte) festgelegt werden.

Die konservative Standardmethode beim Umgang mit fehlenden Werten ist, alle Beobachtungen, bei denen mindestens ein missing value in den für die Untersuchung relevanten Variablen auftritt, aus dem Datensatz zu entfernen. Dies kann man sehr bequem mittels dem Befehl

```
Daten.t <- na.omit(Daten)
```

durchführen. In dem Dataframe `Daten.t` befinden sich nur noch die Zeilen von `Daten`, bei denen kein `NA` vorhanden ist.

Man beachte, dass das Einhalten der Reihenfolge

1. Auswahl der relevanten Variablen und Beobachtungen
2. Entfernen der Beobachtungen mit mindestens einem missing value

entscheidend ist.

Bei Datensätzen mit vielen Variablen kann das Entfernen aller Beobachtungen mit mindestens einem fehlenden Wert dazu führen, dass der Stichprobenumfang sehr klein wird.

Da beim Entfernen aller Beobachtungen mit mindetstens einem fehlenden Wert aus einem Datensatz Information verloren geht, verwendet man insbesondere in der statistischen Modellbildung (z.B. Regressionsanalyse) andere Methoden für den Umgang mit missing values. Man ersetzt z.B. die (rein zufällig) fehlenden Werte geeignet oder bildet eigene missing-Kategorien.

2.11.2 Ausreißer und falsche Werte

Vor einer statistischen Analyse sollten Beobachtungen, die Ausreißer oder falsche Werte (in mindestens einer Variablen enthalten) aus dem Datensatz enfernt werden.

Es gibt verschiedene Definitionen darüber, welchen Wert man als Ausreißer bezeichnet.

Man verwendet z.B.:

Ausreißer sind: alle Beobachtungswerte einer Stichprobe, die größer als das obere Quartil plus das 1,5-fache des Quartilabstands sind und alle Beobachtungswerte einer Stichprobe, die kleiner als das untere Quartil minus das 1,5-fache des Quartilabstands sind.

Diese Sichtweise wird etwa bei Boxplots unterstützt. Eine grafische Datenanalyse eignet sich (bei nicht zu großem Stichprobenumfang) i.A. sehr gut zur Ausreißer-Erkennung.

In der Literatur findet man verschiedene Ausreißer-Tests, die aber immer im Kontext der verwendeten Definition eines Ausreissers und der Verteilungsannahmen zu sehen sind. In der statistischen Modellbildung erhält man nach der Modellbildung mit Modell-Diagnoseverfahren die Möglichkeit Ausreißer zu erkennen und dann die Modellbildung mit einer modifizierten Stichprobe neu zu berechnen.

Für die praktische Anwendung ist ein Ausschluß von **falschen Werten** sehr wichtig. Man beachte, dass falsche Werte eventuell auch als Ausreißer erkennbar sind, aber auch ganz unauffällige Werte sein können. Für die Erkennung von falschen Werten sollten, falls möglich, sachlogische Regeln angewandt werden. So kann z.B. eine Temperatur von -325 [C°] nur ein falscher Wert sein.

2.12 Definition neuer Funktionen

Mit der R-Umgebung können nicht nur bereitgestellte Funktionen angewendet werden, sondern der Anwender kann selbst neue, eigene Funktionen definieren und zur Anwendung bereitstellen. Diese Eigenschaft von R hängt stark mit dem Programmiersprachen-Eigenschaften von R zusammen und bildet einen entscheidenden Unterschied zu rein Menü-geführten Analysesystemen. Die Möglichkeit eigene Funktionen zu erstellen ist z.B. mit der SAS-Makrosprache oder mit VBA innerhalb von MS-Excel vergleichbar.

An dieser Stelle soll nur eine sehr kurze Einführung in diese Thematik gegeben werden. Für detailiertere Informationen sei auf die entsprechenden R Manuals und auf weiterführende Literatur verwiesen.

Die grundlegende R-Syntax zur Definition einer Funktion lautet

```
> Name <- function(Argumente bzw. Parameter){ Funktionskörper }
```

Hierbei bezeichnet `Name` den gewählten Namen für die Funktion. `Argumente bzw. Parameter` sind die Bezeichnungen der Variablen, in die dann beim konkreten Aufruf der Funktion die entsprechenden Werte geschrieben werden und im `Funktionskörper` weiter verarbeitet werden können. Der `Funktionskörper` beinhaltet die eigentlichen Brechnungen (ein zulässiger R-Ausdruck, eventuell mit Kontrollstrukturen) und (oft) den Befehl `return()`, der bewirkt, dass die Funktion beendet wird und der Funktionswert in der R-Console ausgegeben wird. Ohne Ausgabe-Anweisung wird das Ergebnis des letzten Befehls im Funktionskörper an die Stelle des Funktionsaufrufs zurckgegeben.

Zuweisungsanweisungen in einem Funktionskörper generieren jeweils nur lokale, temporäre Objekte, d.h. die Objekte sind nur innerhalb der Funktionsabarbeitung existent. Nach der Funktionsabarbeitung sind die lokalen Objekte gelöscht. Die lokalen Objekte innerhalb einer Funktion haben, selbst bei identischen Namen, keine Auswirkung auf globale Objekte.

Die Funktion wird (über den `function()`-Befehl) automatisch unter dem vergebenen Namen im aktuellen workspace als function-Objekt gespeichert kann durch den Aufruf `Name()` mit Eintrag der entsprechenden Argumente bzw. Parameter verwendet werden. Die Argumente bzw. Paramter werden ohne explizite Nennung ihrer Reihenfolge nach zugeordnet.

Im folgenden, sehr einfachen Beispiel soll die prinzipielle Vorgehensweise verdeutlicht werden.

Beispiel:

```
> meine.funktion <- function(a,b) {
+ # Summe zweier Argumente
+ mein.ergebnis = a + b
+ return(mein.ergebnis)
+ }

> meine.funktion(17,25)
[1] 42
```

Mit obiger Befehlsfolge wurde die Funktion namens `meine.funktion` definiert, die zwei Argumente `a` und `b` besitzt und als Ergebnis die Summe der beiden Argumente ausgibt.

Es folgt ein Beispiel einer Funktion mit default Belegung.

Beispiel:

```
> Funktion1 <- function(a=10,b,c){erg = a*b*c; return(erg)}
# a default 10
> Funktion1(a=2,b=3,c=4)
[1] 24
> Funktion1(b=3,c=4)
[1] 120
> Funktion1(1,1,1)
[1] 1
```

Es folgt noch ein Beispiel einer selbst-definierten Funktion mit Textausgabe in der R-Console, wobei die Funktion `cat()` verwendet wird.

Beispiel:

```
> Kennzahlen <- function(x) {
+ min <- min(x)
+ max <- max(x)
+ cat("Minimum ist",min,"und","Maximum ist", max,"\n")
+ }
> v <- c(1,1,1,2,4,5,6)
> Kennzahlen(x=v)
Minimum ist 1 und Maximum ist 6
```

2.13 Kontrollstrukturen

Wie in jeder Programmiersprache können in R Verzweigungen und Schleifen programmiert werden. Diese Kontrollstrukturen können z.B. innerhalb von selbstdefinierten Funktionen verwendet werden.

2.13.1 Verzweigungen

a) Skalare Bedingung

`> if(Bedingung)` {`Ausdruck1`} else {Ausdruck2}

`Bedingung` ist hier ein R-Ausdruck, der ein skalares `logical`-Resultat liefert. Im Fall einer vektorwertigen `Bedingung` wird nur die erste Komponente beachtet (mit Warnung).

Beispiel:

```
> meine.funktion <- function(a,b) {
+ if(a < b){return("Erstes Argument ist kleiner!")}
+ else {return("Erstes Argument ist nicht kleiner!")}
+ }

> meine.funktion(40,60)
[1] "Erstes Argument ist kleiner!"
```

b) Vektorwertige Operationen

```
ifelse(Bedingung, Ausdruck1, Ausdruck2)
```

Neben der Bedingung können hier auch die `Ausdrücke` vektorwertig sein.

Beispiel:

```
> x <- c(1,1,9,5,2,3,4,7,6,8)
> y <- ifelse(x <= 5,"klein","gross")
> y
 [1] "klein" "klein" "gross" "klein" "klein" "klein" "klein" "gross" "gross"
[10] "gross"
```

c) Die Funktion `switch()`

Die Funktion `switch(EXPR,...)` wird oft verwendet, wenn eine große Zahl von Fällen unterschieden werden soll. Im Argument `EXPR` kann man über eine numerische, ganzahlige Angabe festlegen, welche Ausgabe in der angegeben Reihenfolge ausgewählt wird. Alternativ kann man `EXPR` als Zeichenkette setzen und damit die entsprechend benannte Ausgabe auswählen.

Beispiel:

```
> switch(3,"A","B","C","D","E")
[1] "C"

> switch("xx",x=0,y=1,xy=2,yx=3,xx=4,yy=5)
[1] 4
> switch("xxy",x=0,y=1,xy=2,yx=3,xx=4,yy=5)
NULL
```

2.13.2 Schleifen

a) `for`-Schleife

```
for(i in I){Ausdruck}
```

Im ersten Durchlauf der Schleife wird der Variablen `i` das erste Element in `I` zugewiesen und der `Ausduck` mit diesem Wert für `i` ausgeführt. Beim zweiten Durchlauf der Schleife wird der Variablen `i` das zweite Element in `I` zugewiesen und der `Ausduck` mit diesem Wert für `i` ausgeführt, usw.

Beispiel:

```
> J <- c(2,3,5,7,11,13,17)
> for(i in J){
+ erg <- i^2
+ print(erg)
+ }
[1] 4
[1] 9
[1] 25
[1] 49
[1] 121
[1] 169
[1] 289
```

Beispiel:

```
> x <- c()
> for(i in 1:10) {
+ x[i] <- i^2
+ }

> x
[1] 1 4 9 16 25 36 49 64 81 100
```

Man beachte, dass durch die Anweisung `x[i] <- i^2` bei jedem Schleifendurchlauf eine Komponente des 10-dimensionalen Vektors `x` gesetzt wird. Als erste Komponente 1^2, als zweite Komponente 2^2, als dritte Komponente 3^2 usw.

Als vorzeitige Abbruchbedingung (d.h. sofortiges Verlassen einer Schleife) wird der Befehl `break`, meist verbunden mit einer `Bedingung` in der Form

```
if(Bedingung) break
```

verwendet.

Beispiel:

```
> J <- c(2,3,5,7,11,13,17)
> for(i in J){
+ erg <- i^2
+ if(erg > 50) break
+ }
> erg
[1] 121
```

Der Befehl `next` bewirkt einen Sprung in die nächste Schleifen-Wiederholung.

b) `while`-Schleife

```
> while(Bedingung) {Ausdruck}
```

Der `Ausdruck` wird solange wiederholt, wie die `Bedingung` erfüllt ist.

Beispiel:

```
> i <- 1
> while(i <= 5) {
+ print(i*i);
+ i <- i + 1
+ }
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
```

In den fünf Schleifendurchläufen werden die Quadrate von $1, 2, \ldots, 5$ ausgegeben. Man beachte, dass im Beispiel der Zähler `i` pro Schleifendurchlauf durch die Anweisung
`i <- i+1`
jeweils um eins erhöht wird und die Schleife abbricht, wenn `i` den Wert 6 erreicht hat.

c) Wiederholungen mit `repeat{ }`

```
repeat{Ausdruck}
```

Der `Ausdruck` wird wiederholt. Ohne Abbruch-Bedingung, im Allgemeinen mit `if(Bedingung) break`, wird der `Ausdruck` in einer endlos-Schleife durchlaufen.

Beispiel:

```
> i <- 1
> repeat{
+ x[i] <- i
+ i <- i+1
+ if (i == 20) break
+ }
> x
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
```

d) Wiederholte Ausdrucksauswertung mit `replicate()`

Durch `replicate(n,Ausdruck)` wird der `Ausdruck` `n`-mal, $n \in \mathbb{N}$, wiederholt.

Bemerkung: da es sich bei R um eine Interpretersprache handelt, können Schleifen zu langen Programmlaufzeiten führen. In vielen Fällen ist eine vektorwertige Programmierung hinsichtlich der Laufzeitoptimierung zu bevorzugen. Man beachte, dass viele Bereiche der R-Syntax auch konsequent auf eine vektorwertige Programmierung ausgelegt sind.

Kapitel 3

Grafik mit R

In R stehen grundsätzlich zwei Grafik-Systeme zur Verfügung. Das Basis-Grafiksystem stellt Methoden bereit, die sich an den Möglichkeiten von Papier und Bleistift orientieren. Das erweiterte Lattice-Grafiksystem beinhaltet Methoden, die an einem Kamera-Objekt-Modell orientiert sind. Beide Grafiksysteme in R haben gemeinsam, dass bei den verschiedenen Grafikfunktionen über Parameterwahl die Grafiken vom Anwender vielfältig verändert und individualisiert werden können. Eine weitere wichtige Eigenschaft des R-Grafiksystems ist, dass grundlegende Grafikfunktionen generisch sind. Generische Funktionen liefern je nach dem Objekt-Typ des Arguments verschiedene Outputs.

In dieser Einführung in R soll keine detaillierte Ausarbeitung für die Grafik-Erstellung mit R gegeben werden, sondern nur ein Überblick über die grundlegenden Funktionen dargestellt werden. Mehr Informationen über das Lattice-Grafiksystem erhält man mit der R-Hilfe `help(Lattice)`. Im folgenden werden die grundlegenden Grafikfunktionen des Basis-Grafiksystems vorgestellt.

Die Grafikfunktionen können prinzipiell in drei Klassen aufgeteilt werden.

i) **high level-Funktionen** definieren eine neue Gafikausgabe.

ii) **low level-Funktionen** verändern eine gegebene Grafikausgabe.

iii) **Parametrisierungen** verändern die Grundeinstellungen des Grafiksystems.

3.1 Ausgabe von Grafiken

Vor der Erzeugung einer Grafik muss das Ausgabegerät (`device`) für die Grafik festgelegt werden. Ohne explizite Benennung des device wird die folgende Grafik in dem

in

```
options("device")
```

eingetragenem Gerät ausgegeben. Beim interaktiven Arbeiten mit R ist die default-Einstellung als device die Bildschirmausgabe (`windows()` oder X11()).

In der folgenden Tabelle sind einige, oft verwendete device-Spezifizierungen angegeben.

R-Funktion	Device bzw. Format
postscript()	PostScript
jpeg()	JPEG
pdf()	PDF
X11() und windows()	Grafik in Bildschirmfenster
win.print()	Druckerausgabe
win.metafile()	Windows Metafile
bmp()	Bitmap

Nach der Ausgabe einer Grafik mit dem gewählten Ausgabegerät (device) sollte im Allgemeinen das device wieder durch

```
dev.off()
```

geschlossen werden (im interaktiven R-Betrieb steht dann wieder die Bildschirmausgabe als default-device zur Verfügung).

Alternativ ist es bei R unter Windows möglich, eine im Bildschirmfenster erstellte Grafik menügesteuert (dazu: rechte Maustaste im Grafikfenster) in verschiedenen Formaten (z.B. PostSkript, Bitmap oder Windows Metafile) zu kopieren und zu speichern.

Beispiel:

```
> x <- c(1,2,3,4,5)
> pdf("grafik-test.pdf")
> plot(x)
> dev.off()
null device
          1
```

Die erstellte Grafik wird in der PDF-Datei `grafik-test` im aktuellen Arbeitsverzeichnis (workspace) gespeichert.

Werden in einer Anwendung mehrere Grafiken nacheinander erzeugt, überschreibt das System das Grafikfenster und man sieht jeweils nur die zuletzt erstellte Grafik. Mithilfe des Vorschaltens des Befehls

```
dev.new()
```

vor jeder neuen Grafik öffnet man jeweils ein neues Grafikfenster, in denen dann die Grafiken angezeigt werden.

3.2 High-level Funktionen

High-level Grafikfunktionen erzeugen vollständige Grafiken, die Koordinatenachsen, Beschriftungen und Daten-Darstellungen beinhalten. In der folgenden Tabelle sind einige, insbesondere für die statistische Datenanalysye grundlegende, "high level"-Funktionen aufgelistet. Man beachte, dass bei generischen Funktionen (allen voran die Basis-Funktion `plot()`) verschiedene R-Objekte als Funktionsargument möglich sind und die Funktion bei verschiedenen Objekttypen als Argument auch unterschiedliche Funktionalität besitzen kann. Weitere Informationen (z.B. zu den Argumenten bzw. Parameter der Funktionen) und Anwendungsbeispiele erhält man wieder über die R-Hilfe.

R-Funktion	Bedeutung
`plot()`	generische Funktion zur Grafikerstellung mit vielen Methoden
`plot(x,y)`	Streudiagramm (scatterplot) der Datenvektoren `x` und `y`
`plot(ecdf(x))`	Graph der empirischen Verteilungsfunktion des Datenvektors `x`
`plot(d)`	paarweise Scatterplots der numerischen Datenvektoren im data.frame d
`plot(mod)`	Diagnoseplots für ein Objekt mod vom Typ model
`barplot()`	Stabdiagramm
`pie()`	Kreissegment-Diagramm
`pairs()`	paarweise Streuungsdiagramme, Streudiagramm-Matrix
`coplot()`	Bedingte Streuungsdiagramme
`qqplot()`	Quantil-Quantil-Plot
`qqnorm()`	Normalquantil-Plot
`hist()`	Histogramm
`boxplot()`	Boxplot
`curve()`	Auswertung und Zeichnung des Graphen einer stetigen Funktion
`image()`	3-d-Plot: $z = f(x, y)$, z über Farbe skaliert
`contour()`	3-d-Plot: $z = f(x, y)$, Darstellung über Höhenlinien
`persp()`	3-d-Plot: $z = f(x, y)$, Darstellung als Fläche im Raum $\mathbb{R}^3$.

3.3 Low-level Funktionen

Low-level Grafikfunktionen initialisieren Grafiken und ermöglichen Erweiterungen bzw. Modifikationen von Grafiken, die mit High-level Grafikfunktionen erstellt wurden. Für eine schnelle interaktive Datenanalyse reichen meist die Standardgrafiken der High-level Grafikfunktionen aus.

In der nächsten Tabelle sind einige grundlegende "low level"-Funktionen und ihre Bedeutung zusammengefasst.

R-Funktion	Bedeutung
`points()`	generische Funktion zur Markierung von Punkten an spezifizierten Koordinaten
`lines()`	generische Funktion zur Verbindung von über Koordinaten vorgegebenen Punkten
`abline()`	einem Plot wird eine " schlaue " Gerade zugefügt
`polygon()`	Polygon mit festgelegten Ecken wird einem Plot zugefügt
`axis()`	Achsen werden einem Plot hinzugefügt
`grid()`	Gitternetz
`title()`	Überschrift
`text()`	Text an Koordinaten einfügen
`legend()`	Legenden-Block zufügen
`mtext()`	Text in den Rändern der Grafik einfügen

3.4 Grafik-Parametrisierung

Die R-Funktion `par()` legt die grundlegenden Parameter des R-Grafikausgabe (device) fest.

Eine häufig verwendete Umstellung der Default-Parameter ist in folgendem Beispiel dargestellt.

Beispiel:

Durch den Befehl `par(mfrow = c(n,m))` (n, $m \in \mathbb{N}$) wird das Grafikfenster für $n \cdot m$ Einzelgrafiken mit n Zeilen und m Spalten innerhalb einer Gesamt-Grafik definiert.

Für die Anfertigung von Standard-Grafiken können meist die Grundeinstellungen der Funktionen verwendet werden.

In der folgenden Tabelle sind einige häufig verwendete Parameter von Grafikfunktionen und innerhalb `par()` angegeben. Für mehr Informationen zur Grafik-Parametrisierung vgl. man die R-Hilfe `help(par)`.

Parameter	Bedeutung
`mfrow`	mehrer Grafiken (Matrixstruktur) in einer Grafikausgabe
`axes`	Koordinatenachsen zeichnen: `TRUE` oder `FALSE`
`xlab, ylab`	x-, y-Achsenbeschriftung
`las`	Ausrichtung der Achsenbeschriftung
`xlim, ylim`	Bereiche in x- und y-Richtung, die geplottet werden
`bg`	Hintergrundfarbe
`pch`	Symbol bzw. Zeichen für einen Punkt in einer Grafik
`cex`	Punkt- bzw. Buchstabengröße
`col`	Farben
`lty, lwd`	Linientyp, Linienbreite
`type`	Typ (`l`=Linie, `p`=Punkt,`b`=beides,`n`=nichts)
`grid`	Gitternetz
`title`	Überschrift
`main, sub`	Überschrift, Unterschrift

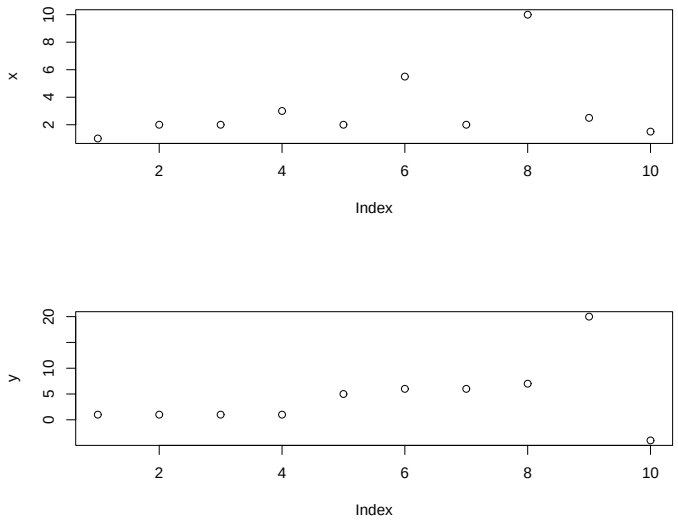
3.5 Praktische Grafikerzeugung

3.5.1 Beispiel

Zwei Vektoren werden gegen ihren Index geplottet. Die beiden Grafiken werden in einer Grafikausgabe erzeugt, vgl. Abbildung 3.1.

```
> x <- c(1,2,2,3,2,5.5,2,10,2.5,1.5)
> y <- c(1,1,1,1,5,6,6,7,20,-4)
> par(mfrow=c(2,1)) # Grafikausgabe in zwei Zeilen und einer Spalte
> plot(x)
> plot(y)
```

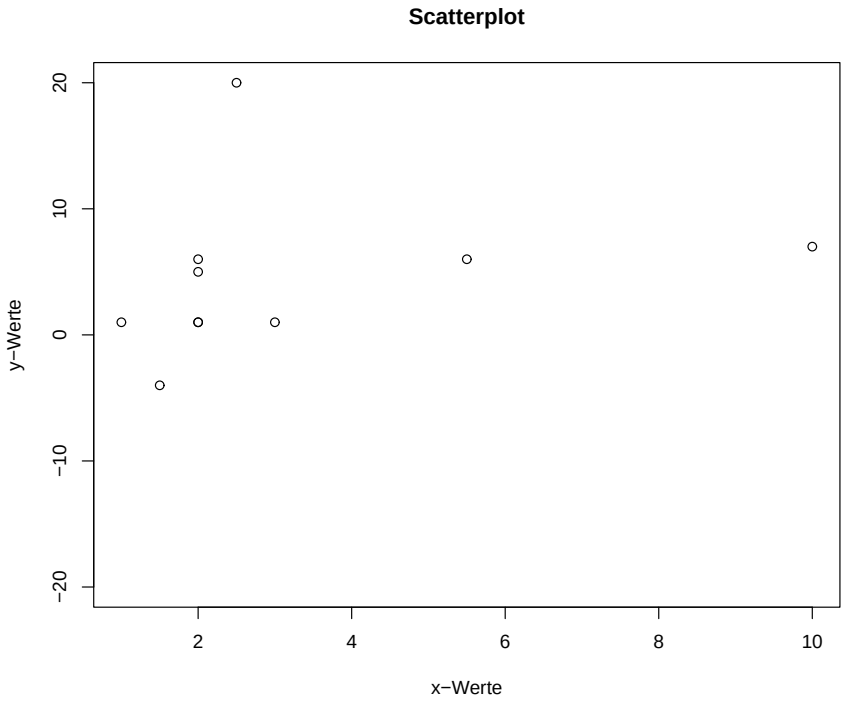
Abbildung 3.1: Grafikausgabe in Matrixstruktur.



Es folgt ein Streuungsdiagramm mit Überschrift, bei dem die Achsenbezeichnungen geändert werden und der Wertebereich der y-Achse vorgegeben wird, vgl. Abbildung 3.2.

```
> par(mfrow=c(1,1)) # Grafikausgabe: eine Grafik
> plot(x,y,main="Scatterplot",xlab="x-Werte",ylab="y-Werte",ylim=c(-20,20))
```

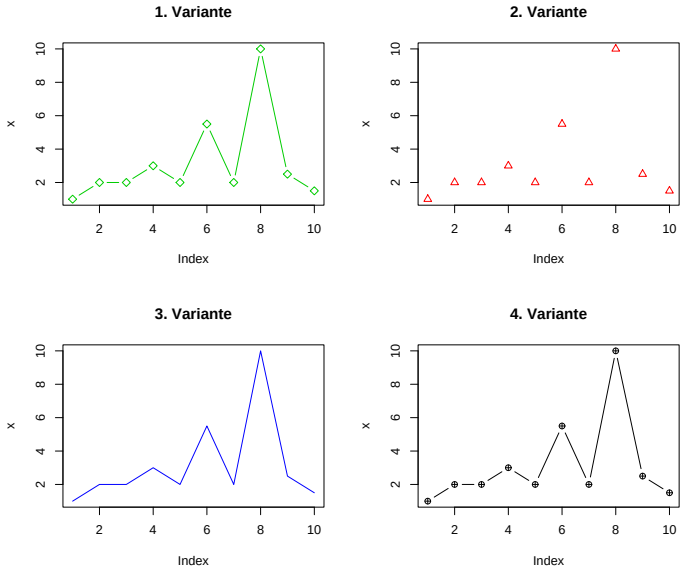
Abbildung 3.2: Angepasstes Streuungsdiagramm.



Die folgenden Variationen zeigen die Wirkung von `pch`, `col` und `type`. Die vier erzeugten Grafiken werden in einer Ausgabe (2 Zeilen und 2 Spalten) zusammengefasst, vgl Abbildung 3.3. Man beachte, dass innerhalb des vorliegende Textes keine Farbdarstellung umgesetzt ist. Beim Ausführen der Befehle erhält man entsprechend farbige Grafiken.

```
> par(mfrow=c(2,2))
> plot(x,main="1. Variante",pch=5,col=3,type="b")
> plot(x,main="2. Variante",pch=2,col=10,type="p")
> plot(x,main="3. Variante",col=20,type="l")
> plot(x,main="4. Variante",pch=10,col=1,type="b")
```

Abbildung 3.3: Variationen von `pch`,`col` und `type`.



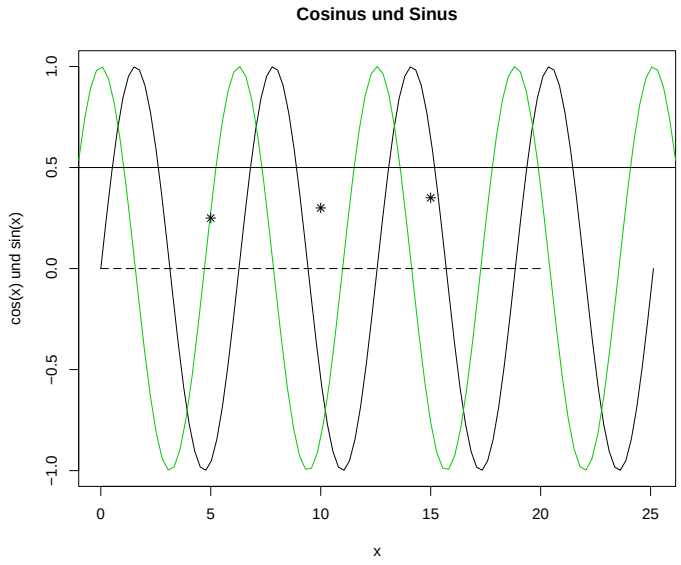
3.5.2 Beispiel

Für die Darstellung von Graphen reellwertiger Funktionen wird die Funktion `curve()` verwendet, vgl. Abbildung 3.4.

```
> curve(sin(x), from=0, to=8*pi)
> curve(cos(x), add=TRUE,col=5)
> lines(x=c(0,20),y=c(0,0),type="l",lty=5)
> abline(h=0.5)
> points(x=c(5,10,15),y=c(0.25,0.30,0.35),pch=8)
```

Durch `add=TRUE` wird die zweite Grafik in die erste Grafikausgabe gezeichnet.

Abbildung 3.4: Graphen von Funktionen zeichnen.

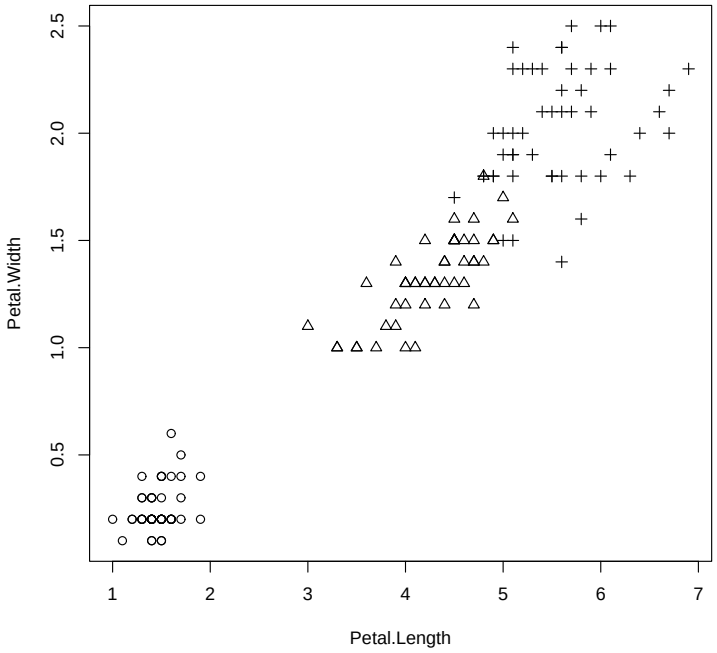


3.5.3 Beispiel

Mithilfe des `pch` oder `col` Parameters kann der Zusammenhang von zwei Variablen unter dem zusätzlichen Einfluss einer dritten Variablen in einem Scatterplot dargestellt werden, vgl. Abbildung 3.5.

```
> data(iris)
> fix(iris)
> attach(iris)
> plot(Petal.Length,Petal.Width,pch = as.numeric(Species))
> # as.numeric() wandelt die factor-Variable \texttt{Species} in
> # in den Level-Index, d.h. insbesondere in numerische Werte, um.
```

Abbildung 3.5: Anwendung von `pch`



Eine weitere häufige Anwendung ist das Ergänzen von Grafiken mit Texten, vgl. Abbildung 3.6. Neben der Beschriftung von Achsen (`xlab=`, `ylab=`) und Überschriften (`main=`) ist hier die `text()`-Funktion sehr hilfreich. Durch

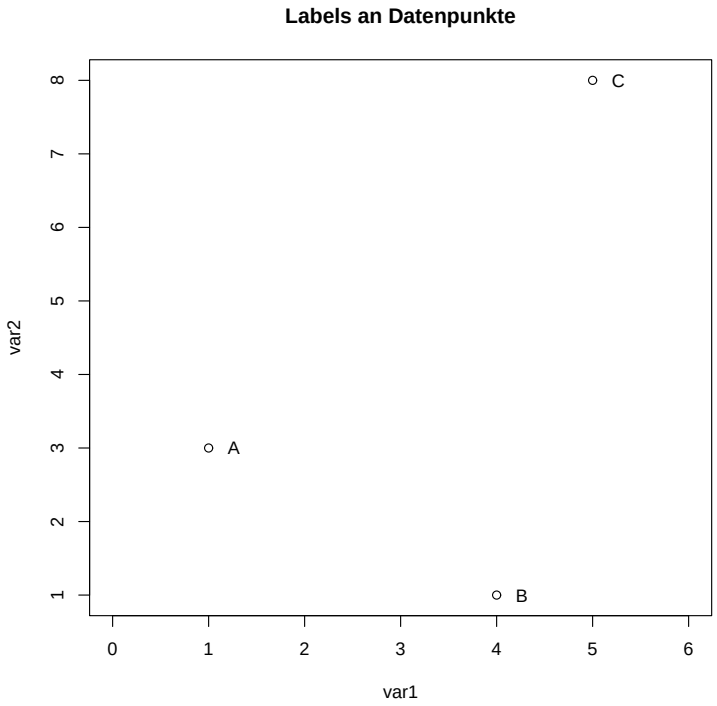
```
> text(x=,y=,labels=,adj=)
```

erzeugt man innerhalb einer Grafik an der Stelle (x, y) einen über `lables` festgelegten Text-Eintrag, wobei man beachte, dass die `text()`-Funktion (wie auch die `plot()`-Funktion) elementweise die Punkte (x, y) der Datenvektoren verarbeitet.

```
> var1 <- c(1,4,5)
> var2 <- c(3,1,8)
> var3 <- c("A","B","C")
> Daten <- data.frame(var1,var2,var3)
> Daten
  var1 var2 var3
1    1    3    A
2    4    1    B
3    5    8    C
```

```
> plot(var1,var2,xlim=c(0,6),main="Labels an Datenpunkte")
> text(x=var1+0.2,y=var2,labels=var3,adj=0)
```

Abbildung 3.6: Beschriftung von Datenpunkten



Kapitel 4

Statistik mit R

Die statistische Datenanalyse ist neben der Erstellung von Grafiken das zentrale Einsatzgebiet von R. Einerseits können viele Standardmethoden (größtenteils implementiert in den Funktionen des `base package`) angewandt werden, andererseits werden viele aktuelle statistische Methoden (aber auch aktuarielle und finanzmathematische Verfahren) in Zusatzpaketen programmiert und stehen jedem R-Anwender zur freien Verfügung. Schließlich kann jeder Anwender, dank der großen Flexibilität von R, seine eigenen Datenanalysefunktionen in R schreiben. Bei der Vielzahl moderner Methoden, die in Zusatzpaketen angeboten werden, scheinen eher mangelnde Methodenkenntnisse und die Informationslogistik zu den Zusatzpaketen (wo ist welche Methode zu finden?) natürliche Grenzen der Ausbreitung der modernen statistischen Vefahren zu sein.

4.1 Verteilungen und Simulation

In R können (Pseudo-)Zufallszahlen generiert werden (mithilfe eines Zufallszahlengenerators) und es können verschiedenste Zufallsexperimente simuliert werden. Für viele häufig verwendete Verteilungen kann man neben der Erzeugung entsprechend verteilter Zufallszahlen auch die Wahrscheinlichkeitsfunktion bzw. Dichte, Quantile und die Verteilungsfunktion berechnen.

Mittels dem Befehl

`set.seed(x)`,

mit `x` eine Zahl, kann der Startwert (engl.: seed) des Zufallszahlengenerators auf `x` gesetzt werden. Damit können bei gleicher seed immer wieder die gleichen Zufallszahlen erzeugt werden und so Ergebnisse reproduziert werden. Die Standard(default)-Anwendung des Zufallszahlengenerators ist allerdings eine Anwendung mit jeweils wech-

selnder seed, was z.B. über eine Deklaration der seed durch die aktuelle Systemzeit in R automatisiert realisiert wird.

Eine sehr komfortable Funktion zur Simulation diskreter Zufallsexperimente ist die R-Funktion

`sample()`.

Mit dieser Funktion können z.B. Urnenmodelle mit bzw. ohne Zurücklegen simuliert werden. Dem Funktionsparameter `x` kann man einen Vektor (auch vom Charakter-Typ) übergeben, aus dem eine Zufallsstichprobe gezogen werden soll. Über den Parameter `size` gibt man den Stichprobenumfang an, der Parameter `replace = TRUE (FALSE)` steuert, ob mit oder ohne Zurücklegen gezogen werden soll und über den Parameter `prob` kann ein Vektor mit Ziehungswahrscheinlichkeiten für die Komponenten des Vektors `x` angegeben werden. Mittels geeigneter Wahl von `prob` kann man z.B. jede Kugel-Verteilung in einer Urne simulieren.

Beispiel:

```
x <- sample(x=c(1,2,3),size=10,replace=TRUE)
```

erzeugt einen Zufallsvektor `x` der Länge 10 mit Komponenten, die jeweils gleichwahrscheinlich die Werte 1, 2 oder 3 annehmen.

Die R-Syntax um Zufallszahlen, Dichte bzw. Wahrscheinlichkeitsfunktion, Quantile und die Verteilungsfunktion bestimmter Verteilungen zu berechnen, folgt immer dem folgendem Schema.

Ein Präfix gibt an, welche Größe man berechnen will (z.B. `d` für Dichte (engl.: density) oder `r` für eine (Pseudo-)Zufallszahl (engl.: random)) und es folgt der Name der speziellen Verteilung (z.B. `norm` für Normalverteilung).

Präfix	Bedeutung
`d`	für density, Werte der Dichte bzw. Wahrscheinlichkeitsfunktion
`p`	für probability, Werte der Verteilungsfunktion
`q`	für quantile, Quantile
`r`	für random, Zufallszahlen

In der folgenden Tabelle sind für einige grundlegende Verteilungen jeweils der Name in der R-Syntax und die Parameter der R-Funktionen (mit default-Werten) angegeben.

Verteilung	R-Name	Parameter	default-Werte
`binom`	Binomial	`size, prob`	
`chisq`	Chi-Quadrat	`df`	
`exp`	Exponential	`rate`	1
`f`	F	`df1, df2`	
`gamma`	Gamma	`shape`	
`geom`	Geometrisch	`prob`	
`hyper`	Hypergeometrisch	`m, n, k`	
`norm`	Normal	`mean, sd`	0, 1
`pois`	Poisson	`lambda`	
`t`	t	`df`	
`unif`	Gleichverteilung	`min, max`	0, 1

In den folgenden Beispielen wird die prinzipielle Anwendung der R-Funktionen zu Verteilungen verdeutlicht. Die Bezeichnungen, Funktionalitäten und defaults der jeweiligen Funktionsparamter können über die R-Hilfe zu den Funktionen eingesehen werden.

Beispiele:

a) `dbinom(x=5,size=100,prob=0.2)` berechnet die Wahrscheinlichkeitsfunktion $f(5) \equiv P(X = 5)$ einer $B(100, 0.2)-$verteilten Zufallsvariablen X.

b) `pnorm(q=4,mean=3,sd=2)` berechnet die Verteilungsfunktion $F(4) \equiv P(X \leq 4)$ einer $N(3, 4)-$verteilten Zufallsvariablen X.

c) `x<-runif(x=100,min=1,max=5)` erzeugt einen Vektor `x` mit 100 unabhängigen Realisationen einer auf $[1, 5]$ gleichverteilten Zufallsvariablen.

4.2 Univariate deskriptive und explorative Analyse

Man betrachtet die Analyse eines Merkmals (Variable), für das n Beobachtungen vorliegen. Als Datenstruktur liegt also ein Datenvektor `x` entweder mit numerischen Komponenten (quantitatives Merkmal) oder Zeichenketten-Komponenten (qualitatives Merkmal) vor.

4.2.1 Qualitatives Merkmal

In den folgenden Tabellen sind die grundlegenden R-Funktionen, zusammen mit einer Kurzbeschreibung, für die Untersuchung der Häufigkeitsverteilung einer Stichprobe eines qualitativen Merkmals angegeben.

R-Funktion	Bedeutung
`table(x)`	absolute Häufigkeitstabelle
`barplot(table(x))`	Balkendiagramm
`pie(table(x))`	Kuchendiagramm

4.2.2 Quantitatives Merkmal

Neben den eventuell auch bei quantitativen Merkmalen sinnvoll anwendbaren Funktionen, die in obiger Tabelle gegeben sind, können folgende R-Funktionen des Basis-Systems bei der Analyse eines quantitativen Merkmals von Nutzen sein.

R-Funktion	Bedeutung
`hist(x)`	Histogramm
`mean(x)`	Arithmetisches Mittel
`var(x)`	Stichprobenvarianz
`sd(x)`	Standardabweichung
`median(x)`	Median
`range(x)`	Extremwerte
`boxplot(x)`	Boxplot
`IQR(x)`	Interquartilsabstand
`quantile(x)`	Quantile
`density(x)`	Kerndichteschätzung
`rug(x)`	Rugplot

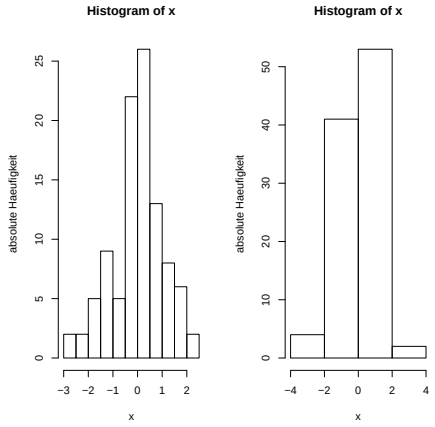
Zur grafischen Darstellung und Untersuchung einer Stichprobenverteilung eines quatitativen Merkmals verwendet man vor allem die folgende Methoden.

a) Histogramme

Beispiel:

```
> x <- rnorm(100)
> par(mfrow=c(1,2))
> hist(x,ylab="absolute Haeufigkeit")
> # Automatisierte Klassenwahl
> hist(x,ylab="absolute Haeufigkeit",breaks=c(-4,-2,0,2,4))
> # Definierte Klassenwahl
```

Abbildung 4.1: Histogramme



Mittels der Parametrisierung `hist(...,freq=F)` erhält man das Histogramm der relativen Häufigkeiten.
Man beachte bei der Anwendung von Histogrammen, dass die Klassen-(Intervall-)Wahl die Interpretation eines Histogramms verändern kann.

b) Dichteschätzungen

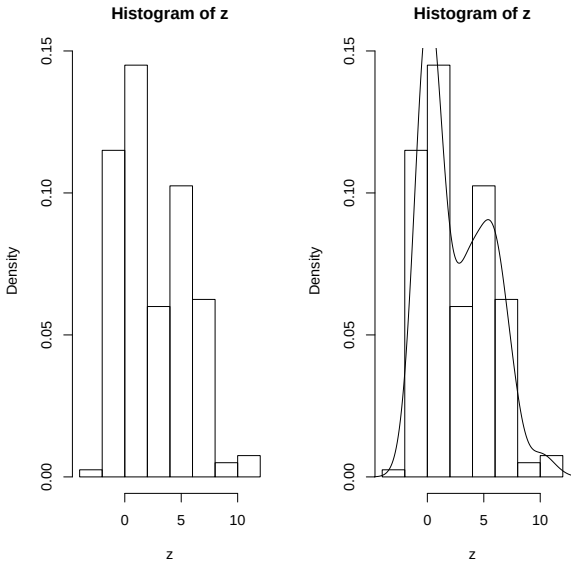
Auf Basis der Stichprobenwerte wird die unbekannte Dichte der zugrundeliegenden Verteilung mit dem Verfahren der *Kerndichteschätzung* geschätzt. Eine Einführung zu Kernschätzern findet man bei Pruscha (2006, S. 208 - 215). Bei der Kerndichteschätzung spielt eine Klasseneinteilung wie bei Histogrammen keine Rolle, allerdings wird die Dichteschätzung durch die Parameter des Schätzalgorithmus (verwendeter Kern, Fensterbreite und Glättungsparameter) beeinflusst.

Beispiele:

Mit dem folgenden R-Code werden zwei Histogramme erzeugt (vgl. Abbildung 4.2), wobei in das zweite Histogramm noch die Kerndichteschätzung der simulierten Stichprobe eingezeichnet wird.

```
> x <- rnorm(100,mean=0,sd=1)
> y <- rnorm(100,mean=5,sd=2)
> z <- c(x,y) # Vektor z = Vektor y wird an x anghaengt
> par(mfrow=c(1,2))
> hist(z,freq=F)
> d <- density(z) # Kerndichteschaetzung
> hist(z,freq=F)
> lines(d) # Dichteschaetzung in Histogramm einzeichnen
```

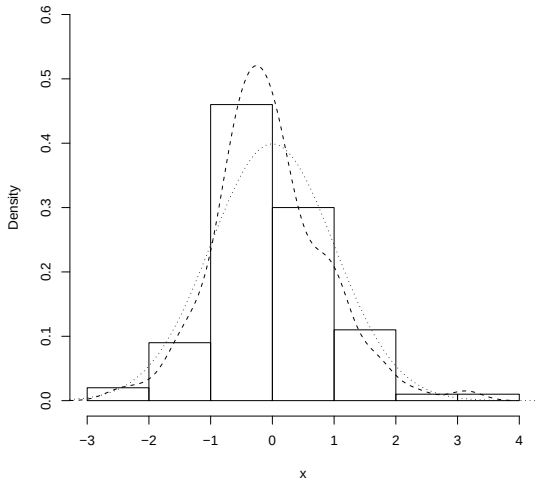
Abbildung 4.2: Histogramm mit Kerndichteschätzung



Mit dem folgenden R-Code wird für eine Stichprobe von 100 simulierten, unabhängigen Realisationen einer standardnormalverteilten Zufallsvariablen das Histogramm, die Kerndichteschätzung und die theoretische Dichte der $N(0,1)-$Verteilung grafisch in der Abbildung 4.3 dargestellt.

```
> par(mfrow=c(1,1))
> x <- rnorm(100)
> hist(x,freq=F,ylim=c(0,0.6))
> lines(density(x),lty=2)
> curve(dnorm(x,0,1), add=TRUE, lty=3)
```

Abbildung 4.3: Kerndichteschätzung und theoretische Dichte



c) Boxplots

Boxplots (genauer: *Box- and Whiskerplots*) beinhalten eine graphische Darstellung der Größen
- Median, Quartile,
- Grenzen des "normalen" Bereichs: unteres Quartil $- 1.5 \cdot$ Interquartilsabstand und oberes Quartil $+ 1.5 \cdot$ Interquartilsabstand,
- Extrembeobachtungen (Ausreisser?).

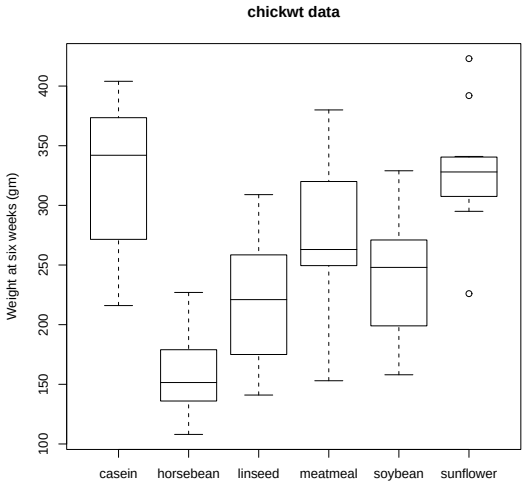
Mit der R-Funktion `boxplot(x)` erzeugt man einen Boxplot des Datenvektors `x` (für mehrere Datenvektoren `x,y,...` verwendet man `boxplot(x,y,...)`). Ist `f` eine Spalte eines Datensatzes vom Faktor-Typ (also eine kategorielle Variable), so liefert die Modifikation `boxplot(x ~ f)` mehrere Boxplots bedingt nach den Ausprägungsklassen der faktoriellen Variable.

Beispiel:

```
> data(chickwts)
> attach(chickwts)
> summary(chickwts)
       weight             feed
 Min.   :108.0   casein   :12
 1st Qu.:204.5   horsebean:10
 Median :258.0   linseed  :12
 Mean   :261.3   meatmeal :11
 3rd Qu.:323.5   soybean  :14
```

```
 Max.   :423.0   sunflower:12
> boxplot(weight ~ feed, main = "chickwt data",ylab = "Weight at six weeks (gm)")
```

Abbildung 4.4: Boxplots

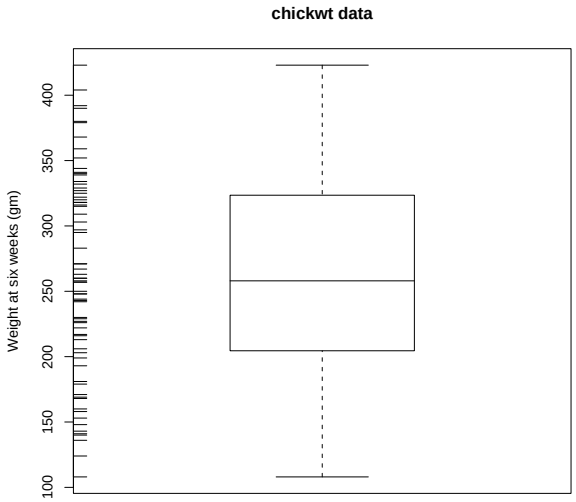


Die `boxplot()`-Funktion bietet eine Vielzahl von Modifikationsmöglichkeiten und Ergänzungen (vgl. `help(boxplot)`).
In manchen Situationen kann auch ein zusätzlicher *Rugplot* in einem Boxplot von Nutzen sein, vgl Abbildung 4.5.

Beispiel:

```
> data(chickwts)
> attach(chickwts)
> boxplot(weight)
> rug(weight,side=2)
```

Abbildung 4.5: Boxplot mit Rugplot



d) Quantil-Quantil-Plots

Mit der Funktion `qqplot(x,y,...)` wird ein *Quantil-Quantil-Plot* (*Q-Q-Plot*) der beiden Datenvektoren (Stichproben) `x` und `y` erzeugt.

In einem Q-Q-Plot werden die (empirischen) Quantile der beiden Stichproben in einem Streudiagramm gegeneinander aufgetragen.

Für den Fall, dass den Stichproben eine identische Verteilung zugrundeliegt, sind die über die empirischen Quantile der Stichproben festgelegten Punkte des Q-Q-Plots annähernd an der Identitätsgeraden (Winkelhalbierenden) $y = x$ ausgerichtet.

Liegen die Punkte eines Q-Q-Plots annähernd auf einer Geraden, weichen aber in der Steigung bzw. im Ordinatenabschnitt von der Winkelhalbierenden ab, könnten sich die zugrundeliegenden Stichprobenvariablen durch eine lineare Transformation ineinander überführen lassen. D.h. man kann z.B. vom gleichen Verteilungstyp mit unterschiedlichen Erwartungswerten und Varianzen ausgehen.

Mittels einer Ausgleichsgeraden (z.B. erstellt durch eine einfache lineare Regression oder durch eine robuste Schätzung etwa mittels der Quartil-Punkte) durch die Punkte eines Q-Q-Plots kann man beim Vorliegen von linear transformierten Stichprobenvariablen aus der Abweichung der Ausgleichsgeraden von der Identitätsgeraden auf Skalenunterschiede (Steigung) und Lageunterschiede (Verschiebung der Geraden bzgl. der Identitätsgeraden) schließen.

Durch Simulation (mithilfe von Pseudozufallszahlen) einer Referenzstichprobe `x` kann man mittels der Funktion `qqplot(x,y,...)` die Stichprobe `y` auch hinsichtlich einer hypothetischen, zugrundeliegenden Verteilung (die man zur Simulation der Referenzstichprobe verwendet) untersuchen. Alternativ kann man für die Referenzstichprobe `x` auch direkt theoretische Quantile der hypothetischen Verteilung verwenden.

Einen Q-Q-Plot, in dem man die empirischen Quantile einer Stichprobe mit den theoretischen Quantilen einer hypothetischen Verteilung vergleicht, nennt man auch *Wahrscheinlichkeits-Plot*.

Mit dem R-Befehl `qqnorm(x)` erzeugt man einen Q-Q-Plot (genauer eigentlich einen Wahrscheinlichkeits-Plot) des Datenvektors `x` bzgl. der Annahme, dass die der Stichprobe zugrundeliegende Verteilung eine (Standard-)Normalverteilung ist. Entstammen die Daten einer Normalverteilung liegen die Punkte im Q-Q-Plot annähernd auf einer Geraden. Mittels `qqline(x)` kann die - bei Vorliegen einer Normalverteilung theoretisch ewartete - Sollgerade zusätzlich in den Q-Q-Plot eingezeichnet werden. In diesem Spezialfall ist der Ordinatenabschnitt der Geraden ein Schätzwert für den Erwartungswert der Stichprobenverteilung und die Steigung der Geraden ein Schätzwert für die Standardabweichung der Stichprobenverteilung (vorausgesetzt es liegt wirklich eine Normalverteilung vor).

In der folgenden Aufzählung sind die Anwendungsmöglichkeiten von Q-Q-Plots zur explorativen Analyse einer Stichprobe mit großem Stichprobenumfang zusammengefasst:

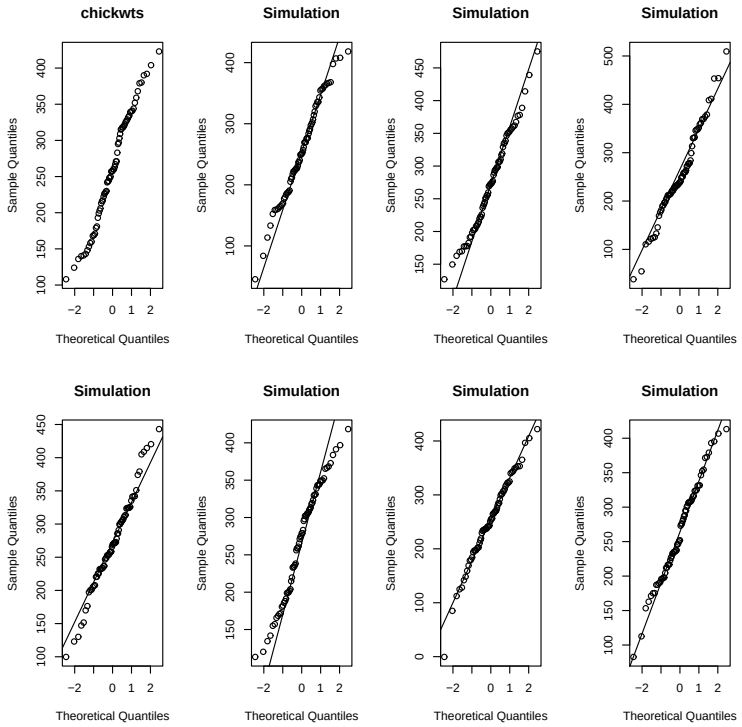
1. Verteilungsannahme: Stimmt die Verteilung der (linear transformierten) Stichprobenvariablen mit der hypothetischen Verteilung überein, zeigen die Punkte des Q-Q-Plots einen linearen Verlauf entlang einer Sollgeraden.
2. Lage- und Skalenunterschiede: Besitzen die Stichprobenvariablen nach einer linearen Transformation tatsächlich die hypothetische Verteilung, können mit dem Ordinatenabschnitt und der Steigung der Sollgeraden grafisch Lage- und Skalierungsparameter der Stichprobenverteilung geschätzt werden.
3. Ausreisser: Enstprechen die Punkte des Q-Q-Plots mehrheitlich einem approximativ linearen Verlauf, so können einzelne abweichende Punkte als potentielle Ausreisser identifiziert werden.
4. Unterschiede in Form und Schiefe: Ein systematisches Abweichen der Punkte im Q-Q-Plot von der Sollgeraden an den Rändern ist ein Hinweis auf Unterschiede an den Rändern der hypothetischen Verteilung und der tatsächlichen Stichprobenverteilung. Besitzt die Stichprobenverteilung im Vergleich zur hypothetischen Verteilung z.B. stärker (schwächer) besetzte Verteilungsränder, verlaufen die Punkte im Q-Q-Plot an den Rändern horizontal (vertikal) von der Sollgeraden weg.

Beispiel: Wir untersuchen die Gewichtsdaten `weight` aus dem Datensatz `chickenwts` auf Normalverteilung. Um den Q-Q-Plot besser einordnen zu können, simulieren wir

7-mal eine entsprechend normalverteilte Stichprobe und plotten zu diesen ebenfalls die Q-Q-Plots, vgl. Abbildung 4.6.

```
> data(chickwts)
> attach(chickwts)
> length(weight)
[1] 71
> mean(weight)
[1] 261.3099
> sd(weight)
[1] 78.0737

> par(mfrow=c(2,4))
> qqnorm(weight,main="chickwts")

> for(i in 1:7)
+ {x = rnorm(71,mean=261.3,sd=78.1)
+ qqnorm(x,main="Simulation")
+ qqline(x)}
```

Abbildung 4.6: Q-Q-Plots



Aus den Vergleichen der Q-Q-Plots folgert man, dass der Q-Q-Plot der Gewichtsdaten nicht gegen eine Normalverteilungsannahme bei den Gewichtsdaten spricht.

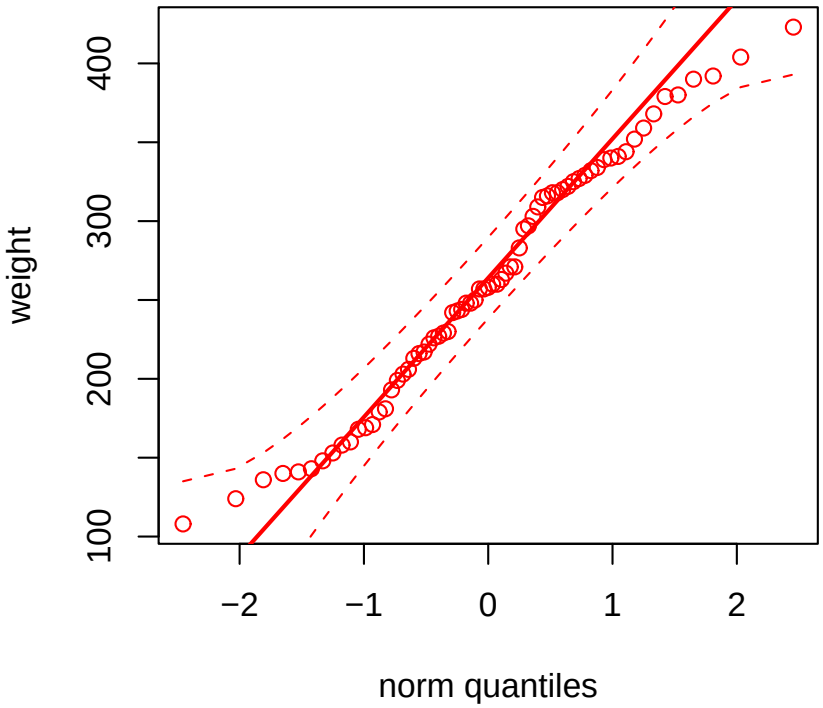
Erweiterung: Alternativ kann man die R-Funktion `qq.plot()` aus dem Paket **car** verwenden. Für die Überprüfung einer zugrundeliegenden Normalverteilung bei einem Datenvektor `x` verwendet man hier den Aufruf

```
qq.plot(x,dist="norm"} # Spezifizierung der Verteilung
```

In der erzeugten Grafik, vgl. Abbildung 4.7, wird zusätzlich eine punktweise *Konfidenzregion* angezeigt, die eine Beurteilung der Abweichungen der Punkte von der Sollgeraden erleichtert. Die Konfidenzregion wird R-intern mittels Simulation errechnet.

```
> data(chickwts)
> attach(chickwts)
> qq.plot(weight,dist="norm")
```

Abbildung 4.7: Q-Q-Plot mit Konfidenzregion



Mittels dem Parameter `dist` in der Funktion `qq.plot` können über die Normalverteilung hinaus auch andere Verteilungsannahmen untersucht werden.

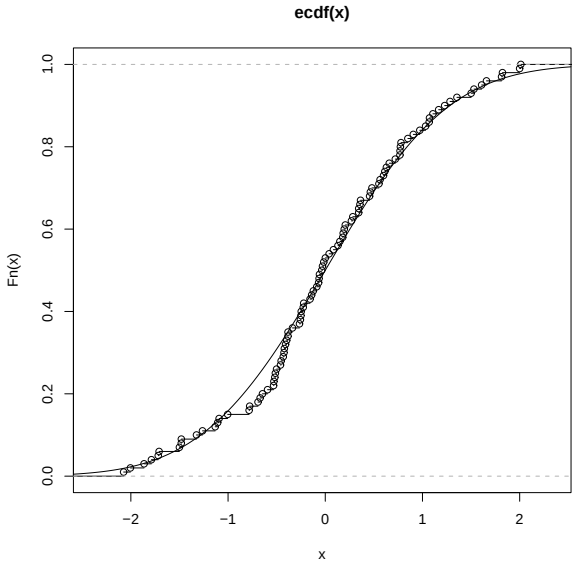
e) Empirische Verteilungsfunktion

Die R-Funktion `ecdf()` bildet das R-Objekt der empirischen Verteilungsfunktion einer Stichprobe. Mithilfe der `plot()`-Funktion erhält man eine Grafik, vgl. Abbildung 4.8, der empirischen Verteilungsfunktion.
Da für große Stichprobenumfänge die empirische Verteilungsfunktion einer unabhängigen Stichprobe gegen die theoretische Verteilungsfunktion konvergiert (*Satz von Glivenko Cantelli*), erhält man durch einen Vergleich der beiden Funktionen eine Einschätzung über die zugrundeliegende Verteilung.

Beispiel:

```
> x <- rnorm(100)
> plot(ecdf(x)) # empirische Verteilungsfunktion der Stichprobe x
> curve(pnorm(x),add=TRUE) # theoretische Verteilungsfunktion N(0,1)
```

Abbildung 4.8: Empirische Verteilungsfunktion



4.3 Multivariate deskriptive und explorative Analyse

Nun betrachtet man den Fall, dass Beobachtungen zu mehreren Merkmalen vorliegen. Betrachtet man nur zwei Merkmale spricht man von einer bivariaten Analyse. Dieser Fall soll hier hauptsächlich diskutiert werden.

Man betrachtet also (mindestens) zwei Daten-Vektoren `x` und `y`, wobei die $i-$te Komponente von `x` die Ausprägung des Merkmals X bzw. die $i-$te Komponente von `y` die Ausprägung des Merkmals Y des $i-$ten Merkmalträgers (der $i-$ten Beobachtung) darstellt.

1. Fall: Quantitative und qualitative Merkmale

In der Situation, dass eines der beiden Merkmale qualitativ ist (z.B. könnten die Ausprägungen von X zwei verschiedene Gruppen (Kategorien, Stufen) sein), und das andere Merkmal quantitativ ist, kann das quantitative Merkmal in den beiden durch das qualitative Merkmal (man sagt auch *Faktor*) definierten Gruppen (man sagt auch *Stufen*) untersucht werden.

Prinzipiell können alle Kennzahlen und Grafiken der univariaten Analyse verwendet werden. Dabei bildet man z.B. in den unterschiedlichen Gruppen Mittelwerte oder Boxplots und vergleicht diese. Insbesondere bei grafischen Methoden ist es vorteilhaft, wenn die Einzel-Grafiken zusammengefasst werden.

Beispiel: Sei `x1` der Datenvektor eines Merkmals X in einer Gruppe 1 und `x2` der Datenvektor eines Merkmals X in einer Gruppe 2. Dann werden durch `boxplot(x1,x2)` die beiden Boxplots der Daten je Gruppe in eine gemeinsame Grafik gezeichnet und können hinsichtlich Lage- und Streuungsunterschiede verglichen werden.

2. Fall: Quantitative Merkmale X und Y

Streudiagramme: Als grundlegende grafische Methode steht hier (insbesondere für metrischen Merkmale) das Streudiagramm (engl.: scatterplot) zur Verfügung. Falls `x` und `y` die Datenvektoren der beiden Merkmale sind, kann man das Streudiagramm sehr einfach mittels der R-Funktion

```
plot(x,y)
```

erstellen.

Um mehr als zwei Merkmale mittels Streudiagrammen zu untersuchen kann man für die Merkmale jeweils paarweise ein Streudiagramm erstellen und in einer gemeinsamen Grafik zur besseren Interpretation gegenüberstellen. Liegen die Daten in einer Matrix oder Datentabelle (`data frame`) vor, wobei in den Spalten jeweils die Ausprägungen der einzelnen Merkmale $X_1, \ldots, X_n$ stehen, so kann man einen Multi-Scatterplot mithilfe der R-Funktion

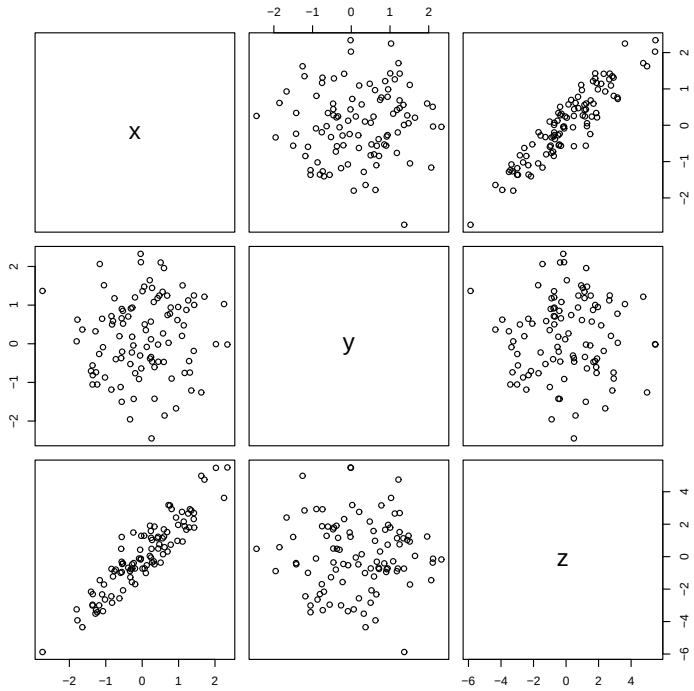
```
pairs()
```

erstellen. Als Argument erwartet die Funktion `pairs()` den Namen der entsprechenden Matrix oder Datentabelle.

Beispiel: Die Befehlsfolge

```
> x<-rnorm(100)
> y<-rnorm(100)
> z<-2*x+rnorm(100)
> A<-cbind(x,y,z)
> pairs(A)
```

führt zu dem paarweisen Streudiagramm



Empirische Korrelationskoeffizienten: Seien `x` und `y` die Datenvektoren zweier Merkmale X und Y. Folgende Korrelationskoeffizienten stehen in der Grundversion von R zur Verfügung:

R-Funktion	Bedeutung
`cor(x,y)`	Bravais-Pearson-Korrelationskoeffizient
`cor(x,y,method="spearman")`	Spearman-(Rang-)Korrelationskoeffizient
`cor(x,y,method="kendall")`	Kendall-(Rang-)Korrelationskoeffizient
Korrelationskoeffizient	**Voraussetzung**
Pearson-Korrelationskoeffizient	X und Y metrisch skaliert
Spearman-(Rang-)Korrelationskoeffizient	X, Y ordinal oder metrisch skaliert
Kendall-(Rang-)Korrelationskoeffizient	X, Y ordinal oder metrisch skaliert

Wird die R-Funktion `cor()` auf eine Datentabelle angewandt, erzeugt die Funktion eine Korrelationsmatrix der numerischen Datenvektoren. Damit wird die Korrelationsstruktur von mehr als zwei Merkmalen untersucht.

Die empirische Kovarianz kann durch die Funktion `var(x,y)` bzw. `cov(x,y)` berechnet werden.

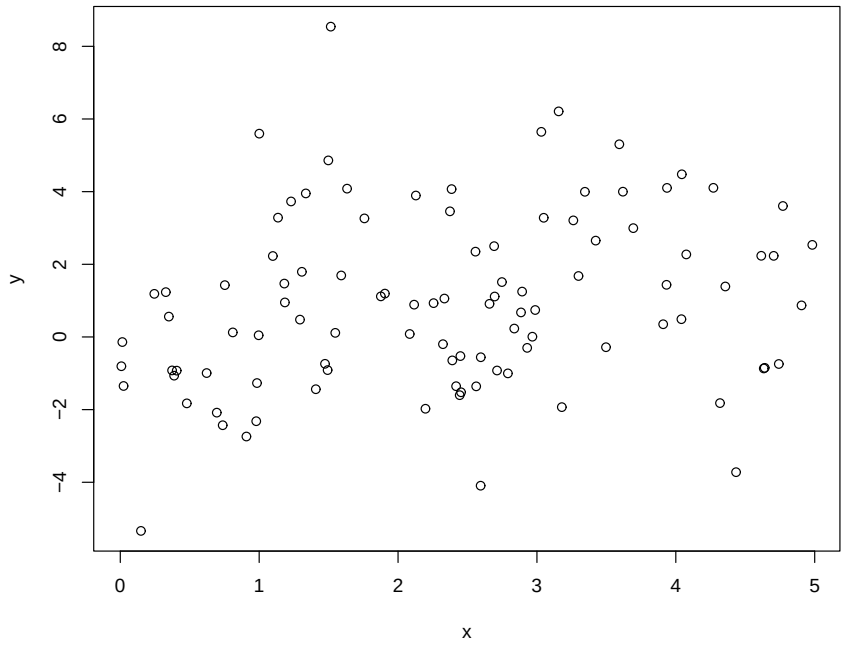
Bemerkung:

Betrachtet man in einem Datensatz zusätzlich zu den quantitativen Merkmalen noch einen Faktor mit mehreren Stufen, so können die Zusammenhangsuntersuchungen für die beiden quantitativen Merkmale jeweils getrennt für die Stufen durchgeführt werden.

Für Streudiagramme ist hier die Grafik-Funktion `coplot()` sehr hilfreich.

Beispiel:

```
> Gruppe <- c(rep("A",20),rep("B",50),rep("C",30))
> x1 <- runif(20,min=0,max=1)
> x2 <- runif(50,min=1,max=3)
> x3 <- runif(30,min=3,max=5)
> x <- c(x1,x2,x3)

> y1 <- -2*x[1:20] + rnorm(20,sd=2)
> y2 <- 5 -2*x[21:70] + rnorm(50,sd=2)
> y3 <- 10 -2*x[71:100] + rnorm(30,sd=2)
> y <- c(y1,y2,y3)
> plot(x,y)
> cor(x,y)
[1] 0.2234462
```

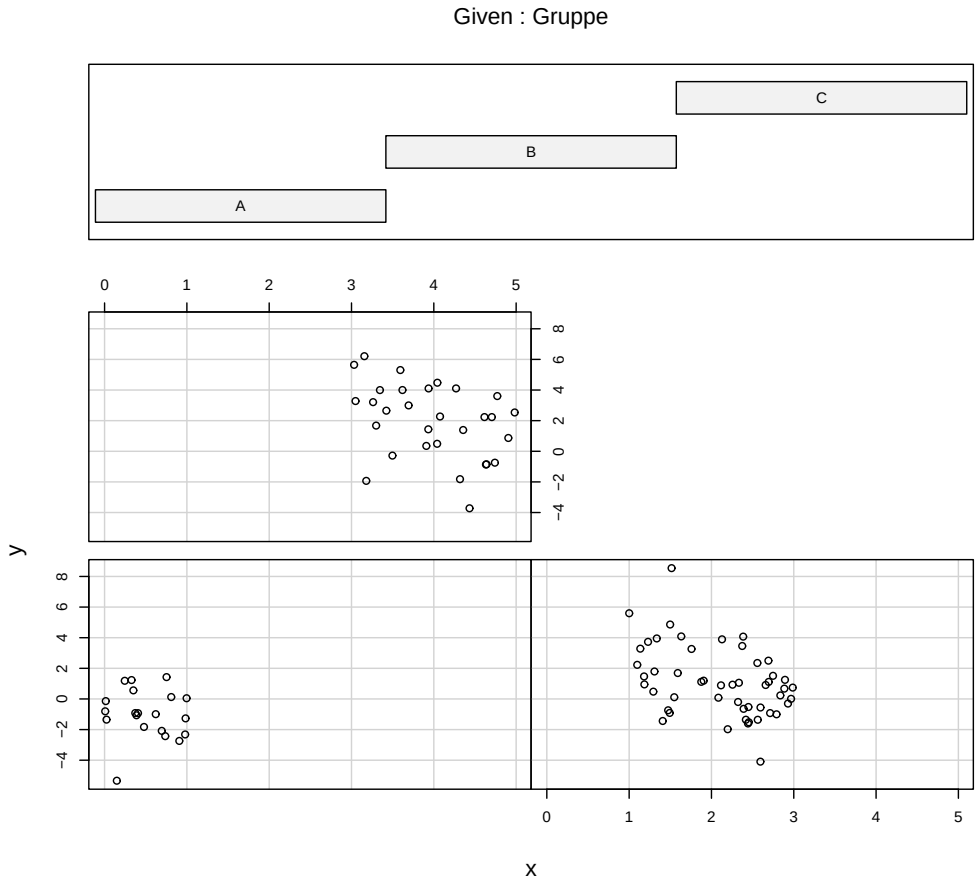



```
> daten <- data.frame(Gruppe,x,y)
> attach(daten)
> coplot(y ~ x | Gruppe)

>  cor(x[Gruppe=="C"],y[Gruppe=="C"])
[1] -0.3848766
> cor(x[Gruppe=="A"],y[Gruppe=="A"])
[1] -0.05260065
> cor(x[Gruppe=="B"],y[Gruppe=="B"])
[1] -0.4240954
```

Beachte: Ohne Einbezug der Faktor-(Gruppen-)Variable führt die Analyse zu einer Fehlinterpretation (man spricht von einer *Scheinkorrelation*). Die Frage, welche Untersuchungstiefe man in einer statistischen Untersuchung wählt, ist von grundlegender Wichtigkeit. Ohne Kenntis von Schlüsselvariablen kann es zu Fehlinterpretationen kommen. Eine weitere typische Fehlinterpretation bei Korrelationsanalysen sind sogenannte *versteckte Variablen*, wo nicht in die Untersuchung involvierte Variablen im Hintergrund die eigentliche kausale Wirkung besitzen und nicht diejenigen Variablen, die man untersucht hat.

Wichtig ist insbesondere der Grundsatz:

$$\text{Korrelation} \neq \text{kausaler Zusammenhang}$$



3. Fall: Qualitative Merkmale X und Y

Seien `x` und `y` die Datenvektoren der nominalen Merkmale X und Y. Als grundlegende Analysemethode bietet sich hier eine Kontingenztabelle an.

Häufigkeits-(Kontingenz-)Tabellen: Mittels

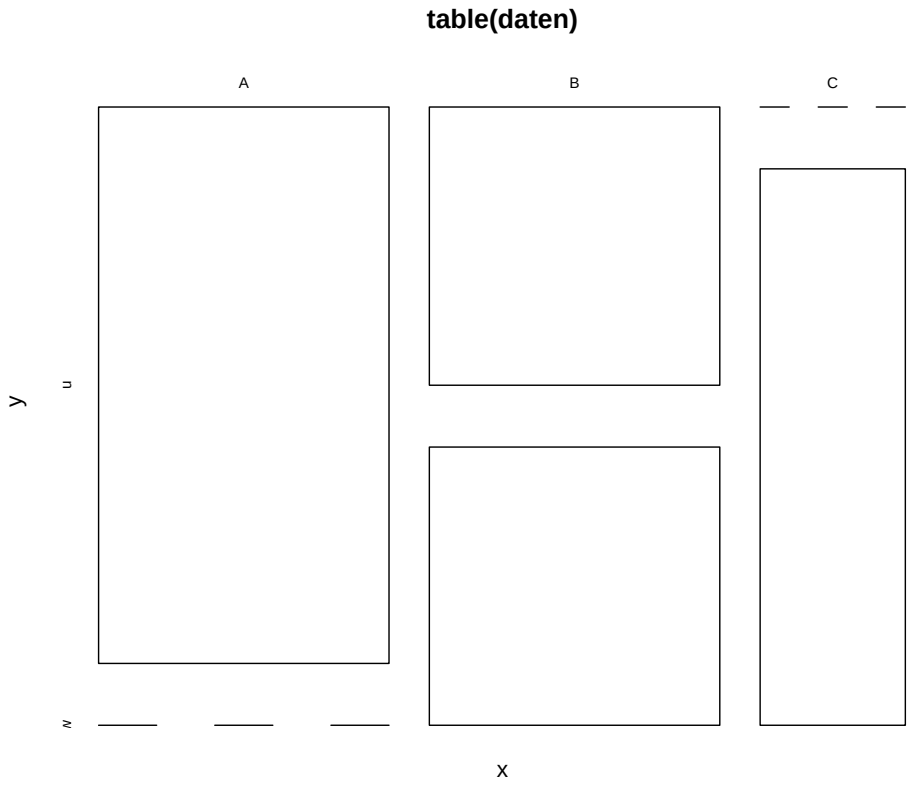
```
table(x,y)
```

wird die absolute Häufigkeitstabelle berechnet.
Die R-Funktion `prop.table()` ermöglicht neben der Berechnung der relativen Häufigkeitstabelle auch die Berechnung der bedingten Verteilung der Spalten bzw. Zeilen. Als Argument benötigt die Funktion `prop.table()` die absolute Häufigkeitstabelle. Über den Parameter `margin` kann durch die Zuweisung `margin=1` die bedingte Verteilung der Zeilen, durch die Zuweisung `margin=2` die bedingte Verteilung der Spalten oder bei `margin=NULL` (=default-Einstellung) die relative Häufigkeitstabelle erstellt werden.

Zur grafischen Darstellung von (bedingten) Häufigkeitstabellen kann man vergleichende Säulendiagramme oder Mosaikplots (vgl. R-Funktion `mosaicplot()`) heranziehen.

Beispiel:

```
> x <- c("A","A","A","A","B","B","B","B","C","C")
> y <- c("u","u","u","u","u","u","w","w","w","w")
> daten <- data.frame(x,y)
> daten
   x y
1  A u
2  A u
3  A u
4  A u
5  B u
6  B u
7  B w
8  B w
9  C w
10 C w
> table(daten)
   y
x   u w
  A 4 0
  B 2 2
  C 0 2
> prop.table(table(daten))
   y
x   u   w
  A 0.4 0.0
  B 0.2 0.2
  C 0.0 0.2

> prop.table(table(daten),margin=2)
   y
x   u         w
  A 0.6666667 0.0000000
  B 0.3333333 0.5000000
  C 0.0000000 0.5000000

> mosaicplot(table(daten))
```



χ^2-Koeffizient, Kontingenzkoeffizient und weitere Zusammenhangsmaße:

Der (Pearson-)χ^2-Koeffizient und χ^2-Unabhängigkeitstests können mittels der R-Funktion `chisq.test()` berechnet werden. Als Argument erwartet die Funktion die betreffende absolute Kontingenztabelle. Falls man den üblichen χ^2-Koeffizienten berechnen möchte, muß man den Funktionsparameter `correct` auf `correct=FALSE` stellen, ansonsten wird eine sogenannte Stetigkeitskorrektur durchgeführt.

Beispiel:

Eine $2 \times 2-$Kontingenztabelle sei definiert durch

```
> x <- matrix(c(45,33,75,49), nc = 2)
```

Zur Berechnung des χ^2-Koeffizienten bzw. zur Durchführung des χ^2-Tests verwendet man den Aufruf

```
> chisq.test(x,correct=FALSE)
```

und erhält die Ausgabe

```
        Pearson's Chi-squared test

data:  x X-squared = 0.1547, df = 1, p-value = 0.694
```

In dem Paket `vcd` wird die R-Funktion `assocstats()` zur Verfügung gestellt. Als Argument wird die untersuchte absolute Häufigkeitstabelle in die Funktion eingesetzt. Mit der Funktion kann man unter anderem den Kontingenzkoeffizienten oder Cramer's V für zwei Merkmale berechnen.

4.4 Signifikanztests

Im Rahmen dieser Einführung werden nur diejenigen R-Funktionen vorgestellt, die zur Durchführung von grundlegenden und weit verbreiteten statistischen Signifikanttests verwendet werden. In speziellen Packages bietet R viele Möglichkeiten, auch speziellere Testprobleme umzusetzen. Ausserdem kann der Anwender spezielle Testverfahren natürlich selbst programmieren.

4.4.1 Test auf Lage- und Streuungsparameter in der Ein-Stichprobensituation

Gegeben sei eine Stichprobe $x_1, \ldots, x_n$ von n i.i.d. Stichprobenvariablen $X_1, \ldots, X_n$. Je nach speziellem Test müssen die Stichprobenvariablen noch bestimmte Voraussetzungen, z.B. an ihre Verteilung (z.B. $N(\mu, \sigma^2)-$verteilt) erfüllen. Die Stichprobe sei in dem Datenvektor `x` erfasst.

4.4.1.1 t-Test

Der t-Test zum Testen von Hypothesen über den unbekannen Erwartungswert μ einer zugrundeliegenden $N(\mu, \sigma^2)-$Verteilung bei unbekannter Varianz σ^2 wird in R mit der Funktion `t.test()` durchgeführt.

Die grundlegenden Argumente und Parameter der Funktion `t.test()` sind:

1. Datenvektor `x`.

2. `alternative` kann auf `"two.sided"`, `"less"` oder `"greater"` gesetzt werden und definiert die Alternative H_1 im Test.
3. `mu` definiert den hypothetischen Erwartungswert μ_0, mit dem die Hypothesen gebildet werden.
4. `conf.level` gibt die Sicherheitswahrscheinlichkeit $= 1 - \alpha$ an (default-Wert: $1 - \alpha = 0.95$) für das, mit dem Signifikanzest einhergehende, Konfidenzintervall an.

Die Parameter `y`, `paired` und `var.equal` werden für die Zweistichprobensituation verwendet (s.u.).

Beispiel: Es wird eine Stichprobe x von 100 $N(5,1)-$verteilten (Pseudo-)Zufallszahlen erzeugt. Mittels `t.test()` wird der t-Test zum Signifikanzniveau $\alpha = 5\%$ für die Hypothesen

$$H_0 : \mu_0 = 5 \text{ versus } H_1 : \mu_0 \neq 5$$

durchgeführt.

```
> x <- rnorm(100,mean=5,sd=1)
>
> t.test(x,mu=5,alternative="two.sided",conf.level=0.95)

        One Sample t-test

data:  x
t = -0.5121, df = 99, p-value = 0.6098

alternative hypothesis: true mean is not equal to 5

95 percent confidence interval:
 4.733212 5.157338
sample estimates: mean of x
 4.945275
```

Da der `p-value` gleich 0.6098 ist und damit größer als $\alpha = 0.05$, wird in diesem Beispiel die Alternative $H_1 : \mu_0 \neq 5$ nicht angenommen.

4.4.1.2 Exakter Binomialtest

Mithilfe der R-Funktion `binom.test()` kann ein Signifikanztest zu Hypothesen über den unbekannten Anteilswert einer Grundgesamtheit bzw. den unbekannten Parameter $p \in]0,1[$ (= Trefferwahrscheinlichkeit in dem entsprechenden Bernoullischen Versuchsschema) einer zugrundeliegenden $B(n,p)-$Verteilung durchgeführt werden. Man

beachte, dass in Anwendungen, z.B. in der Qualitätskontrolle, (nach Übeprüfung geeigneter Voraussetzungen) oft eine eigentlich vorliegende Hypergeometrische Verteilung (Ziehen ohne Zurücklegen) durch eine Binomialverteilung (Ziehen mit Zurücklegen) approximiert wird. Innerhalb der Funktion `binom.test()` ist ein exakter Signifikanztest realisiert, d.h. die für die Entscheidungsregeln notwendigen Quantile werden aus der Binomilaverteilung bestimmt und nicht über eine Normalverteilungsapproximation, die großen Stichprobenumfang voraussetzen würde.

Die grundlegenden Argumente und Parameter der Funktion `binom.test()` sind:

1. `x` Anzahl der Versuche mit Treffer bzw. Anzahl der Eintritte des in dem zugrundeliegenden Bernoullischen Versuchsschema betrachteten Ereignisses mit zu schätzender Eintrittswahrscheinlchkeit p.
2. `n` Stichprobenumfang bzw. Anzahl der Versuche
3. `alternative` kann auf `"two.sided", "less"` oder `"greater"` gesetzt werden und definiert die Alternative H_1 im Test.
4. `p` definiert den hypothetischen Anteilswert p_0 bzw. die hypothetische Eintrittswahrscheinlichkeit p_0 mit dem die Hypothesen gebildet werden.
5. `conf.level` gibt die Sicherheitswahrscheinlichkeit $= 1 - \alpha$ an (default-Wert: $1 - \alpha = 0.95$)

Beispiel: Aus einer Grundgesamtheit von 3 Mio. Produkten wurden $n = 5000$ Produkte ausgewählt und auf ihre Funktionsfähigkeit getestet. Es wurde festegestellt, dass 480 Produkte defekt sind. Es sollen Aussagen bzgl. dem unbekannten Anteil p_0 von defekten Produkten in der Grundgesamtheit gemacht werden. Dazu wird z.B. ein Signifikanztest zum Signifikanzniveau $\alpha = 1\%$ bzgl. der Hypothesen

$$H_0 : p_0 \geq 0.10 \text{ versus } H_1 : p_0 < 0.10$$

durchgeführt.

```
> binom.test(x=480,n=5000,p=0.10,alternative="less",conf.level=0.99)

        Exact binomial test

data:  480 and 5000
number of successes = 480, number of trials = 5000,
p-value = 0.1792

alternative hypothesis: true probability of success is less than 0.1

99 percent confidence interval:
```

```
 0.0000000 0.1061145
sample estimates:
probability of success
                 0.096
```

Da `p-value = 0.1792` und somit größer als 1% kann die Alternative $H_1 : p_0 < 0.10$ nicht angenommen werden.

4.4.1.3 Vorzeichentest

Mithilfe des (verteilungsfreien) Vorzeichentests können Hypothesen über den Median der zugrundeliegenden Verteilung getestet werden.

Der Vorzeichentest kann mithilfe der R-Funktion `binom.test()` durchgeführt werden. Dazu muß der Parameter `p=0.5` gesetzt werden (das ist die default-Einstellung), der Parameter `n` gibt die Stichprobenanzahl, deren Werte ungleich dem zu testenden hypothetischen Median M_0 sind, an und über den Parameter `x` in der Funktion `binom.test()` gibt man an, wieviele Stichprobenwerte kleiner (alternativ: größer) als der hypothetische Median M_0 sind.

Beispiel: Eine Stichprobe von 10 i.i.d. Zufallsvariablen mit stetiger Verteilung ergab die Werte

$$1, 4, 3, 5, 2, 7, 6, 0, 8, 9.$$

Für den Median M der zugrundeliegenden Verteilung sollen die Hypothesen

$$H_0 : M \geq 5 \text{ versus } H_1 : M < 5$$

zum Signifikanzniveau $\alpha = 0.05$ getestet werden.

```
> binom.test(x=5,n=9,p=0.5,alternative="greater")

        Exact binomial test

data:  5 and 9
number of successes = 5, number of trials = 9, p-value = 0.5
alternative hypothesis: true probability of success is greater than 0.5
95 percent confidence interval:
 0.2513676 1.0000000
sample estimates:
probability of success
             0.5555556
```

Man beachte, dass im Aufruf eine Stichprobenanzahl von `n=9` zu wählen ist, da der Beobachtungswert 5 identisch dem zu testenden Median ist. Als Wert der Teststatistik (= Anzahl der Stichprobenwerte, die kleiner als 5 sind) erhält man hier 5. Da `p-value = 0.5` kann die Alternative $H_1 : M < 5$ zum Signifikanzniveau $\alpha = 0.05$ nicht angenommen werden.

Man beachte, dass für den Fall der hier vorliegenden Alternative

$$H_1 : M < 5$$

eine Trefferwahrscheinlichkeit p für das Ereignis:

$$\{\text{Stichprobenwert } X_i < 5\}$$

vorliegen müsste, die größer als $\frac{1}{2}$ ist. Daher setzt man in diesem Beispiel im Binomialtest den Parameter `alternative` auf `greater`.

4.4.1.4 Wilcoxon-Vorzeichen-Rangtest

Ist die zugrundeliegende Verteilung der Stichprobenvariablen zusätzlich zur Stetigkeit noch symmetrisch, kann der Wilcoxon-Vorzeichen-Rangtest zum Testen von Hypothesen über den unbekannten Median der Verteilung verwendet werden. Als nichtparametrischer (verteilungsfreier) Test benötigt man keine exakte Verteilungsspezifizierung (z.B. Normalverteilung). Im Unterschied zum Vorzeichentest nutzt der Wilcoxon-Vorzeichen-Rangtest nicht nur die Tatsache, ob Stichprobenwerte größer oder kleiner als ein hypothetischer Median sind, aus, sondern die Informationen über die Ränge der Stichprobenwerte gehen in die Berechnung mit ein.

In der Literatur wird der Wilcoxon-Vorzeichen-Rangtest auch als *Mann-Whitney U-Test* bezeichnet.

Der Test wird mit der Funktion `wilcox.test()` durchgeführt, wobei über den Paramter `mu` der hypothetische Median für die Testhypothesen gesetzt wird.

Für den Fall, dass Bindungen vorliegen, müssen Korrekturen verwendet werden. Der logische Parameter `correct` gibt an, ob bei der Normalapproximation eine Stetigkeitskorrektur verwendet werden soll. Der logische Parameter `exact` entscheidet, ob exakte oder approximierte p-Werte berechnet werden.

Beispiel:

```
> x <- rnorm(50)
> wilcox.test(x,mu=0) # alternative = "two.sided" ist default
```

```
        Wilcoxon signed rank test with continuity correction

data:  x
V = 579, p-value = 0.5756
alternative hypothesis: true mu is not equal to 0

> wilcox.test(x,mu=1,alternative="less")

        Wilcoxon signed rank test with continuity correction

data:  x
V = 52, p-value = 8.156e-09
alternative hypothesis: true mu is less than 1
```

4.4.1.5 χ^2-Varianz-Test

Voraussetzung: Stichprobe $x_1, \ldots, x_n$ von n i.i.d. Stichprobenvariablen $X_1, \ldots, X_n$, mit $X_i \sim N(\mu, \sigma^2)$. μ und σ^2 unbekannt.

Hypothesen (als Beispiel ein eineitiger Test):

$$H_0 : \sigma^2 = \sigma_0^2 \text{ vs. } H_1 : \sigma^2 > \sigma_0^2$$

Teststatistik:

$$\widehat{\chi}^2 := \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{\sigma_0^2} = \frac{(n-1)s^2}{\sigma_0^2}$$

Entscheidung für H_1, falls

$$\widehat{\chi}^2 > \chi^2_{n-1,1-\alpha}$$

4.4.2 Test auf Lage- und Streuungsparameter in der Zwei-Stichprobensituation

4.4.2.1 Verbundene Stichproben

Gegeben ist die bivariate, verbundenen Stichprobe $(x_1, y_1), \ldots, (x_n, y_n)$. Man bildet nun die Differenzenstichprobe

$$d_i := x_i - y_i,\ i = 1, \ldots, n,$$

und wendet auf die Stichprobe der Differenzen d_i die entsprechenden Signifikanztests (t-Test, Vorzeichentest oder Wilcoxon-Vorzeichen-Rangtest) aus der Einstichprobensituation an.

Um z.B. die Frage, ob der Erwartungswert μ_x der $x-$Stichprobe zugrundeliegenden Verteilung identisch dem Erwartungswert μ_y der $y-$Stichprobe zugrundeliegenden Verteilung ist, können (unter entsprechenden theoretischen Voraussetzungen an die Verteilungen) die Hypothesen

$$H_0 : \mu_x = \mu_y \text{ versus } H_1 : \mu_x \neq \mu_y$$

wiefolgt getestet werden. Man führt mittels der R-Funktion `t.test()` einen Signifikanztest mit der Differenzenstichprobe d_i zu den Hypothesen

$$H_0 : \mu_d = 0 \text{ versus } H_1 : \mu_d \neq 0$$

durch.
Durch Setzten des Funktionsparameters `paired = TRUE` innerhalb der R-Funktion `t.test` bzw. in `wilcox.text()` werden die Test-Varianten für verbundene Stichproben angesprochen.

4.4.2.2 Unverbundene Stichproben

Für unverbundenen unabhängige Stichproben können in R (innerhalb des in der Basis-Version von R bereits installierten Package `ctest`) die Zweistichprobenvarianten des $t-$Test und des Wilcoxon-Vorzeichen-Rangtest durchgeführt werden. Man verwendet dazu die schon aus der Einstichprobensituation bekannten R-Funktionen `t.test()` bzw. `wilcox.test()`

Falls die Daten der zu vergleichenden Gruppen G (Gruppe 1) und H (Gruppe 2) (mit unbekannten Erwartungswerten μ_1 und μ_2) in zwei Datenvektoren `g` (mit Stichprobenumfang n_1) und `h` (mit Stichprobenumfang n_2) gegeben sind, ergibt sich z.B. für den $t-$Test der Hypothesen

$$H_0 : \mu_1 \geq \mu_2 \text{ versus } H_1 : \mu_1 < \mu_2$$

der Aufruf

```
t.test(x=g,y=h,alternative="less",var.equal=TRUE)
```

Wie in der Einstichprobensituation kann man über den Parameter `alternative` die Alternativhypothese festlegen.

Wie in der Einstichprobensituation kann über den Parameter `conf.level` die Sicherheitswahrscheinlichkeit $1-\alpha$ für das entsprechende Konfidenzintervall spezifiziert werden.

Der logische Parameter `var.equal` in `t.test()` kann auf `TRUE` gesetzt werden, falls man in beiden Gruppen gleiche Varianzen annehmen kann (diesen Signifikanztest nennt man *2-Stichproben-t-Test*). Im Fall von ungleichen Varianzen in den Gruppen setzt man `var.equal=FALSE` (das ist auch die default-Einstellung). In den Berechnungen von R wird dann eine Modifikation des Test (der nur approximativ gültige *Aspin-Welch-Test*) durchgeführt.

2-Stichprobenvarianz-Test (Varianz-Quotiententest, F-Test)

Der 2-Stichprobenvarianz-Test wird als *Vorschalttest* bzgl. der Varianzhomogenität in den Gruppen für den 2-Stichproben-t-Test (*Haupttest*) bei unverbundenen Stichproben verwendet. Man setzt neben der Normalverteilung die Unabhängigkeit der Gesamtstichprobe voraus.

$$H_0 : \sigma_1^2 = \sigma_2^2 \text{ versus } H_1 : \sigma_1^2 \neq \sigma_2^2,$$

wobei σ_1^2 und σ_2^2 die Varianzen der Stichprobenvariablen der beiden Gruppen bezeichnen.

Die verwendete Teststatistik

$$F := \frac{s_1^2}{s_2^2}$$

ist unter H_0 asymptotisch $F_{n_1-1,n_2-1}-$verteilt.

In der Anwendung wählt man für den Vorschalttest (da man hier mit H_0 weiterarbeiten muss) ein eher größeres Signifikanzniveau α, z.B. $\alpha = 0,1;\ 0,2$.

R-Funktion: `var.test()`

Falls möglich, sollte der Vorschalt- und der Haupttest an unterschiedlichen Stichproben durchgeführt werden. Ist dies nicht möglich, muss im Haupttest mit einem größerem Fehler 1. Art, als das gewählte Signifikanzniveau α suggeriert, gerechnet werden.

Lehnt der Vorschalttest die Varianzhomogenität ab, gibt es folgende Vorgehensweisen:

- Varianzhomogenisierende Transformationen (z.B. log)

- Aspin-Welch-Test

- Nichtparametrischer Tests: Zweistichproben-Wilcoxon-Vorzeichen-Rangtest

Zweistichproben-Wilcoxon-Vorzeichen-Rangtest:

Die im Test verwendete Teststatistik wird auch als *Mann-Whitney $U-$Statistik* bezeichnet und basiert auf den Rangsummen der Teilstichproben.

Hypothesen:
$H_0: F(x) = G(x) \, \forall x \in \mathbb{R}$, d.h. die Verteilungen sind identisch
vs.
$H_1: F(x) \neq G(x)$, d.h. die Verteilungen unterscheiden sich
und einseitige Variationen (vgl. Literatur).

Anwendung: Der Test deckt vor allem Unterschiede in der zentralen Lage der Grundgesamtheiten auf und ist damit eine nichtparametrische Alternative zum 2-Stichproben t-Test.

Der Wilcoxon-Vorzeichen-Rangtest kann mithilfe der R-Funktion `wilcox.test()` durchgeführt werden.

Bemerkung: a) Der Test erkennt auch bei großen Stichprobenumfängen nicht jeden Unterschied zwischen den Verteilungen. Er erkennt aber Lage-Unterschiede sehr gut. Daher fordert man in der Anwendung oft noch zusätzlich, dass die Verteilungen der Teilstichproben bis auf die Lage ähnlich sind. Somit ist über den Test ein Vergleich der Mediane möglich. Bei symmetrischen Verteilungen kann man über die Mediane auch auf die Erwartungswerte der Teilstichproben schließen.

b) Für den Fall, dass Bindungen vorliegen, müssen Korrekturen verwendet werden. Der logische Parameter `correct` gibt an, ob bei der Normalapproximation eine Stetigkeitskorrektur verwendet werden soll. Der logische Parameter `exact` entscheidet, ob exakte oder approximierte p-Werte berechnet werden.

c) Mithilfe der R-Funktion `wilcox.exact()` aus dem Package `exactRankTests` (das man zunächst installieren muß) kann man einen exakten Wilcoxon-Vorzeichen-Rangtest (sowohl in der Ein-, als auch in der Zweistichprobensituation) auch bei Bindungen durchführen.

d) Hinweis für die praktische Anwendung: Sind die Voraussetzungen des $t-$Tests zweifelhaft, dann sollte man immer zum $t-$Test auch den Wilcoxon-Vorzeichen-Rangtest rechnen. Vorsicht ist geboten, wenn beide Tests eine unterschiedliche Entscheidung liefern.

Ein entsprechender Aufruf für den Zweistichproben-Wilcoxon-Vorzeichen-Rangtest hat hier die Form

```
wilcox.test(x=g,y=h,alternative="two.sided",correct=F)
```

4.4.2.3 Exkurs zu Versuchdesigns: Randomisierung und Blockexperiment

Problemstellung: Auf Basis zweier Stichproben

$$x_i, \, i = 1, \ldots, n, \text{ und } y_i, \, i = 1, \ldots, m$$

soll entschieden werden, ob sich die Verteilungen der zugrundeliegenden Stichprobenvariablen X und Y in ihren Erwartungswerten (oder Medianen oder Varianzen oder Quantilen oder ...) unterscheiden.

Oft ist die praktische Fragestellung, ob sich ein und dieselbe metrische Messgröße (z.B. Schadenquote) in zwei verschiedenen Situationen (z.B. Wohngebäudetyp A und B) unterscheidet. In der praktischen, statistischen Analyse werden im Versuchsdesign zur Datengewinnung zwei grundlegende Konzepte angewandt (siehe (U) und (G) unten).

Voraussetzung (U) Man setzt voraus, dass die Stichproben Realisationen unabhängiger Zufallsvariablen X_i, Y_j, $i = 1, \ldots, n$, $j = 1, \ldots, m$, sind. Man betrachtet also die Situation unverbundener (ungepaarter) Stichproben.

Das Versuchsdesign bei (U) entspricht dem Konzept des *Randomisierens*. Durch die zufällige Auswahl der Objekte hofft man, den Einfluss aller Grössen - bis auf die Gruppen-bildenden Variable (z.B. Wohngebäudetyp) - ausblenden zu können. Man beachte, dass in diesem Konzept durch die Anwendung einer echten Zufallsauswahl ein sogenannter *Stichprobenbias* vermieden werden muß. Werden die Gruppeneinteilungen nicht wirklich zufällig getroffen, kann einen systematische Verzerrung eintreten, die i.A. innerhalb der folgenden statisitschen Untersuchung nicht mehr korregiert werden kann.

Voraussetzung (G) In dem zweiten klassischen Vorgehen bildet man ein Versuchsdesign, bei dem eine metrische Messgröße in zwei Situationen an ein und demselben Objekt gemessen wird, z.B. die versicherte Summe in einer Hausratversicherung vor und nach einem Informationsgespräch jeweils gemessen an n Verträgen bzw. Versicherungskunden. Es entsteht eine verbundene (gepaarte) bivariate Stichprobe $(x_1, y_1), \ldots, (x_n, y_n)$. Hier fordert man dann, dass die zugehörigen Zufallspaare (X_i, Y_i), $i = 1, \ldots, n$, unabhängig sind. Man beachte, dass X_i und Y_i (für den gleichen Index i) i.A. nicht unabhängig sind.

Bei (G) spricht man von einem *Blockexperiment* und man hofft durch die Verwendung der identischen Objekte nur den Einfluss der einen interessierenden Variable auf die metrische Größe zu erhalten. Durch Blockexperimente wird die störende Streuung zwischen den Objekten (*within-sample variability*) reduziert. Man beachte aber, dass man in bestimmten Fällen in der Praxis keine Blockexperimente durchführen kann.

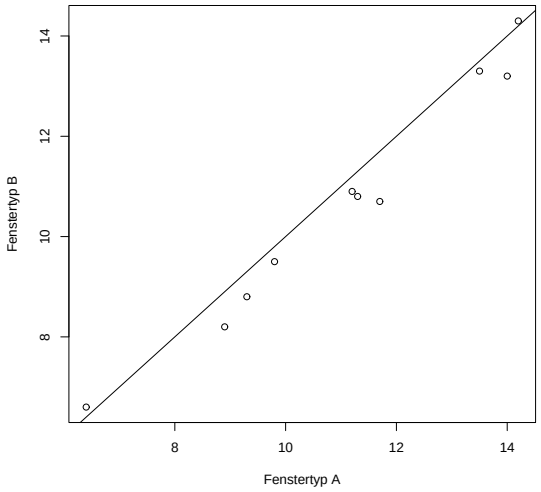
Beispiel: Verbundene Stichproben (Blockexperimente) bilden ein Versuchsdesign, das die Streuungen zwischen den Objekten aus den Daten reduziert und so auch sehr geringe Lage- bzw. Verteilungsunterschiede aufgedeckt werden können.

Wir untersuchen einen fiktiven Datensatz. Bei 10 Häusern werden jeweils in einem Zimmer Fenster vom Typ A und in einem anderen Zimmer Fenster vom Typ B eingebaut. Für jedes Haus und jedes der beiden Zimmer wird eine pyhsikalische Größe Y (z.B. bzgl. des Raumklimas) gemessen. Die Zimmer Z_1 und Z_2 sollen in ihrem Einfluss auf Y vergleichbare Eigenschaften besitzen. Ziel der Untersuchung ist die Frage, ob sich Y

- genauer $E(Y)$ bzw. die Verteilung von Y - mit den verwendeten Fenstertypen A oder B verändert. Es ergibt sich folgende Beobachtungstabelle

Haus	Fenstertyp A	Fenstertyp B
1	14.0 (Zimmer Z_1)	13.2 (Zimmer Z_2)
2	8.9 (Zimmer Z_1)	8.2 (Zimmer Z_2)
3	11.2 (Zimmer Z_2)	10.9 (Zimmer Z_1)
4	14.2 (Zimmer Z_1)	14.3 (Zimmer Z_2)
5	11.7 (Zimmer Z_2)	10.7 (Zimmer Z_1)
6	6.4 (Zimmer Z_1)	6.6 (Zimmer Z_2)
7	9.8 (Zimmer Z_2)	9.5 (Zimmer Z_1)
8	11.3 (Zimmer Z_1)	10.8 (Zimmer Z_2)
9	9.3 (Zimmer Z_1)	8.8 (Zimmer Z_2)
10	13.5 (Zimmer Z_2)	13.3 (Zimmer Z_1)

```
> A <- c(14.0,8.9,11.2,14.2,11.7,6.4,9.8,11.3,9.3,13.5)
> B <- c(13.2,8.2,10.9,14.3,10.7,6.6,9.5,10.8,8.8,13.3)
> plot(A,B,xlab="Fenstertyp A",ylab="Fenstertyp B")
> abline(0,1) # Gerade y = x in Scatterplot einzeichnen
```



In der resultierenden Grafik sieht man, dass sich die physikalische Größe Y pro betrachtetes Haus kaum mit dem Fenstertyp ändert (Punkte im Scatterplot liegen annähernd auf der Winkelhalbierenden). Allerdings ändert sich die Größe Y, wenn man sie über alle Häuser hinweg betrachtet.

Zu den Tests:

```
# Zwei-Stichproben t-Test fuer unabhaengige Stichproben
```

```
> t.test(A,B,var.equal=TRUE)

        Two Sample t-test

data:  A and B t = 0.3617, df = 18, p-value = 0.7218

alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.923522  2.723522
sample estimates: mean of x mean of y
    11.03     10.63

# Zwei-Stichproben t-Test fuer verbundene Stichproben
> t.test(A,B,var.equal=TRUE,paired=TRUE)

        Paired t-test

data:  A and B t = 3.3282, df = 9, p-value = 0.008824

alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.1281225 0.6718775
sample estimates: mean of the differences
                  0.4

# Wilcoxon-Test fuer verbundene Stichproben
> wilcox.test(A,B,paired=TRUE)

        Wilcoxon signed rank test with continuity correction

data:  A and B V = 51.5, p-value = 0.01646

alternative hypothesis: true mu is not equal to 0

Warning message: Cannot compute exact p-value with ties in:
wilcox.test.default(A, B, paired = TRUE)
```

Folgerung: Der Zwei-Stichproben $t-$Test für unabhängige Stichproben findet keinen Unterschied der Erwartungswerte von Y je nach Fenstertyp. Die Tests für verbundene Stichproben reduzieren die Streuung zwischen den unterschiedlichen Häusern und finden einen Unterschied je nach Fenstertyp. Für die Praxis bleibt allerdings die Frage, ob der kleine Unterschied von $\Delta E(Y) = 0.4$ überhaupt für die Anwendung interessant ist.

4.4.3 Anpassungstests, Verteilungstests

Kolmogorov-Smirnov Test

Einstichproben-Situation:

Nullhypothese H_0 : Stichprobenwerte $x_1, \ldots, x_n$ sind unabhängige Realisationen einer Zufallsvariablen mit Verteilungsfunktion F_0. F_0 ist fest vorgegeben und wird als stetig vorausgesetzt.

$$\text{Teststatistik: } d_n := \sup_{x \in \mathbb{R}} |F_n(x) - F_0(x)|$$

R-Funktion: `ks.test()`

Im der R-Funktion `ks.test()` im package *ctest* muss die hypothetische Verteilung (genauer die Verteilungsfunktion) exakt vorgegeben werden, d.h. zur Verteilung müssen auch die Verteilungsparameter im Funktionsaufruf angegeben werden. In modifizierten Funktionen zum Kolmogorov-Smirnov Test reicht eine Angabe der Verteilung, die Verteilungsparameter werden dann automatisch geschätzt. In diesem Fall müssen allerdings die kritischen Bereiche angepasst werden (*Lilliefors-Modifikation des Kolmogorov-Smirnov Tests*).

R-Funktion: `lillie.test()` im Paket `nortest`.

Beispiele:

```
> x <- rnorm(50)
> ks.test(x,"pnorm",mean(x),sd(x))

        One-sample Kolmogorov-Smirnov test

data:  x
D = 0.0754, p-value = 0.9385
alternative hypothesis: two.sided

> ks.test(x,"pnorm",0,1)

        One-sample Kolmogorov-Smirnov test

data:  x
D = 0.1267, p-value = 0.3981
alternative hypothesis: two.sided
```

Zweistichproben-Situation:

Nullhypothese H_0 : Stichprobenwerte $x_1, \ldots, x_n$ und Stichprobenwerte $y_1, \ldots, y_m$ sind unabhängige Realisationen einer Zufallsvariablen mit stetiger Verteilungsfunktion, d.h. die beiden Stichproben basieren auf identisch verteilte Stichprobenvariabeln. Die Verteilungen bzw. die Verteilungsfunktionen F und G in beiden Gruppen sind identisch.

$$\text{Teststatistik: } d_n := \sup_{x \in \mathbb{R}} |F_n(x) - G_n(x)|,$$

wobei F_n und G_n die empirischen Verteilungsfunktionen der beiden Teilstichproben bezeichnen.

Beispiel:

```
> x <- rnorm(30)
> y <- runif(n=50,min=-1,max=1)
> ks.test(x,y)

        Two-sample Kolmogorov-Smirnov test

data:  x and y D = 0.1867, p-value = 0.4809
alternative hypothesis: two.sided
```

Bemerkung: Der Zwei-Stichproben Kolmogorov-Smirnov Test deckt (bei genügend großen Stichprobenumfängen) beliebige Abweichungen der Verteilungen auf. Der Zweistichproben-t-Test und der Zweistichproben-Wilcoxon-Vorzeichen-Rangtest sind vor allem sensibel bzgl. Lageunterschiede in den Verteilungen.

χ^2-Anpassungstests

a) Anpassungstest bei kategorialem Merkmal, $1 \times k$-Kontingenztafel

Anwendung als Anpassungstest auf Verteilungen mit endlicher Wertemenge.

Voraussetzung: Stichprobe $x_1, \ldots, x_n$ von n i.i.d. wie X verteilten Stichprobenvariablen X_i mit (eventuell nach Transformation) Wertemenge $X(\Omega) = \{1, \ldots, k\} \subset \mathbb{N}$. Das betrachtete Merkmal ist nominal skaliert. Es wird ein großer Stichprobenumfang n vorausgesetzt (asymptotischer Test) und weiter wird eine ausreichend große Besetzung der resultierenden Zellen angenommen.

Nullhypothese:

$$H_0 \,:\, P(X = i) = \pi_i, \, \forall i = 1, \ldots, k$$

Alternative:

$$H_1 \,:\, P(X = i) \neq \pi_i \text{ für mind. ein } i,$$

wobei die $0 < \pi_i < 1$ mit $\sum_{i=1}^{k} \pi_i = 1$ fest vorgegeben sind.

Teststatisik: χ^2-Koeffizient-Teststatistik ist unter H_0 approximativ $\chi^2(k-1)$-verteilt.

Regel für Anwendbarkeit der Approximation: $n\pi_i \geq 1$ für alle i und $n\pi_i \geq 5$ für mindestens 80% der Zellen.

R-Funktion: `chisq.test()`

Beispiel: Liefert `sample()` einen Laplace-Würfel?

```
> x <- sample(x=c(1,2,3,4,5,6),size=100,replace=TRUE)

> table(x)
x
 1  2  3  4  5  6
19 14 22 18 18  9

> hfkt <- table(x)
> chisq.test(x=hfkt,p=c(1/6,1/6,1/6,1/6,1/6,1/6))

        Chi-squared test for given probabilities

data:  hfkt
X-squared = 6.2, df = 5, p-value = 0.2872
```

b) Anpassungstest für Verteilungen mit unendlicher oder überabzählbarer Wertemenge

Vorgehen und Voraussetzungen: Wie in a), aber zunächst Gruppierung der n-Stichprobenwerte in k Intervalle. Die hypothetischen Wahrscheinlichkeiten $\pi_1, \ldots, \pi_k$ zur Formulierung der Nullhypothese werden über die vollständig vorgegebene, hypothetische Verteilung bestimmt,

$$\pi_j = P\left(\text{Beobachtung liegt in Intervall } j\right).$$

Modifikation: Aus der Stichprobe werden noch unbekannte Parameter der hypothetischen Verteilung geschätzt. Nach der *Regel von Fisher* wird dann der Ablehnungbereich von H_0 im χ^2-Test angepasst. Beim χ^2-Anpassungstest auf Vorliegen einer Normalverteilung werden z.B. der Erwartungswert und die Varianz geschätzt. Für den Ablehnungsbereich verwendet man dann das $(1-\alpha)$-Quantil der χ^2_{k-3}-Verteilung (wobei k die Anzahl der betrachteten Intervalle ist). D.h. es sind nun (wegen der zusätzlichen Schätzung der beiden Parameter) $k-3$ Freiheitsgrade im Unterschied zu $k-1$ Freiheitsgrade (bei einer vollständig spezifizierten, hypothetischen Verteilung) zu betrachten.

Bei einer Schätzung von p Verteilungsparametern zur Festleung der hypothetischen Verteilung verwendet man also nach der Regel von Fisher das $(1-\alpha)$-Quantil der χ^2_{k-1-p}-Verteilung

Anwendung: Z.B. Test auf Normalverteilung

R-Funktion: `chisq.test()`

Bemerkung: Der logische Parameter `correct` der Methode `chisq.test()` gibt an, ob beim Bilden der Teststatistik eine Stetigkeitskorrektur verwendet werden soll. Diese Korrektur ist dann angebracht, wenn die zu testende Verteilung eigentlich stetig ist und für den Test zunächst gruppiert d.h. diskretisiert wird.

Shapiro-Wilk Test

Der Shapiro-Wilk Test prüft die Nullhypothese, dass die Stichprobe $x_1, \ldots, x_n$ unabhängige Realisationen einer normalverteilten Zufallsvariablen sind.

Die Teststatistik im Shapiro-Wilk Test ist der Quotient zweier unterschiedlicher Schätzer für die Varianz der Stichprobenvariabeln. Zum einen wird die Varianz über einen KQ-Schätzer der Steigung einer Regressionsgeraden im QQ-Plot geschätzt, zum anderen über die Stichprobenvarianz s^2. Unter H_0 sollten beide Schätzwerte für die Varianz annähernd identisch sein, d.h. der Quotient der Schätzwerte sollte ungefähr 1 sein.

Der Shapiro-Wilk Test besitzt im Vergleich zu anderen Tests auf Normalverteilung eine große Power.

R-Funktion: `shapiro.test()`

Anderson-Darling Test

Test auf spezielle Verteilung (z.B. Normalverteilung) der der Stichprobe zugrundeliegenden Stichprobenvariablen.

R-Funktion: `ad.test()` im Paket `nortest`.

4.4.4 χ^2-Tests auf Unabhängigkeit und Homogenität - nominale Merkmale

Mithilfe der Funktion `chisq.test()` können χ^2-Tests auf Kontingenztabellen durchgeführt werden. Dabei wird der χ^2-Koeffizient als Teststatistik verwendet.

a) Test auf Unabhängigkeit zweier nominaler Merkmale

Datensituation und Voraussetzung:

Bivariate Stichprobe $(x_1, y_1), \ldots, (x_n, y_n)$ zweier nominaler Merkmale X und Y. X besitzt die möglichen Ausprägungen $1, \ldots, M$ und Y die möglichen Ausprägungen $1, \ldots, N$. Die Teilstichproben $(x_1, \ldots, x_n)$ und $(y_1, \ldots, y_n)$ werden jeweils als unabhängig vorausgesetzt. Die Daten ergeben eine $M \times N$-Kontingenztabelle. Weiter setzt man einen großen Stichprobenumfang n und eine genügend große Zellenbesetzung voraus (asymptotischer Test).

Nullhypothese:

$H_0 : P(X = i, Y = j) = P(X = i) \cdot P(Y = j),$ für alle $i \in \{1, \ldots, M\}, j \in \{1, \ldots, N\}$

Also: X und Y sind unabhängig.

Alternative:

$$H_1 : P(X = i, Y = j) \neq P(X = i) \cdot P(Y = j) \text{ für mind. ein } i, j$$

Beispiel: $n = 227$ Personen mit den Berufen A oder B werden befragt, welche der 3 Anlageformen u, v, w sie bevorzugen.
Frage: Gibt es einen Zusammenhang zwischen dem Beruf und der bevorzugten Anlageform?

```
> daten <- matrix(c(30,40,25,56,30,46),nrow=2)
> daten
     [,1] [,2] [,3]
[1,]   30   25   30
[2,]   40   56   46

> dimnames(daten) <- list(c("A","B"),c("u","v","w"))
> daten
   u  v  w
A 30 25 30
B 40 56 46
> chisq.test(daten)

        Pearson's Chi-squared test

data:  daten
X-squared = 2.5065, df = 2, p-value = 0.2856
```

b) Homogenitätstest

Datensituation und Voraussetzungen:

Für I Gruppen (unabhängige Stichproben) ist jeweils die Anzahl des Auftretens von K möglichen Ausprägungen eines nominalen Merkmals Y gegeben. Die Häufigkeits-Daten sind in einer $I \times K$-Kontingenztabelle darstellbar. Man setzt einen großen Stichprobenumfang n und eine genügend große Zellenbesetzung voraus (asymptotischer Test).

Nullhypothese: Alle Gruppen besitzen die identische Wahrscheinlichkeitsverteilung bzgl. der K Ausprägungsalternativen, d.h. die Gruppen sind homogen.

Alternative: mind. zwei Gruppen besitzen unterschiedliche Wahrscheinlichkeitsverteilungen bzgl. der Ausprägungsalternativen.

Beispiel: Die $n = 2$ Mio. Kunden von $I = 4$ Versicherungsunternehmen U_1, U_2, U_3, U_4 werden hinsichtlich ihrer Berufsstruktur

$$A = \text{ Angestellter}, B = \text{ Beamter}, C = \text{ Arbeitslos},$$

untersucht. Frage: Besitzen die 4 Unternehmen bzgl. der Berufsstruktur identische Kundengruppen?

```
> daten <- matrix(c(150000,40000,10000,280000,20000,5000,300000,95000,10000,
+ 80000,8000,2000),nrow=4)

> daten
       [,1]   [,2]  [,3]
[1,] 150000  20000 10000
[2,]  40000   5000 80000
[3,]  10000 300000  8000
[4,] 280000  95000  2000

> dimnames(daten) <- list(c("U1","U2","U3","U4"),c("A","B","C"))

> daten
        A      B     C
U1 150000  20000 10000
U2  40000   5000 80000
U3  10000 300000  8000
U4 280000  95000  2000

> chisq.test(daten)

        Pearson's Chi-squared test

data:  daten
X-squared = 977276.9, df = 6, p-value = < 2.2e-16
```

c) Spezialfall: Homogenitätstests in $(k \times 2)-$Kontingenztafeln

Datensituation: In k Gruppen wird eine *binäre Variable* (Wertemenge $\{0,1\} \equiv \{$ Treffer, Niete$\}$) beobachtet. Die Stichprobe kann in einer $(k \times 2)-$Kontingenztafel zusammengefasst werden.

Beispiel: Für 100 Produkte, gefertigt aus dem Material A, B, C oder D, wird nach einem Belastungstest die Funktionsfähigkeit $\{0,1\} = \{$ Nein, Ja$\}$ geprüft.

Gruppe (Material)	funktionsfähig	nicht funktionsfähig	$\widehat{p}_j$
1 (A)	16	10	$\frac{16}{26} \approx 0.62$
2 (B)	22	15	$\frac{22}{37} \approx 0.59$
3 (C)	16	33	$\frac{16}{49} \approx 0.33$
4 (D)	5	22	$\frac{5}{27} \approx 0.19$

Ziel der Untersuchung ist eine Überprüfung, ob die Funktionsfähigkeits-Wahrscheinlichkeiten p_j, $j = 1,2,3,4$, bei allen Material-Gruppen identisch sind, oder sich unterscheiden. Man befindet sich also sozusagen in der Situation eines *Mehr-Stichproben Binomialtests*.

Falls ein Unterschied in den p_j vorhanden ist, soll weiter untersucht werden, in welchen Gruppen (paarweise) die Wahrscheinlichkeiten unterschiedlich sind.

Der im folgenden angewandte *Proportionstest* (`prop.test()`) verwendet die Hypothesen
$H_0 : p_1 = p_2 = \ldots = p_k$
vs.
$H_1 : \exists\, i,j \in 1,\ldots,k, : p_i \neq p_j$.
Einseitige Varianten sind möglich.
Für paarweise Vergleiche der p_j wird der modifizierte Test mit einer $p-$Wert-Adjustierung (*Bonferroni-* oder default: *Holm*-Methode) verwendet.

Die Voraussetzungen der Tests sind ein genügend großer Gesamt-Stichprobenumfang (asymptotischer Test!), genügend große Besetzung der Zellen der Kontingenztafel und die Unabhängigkeit der Beobachtungen. Bei kleinerem Gesamt-Stichprobenumfang verwendet man die *Yates-Korrektur* (vgl. Literatur). Es folgen die Analysen des obigen Beispiels mit den entsprechenden R-Befehlen.

```
> # Kontingenztabelle als Daten-Matrix defnieren
> funktionsfaehig <- c(16,22,16,5)
> n.funktionsfaehig <- c(10,15,33,22)
> Tabelle.d <- cbind(funktionsfaehig,n.funktionsfaehig)
> rownames(Tabelle.d) = c("A","B","C","D")
> Tabelle.d
  funktionsfaehig n.funktionsfaehig
```

```
A              16                10
B              22                15
C              16                33
D               5                22

> # Propotionstest
> prop.test(Tabelle.d)

        4-sample test for equality of proportions without continuity
        correction

data:  Tabelle.d X-squared = 16.5149, df = 3, p-value = 0.0008891
alternative hypothesis: two.sided
sample estimates:
   prop 1    prop 2    prop 3    prop 4
0.6153846 0.5945946 0.3265306 0.1851852

> # Paarweise Vergleiche
> pairwise.prop.test(Tabelle.d)

        Pairwise comparisons using Pairwise comparison of proportions

data:  Tabelle.d

  A     B     C
B 1.000 -     -
C 0.096 0.096 -
D 0.017 0.015 0.587

P value adjustment method: holm
```

d) Spezialfall: (2×2)**-Kontingenztafel**

Für (2×2)-Kontingenztafeln (Vierfeldertafeln) kann man auch bei geringem Gesamt-Stichprobenumfang $n < 20$ den *exakten Test von Fisher* anwenden.

R-Funktion: `fisher.test()`.

4.4.5 Assoziationstests (Korrelations- und Unabhängigkeitstests) bei metrischen und ordinalen Merkmalen

a) Unabhängigkeitstest bei zweidimensionaler Normalverteilung

Voraussetzung: Der Zufallsvektor der Stichprobenvariablen $(X_i, Y_i)^\top$, $i = 1, \ldots, n$, ist identisch 2-dim. normalverteilt. Die Teilstichproben-Variablen X_i, $i = 1, \ldots, n$, und Y_i, $i = 1, \ldots, n$ sind jeweils unabhängig.

Ziel: Beurteilung, ob die Zufallsvariablen X_i und Y_i unabhängig sind.

Unter der Normalverteilungsannahme gilt, dass

$$X_i, Y_i \text{ unkorreliert } \Leftrightarrow X_i, Y_i \text{ unabhängig .}$$

X_i und Y_i heißen unkorreliert, falls fur den Korrelationskoeffizienten gilt, dass

$$\varrho(X_i, Y_i) = 0.$$

Teststatistik: Mit dem empirischen Korrelationskoeffizienten

$$r_{xy} = \frac{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{s_x s_y},$$

dieser ist ein Schätzer für $\varrho(X_i, Y_i)$, betrachtet man die Teststatistik

$$T = \sqrt{n-2}\frac{r_{xy}}{\sqrt{1 - r_{xy}^2}}.$$

Hypothesen:

$$H_0 : \varrho(X_i, Y_i) = 0 \text{ vs. } H_1 : \varrho(X_i, Y_i) \neq 0.$$

Ablehnugsbereich:

Unter H_0 gilt, dass $T \sim t_{n-2}$. Man lehnt H_0 ab, falls

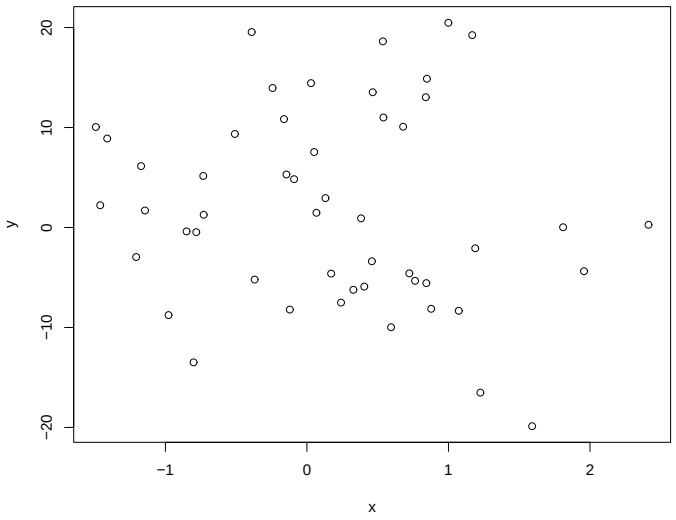
$$|T| \geq t_{n-2,1-\frac{\alpha}{2}}$$

In R kann der Test mit der Funktion `cor.test()` durchgeführt werden, wobei man den Funktionsparameter `method` auf `pearson` setzt.

Beachte: Der Test ist nicht robust bzgl. Abweichungen von der Normalverteilungsannahme.

Beispiel:

```
> x <- rnorm(50)
> y <- -2*x + rnorm(50,sd=10)
> plot(x,y)
```




```
> cor.test(x,y,method="pearson")

        Pearson's product-moment correlation

data:  x and y
t = -0.9677, df = 48, p-value = 0.3380
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.4012306  0.1456229
sample estimates:
       cor
-0.1383320
```

Beispiel:

```
> x <- rnorm(50)
> y <- x[2:50] # 2. bis 50. Komponente
> z <- x[1:49] # 1. bis 49. Komponente
> cor.test(y,z,method="pearson")

        Pearson's product-moment correlation

data:  y and z
t = 0.2688, df = 47, p-value = 0.7892
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.2447074  0.3168904
```

```
sample estimates:
       cor
0.03918556
```

b) Spearman-Rang-Zusammenhangstest

Dieser Test kann auch für ordinale Merkmale verwendet werden. Der Test verwendet als Teststatistik die *Hotelling-Pabst-Statisitk*, die bis auf eine lineare Transformation dem *Spearman-Rangkorrelationskoeffizienten* entspricht.

Während der Pearson-Korrelationskoeffizient den linearen Zusammenhang misst, betrachtet man hier nun den monotonen Zusammenhang zweier Variablen.

Der Test wird in R mit der Funktion `cor.test()` durchgeführt, wobei man den Funktionsparameter `method` auf `spaerman` setzt.

4.5 Konfidenzintervalle

Konfidenzintervalle werden in R oft durch die Funktionen der entsprechenden Signifikanztests mitberechnet und ausgegeben, vgl. `t.test()` oder `binom.test()`.

Innerhalb der statistischen Modellbildung, insbesondere bei Regressionsmodellen, werden Konfidenzintervalle automatisch in den Modellbildungs-Funktionen mitberechnet.

Für speziellere Schätzprobleme werden Konfidenzintervalle in Zusatzpaketen bereitgestellt oder können natürlich auch vom Anwender selbst programmiert werden.

Beispiel:

Bezeichne `x` den Datenvektor, dann erhält man durch

```
t.test(x)[[4]]
```

ein Konfidenzintervall für μ zum Konfidenzniveau $1 - \alpha = 95\%$.

Die optionale Einschränkung `[[4]]` im Aufruf `t.test(x)[[4]]` bewirkt, dass nur die Informationen zum Konfidenzintervall ausgegeben werden und die Ergebnisse zum entsprechenden Signifikanztest ($t-$Test) in der Anzeige unterdrückt werden. Alternativ kann man das Konfidenzintervall aus der Ausgabeliste von `t.test()` auch durch

```
t.test(x)$conf.int
```

isolieren.

Es wird eine Stichprobe x von 20 $N(0,1)-$verteilten (Pseudo-)Zufallszahlen erzeugt. Innerhalb der R-Funktion `t.test()` wird das Konfidenzintervall für den unbekannten Erwartungswert μ berechnet.

```
> x<-rnorm(20)
> t.test(x)[[4]]

[1] -0.4751918  0.4901579
attr(,"conf.level")
[1] 0.95
```

Das Konfidenzniveau kann über den Parameter `conf.level` spezifiziert werden (default-Einstellung: Konfidenzniveau $= 1 - \alpha = 95\%$.) Man erhält z.B. das Konfidenzintervall zum Konfidenzniveau 99% durch den Aufruf

```
> t.test(x,conf.level=0.99)[[4]]
```

und damit z.B. mit der Stichprobe x des obigen Beispiels die Ausgabe

```
[1] -0.7446247  0.9529400
attr(,"conf.level")
[1] 0.99
```

4.6 Statistische Modellbildung

4.6.1 Lineares Modell

Mit dem R-Basis-System können Lineare Modelle (LM) sehr komfortablel und effizient analysiert werden.

Viele der in diesem Abschnitt aufgeführten R-Methoden und Vorgehensweisen (z.B. die Spezifikation von Modellgleichungen) werden ganz analog auch im Generalisierten Linearen Modell (GLM) angewandt.

In einem Linearen Modell wird eine metrische Zielgröße (Response, abhängige Variable, Kriteriumsvariable) Y in Beziehung zu einer oder mehreren Einflussgrößen $x_1, \ldots, x_p$

(unabhängige Variablen, Kovariablen, erklärende Variablen, Regressoren, Faktoren) gesetzt. Man betrachtet die Strukturgleichung

$$Y = X \cdot \beta + e,$$

wobei Y der $n-$dimensionale Vektor der Zielgröße (man geht von einer Stichprobe vom Umfang n aus), X die $n \times p-$Matrix der bekannten Einflussgrößen (Designmatrix), β der $p-$dimensionalen Vektor der unbekannten Parameter und $e := (e_1, \ldots, e_n)^T$ der $n-$dimensionale Fehlervektor ist.

Die Strukturgleichung zeigt, dass der Einfluss der Kovariablen über die Komponenten $\beta_1, \ldots, \beta_p$ des Parametervektors β linear erfolgt.

Für die auftretenden Dimensionen und Größen gilt

$$\text{rang}(X) \leq p < n.$$

Man setzt insbesondere voraus, dass der Stichprobenumfang n größer als die Anzahl der unbekannten und zu schätzenden Modellparameter p ist.

Neben der Strukturgleichung setzt man voraus:

(LM) $e_i, i = 1, \ldots, n$, sind unabhängig mit $E(e_i) = 0$ und $Var(e_i) = \sigma^2 > 0$.

oder stärker

(NLM) $e_i, i = 1, \ldots, n$, sind unabhängig und identisch $N(0, \sigma^2)-$verteilt.

Die Voraussetzung (NLM) ist hinsichltich der Anwendung von exakten Konfidenzintervallen und Signifikanztests notwendig. Ohne die Normalverteilungsannahme können im LM nur asymptotische Verfahren angewandt werden, für die (neben anderen Regularitätsvoraussetzungen) insbesondere ein großer Stichprobenumfang n vorliegen muss.

Eine weitere günstige Modelleigenschaft ist

(VR) Designmatrix X besitzt vollen Rang p.

Gilt (VR) in einem Modell nicht, ist der übliche MQ-Schätzer $\widehat{\beta}$ für den Parametervektor β nicht mehr erwartungstreu und der Schätzer $\widehat{\beta}$ ist zumindest teilweise nicht mehr identifizierbar (d.h. es gibt mehrere Lösungen der Schätzgleichungen und die Schätzungen werden instabil). Durch geeignete Konstruktion des linearen Modells (z.B. Auswahl der im Modell involvierten Kovariablen) kann der Anwender meist eine Verletzung von (VR) vermeiden.

Regression, ANOVA und ANCOVA

Bei einer metrischen Kovariablen spricht man von einer einfachen linearen Regression. Bei mehreren metrischen Kovariablen spricht man von multiplen Regressionsmodellen.

Betrachtet man nur kategorielle (qualitative) Kovariablen, so bezeichnet man die Modellanalyse als Varianzanalyse (ANOVA). Die Kovariablen werden in der ANOVA als Faktoren bezeichnet und man untersucht den Mittelwert-Einfluss der Faktoren auf den Response Y. Je nach Anzahl $1, 2, \ldots$ der Faktoren spricht man von einer einfachen, zweifachen, ... Varianzanalyse.

Falls die Zielgröße Y in Abhängigkeit von metrischen und kategoriellen Kovariablen modelliert wird, nennt man die Modellanalyse eine Kovarianzanalyse.

Über die klassische Varianzanalyse, Kovarianzanalyse und Regressionsanalyse hinaus werden in der angewandten, statistischen Modellbildung in Regressionsmodelle auch kategorielle Kovariablen (also Faktoren) über sogenannte *Dummyvariablen* als Quasi-Regressoren in lineare Modelle integriert.

Ein einfaches Beispiel für ein lineares Modell ist die einfache lineare Regression

$$Y_i = \alpha + \beta \cdot x_i + e_i,\ i = 1, \ldots, n.$$

Die üblichen Datengrundlage sind hier n Messpaare (y_i, x_i), auf denen die Analysen zu den unbekannten Parametern $\alpha \in \mathbb{R}$ und $\beta \in \mathbb{R}$ basiert.

Die Ziele und Aufgaben der statistischen Inferenz im LM sind:

- Punkt- und Intervallschätzung der unbekannten Parameter
- Punkt- und Intervallschätzung für Y bzw. $E(Y)$ bei gegebenen Kovariablenwerten (Prognosen)
- Signifikanztests zu den Parametern (Überprüfung der Signifikanz der Kovariablen)
- Auswahl der Kovariablen - Modellwahl
- Überprüfung der Modellvoraussetzungen
- Einschätzung der Modellgüte

4.6.1.1 Modellspezifikation

Mit dem Operator $\sim$ wird die Abhängigkeitsstruktur zwischen der abhängigen Variable Y und den unabhängigen Variablen $x_1, \ldots, x_p$ spezifiziert.

Man nehme für das Folgende an, dass die Stichprobe des Beobachtungsvektors Y in dem R-Datenvektor `y` und die Kovariablenwerte in den R-Datenvektoren `x1`,`x2`,... abgelegt sind. Typischerweise sind die Datenvektoren als Spalten eines dataframe organisiert.

Beispiel:

```
mod1.gl <- y ~ x1 + x2 + x3
```

Der Strukturgleichung wird der Namen `mod1.gl` zugewiesen. Dem Modell liegt die multiple lineare Regressionsbeziehung

$$Y = \alpha + \beta_1 \cdot x1 + \beta_2 \cdot x2 + \beta_3 \cdot x3 + e$$

zugrunde.

Die Einflussgrößen $x_1, \ldots$ können metrisch (Regressoren) oder nominal (Faktoren) sein. Ein Regressor hat in R eine `numerische` Datenstruktur und ein Faktor besitzt eine `factor`-Datenstruktur.

Faktoren werden in R innerhalb von linearen Modellen automtisch durch dichotome Hilfsvariablen (*Dummyvariablen*) codiert, sodass die mathematische Strukturgleichung auch bei nominalen Einflussgrößen inhaltlich konsistent bleibt. Ein Faktor mit $k \geq 2$ Stufen (`Levels`) wird durch $k - 1$ dichotome Dummyvariablen modelliert.

Möchte man explizit ein Modell ohne Intercept α, so muss die Definitionsgleichung zu

```
mod1 <- y ~ 0 + x1 + x2 + x3
```

oder alternativ

```
mod1 <- y ~ x1 + x2 + x3 - 1
```

modifiziert werden. In der Default-Einstellung von R werden lineare Modelle mit Intercept aufgestellt.

Folgende Operatoren werden für die Modellspezifikation im LM und auch im GLM verwendet.

R-Operator	Funktionalität
`+`	Aufnahme der Variable in das Modell
`-`	Entfernen der Variable, -1 für Intercept
`:`	Wechselwirkung (Interaktion)
`*`	Aufnahme der Variablen und derer Wechselwirkungen
`/`	nested, z.B. `y/z` bedeuted `z` hat nur Wirkung in den Stufen von `y`
`^`	alle Wechselwirkungen bis zum geg. Grad
`.`	alle Variablen aus dem dataframe werden aufgenommen
`I()`	innerhalb von `I()` haben arithmetische Operatoren die ursprüngliche Wirkung

4.6.1.2 Parameterschätzung, Konfidenzintervalle und Signifikanztests

Zur statistischen Inferenz (Schätzen und Testen) innerhalb linearer Modelle stehen in R folgende Funktionen zur Verfügung.

R-Funktion	Modellklasse
`lm()`	Lineares Modell (Regression)
`anova()`	Varianzanalyse
`aov()`	Varianzanalyse `anova()` inklusive des Aufrufs von `lm()`

Die Funktionen `lm` und `aov` erwarten als Argument jeweils eine Modellspezifikation wie oben besprochen (vgl. `mod1.gl` oben) und noch weitere Parameter (z.B. die Datenvektoren bzw. das dataframe mit denen die Modellrechnungen durchgeführt werden sollen). Das zugrundeliegende dataframe wird dabei i.A. über den Parameter `data` festgelegt.

Das Ergebnis der Methode `lm` und `aov` ist ein Objekt der Klasse `lm`, d.h. ein Modell-Objekt, auf das wieder weitere Methoden angewandt werde können.

Die Argumente der Funktion `anova` sind Modell-Objekte, z.B. der Klasse `lm` (oder auch `glm`, siehe unten).

Durch z.B. die Zuweisung

```
mod1 <- lm(y ~ 0 + x1 + x2 + x3)
```

wird das Modell berechnet und man erhält ein R-Objekt mit Namen `mod1` vom Typ Modell, in dem die Berechnungsergebnisse abgelegt sind und zur weiteren Analyse verwendet werden kann.

Um die detaillierten Ergebnisse z.B. bzgl.

- Parameterschätzungen
- Teststatistiken
- $p-$Werte
- Standardfehler
- Modellgüte-Maße (z.B. R^2)
- Streuungszerlegung

zu erhalten, verwendet man am einfachsten die R-Funktion `summary()`, eine Methode, die als Argument ein R-Objekt der Klasse `lm` verarbeitet.

Für Regressionsmodelle werden neben den Punktschätzern der Modellparameter die Ergebnisse des globalen F-Tests und der partiellen F-Tests (t-Value) ausgegeben.

Die partiellen F-Tests testen jeweils, ob die Kovariable x_i unter Modelleinschluss aller anderen in der Modellgleichung spezifizierten Regressoren einen signifikant von 0 verschiedenen Regressionskoeffizienten β_i besitzt. Das Testproblem lautet also

$$H_0 \,:\, \beta_i = 0 \text{ vs. } H_1 \,:\, \beta_i \neq 0$$

Der globale F-Test untersucht die Hypothese H_0, ob alle Regressionskoeffizienten identisch 0 gesetzt werden können. Ein kleiner p-Wert beim globalen F-Test ist also eine Minimalanforderung an den Modellierungsansatz.

Bei Anwendung der Funktion `aov` erhält man die Tafel der Varianzanalyse und die entsprechenden F-Tests der Varianzanalyse bzw. Kovarianzanalyse.

Gibt man das Modell-Objekt nur in die Konsole ein (z.B. `mod1`), so erhält man nur eine eingeschränkte Auswahl der Berechnungsergebnisse.

Auf die Komponenten der Ausgabe-Liste der Funktionen (z.B. `summary`) kann wie üblich zugegriffen werden. Wichtige Modell-Größen können über folgende Funktionen, die jeweils ein Modell-Objekt als Argument erwarten, direkt abgefragt bzw. ausgegeben werden.

R-Funktion	Inhalt
`AIC()`	Ausgabe des Akaike Information Criterion
`coef()`	geschätze Koeffizienten
`confint()`	Konfidenzintervall für die Modellparameter
`fitted()`	mit Modell angepasste (geschätzte) Kriteriumswerte
`model.matrix()`	Designmatrix
`residuals()`	Residuen
`rstandard()`	standardisierte Residuen
`rstudent()`	studentisierte Residuen

Mithilfe von *effect plots* (*effect displays*), vgl.Fox (2003), kann in multiplen Regressionsmodellen (LM und GLM) der Einfluss der einzelnen Kovariablen grafisch dargestellt werden. Das R-Package `effects` bietet eine einfache Möglichkeit zur Erstellung von effect plots.

Beispiel:

```
plot(allEffects(Modellname), ask=FALSE)
```

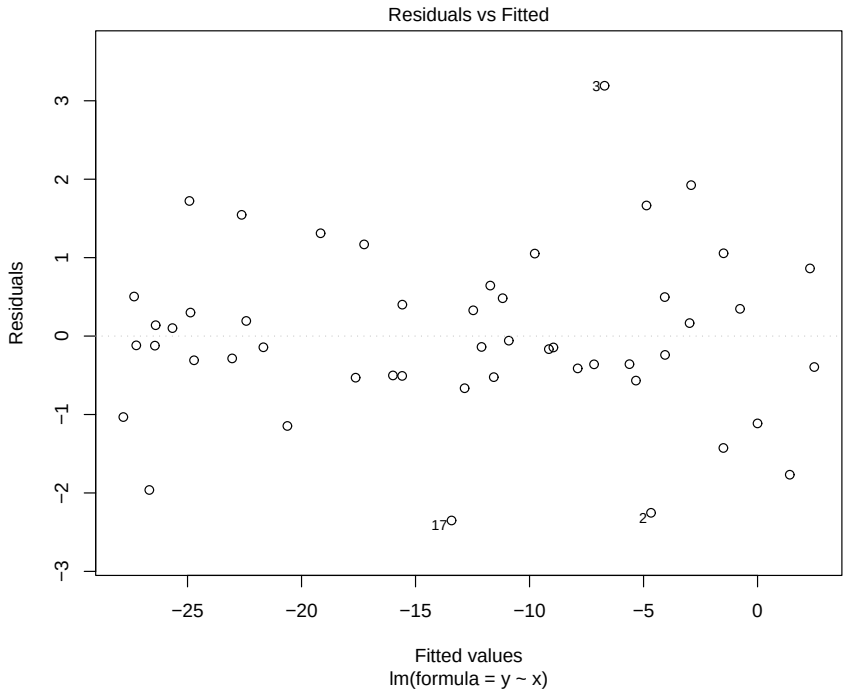
Der Parameter `Modellname` ist der Name eines Modellobjekts. Der Plot von `allEffects` erzeugt eine Grafik mit effect plots für alle Kovariablen des Modells. In einem effect plot werden default alle anderen Kovariablen auf das jeweilige Stichprobenmittel der Kovariablen gesetzt.

4.6.1.3 Überprüfung der Modellvoraussetzungen

Zur grafischen Analyse der Modellvoraussetzungen (Residualanalyse) kann man bei linearen Modellen die generische Grafik-Funktion `plot()` anwenden und erhält durch

```
plot(mod1)
```

verschiedene Diagnostik-Plots. Die grafische Analyse der Residuen (als Schätzer der



Fehlervariablen) ist das Standard-Vorgehen bei der Einschätzung der Voraussetzungen eines linearen Modells. Im Allgemeinen betrachtet man einen Plot der Residuen bzw. der standardisierten Residuen über den Prädiktionswerten der Kriteriumsvariable (*Tukey-Anscombe-Plot*) und über den Kovariablenwerten. Zur Überprüfung der Normalverteilungsannahme erstellt man einen entsprechenden QQ-Plot.

Interpretation: Der Residuenplot sollte eine horizontal ausgerichtete Punktwolke zeigen, die unsystematisch um den Wert 0 verteilt ist. Standardisierte Residuen besitzen den Vorteil, dass man die Größenordnung hinsichtlich Ausreisser an den $k \cdot \sigma-$Bereichen (mit $\sigma = 1$) ausrichten kann. Abweichungen über $2\sigma = 2$ gelten als Ausreisserverdächtig. Eine systematische Veränderung der Streuung der Residuenwerte um 0 deutet auf eine vorliegende Heteroskedastizität hin.

Ergibt die Residuenanalyse Zweifel an den Voraussetzungen, können Variablentransformationen (z.B. $\log-$Transformation bei Heteroskedastizität) weiterhelfen oder es

muss eine andere Modellformulierung (z.B. Aufnahme von weiteren Kovariablen, etwa Potenzen von Kovariablen) verwendet werden. Eventuell muss man auch auf eine Modellierung außerhalb des LM (z.B. nichtlineare Regression) ausweichen.

Ein eventuelles Vorliegen von Autokorrelation der Fehler kann mithilfe der *Durbin-Watson Statistik* geprüft werden. Weitere (induktive) Verfahren zur Überprüfung der Voraussetzungen sind z.B. Tests auf Varianzhomogenität (in der ANOVA) oder Anpassungstests oder Linearitätstests usw.

4.6.1.4 Modellvergleiche, Modellwahl und Variablenselektion

R stellt Funktionen bereit, mit deren Hilfe man sukzessive verschiedenen lineare Modelle bilden und vergleichen kann. Über den Vergleich verschiedener Modelle kann der Anwender entscheiden, welche Kombination von Einflußgrößen zur Vorhersage der Zielgröße am besten geeignet ist. Ferner können Aussagen hinsichtlich der Stärke des erklärenden Einflusses der verschiedenen Kovariablen auf die Zielgröße getroffen werden.

R-Funktion	Bedeutung
`step()`	schrittweise Regression auf einem Modell-Objekt
`add1()`	eine Kovariable dem Modell hinzufügen mit Änderung der Modellgüte
`drop1()`	eine Kovariable aus dem Modell entfernen mit Änderung der Modellgüte
`anova()`	Vergleich zweier hierarchisch geschachtelter Modelle über den partiellen F-Test

Bemerkung zum Modellvergelich mittels `anova()`:

Mittels `anova()` können hierarchisch geschachtelte (*nested*) Modelle bzgl. der Modellgüte (Prognosegüte) verglichen werden.

Beispiel:

Sei `mod1` ein LM mit den Kovariablen x_1, x_2 und `mod2` ein LM mit den Kovariablen x_1, x_2, x_3, x_4. In diesem Fall liefert `anova(mod1,mod2)` einen simultanen F-Test, ob die Kovariablen x_3 und x_4 in das Modell aufgenommen werden sollen.

Die Anwendung von `anova(mod2)` ergibt Modelltests, bei denen in der Reihenfolge der Kovariablen, wie sie innerhalb der Modellspezifikation von `mod2` festgelegt wurde, sukzessive eine Variable mehr in das Modell integriert wird.

Für den Vergleich beliebiger (nicht hierarchisch geschachtelter) Modelle verwendet man Maßzahlen der Prognosegüte, wie z.B. *AIC* (*Akaike Informationskriterium*), *BIC* (*Bayesianisches Informationskriterium*) oder *korrigiertes* R^2. In dem package `MASS`

wird die Funktion `step.AIC()` zur Verfügung gestellt, mit der man eine Vorwärts-Variablenselektion (*forward stepwise*), eine Rückwärts-Variablenelimination (*backward stepwise*) und eine Stepwise-Selektion (kombiniert: backward und forward) auf Basis des AIC durchführen kann.

Definition:

$$\begin{aligned} AIC &:= -2\log(\text{likelihood}) + 2 \cdot K, \\ BIC &:= -2\log(\text{likelihood}) + \ln(n) \cdot K, \end{aligned}$$

wobei likelihood der Wert der Likelihood-Funktion des Modells mit den bestimmten Parameterschätzern bezeichnet, n der Stichprobenumfang und K die Anzahl der im Modell geschätzten Parameter ist.

Es gilt, dass ein Modell umso besser bewertet wird, je kleiner der AIC-Wert bzw. der BIC-Wert ist. Die Anzahl der geschätzten Modellparameter K wird in den Kennzahlen bestrafend in die Bewertung eingerechnet. Für $n > e^2 \approx 7$ bestraft das BIC komplexere Modelle (d.h. Modell mit mehr Parameter) stärker als das AIC. In der Praxis zeigt sich, dass Modelle, die über das AIC entwickelt wurden, eher zu Overfitting neigen, als Modelle, die auf Basis des BIC gewählt wurden. Das BIC kann allerdings auch zu zu kleinen, d.h. zu zu einfachen, Modellen tendieren.

4.6.1.5 Vorhersagen (Prediction)

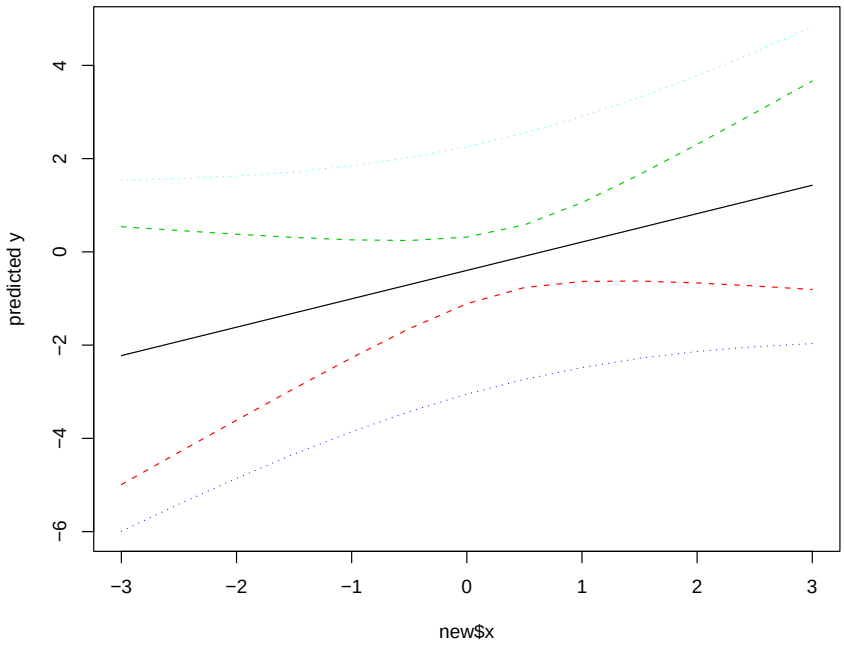
Wenn ein lineares Modell für einen Enwicklungs-Datensatz angepasst ist, können mithilfe der R-Funktion `predict()` leicht Vorhersagen (d.h. Schätzungen für die Zielgröße $E(Y)$) durchgeführt werden. Dies ist insbesonderer für andere Datensätze als den Entwicklungsdatensatz (z.B. zur Modellvalidierung auf einem Test-Datensatz) interessant.

```
predict(object, newdata, se.fit = FALSE, scale = NULL, df = Inf,
interval = c("none", "confidence", "prediction"), level = 0.95,
type = c("response", "terms"), terms = NULL, na.action = na.pass,
...)
```

Beispiel:

```
# Daten f"ur Modellbildung erzeugen

> x <- rnorm(15)
> y <- x + rnorm(15)

> predict(lm(y ~ x))
```

```
# Neue Daten f"ur Prediction

> new <- data.frame(x = seq(-3, 3, 0.5))

# Prediction Kriteriumsvariable

> predict(lm(y ~ x), new, se.fit = TRUE)

# mit Konfidenzintervalle bzw. Prognoseintervalle

> pred.w.plim <- predict(lm(y ~ x), new, interval="prediction")
> pred.w.clim <- predict(lm(y ~ x), new, interval="confidence")

> matplot(new$x,cbind(pred.w.clim, pred.w.plim[,-1]),
> + lty=c(1,2,2,3,3), type="l", ylab="predicted y")
```



4.6.1.6 Bewertung der Modellgüte bei großem Stichprobenumfang

Für den Fall, dass genügend Daten (Stichprobenumfang) für die Modellentwicklung zur Verfügung stehen, kann man einen Gesamt-Datensatz D mithilfe einer Zufallsauswahl in 2 (oder mehr) disjunkte Teildatensätze $D_1, D_2, \ldots$ zerlegen. Mit den folgenden

Verfahren testet man die Stabilität eines Modells bzw. die Tendenz des Modells auf Overfitting.

Die aufgeführten Verfahren sind unabhängig von der speziellen Methode der Modellbildung und so z.B. auch für GLM oder nicht-statistische Modellbildung einsetzbar.

a) Überprüfung der Modellstabilität durch Parallelentwicklungen

Man entwickelt an mehreren Entwicklungsdatensätzen $D_1, D_2, \ldots$ den selben Modell-Typ und vergleicht die resultierenden Modelle, d.h. z.B. die Paramaterschätzer in Regressionsmodellen. Treten dann große Abweichungen auf, zeigt dies eine starke Datenabghängigkeit bzw. Instabilität der Modelle.

b) Vergleich: Prognose mit Beobachtung

Ist ein Modell auf dem Entwicklungsdatensatz D_1 entwickelt worden, verwendet man einen disjunkten Validierungsdatensatz D_2, um die Modellgüte (auf nicht in der Modellentwicklung verwendeten Daten) zu bewerten.

Für den Fall einer metrischen oder dichotomen (binären) Responsevariable kann man für den Vergleich der Prognose-Ist-Werte auf dem Validierungsdatensatz einen sogenannten *Liftplot* verwenden.

Liftplot: Ausgangspunkt ist ein Datensatz mit m Beobachtungen, der in einer Spalte I die Ist-Werte der Responsevariable und in einer zweiten Spalte P die zu den Beobachtungen gehörigen Prognosewerte des Modells enthält.

1. Schritt: Ordne die m Beobachtungen des Datensatzes nach den Prognosewerten P.

2. Schritt: Bilde k, z.B. $k = 10$, ungefähr gleichbesetzte Klassen $K_1, \ldots, K_k$ der Beobachtungen nach der durch die Prognosewerte festgelgten Reihenfolge. D.h. in der 1. Klasse K_1 befinden sich z.B. diejenigen 10% der Beobachtungen mit den größten Prognosewerten und in der letzten 10. Klasse K_{10} befinden sich diejenigen 10% der Beobachtungen mit den kleinsten Prognosewerten.

3. Schritt: Bilde für jede der Klasse $K_1, \ldots, K_k$ die arithmetischen Mittel der Prognosewerte $\overline{P}_k$ und der Ist-Werte $\overline{I}_k$ der Responsevariable.

4. Schritt: Plotte über jeder der k Klassen die Werte $\overline{P}_k$ und $\overline{I}_k$ (und verbinde die Punktefolgen linear zu zwei Kurven).

Interpretation:

i) Der Lift von $\overline{I}_k$ der ersten Klasse(n) zu den letzten Klassen bzw. zum Mittelwert über alle Ist-Werte verdeutlicht die Trennungseigenschaft des Modells.

ii) Die Abweichungen von $\overline{P}_k$ und $\overline{I}_k$ innerhalb der selben Klassen beschreiben Unter- bzw. Überschätzungen des Modells.

4.6.1.7 Beispiele: Einfache und multiple lineare Regression mit metrischen Regressoren

Wir verwenden den R-internen Datensatz `trees`.

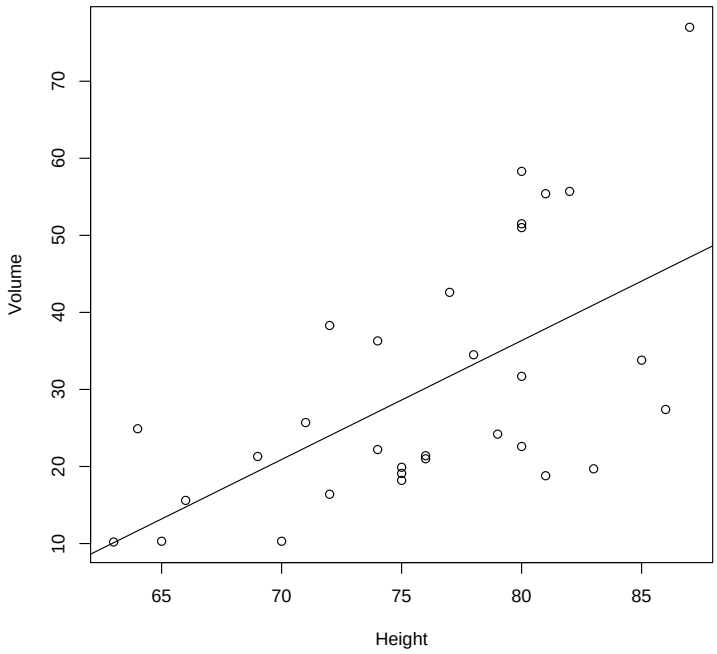
```
trees                   package:base                    R
Documentation

Girth, Height and Volume for Black Cherry Trees

Description:

     This data set provides measurements of the girth, height and
     volume of timber in 31 felled black cherry trees.  Note that girth
     is the diameter of the tree (in inches) measured at 4 ft 6 in
     above the ground.

Usage:

     data(trees)

Format:

     A data frame with 31 observations on 3 variables.

       '[,1]'  'Girth'   numeric  Tree diameter in inches
       '[,2]'  'Height'  numeric  Height in ft
       '[,3]'  'Volume'  numeric  Volume of timber in cubic ft
```

Ziel der Modellbildung ist, das Holz-Volumen eines Baums über die Variablen `Girth` und `Height` zu erklären und zu prognostizieren.

1. Modell: einfache lineare Regression, Kovariable `Height`

```
> data(trees)
> attach(trees)

> mod1 <- lm(Volume ~ Height) # einfache lineare Regression
> summary(mod1)

Call: lm(formula = Volume ~ Height)

Residuals:
```

```
    Min      1Q  Median      3Q     Max
-21.274  -9.894  -2.894  12.067  29.852

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -87.1236    29.2731  -2.976 0.005835 **
Height        1.5433    0.3839   4.021 0.000378 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 13.4 on 29 degrees of freedom

Multiple R-Squared: 0.3579, Adjusted R-squared: 0.3358
F-statistic: 16.16 on 1 and 29 DF,  p-value: 0.0003784

> plot(Height,Volume) # Scatterplot
> abline(mod1) # Regressionsgerade einzeichnen
```



Interpretation von `summary(mod1)`:

- Globaler $F-$Test: p-value: 0.0003784, das Modell ist besser als das Nur-Intercept-Modell.

- Partieller $F-$Test: p-value: 0.000378, der Koeffizient zur Variable `Height` ist hochsignifikant von 0 verschieden.

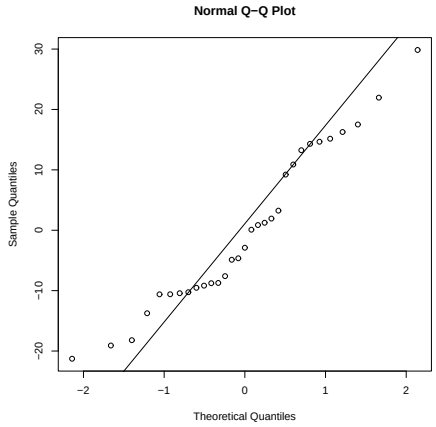
- Multiple R-Squared: 0.3579, ca. 36% der Streuung der Volumen-Daten wird alleine durch den linearen Zusammenhang und die Streuung der Höhen-Daten erklärt.

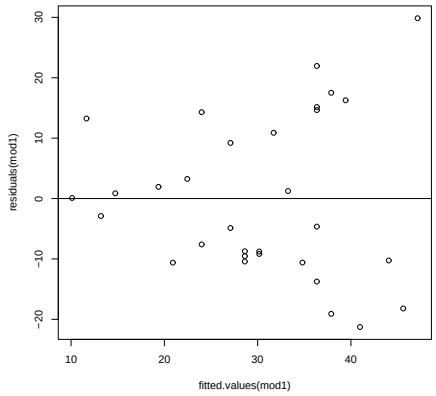
- Regressionsgerade (= Modell): $\widehat{Y} = -87.1236 + 1.5433 \cdot \text{Height}$

Überprüfung der Modellvoraussetzungen - Residualanalyse:

```
> qqnorm(residuals(mod1))
> qqline(residuals(mod1))

> plot(residuals(mod1) ~ fitted.values(mod1)) # Tukey-Anscombe-Plot
> abline(h=0) # Gerade y = 0 einzeichnen
```

Man erhält die folgenden Grafiken.





Prognosen - Prediction:

```
> predict(mod1, newdata = data.frame(Height=c(65,70,75,80,85)))
       1        2        3        4        5
13.19412 20.91087 28.62762 36.34437 44.06112
```

2. Modell: einfache lineare Regression, Kovariable `Height`, ohne Intercept

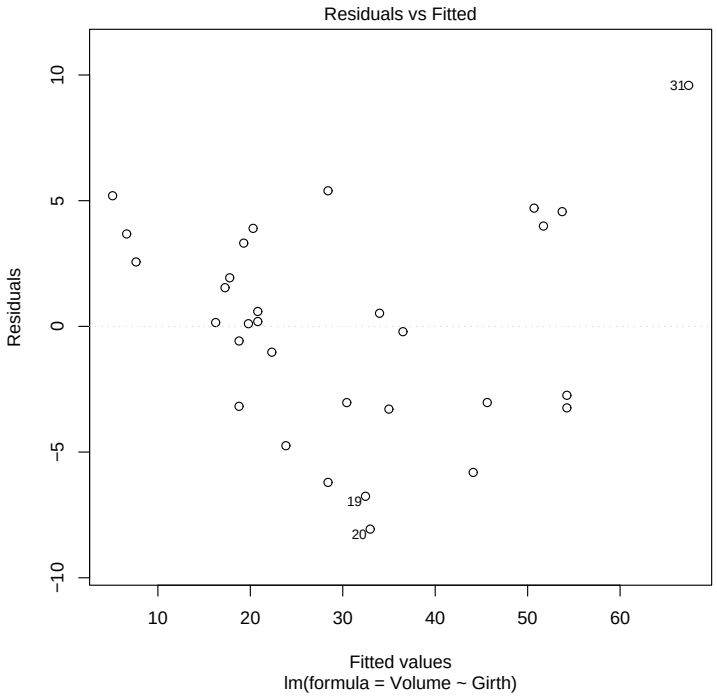
```
> mod2 <- lm(Volume ~ Height - 1)
> summary(mod2)

Call: lm(formula = Volume ~ Height - 1)

Residuals:
    Min      1Q  Median      3Q     Max
-18.031 -11.184  -7.407   7.755  41.788

Coefficients:
       Estimate Std. Error t value Pr(>|t|)
Height  0.40473    0.03545   11.42 1.91e-12 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 15.05 on 30 degrees of freedom
Multiple R-Squared: 0.8129, Adjusted R-squared: 0.8067
F-statistic: 130.4 on 1 and 30 DF,  p-value: 1.910e-12
```

3. Modell: einfache lineare Regression, Kovariable `Girth`

```
> mod3 <- lm(Volume ~ Girth)
> summary(mod3)

Call: lm(formula = Volume ~ Girth)

Residuals:
    Min      1Q  Median      3Q     Max
-8.0654 -3.1067  0.1520  3.4948  9.5868

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -36.9435    3.3651  -10.98 7.62e-12 ***
Girth         5.0659      0.2474  20.48  < 2e-16  ***
---
```

```
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4.252 on 29 degrees of freedom
Multiple R-Squared: 0.9353,    Adjusted R-squared: 0.9331
F-statistic: 419.4 on 1 and 29 DF,  p-value: < 2.2e-16
```

Man erhält für das Modell 3 den folgenden Tukey-Anscombe-Plot, der auf einen quadratischen Zusammenhang von Volumen mit Durchmesser (`Girth`) hinweist.



4. Modell: multiple lineare Regression, Kovariablen `Girth` und `Girth`^{2}, polynomiale (quadratische) lineare Regression

```
> mod4 <- lm(Volume ~ Girth + I(Girth^2))
> summary(mod4)

Call: lm(formula = Volume ~ Girth + I(Girth^2))

Residuals:
    Min      1Q  Median      3Q     Max
-5.4889 -2.4293 -0.3718  2.0764  7.6447

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
```

```
(Intercept) 10.78627   11.22282   0.961 0.344728
Girth       -2.09214    1.64734  -1.270 0.214534
I(Girth^2)   0.25454    0.05817   4.376 0.000152 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.335 on 28 degrees of freedom
Multiple R-Squared: 0.9616,     Adjusted R-squared: 0.9588
F-statistic: 350.5 on 2 and 28 DF,  p-value: < 2.2e-16
```

5. Modell: multiple lineare Regression, Kovariablen `Girth`, $\texttt{Girth}^2$ und `Height`.

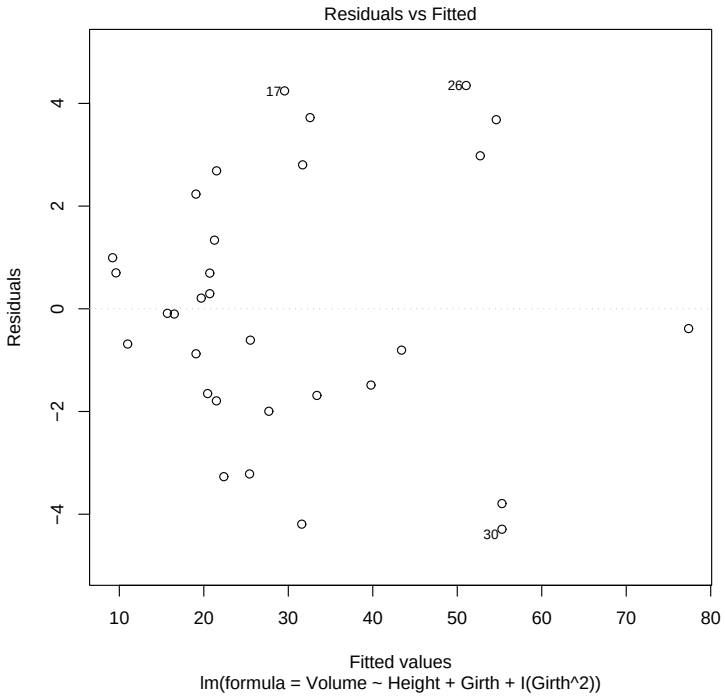
```
> mod5 <- lm(Volume ~ Height + Girth+ I(Girth^2))
> summary(mod5)

Call: lm(formula = Volume ~ Height + Girth + I(Girth^2))

Residuals:
    Min      1Q  Median      3Q     Max
-4.2928 -1.6693 -0.1018  1.7851  4.3489

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -9.92041   10.07911  -0.984 0.333729
Height       0.37639    0.08823   4.266  0.000218 ***
Girth       -2.88508   1.30985   -2.203 0.036343 *
I(Girth^2)  0.26862    0.04590   5.852  3.13e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.625 on 27 degrees of freedom
Multiple R-Squared: 0.9771,     Adjusted R-squared: 0.9745
F-statistic: 383.2 on 3 and 27 DF,  p-value: < 2.2e-16
```

Die Regressionsfunktion (= Modell) lautet also:

$$\widehat{Y} = -9.92041 + 0.37639 \cdot \texttt{Height} - 2.88508 \cdot \texttt{Girth} + 0.26862 \cdot \texttt{Girth}^2$$

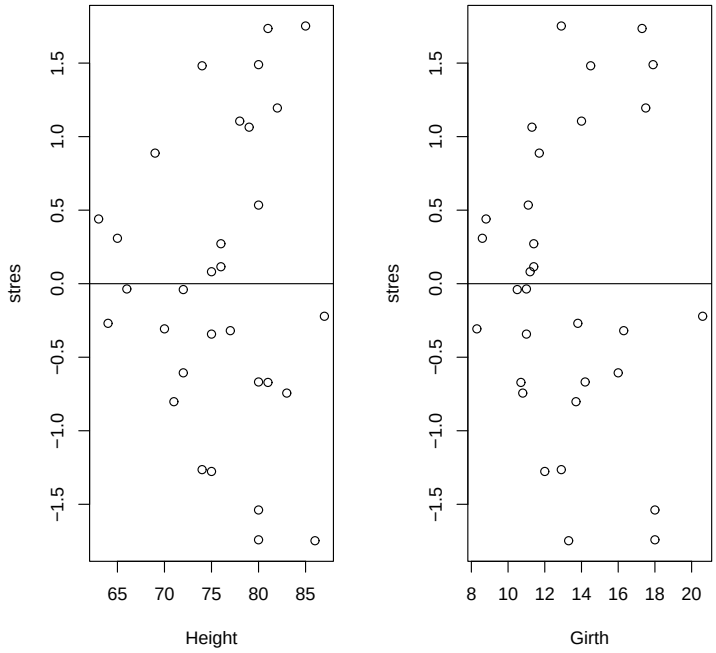
Alle drei Regressoren sind signifikant auf dem 5%-Level. Das Modell 5 erklärt ca. 98% der Streuung der Volumendaten.

Es ergibt sich ein sehr akzeptabler Tukey-Anscombe-Plot



Im Folgendem werden noch die Plots der **standardisierten** Residuen des 5. Modells gegen die Kovariablen `Height` und `Girth` geplottet. Dazu werden zunächst die standardisierten Residuen aus Variablen der Objekte `summary(mod5)` und `lm.influence(mod5)` berechnet.

```
> # Predictions und Residuen aus dem lm-Objekt mod5 auslesen
> pred <- fitted.values(mod5)
> res <- residuals(mod5)
> # Standardisierte Residuen bilden
> si <- summary(mod5)$sigma
> ha <- lm.influence(mod5)$hat
> stres <- res/(si*sqrt(1-ha))
> # Grafiken erzeugen
> par(mfrow=c(1,2))
> plot(Height,stres)
> abline(h=0)
> plot(Girth,stres)
> abline(h=0)
```



Man folgert aus den beiden Plots, dass für das Modell 5 die Modell-Voraussetzungen sehr gut erfüllt sind.

Konfidenzintervalle und Prognoseintervalle:

Zur Einschätzung der Genauigkeit der geschätzten Koeffizienten bzw. der mit dem Modell erstellten Prognosen für Y oder $E(Y)$ verwendet man Prognoseintervalle bzw. Konfidenzintervalle. Diese können mithilfe der Prognosewerte (`fit`), Standardfehler (`fit.se`) (enthalten in dem R-lm-Objekt `predict`) und den passenden Quantilen vom Anwender selbst berechnet werden (vgl. Pruscha (2006, S. 114 - 118).

Alternativ können Konfidenzintervalle und Prognoseintervalle für die Kriteriumsvariable bzw. den Erwartungswert der Kriteriumsvariable bei gegebenen Kovariablenwerten auch direkt über die Methode `predict()` ausgegeben werden.

Konfidenzintervalle für die Parameterschätzer eines linearen Modells können mithilfe der R-Funktion `confint()` berechnet werden. Die Methode `confint()` erwartet als Argument ein `lm` (oder `glm`) Objekt.

```
> data(trees)
> attach(trees)

> mod <- lm(Volume ~ Height + Girth + I(Girth^2))
```

```
> confint(mod, level=0.95)
                  2.5 %         97.5 %
(Intercept)      -30.6010395   10.7602276
Height            0.1953502     0.5574244
Girth            -5.5726707    -0.1974868
I(Girth^2)        0.1744336     0.3628112
```

4.6.1.8 Beispiel: Multiple lineare Regression mit kategorialen Kovariablen (Faktoren)

Der Fall, dass nur ein Faktor als Einflussgröße betrachtet wird entspricht der einfachen ANOVA. Bei mehreren Faktoren liegt eine mehrfache Varianzanalyse vor. Bei einem Mix aus metrischen Regressoren und Faktoren spricht man von einer Kovarianzanalyse.

Bei den Modellen der mehrfachen Varianzanalyse und der Kovarianzanalyse ist besonders auf die korrekte, statistische Behandlung von Interaktionen (Wechselwirkungen) der Kovariablen zu achten. Ausserdem müssen bei diesen Modell-Typen spezielle Datendesigns (Versuchspläne) vorliegen bzw. unterschieden werden.

Im folgenden Beispiel werden die Stufen eines Faktors als dichotome Kovariablen innerhalb eines multiplen, linearen Regressionsmodell behandelt. Die betrachtete kategoriale Kovariable mit `factor`-Datentyp wird dazu (automatisiert) dummy-codiert.

Wir verwenden der R-internen Datensatz `chickwts`.

```
> data(chickwts)
> attach(chickwts)

> summary(chickwts)
     weight             feed
 Min.   :108.0   casein   :12
 1st Qu.:204.5   horsebean:10
 Median :258.0   linseed  :12
 Mean   :261.3   meatmeal :11
 3rd Qu.:323.5   soybean  :14
 Max.   :423.0   sunflower:12

> boxplot(weight~feed)

> mod1 <- lm(weight ~ feed)
> summary(mod1)

Call: lm(formula = weight ~ feed)
```

```
Residuals:
     Min       1Q   Median       3Q      Max
-123.909  -34.413    1.571   38.170  103.091

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)    323.583     15.834  20.436  < 2e-16 ***
feedhorsebean -163.383     23.485  -6.957 2.07e-09 ***
feedlinseed   -104.833     22.393  -4.682 1.49e-05 ***
feedmeatmeal  -46.674      22.896  -2.039 0.045567 *
feedsoybean   -77.155      21.578  -3.576 0.000665 ***
feedsunflower 5.333        22.393   0.238 0.812495
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 54.85 on 65 degrees of freedom
Multiple R-Squared: 0.5417,     Adjusted R-squared: 0.5064
F-statistic: 15.36 on 5 and 65 DF,  p-value: 5.936e-10
```

Man beachte, dass die Stufe `feed casein` die Referenz-Kategorie darstellt und deren Effekt im Intercept enthalten ist. Die Variablen feedhorsebean ($:= D_1$), feedlinseed ($:= D_2$), feedmeatmeal ($:= D_3$), feedsoybean ($:= D_4$) und feedsunflower ($:= D_5$) sind dichotome Dummyvariablen, die beim Vorliegen der entsprechenden Stufe den Wert 1 besitzen und sonst identisch 0 sind.

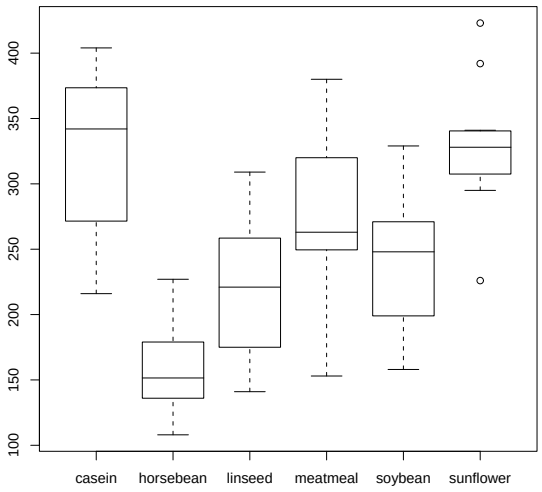
Die resultierende Regressionsfunktion ist hier mit $Y \equiv$ weight

$$\widehat{Y} = 323.583 - 163.383 \cdot D_1 - 104.833 \cdot D_2 - 46.674 \cdot D_3 - 77.155 \cdot D_4 + 5.333 \cdot D_5$$

Varianzanalytische Betrachtung:

```
> anova(mod1)
Analysis of Variance Table

Response: weight
          Df Sum Sq Mean Sq F value    Pr(>F)
feed       5 231129   46226  15.365 5.936e-10 ***
Residuals 65 195556    3009
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Beachte: Für eine Verwendung der Ergebnisse müsste man noch eine Überprüfung der Modellvoraussetungen durchführen.



4.6.1.9 Beispiel: Einfache Varianzanalyse

a) Einfache ANOVA mit Normalverteilungsannahme

- Voraussetzung: Die zugrundeliegenden Zuvallsvariablen Y_{ij}, $i = 1, \ldots, k$ (Stufen), $j = 1, \ldots, n_i$ (Umfang in Gruppe i), sind unabhängig und $Y_{ij} \sim N(\mu_i, \sigma^2)$. Man beachte, dass die Varianzen für alle Gruppen als identisch vorausgesetzt werden.

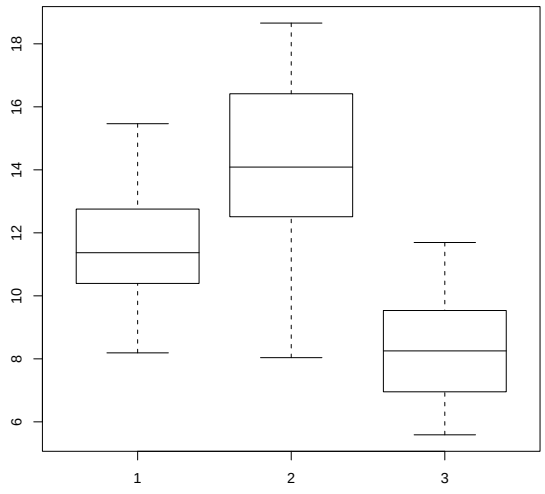
- Globaler $F-$Test der ANOVA

Hypothesen:
$H_0 : \mu_1 = \mu_2 = \ldots = \mu_k$
vs.
$H_1 :$ mindestens 2 der μ_i unterscheiden sich

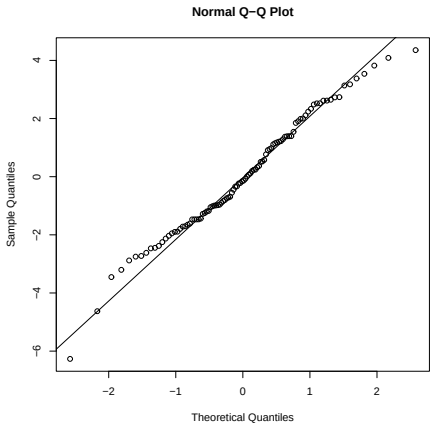
Anwendung: H_1 besagt, dass der Faktor insgesamt (global) einen Einfluss auf Y hat.

- Beispiel:

```
# Daten erzeugen
> A <- rnorm(30,mean=12,sd=2)
> B <- rnorm(38,mean=15,sd=2.2)
> C <- rnorm(32,mean=8,sd=1.98)

> boxplot(A,B,C)
```

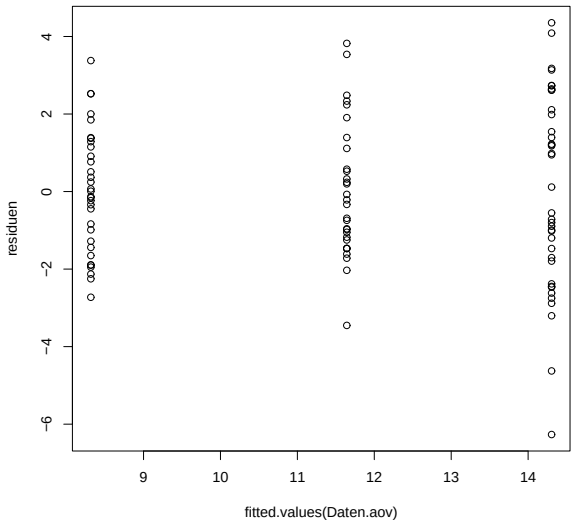



```
# Dataframe fuer ANOVA erzeugen
> Y <- c(A,B,C)
> Daten <- data.frame(Faktor=c(rep("A",30),rep("B",38),rep("C",32)),Y)

> summary(Daten)
 Faktor     Y
 A:30       Min.   : 5.588
 B:38       1st Qu.: 9.406
 C:32       Median :11.426
            Mean   :11.590
            3rd Qu.:13.631
            Max.   :18.659

# ANOVA
> Daten.aov <- aov(Y ~ Faktor, data=Daten)
> summary(Daten.aov)
            Df Sum Sq Mean Sq F value    Pr(>F)
Faktor       2 623.57  311.79   75.81    < 2.2e-16 ***

Residuals   97 398.94 4.11
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

# Ueberpruefung der Voraussetzungen
> residuen <- residuals(Daten.aov)
> qqnorm(residuen)
> qqline(residuen)
```



Bemerkung: Ein weiteres Verfahren zur Überprüfung der Voraussetzungen vor allem in Hinblick auf die Varianzhomogenität ist ein Plot der Residuen gegen die geschätzten Gruppenmittelwerte.

```
> plot(residuen ~ fitted.values(Daten.aov)}
```



Als Signifikanztests zur Überprüfung der Varianzhomogenität kann man den parametrischen *Bartlett-Test* (R: `bartlett.test()`) oder den, bzgl. Abweichungen von der Normalverteilungsannahme robusten, `Fligner-Killeen-Test` (R: `fligner.test()`) verwenden.

```
> bartlett.test(Daten$Y,Daten$Faktor)
```

```
Bartlett test for homogeneity of variances

data:  Daten$Y and Daten$Faktor

Bartlett's K-squared = 8.7192, df = 2, p-value = 0.01278
```

b) Kruskall-Wallis-Test, Einfache ANOVA ohne Normalverteilungsannahme

- Voraussetzung: wie in a), aber anstelle der Normalverteilung wird nur vorausgesetzt, dass die Messgröße in den Gruppen jeweils stetig mit Dichten $f_1, \ldots, f_k$ verteilt ist.

- Hypothesen:
$H_0 : f_1 = f_2 = \ldots = f_k$
vs.
H_1 : mindestens 2 der f_i unterscheiden sich

- Beispiel:

```
> kruskal.test(Y ~ Faktor, data=Daten)

        Kruskal-Wallis rank sum test

data:  Y by Faktor

Kruskal-Wallis chi-squared = 63.757, df = 2,p-value = 1.43e-14
```

c) Paarweise Vergleiche

Falls der globale $F-$Test der ANOVA signifikant auf einen Unterschied der Gruppen-Erwartungswerte testet (d.h. man nimmt H_1 an, also kleiner $p-$Wert), möchte man i.A. wissen, welche Gruppen-Erwartungswerte sich unterscheiden.
Dazu kann man den entsprechend angepassten 2-Stichproben-$t-$Test und Wilcoxon-Test verwenden. Die Anpassung betrifft die Adjustierung der $p-$Werte, da es sich hier um ein multiples Testproblem handelt.

- Beispiel: vgl. oben!

```
> A <- rnorm(30,mean=12,sd=2)
> B <- rnorm(38,mean=15,sd=2.2)
> C <- rnorm(32,mean=8,sd=1.98)
> Y <- c(A,B,C)
> Daten <- data.frame(Faktor=c(rep("A",30),rep("B",38),rep("C",32)),Y)
```

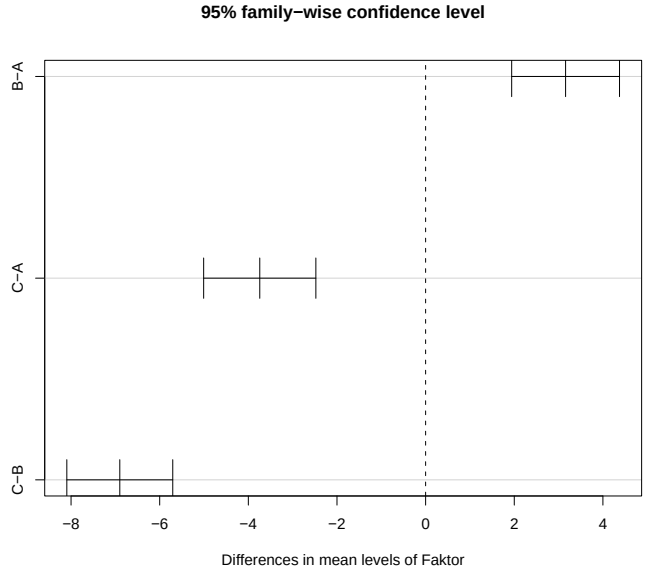
```
# Falsch: ohne p-Wert Adjustierung
> pairwise.t.test(Daten$Y,Daten$Faktor,p.adjust.method="none")

        Pairwise comparisons using t tests with pooled SD

data:  Daten$Y and Daten$Faktor

  A       B
B 2.5e-08 -
C 9.0e-12 < 2e-16

P value adjustment method: none

# Richtig: mit p-Wert Adjustierung
> pairwise.t.test(Daten$Y,Daten$Faktor, p.adjust.method="bonferroni")

        Pairwise comparisons using t tests with pooled SD

data:  Daten$Y and Daten$Faktor

  A       B
B 6.5e-06 -
C 1.0e-09 < 2e-16

P value adjustment method: bonferroni

# Nichtparametrische Variante
> pairwise.wilcox.test(Daten$Y,Daten$Faktor, p.adjust.method="bonferroni")

        Pairwise comparisons using Wilcoxon rank sum test

data:  Daten$Y and Daten$Faktor

  A       B
B 7.2e-06 -
C 4.7e-09 < 2e-16

P value adjustment method: bonferroni
```

Die *Bonferroni-Methode* zur $p-$Wert-Adjustierung (hier werden einfach die $p-$Werte mit der Anzahl der Vergleiche multipliziert) ist eine Standard-Methode, die bzgl. der Test-Güte oft nicht optimal ist. In R können auch andere, weiter-entwickelte Adjustierungen gewählt werden.
Im Fall einer einfachen ANOVA sollte man die *Tukey-Methode* verwenden, die aller-

dings keine Testentscheidung, sondern *simultane Konfidenzintervalle* für die Erwartungswertdifferenzen liefert.

- Beispiel: Daten von oben.

```
> Daten.aov <- aov(Y ~ Faktor, data=Daten)
> TukeyHSD(Daten.aov, conf.level=0.95)
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = Y ~ Faktor, data = Daten)

$Faktor
         diff       lwr       upr
B-A  2.923410  1.542329  4.304491
C-A -4.221009 -5.658076 -2.783941
C-B -7.144419 -8.501167 -5.787670

> plot(TukeyHSD(Daten.aov, conf.level=0.95))
```



Keines der 3 Konfidenzintervalle überdeckt die 0, daher folgert man mit einer Sicherheitswahrscheinlichkeit von 95%, dass alle 3 Gruppen-Erwartungswerte verschieden sind.

Varianzhomogenität: Die Voraussetzung der Varianzhomogenität in allen betrachteten Gruppen überprüft man in der ANOVA mittels:

- *Levene-Test*
- *Bartlett-Test* (R-Funktion `bartlett.test()`)
- Diagnostischer s-m Plot (vgl. Pruscha (2006, S. 69 - 70)).

Erweiterungen: Ausgehend von dem Grundmodell der einfachen Varianzanalyse werden folgende Modell-Erweiterungen verwendet:

- $m-$fache ANOVA ($m > 2$ Faktoren):
Im Modell der mehfachen Varianzanalyse müssen im Vergleich zur einfachen Varianzanalse die Wechselwirkungen der Faktoren geeignet berücksichtigt werden. Bei der Modell-Interpretation einer mehrfachen ANOVA betrachtet man zuerst den Wechselwirkungseffekt. Die Haupteffekte werden nur dann einzeln interpretiert, wenn keine signifikanter Wechselwirkungseffekt vorliegt.

Beispiel: 2-fache ANOVA

Daten:

```
> summary(daten)
       Y          Faktor1 Faktor2
 Min.   : 0.000   1: 70   1:101
 1st Qu.: 4.000   2:122   2: 91
 Median : 7.000
 Mean   : 7.031
 3rd Qu.: 9.000
 Max.   :19.000
```

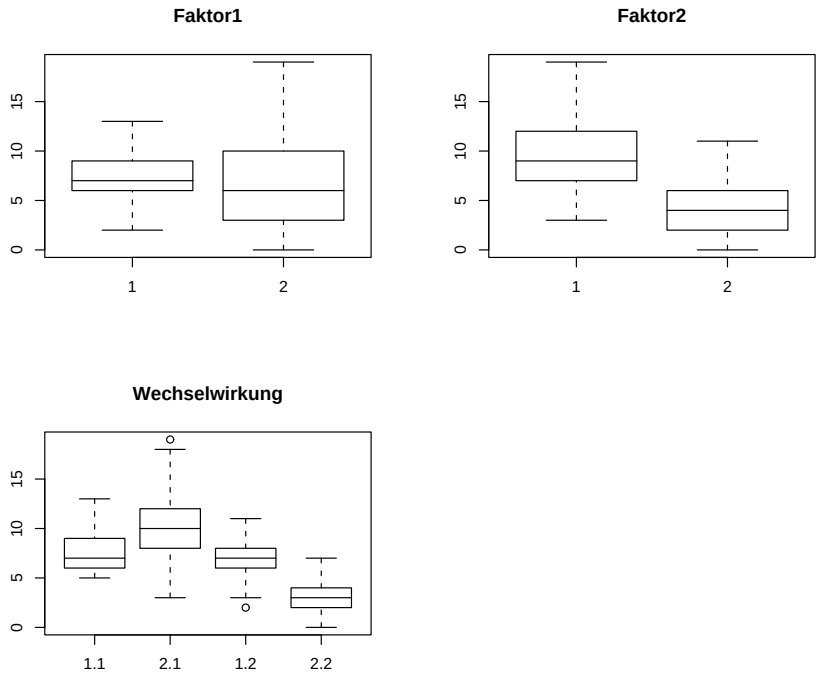
Boxplots:

```
> boxplot(Y ~ Faktor1,main="Faktor1")
> boxplot(Y ~ Faktor2,main="Faktor2")
> boxplot(Y ~ (Faktor1 + Faktor2),main="Wechselwirkung")
```

Interaktionsplot (Mittelwertplot):

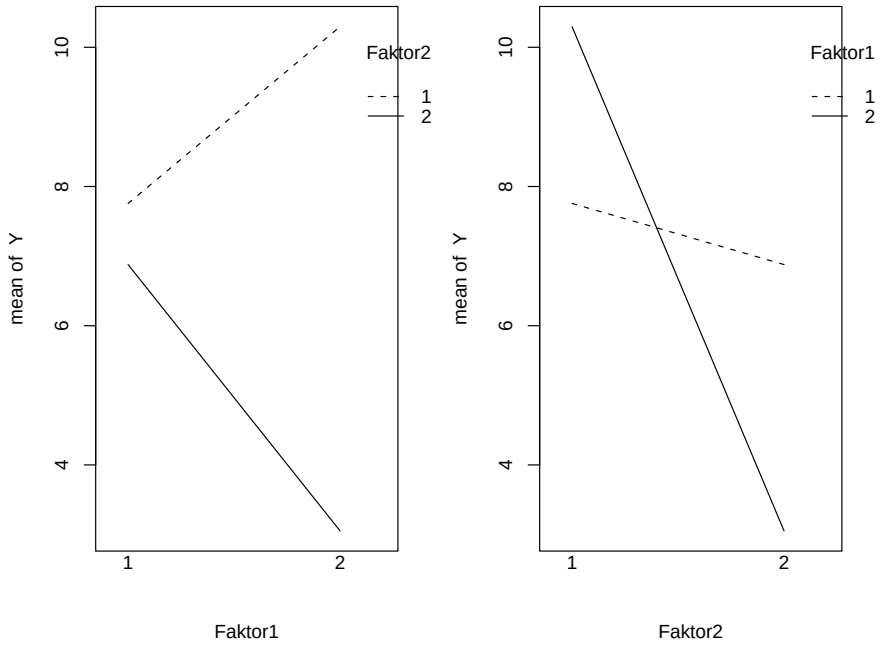
```
> par(mfrow=c(1,2))
> interaction.plot(Faktor1,Faktor2,Y)
> interaction.plot(Faktor2,Faktor1,Y)
```

2-fache ANOVA:




```
> anova(lm(Y ~ Faktor1 * Faktor2))
Analysis of Variance Table

Response: Y
                 Df  Sum Sq Mean Sq  F value    Pr(>F)
Faktor1           1   10.70   10.70   1.6004    0.2074
Faktor2           1 1161.11 1161.11 173.7235 < 2.2e-16 ***
Faktor1:Faktor2   1  449.47  449.47  67.2497 3.663e-14 ***
Residuals       188 1256.53    6.68
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Beachte: Vor der Interpretation der Ergebnisse müssten zunächst die Voraussetzungen der ANOVA überprüft werden.



- ANOVA für Messwiederholungen (Blockvarianzanalyse):
Hier betrachtet man typischerweise (zeitliche) Messwiederholungen (*repeated measurements*) an identischen Individuen (vgl. $t-$Test für vebundene Stichproben). Mit dieser Methode kann man störende Streuungen zwischen den Bebachtungen reduzieren und erhält eine größere Power zur Lokalisierung von Erwartungswertunterschieden zwischen den Gruppen. Als nichtparametrische Alternative (also ohne Voraussetzung von Normalverteilung) zum Vergleich mehrerer verbundener Stichproben kann man den `Friedman-Test` (R-Funktion **friedman.test()**) verwenden.
Mehr zur Blockvarianzanalyse und dem Friedman-Test findet man z.B. bei Sachs und Hedderich (2006, S. 454 - 465).

4.6.1.10 Beispiel: Kovarianzanalyse

- Beispiel: vgl. oben!

```
> A <- rnorm(30,mean=12,sd=2)
> B <- rnorm(38,mean=15,sd=2.2)
> C <- rnorm(32,mean=8,sd=1.98)
> Y <- c(A,B,C)

> x <- runif(100) # Ein Regressor x
```

```
> Y1 <- Y + 3*x # Neue Kriteriumsvariable

> Daten <- data.frame(Faktor=c(rep("A",30),rep("B",38),rep("C",32)),Y1,x)

> mod <- lm(Y1 ~ x + Daten$Faktor + x*Daten$Faktor)
> summary(mod)

Call: lm(formula = Y1 ~ x + Daten$Faktor + x * Daten$Faktor)

Residuals:
    Min      1Q  Median      3Q     Max
-5.4477 -1.0618 -0.1099  1.1056  5.9327

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)      11.7064     0.7998  14.636  < 2e-16 ***
x                 3.0793     1.4113   2.182 0.031608 *
Daten$FaktorB     2.6279     1.0266   2.560 0.012070 *
Daten$FaktorC    -4.2431     1.0450  -4.061 0.000101 ***
x:Daten$FaktorB   1.0155     1.7583   0.578 0.564951
x:Daten$FaktorC   1.1979     1.9355   0.619 0.537451
```

4.6.2 Generalisiertes Lineares Modell (GLM)

Im Vergleich zu linearen Modellen (LM) betrachtet man bei GLM nicht-lineare Strukturgleichungen und erweitert die möglichen Verteilungen der Responsevariablen Y_i im Hinblick auf die Normalverteilung im linearen Modell.

Man setzt im GLM voraus, dass die Verteilung der Responsevariablen eine Dichte aus einer Exponentialfamilie besitzt. So können z.B. Binomialverteilte, Gammaverteilte oder Poisson-verteilte Kriteriumsvariablen in einem GLM modelliert werden.

Die Strukturgleichung im GLM wird als

$$E(Y_i) \equiv \mu_i = h(x_i \cdot \beta),\ i = 1, \ldots, n,$$

formuliert, wobei h eine Modelltyp-spezielle (zu wählende) *Response-Funktion* (Umkehrfunktion der sogenannten *Link-Funktion* g) ist, x_i der Vektor der Kovariablen der $i-$ten Beobachtung ist und $\beta \in \mathbb{R}^p$ den zu schätzenden Parametervektor bezeichnet.

Mit der Linkfunktion g schreibt sich die Strukturgleichung als

$$g(\mu_i) = x_i \cdot \beta \equiv \eta_i,$$

d.h. der Erwartungswert ist erst nach einer Transformation eine lineare Funktion der Modellparameter β. Die Größe η_i nennt man den linearen Prädiktor.

In der Anwendung wählt man oft zu einer (aufgrund der Sachlage angenommenen) Verteilungsannahme eine spezielle Linkfunktion, die sogenannte *natürliche Linkfunktion.*

Die Parameterschätzer $\widehat{\beta}_1, \ldots, \widehat{\beta}_p$ können im Unterschied zum LM nicht mehr explizit, sondern nur durch numerische Iterationsalgorithmen (z.B. *Fisher-Scoring*) als Näherungen aus den Schätzgleichungen berechnet werden. Die Schätzgleichungen (ML-Gleichungen) werden im GLM über die Maximumlikelihood-Methode gewonnen.

4.6.2.1 Modellspezifikation

In Analogie zur Methode `lm` im linearen Modell verwendet man im GLM die Methode `glm`.

- Festlegung der Strukturgleichung analog zu `lm` über `formula`

- Spezifizierung der Verteilung in `glm` mittels dem Parameter `family`, z.B. `family = binomial`

- Wahl der Linkfunktion über `family = binomial(link = )`
Ohne expliziten Setzen von `link` wird die natürliche Linkfunktion verwendet.

```
Family Objects for Models

Description:

     Family objects provide a convenient way to specify the details of
     the models used by functions such as 'glm'.  See the documentation
     for 'glm' for the details on how such model fitting takes place.

Usage:

     family(object, ...)

     binomial(link = "logit")
     gaussian(link ="identity")
     Gamma(link = "inverse")
     inverse.gaussian(link = "1/mu^2")
     poisson(link = "log")
     quasi(link = "identity", variance = "constant")
     quasibinomial(link = "logit")
     quasipoisson(link = "log")
```

```
link: a specification for the model link function. The 'gaussian'
          family accepts the links '"identity"', '"log"' and
          '"inverse"'; the 'binomial' family the links '"logit"',
          '"probit"', '"log"' and '"cloglog"' (complementary log-log);
          the 'Gamma' family the links '"inverse"', '"identity"' and
          '"log"'; the 'poisson' family the links '"log"',
          '"identity"',  and '"sqrt"' and the 'inverse.gaussian' family
          the links '"1/mu^2"', '"inverse"', '"inverse"' and '"log"'.
```

Analogien zum LM:

Prinzipiell gilt, dass die meisten im LM bekannten Methoden ganz analog auch im GLM angewandt werden können. Insbesondere ist das Ergebnis von `glm` ein Modell-Objekt, auf das wieder weitere Methoden (z.B. `summary`) angewandt werden können. Ausserdem kann man auf die Ergebnisse in einem `glm`-Modell-Objekt wieder wie üblich zugreifen.

Die Vorgehensweisen zur Modellwahl entsprechen denen im LM. Man vergleiche z.B. die Methoden `drop1()`, `add1()` und `step()`. Eine weiter Methode zur automatisierten Variablenauswahl ist z.B. `stepAIC` aus dem Paket `MASS`.

Die Bewertung der Modelgüte kann über diagnostische Plots (`plot(mod)`, mit `mod` Modell-Objekt) oder über Kreuzvalidierung (z.B. Lift-Plots) erfolgen.

Unterschiede:

In GLM werden sogenannte `Devianz-Residuen` betrachtet und anstelle der Varianzanalyse des LM führt man im GLM eine Devianz-Analyse durch (wieder über die Methode `anova`). Die *Devianz* im GLM entspricht der Residuenquadratsumme im linearen Regressionsmodell. Zum Devianz-Begriff vgl. man z.B. Sachs und Hedderich (2006, S. 590 ff.).

Als (partielle) Parameter-Tests werden `Wald-Tests` (vgl. `summary`) bzw. über

```
anova(...,test = "Chisq")
```

`Likelihood-Quotienten-Tests` durchgeführt. Zusätzlich können auch noch sogenannte `Score-Tests` durchgeführt werden.

Die Devianz-Analyse mit `anova` (LQ-Tests) kann zum Modellvergleich von sogenannten *nested* (*hierarchischen*) Modellen verwendet werden. Dabei nennt man ein Modell $M1$ nested im Modell $M2$, falls die Modellparameter von $M1$ eine Teilmenge der Modellparameter von $M2$ sind. In den Tests wird als Teststatistik die Devianz-Differenz der Devianzen von $M1$ und $M2$ verwendet.

Man beachte, dass alle Tests im GLM (nur) asymptotische Tests sind.

4.6.2.2 Parameterschätzung, Konfidenzintervalle und Parameter-Tests

Durch `glm` erfolgen die Berechnungen und werden in dem Modellobjekt `mod1`

```
mod1 <- glm(...)
```

abgelegt. Durch Anwendung von `summary(mod1)` erhält man Detailinformationen.

`confint(mod1)` liefert Konfidenzintervalle für die Parameterschätzer.

Prognosen:

Prognosen können wieder analog zum LM über die Methode `predict` erstellt werden. Man beachte, dass über den Parameter

```
type = "response"
```

eine Prognose auf der Ebene der modellierten Kriteriumsvariablen erstellt wird, während über

```
type = "link"
```

eine Prognose auf der Ebene des linearen Prädiktors erstellt wird.

Konfidenz- und Prognoseintervalle für die Prognosen können mit den Methoden der base-packages nicht analog zum linearen Modell erstellt werden. Die entsprechenden Konfindenz- bzw. Prognoseintervalle können aber vom Anwender selbst programmiert werden.

4.6.2.3 Beispiele

a) Binäre logistische Regression

Situation: Man betrachtet eine dichotome (binäre) Responsevariable

$$Y \in \{0, 1\} \equiv \{\text{Erfolg, Mißerfolg}\} \equiv \{\text{Treffer, Niete}\}$$

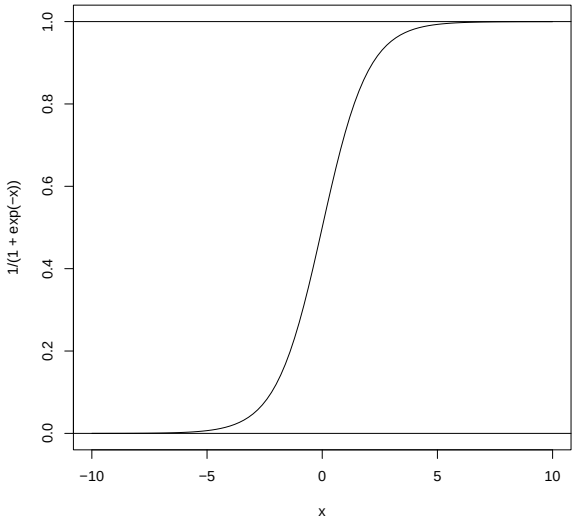
und p Einflussgrößen (Regressoren oder verallgemeinert Regressoren und Faktoren). Die Responsevariable ist also Bernoulli-verteilt.
Mit der logistischen Link-Funktion erhält man die Strukturgleichung

$$E(Y_i \,|\, x_i) = P(Y_i = 1 \,|\, x_i) = \frac{1}{1 + \exp\left(-x_i^T \cdot \beta\right)}.$$

Der Parametervektor $\beta = (\alpha, \beta_1, \ldots, \beta_p)^T$ enthält im Allgemeinen einen Intercept α und Parmeter β_i, die die Wirkung der Kovariablen bestimmen.

Die Kriteriumsvariablen $Y_i, \ldots, Y_n$ sind als unabhängige Zufallsvariablen vorausgesetzt.

In der folgenden Grafik ist die Responsefunktion $h(x) = \frac{1}{1+\exp(-x)}$ dargestellt, die reellwertige x auf die Wertemenge $]0, 1[$ abbildet.



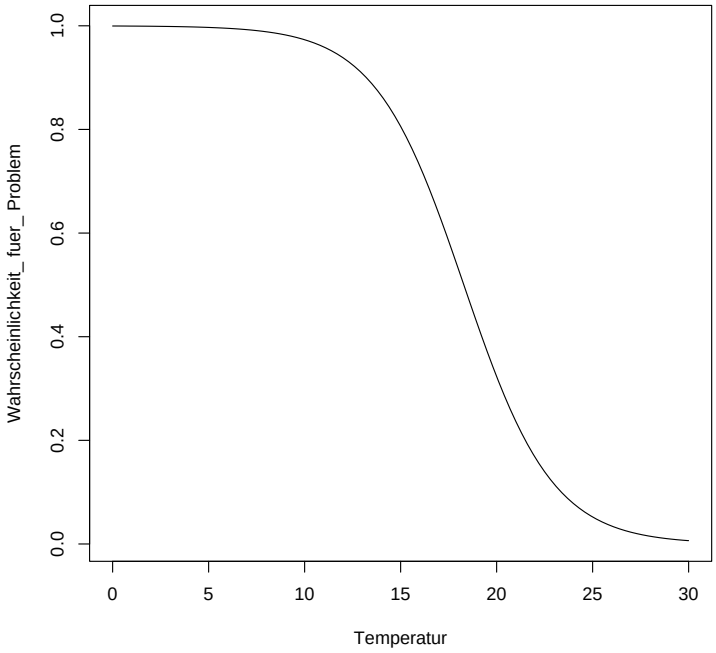
Also: Man erhält über das Modell eine Gleichung (mit den Schätzern $\widehat{\beta}$), die die Wahrscheinlichkeit von $Y = 1$ in Abhängigkeit von den Kovariablenwerten x_i ausdrückt. Aufgrund der Form der Response-Funkion gilt, dass

$$\widehat{P}(Y_i = 1 \,|\, x_i) = \frac{1}{1 + \exp\left(-x_i^T \cdot \widehat{\beta}\right)} \in]0, 1[$$

Man beachte, dass eine lineare Regression (vgl. oben!) für diesen Zweck nicht geeignet ist, da hier für Wahrscheinlichkeiten Prognosen außerhalb des sinnvollen Intervalls $[0, 1]$ auftreten könnten.

Daten-Beispiel: Challenger-Katastrophe am 28.01.1986.
Daten: Temperatur-Daten in Grad C° und Kennung, ob beim Start Probleme mit den Dichtungsringen (1 = Ja, 0 = Nein) auftraten für 23 Starts der Raumfähren.

```
# Daten eingeben
> Temp <- c(12,14,14.5,17,21,21,24,
+ 19,19.5,19.5,19.5,20,20.5,21,22,22,23,23,24.5,24.5,25.5,26,27)
> Problem <- c(1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
> Daten <- data.frame(Temp,Problem)

# Modellbildung
> mod <- glm(Problem ~ Temp, family=binomial)
> summary(mod)

Call: glm(formula = Problem ~ Temp, family = binomial)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.0505  -0.7709  -0.3632   0.4477   2.2586

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)   7.9144     4.0281   1.965   0.0494 *

 Temp         -0.4327    0.1994  -2.169   0.0300 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 28.267  on 22  degrees of freedom

Residual deviance: 20.102  on 21  degrees of freedom

AIC: 24.102

Number of Fisher Scoring iterations: 5

# Regressionskurve zeichnen
> a <- mod$coefficients[1]
> b <- mod$coefficients[2]
> a
(Intercept)
   7.914364
> b
     Temp
-0.432652

> curve(1/(1+exp(-1*(a+b*x))),from=0, to=30,
+ xlab="Temperatur", ylab="Wahrscheinlichkeit_ fuer_ Problem")
```



Das Modell liefert als Schätzfunktion für die Wahrscheinlichkeit eines Problems mit den Dichtungsringen beim Start in Abhängigkeit von der Temperatur T die Funktion

$$\widehat{P}\,(Y = 1 \,|\, T) = \frac{1}{1 + \exp(-7.914364 + 0.432652 \cdot T)}.$$

Modellgüte-Bewertung mithilfe der *confusion matrix*

Bei Modellen mit dichotomer Responsevariablen Y verwendet man zur Einschätzung der Modellgüte gerne eine sogenannte *confusion matrix*, in der die Prognosen $\widehat{Y}$ und die Ist-Werte für Y verglichen werden.
Dazu müssen in einem ersten Schritt die Prognosewerte für die Erfolgswahrscheinlichkeit $\widehat{P}\,(Y = 1 \,|\, x)$ zu dichotomen Werten $\widehat{Y} \in \{0, 1\}$ erweitert werden.
Eine Möglichkeit ist z.B. allen Werten $\widehat{P}\,(Y = 1 \,|\, x) \geq \frac{1}{2}$ die Prognose $\widehat{Y} = 1$ und allen Werten $\widehat{P}\,(Y = 1 \,|\, x) < \frac{1}{2}$ die Prognose $\widehat{Y} = 0$ zuzuordnen. Eine andere Methode wäre eine analoge Zuordnung von $\widehat{Y} \in \{0, 1\}$ über den Median der Prognosen $\widehat{P}\,(Y = 1 \,|\, x)$.

Daten-Beispiel: Confusion matrix, Challenger-Katastrophe (vgl. oben!)

```
> Temp <- c(12,14,14.5,17,21,21,24,
+  19,19.5,19.5,19.5,20,20.5,21,22,22,23,23,24.5,24.5,25.5,26,27)
> Problem <- c(1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
```

```
> mod <- glm(Problem ~ Temp, family=binomial)

# confusion matrix
> table(Problem, round(predict(mod, type="response")))
Problem

        0   1
      0 16  0
      1  3  4
```

Die confusion matrix zeigt, dass 3 Prognosen falsch negativ sind und keine Prognose falsch positiv ist. Der Idealfall (der allerdings in der Realität kaum vorkommt) wäre, dass die confusion matrix nur in der Hauptdiagonalen positive Einträge besitzt.
Im Zusammenhang mit der confusion matrix sei auf das prinizipelle Problem von *Sensitivität* und *Spezifität* von Testverfahren hingewiesen.

Modellerweiterungen:

- Wie in der linearen Regression kann man eine multiple logistische Regression durchführen. Neben Regressoren können in das Modell auch kategoriale Einflußgrößen, d.h. Faktoren, einbezogen werden.

Daten-Beispiel: Multiple binäre logistische Regression mit Faktor-Kovariable

```
> Daten
      Y    X1  X2  X3  X4

1     Ja   10  2   B   40
2     Ja   25  2   A   30
3     Ja   40  3   B   50
4     Ja   45  3   B   20
.
.
.
66  Nein   15  2   A   30
67  Nein   -4  0   B   10

> mod <- glm(Y ~ X1 + X2 + X3 + X4, family = binomial)
> summary(mod)

Call: glm(formula = Y ~ X1 + X2 + X3 + X4, family = binomial)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
```

```
-1.8424  -0.8288  -0.3950   0.7276   2.4462

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -2.34268    0.65829  -3.559 0.000373 ***

X1          0.09703    0.04743   2.046  0.040793 *

X2          0.22231    0.55866   0.398  0.690680

X3B         1.04516    0.46406   2.252  0.024308 *

X4          -0.01127   0.02501   -0.451 0.652134
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 164.29  on 118  degrees of freedom

Residual deviance: 120.55  on 114  degrees of freedom

AIC: 130.55

Number of Fisher Scoring iterations: 5

> mod1 <- glm(Y ~ X1 + X3, family = binomial)
>
> summary(mod1)

Call: glm(formula = Y ~ X1 + X3, family = binomial)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.7750  -0.8505  -0.4047   0.7178   2.4711

Coefficients:
            Estimate    Std. Error  z value     Pr(>|z|)
(Intercept) -2.46126    0.51810     -4.751      2.03e-06 ***

X1          0.10871     0.02237     4.859       1.18e-06 ***

X3B         1.07705     0.45779     2.353       0.0186 *
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

```
(Dispersion parameter for binomial family taken to be 1)

Null deviance: 164.29  on 118  degrees of freedom

Residual deviance: 120.85  on 116  degrees of freedom

AIC: 126.85

Number of Fisher Scoring iterations: 5

> anova(mod1,mod, test = "Chisq")
Analysis of Deviance Table

Model 1: Y ~ X1 + X3
Model 2: Y ~ X1 + X2 + X3 + X4
  Resid. Df Resid. Dev  Df Deviance P(>|Chi|)
1       116    120.852
2       114    120.548   2    0.303     0.859
```

- In der *Multikategorialen logistischen Regression* können Responsevariablen (Kriteriumsvariablen) betrachtet werden, die nicht nur dichotom (0 und 1) sind, sondern $k > 2$ Ausprägungskategorien besitzen.

b) Poisson Regression

Voraussetzungen:

Die Kriteriumsvariablen $Y_1, \ldots, Y_n$ sind unabhängig $P(\lambda_i)$-verteilte Zufallsvariablen. Man setzt die Strukturgleichung

$$\ln\left(E(Y_i \,|\, x_i)\right) = x_i^\top \cdot \beta$$

zur Modellbildung voraus. Dabei ist

$$\beta = (\alpha, \beta_1, \ldots, \beta_p)^\top$$

der unbekannte Vektor der Modellparameter (Intercept und p Regressionskoeffizienten der Einflussgrößen) und x_i der Vektor der Einflussgrößsen in der i-ten Beobachtung (erste Komponente identisch 1 wegen Intercept). Der Vektor x_i stellt die i-te Zeile der Designmatrix X dar. Die Logarithmusfunktion $g(x) = \ln(x)$ ist zur Poissonverteilung die natürliche Linkfunktion.

Mit der Responsefunktion

$$h(x) \equiv g^{-1}(x) = \exp()$$

erhält man aus der Strukturgleichung die Gleichung

$$E(Y_i \,|\, x_i) = \exp\{\alpha + \beta_1 x_{1i} + \ldots + \beta_p x_{pi}\}.$$

Mit den ML-Schätzungen

$$\widehat{\beta} = (\widehat{\alpha}, \widehat{\beta}_1, \ldots, \widehat{\beta}_p)$$

erhält man für jede gegebene Kovariablenkombination

$$x_0 := (x_{10}, \ldots, x_{p0})$$

eine Schätzung der Kriteriumsvariable (bzw. deren Erwartungswert)

$$\widehat{\mu}(x_0) \equiv \widehat{E}(Y \mid x_0) = \exp\{\widehat{\alpha} + \widehat{\beta}_1 x_{10} + \ldots + \widehat{\beta}_p x_{p0}\}.$$

Neben den Schätzungen der Modellparameter und der damit möglichen Vorhersage von Werten der Kriteriumsvariablen Y ist vor allem die Auswahl der bzgl. Y prädiktiven Kovariablen (z.B. über partielle Waldtests oder Devianztests) bzw. deren optimale Kombination (Modellbildung mittels AIC oder Devianzanalyse) ein Hauptziel der Modellbildung.

Modell-Berechung: Für einen strukturellen Zusammenhang der Kriteriumsvariablen Y mit 2 Einflussgrößen $X1$ und $X2$ lautet die Syntax

```
mod <- glm(Y ~ X1 + X2, family = poisson)
```

Die weiterführenden Methoden (angewandt auf das GLM-Objekt `mod`), wie z.B. `summary`, `plot`, `predict` usw., können ganz anlog zur logistischen Regression verwendet werden.

Bei der Anwendung von `predict` sei darauf hingewiesen, dass die Ausgabe von Konfidenz- bzw. Prognoseintervallen für $E(Y)$ in GLM nicht analog zu LM verfügbar ist.

Weiter ist bei der Anwendung von `predict` darauf zu achten, dass man in der Methode `predict` über den Parameter `type =` die gewünschte Skala der Prädiktion wählt.

```
type=c("link","response")
```

`link` definiert die Skala des linearen Prädiktors und `response` setzt die Skala der Kriteriumsvariable.

Wie in der logistischen Regression können auch Faktoren von R automatisiert dummycodiert in die Modellbildung integriert werden.

c) Gamma-Regression

Voraussetzungen:

Die Kriteriumsvariablen $Y_1, \ldots, Y_n$ sind unabhängig $\Gamma(\alpha_0, \beta_0)$-verteilte Zufallsvariablen, $\alpha_0 > 0$, $\beta_0 > 0$. Man verwendet die Strukturgleichung

$$\frac{1}{(E(Y_i \,|\, x_i))} = x_i^\top \cdot \beta$$

zur Modellbildung, wobei β wieder den unbekannten Vektor der Modellparameter bezeichnet. x_i ist der Vektor der Einflussgrößsen in der i-ten Beobachtung (erste Komponente identisch 1 wegen Intercept). Der Vektor x_i stellt die i-te Zeile der Designmatrix X dar. Die inverse Funktion $g(x) = \frac{1}{x}$ ist die zur Gamma-Verteilung natürliche Linkfunktion.

Mit der Responsefunktion

$$h(x) \equiv g^{-1}(x) = \frac{1}{x}$$

erhält man aus der Strukturgleichung die Gleichung

$$E(Y_i \,|\, x_i) = \frac{1}{\alpha + \beta_1 x_{1i} + \ldots + \beta_p x_{pi}}.$$

Modell-Berechung: Für einen strukturellen Zusammenhang der Kriteriumsvariablen Y mit 2 Einflussgrößen $X1$ und $X2$ lautet die Syntax

```
mod <- glm(Y ~ X1 + X2, family = gamma)
```

4.6.3 Nichtlineare MQ-Regression

Im Unterschied zur linearen Regressionsanalyse im LM betrachet man hier einen strukturellen Zusammenhang der Form

$$Y_i = \mu_i(\beta) + e_i,\ i = 1, \ldots, n,$$

mit unabhängigen, zentrierten und homoskedastischen Fehlervariablen e_i, $i = 1, \ldots, n$. Die bekannten, reellwertigen Regressionsfunktionen $\mu_i \equiv \mu(\beta, x_i)$ beinhalten im Allgemeinen die Regressorwerte $x_i = (x_{1i}, \ldots, x_{pi})^\top$ und den unbekannten Parametervektor $\beta \in \mathbb{R}^p$. Der Erwartungswert $E(Y_i)$ ist nicht mehr als lineare Funktion der Koeffizienten β_j, $j = 1, \ldots, p$, vorausgesetzt.

Beispiel:

$$Y_i = \beta_1 \cdot e^{\beta_2 \cdot x_i} + e_i,\ i = 1, \ldots, n.$$

Dieses nichtlineare Modell besitzt nur einen Regressor x und zwei Modellparameter β_1 und β_2.

Minimum Quadrat Methode (least squares):

Der MQ-Schätzer $\widehat{\beta}$ wird (falls existent) über die Minimierungseigenschaft

$$\widehat{\beta} = \text{argmin}_{\beta \in \mathbb{R}^p} \left\{ \sum_{i=1}^{n} (Y_i - \mu(\beta, x_i))^2 \right\}$$

bestimmt. Wie im GLM sind die Schätzgelichungen im Allgemeinen nicht explizit lösbar und es werden iterative, numerische Verfahren (Newtonverfahren bzw. Erweiterungen) zur Bestimmung der Schätzwerte verwendet. In der Praxis zeigt sich, dass die Wahl der Startwerte für die Konvergenz der Schätzalgorithmen eine wichtige Rolle spielt.

Für nichtlineare Regressionsmodelle stehen Konfidenzintervalle und Signifikanztests ähnlich zum LM oder GLM zur Verfügung. Für die notwendigen Verteilungsaussagen verwendet man eine Normalverteilungsannahme und konstruiert auch ohne Normalverteilungsaannahme (unter geeigneten Regularitätsvoraussetzungen) asymptotische Konfidenzintervalle. Eine Einführung in nichlineare Regression mit Konfidenzintervallen und Tests findet man bei Pruscha (2006) S. 200 ff.

Innerhalb von R können nichtlineare MQ-Regressionmodelle mit der Methode

`nls()`

bearbeitet werden. Die Syntax (bzgl. Modellspezifikation etc.) ist wieder ähnlich den Methoden `lm()` und `glm()`.

Ausblick:

Neben den in diese Skriptum aufgeführten parametrischen Regressionsmodellen spielen in der Anwendung immer mehr auch *nicht-parametrische* oder *semi-parametrische Modelle* eine Rolle. Bei nicht-parametrischen Methoden werden die Regressionsfunktionen (bis auf eine, aus theoretischen Gründen geforderte, Zugehörigkeit zu sehr allgemeinen Funktionenräumen) als völlig unbekannt betrachtet. Nichtparametrische Verfahren der Statistik zur Bildung von Regressionmodellen sind z.B. *Kernschätzer*, *Splineschätzer* oder *additive Verfahren*. Nicht-parametrische Verfahren, die eher aus dem Bereich des *maschinellen Lernen* kommen, sind z.B. *Klassifikations-,Regressionsbäume* oder *random forrest*. Letztere Methode verbindet Entscheidungsbäume mit *bootstrap*-Verfahren (vgl. *Resampling-Methoden*), die auch bei parametrischen Regressionsverfahren eingesetzt werden können. Auch für nichtparametrischen Methoden (insbesondere auch aus dem Bereich des maschinellen Lernens)und für resampling Verfahren stehen geeignete R-packages zur Verfügung.

Literaturverzeichnis

[1] Fox, J. (2003): Effect displays in R for generalised linear models. *Journal of Statistical Software*, 8(15):1-27.

[2] John Fox, with contributions from Michael Ash, Theophilius Boye, Stefano Calza, Andy Chang, Philippe Grosjean, Richard Heiberger, G. Jay Kerns, Renaud Lancelot, Matthieu Lesnoff, Samir Messad, Martin Maechler, Duncan Murdoch, Erich Neuwirth, Dan Putler, Brian Ripley, Miroslav Ristic and and Peter Wolf. (2008). Rcmdr: R Commander. R package version 1.3-15. http://www.r-project.org, http://socserv.socsci.mcmaster.ca/jfox/Misc/Rcmdr/

[3] Fox, J. (2005): The R commander: A basic-statistics graphical user interface to R. *Journal of Statistical Software*, 14(9):1-42.

[4] Ligges, U. (2007): Programmieren mit R. Berlin, Heidelberg: Springer Verlag.

[5] Pruscha, H. (2006): Statistisches Methodenbuch. Berlin, Heidelberg: Springer Verlag.

[6] R Development Core Team (2009a): R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org.

[7] R Development Core Team (2009b): R Data Import/Export. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org.

[8] R Development Core Team (2009c): R Installation and Administration. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org.

[9] R Development Core Team (2009d): R Language Definition. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org.

[10] R Development Core Team (2009e): Writing R Extensions. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org.

[11] Sachs, L. und Hedderich, J. (2006): Angewandte Statistik, Methodensammlung mit R. Berlin, Heidelberg: Springer Verlag.

[12] Venables, W. N., Smith, D. M., und the R Development Core Team (2009): An Introduction to R. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org.