Statistik und W-Theorie mit R

Schnelleinstieg

Statistik Betriebsökonomie HES

HES-SO Standort Siders

zusammengestellt von

Paul Ruppen

Version vom 6. März 2020

Dieses Dokument wurde mit LaTeX gesetzt.

Inhaltsverzeichnis

1 Herunterladen des Programms

Home-page: http://www.r-project.org/ (diese finden Sie auch schnell durch Eingabe von R in Google). Dann geht man unter: download: CRAN; man wählt den Server der ETHZ. Wählt Betriebssystem, dann „base“, dann „Download R x.x.x for Windows“ (wenn man auf Windows installiert; x.x.x steht für irgend eine Version von **R**).

Im folgenden wird ein R-Paket verwendet, das „varia“ heisst. Sie finden das Paket auf der Homepage http:/math.logik.ch unter *Statistik*. Das Paket kann wie folgt installiert werden: Paket herunterladen (ohne zu entzippen). In **R** auf „Pakete“ und „installiere Paket(e) aus lokaler zip-Datei“, den Speicherort des gezipten Paketes suchen, anklicken und die Sache sollte geritzt sein. Sollte dies nicht klappen, kann man das Paket entzippen und das Verzeichnis „varia“ (Achtung: nicht das durch das Entzippen vermutlich geschaffene gleichnamige Verzeichnis, sondern das entzippte Verzeichnis! Die richtige Verzeichnis enthält unter anderem die Verzeichnisse „chtlm“, „data“, „R“ und „man“) in der **R**-Installation im Verzeichnis „library“ abspeichern.

Das Paket kann verwendet werden, indem man „library(varia)“ in den workspace schreibt. Die Erläuterungen zum Paket erhält man, indem man nach der Einbindung des Pakets „`help(varia)`“ schreibt oder einen der Befehle des Pakets mit „`help(befehl)`“ verwendet. Das Paket muss jedesmal, wenn man es braucht, mit „`library(varia)`“ eingebunden werden.

Bemerkung 1. *Bei Fragen zu **R** findet man viel auf der Home-Page (Handbücher) und unter „mailing-lists“. Oft kommt man schnell zum Ziel, indem man in Google R und dann ein Stichwort eingibt.*

2 Einlesen von Datenmatrizen

Kopieren der Daten und Einfügen in den „Editor“. Abspeichern als Text-Datei. Einlesen in **R** durch den Befehl: `dataframename=read.table(„c:/dateiname.txt", header=T)` („=“ weist das Objekt rechts dem Namen links zu. Es entspricht damit dem Definitionszeichen „:=“, d.h. „gleich durch Definition“. Statt „=“ kann auch „<-“ verwendet werden. Dieses Zeichen in R-Quell-Codes ist weit verbreitet, da es das ursprüngliche Zuweisungszeichen von **R** ist. Testen auf Gleichheit wird in **R** durch „==“ ausgedrückt).
Zu beachten ist:

- Die Zeile mit den Variablennamen muss gleichviele Wörter enthalten wie die Tabelle Spalten aufweist. Ein Leerschlag macht aus einer Zeichenfolge zwei Wörter. Der Variablenname „Zufriedenheit mit Arbeit“ besteht aus drei Wörtern. Er kann mit „Zufriedenheit_mit_Arbeit“ ausgedrückt werden - wobei dieser Name fürs praktische Arbeiten in R zu lange ist, da man in R Befehle schreiben muss.
- Sind bei nominal- oder ordinalskalierten Variablen die Ausprägungen benannt, so dürfen diese nur aus einem Wort bestehen (statt „sehr gut“ als z.B. „sehr_gut“).
- Durch die beschriebene Methode wird die Datei als ein sogenanntes Dataframe eingelesen, ein spezielles **R**-Objekt (wie auch Vektoren, Matrizen, Listen, Funktionen, etc. **R**-Objekte sind).
- Im Beispiel wurde fürs Abspeichern der Datei das Laufwerk **c:** ohne Unterverzeichnis gewählt.
- Wählen Sie eine günstige Speicherstelle (weniger Fehlerquellen!)
- **R** unterscheidet zwischen Gross- und Kleinschreibung
- Beachten Sie den verwendeten Verzeichnis-Trenn-Strich (nicht den rückwärtigen Schrägstrich (backslash), sondern den Schrägstrich (slash)).
- „`header=T`“ bedeutet, dass die erste Zeile die Variablennamen enthält. Vermeiden Sie, wenn möglich, Umlaute.

- Liegen fehlende Wert vor, so sollten diese vorgängig in Excel durch „NA“ ersetzt werden (NA steht für „not available“).
- Es gibt weitere Einlesemethoden (s. PDF auf der R-Home-Page; manuals). Die eben beschriebene funktioniert gut!

3 Umbenennen von Variablen und Ausprägungen in Dataframes

Möchte man ein Dataframe umbenennen, wird diesem einfach ein neuer Name zugewiesen:
`NeuerName=AlterName`;
Das alte Dataframe kann dann gelöscht werden durch
„`rm(AlterName)`“.

Möchte man in einem Dataframe eine Variablennamen ändern, so schreibt man:
`names(Dataframename)[names(Dataframename)== Altervariablenname]=`
`„NeuerVariablenname".` (zwei Gleichheitszeichen bedeuten „Überprüfung auf Gleichheit“, ein Gleichheitszeichen „Zuweisung durch Definition“, die eckigen Klammern braucht man bei Dataframes, Vektoren und Matrizen, um die Stellen anzugeben (bei Matrizen braucht es also zwei Stellen, die durch Komma abgetrennt werden). Hier wird an der Stelle des Dataframes, wo der Name der Dataframevariable mit dem alten Variablennamen identisch ist, der alte durch den neuen Variablennamen ersetzt).

R unterscheidet zwischen metrischen Variablen (bei `str()` durch „num“ angegeben), nominalskalierten Variablen (factor) und ordinalskalierten Variablen (ord.factor). Bei den metrischen Variablen werden noch Variablen mit ganzzahligen Ausprägungen angegeben (int).
Man kann den Variablentyp verändern durch
Verwandlung in numerische Variable: `dat$v1=as.numeric(dat$v1)`,
Dadurch wird die Variable `dat$v1` umgewandelt und die alte steht nicht mehr zur Verfügung. Es ist im allgemeinen besser, eine neue Variable zu schaffen, dann kann immer auf die alte zurückgegriffen werden. Man würde also z.B. schreiben. `dat$v1num=as.numeric(dat$v1)`
Verwandlung in nominalskalierte Variable: `dat$v1f=factor(dat$v1)`
Verwandlung in ordinalskalierte Variable:
`dat$v2ord=ordered(dat$v2)` oder `dat$v2ord=factor(dat$v2,ordered=T)`.

Es empfielt sich, bei R nur Zahlen einzulesen. Man hat dann nur int- und num-Variablen und man ist flexibler. Man kann immer im nachhinein auf der Grundlage der eingelesenen Variablen diese entsprechend umzuwandeln oder Graphiken geeignet beschriften.

Man kann aber auch mit nominal- und ordinalskalierten Variablen arbeiten, deren Ausprägungen benannt sind (z.B. so eingelesen; beim Einlesen dürfen diese keine Leerzeichen enthalten!). Möchte man eine int- oder num-Variable (mit wenig Ausprägungen) in einen Faktor mit benannten Ausprägungen verwandeln, so schreibt man:
`Dataframename$VariablennameF=factor(Dataframename$Variablenname,`
`labels= c(„Ausprägungsname1", „Ausprägungsname2", ...., „AusprägungsnameM"))`
Die Namen der Ausprägungen der Variable werden so zugeordnet, dass der kleinsten Zahl der erste Name des Label-Vektors, der zweitkleinsten der zweite Name des Label-Vektors, etc. zugeordnet werden.

Muss man einmal eine als String (d.h. die Ausprägungen sind benannt) eingelesene ordinalskalierte Variable bearbeiten, ist folgendes zu beachten. Den Strings werden beim Einlesen vom Programm im Hintergrund Zahlen zugeordnet, und zwar in alphanumerischer Reihenfolge (die Anfangsbuchstaben des Strings bestimmen die Zuordnung der Zahl, bei „anton“, „ap“ wird z.B. „anton“ 1 zugeordnet, „ap“ 2, etc). Bei Barplots wird die Reihenfolge der Balken durch diese Zahlen bestimmt. Bei ordinalskalierten Variablen entspricht deshalb diese Ordnung oft nicht der Ordnung der Ausprägungen. Will man dies ändern, gibt es u.a. zwei Möglichkeiten:

1. Wir betrachten ein Beispiel: `a=data.frame(dat=1:10,zf=c("zu","nz","nz","mz","nz", "mz","sz","sz","mz","mz"))`. Das Programm ordnet im Hintergrund "zu" die Zahl 4 zu,

"nz" die Zahl 2, "mz" die Zahl 1 und "sz" die Zahl 3. Dies sieht man durch das Ausführen von str(a) und a. Mit a$zf1=factor(a$zf,levels=c("nz","mz","zu","sz")) Wird eine Variable zf1 geschaffen, so dass nun "nz" die Zahl 1, "mz" die Zahl 2, "zu" die Zahl 3 und "sz" die Zahl 4 zugeordnet ist. Verwendet man derart `factor` wird eine nominalskalierte Variable geschaffen, die aber die der ordinalskalierten Variable entsprechende Ordnung der zugeordneten Zahlen aufweist. Statt `factor` kann man aber auch `ordered` verwenden, um eine fürs System ordinalskalierte Variable zu schaffen.

2. Die zweite Variante ist etwas umständlicher, zeigt aber, wie man Variablen rekodieren kann. Man verwandelt die eingelesene String-Variable `data$V1` durch z.B. `data$V1num=as.numeric (data$V1)` in eine numerische Variable. An der neuen Variable, kann man ablesen, welche Zahlen den Ausprägungen von `data$V1` zugeordnet sind. Dann werden die Ausprägungen der Variable `data$V1num` umkodiert: Soll die Variable `data$V1numrecod` an der Stelle, an der die Variable `data$V1num` den Wert 1 annimmt, den Wert 2 annehmen, schreibt man `data$V1numrecod[data$V1num==1]=2` (diesen Befehl gibt man für alle Ausprägungen ein, bis man die `data$V1num` so umgewandelt hat, dass die Reihenfolge der Zahlen der Reihenfolge der Ausprägungen entspricht; überprüfen mit `head(data,10)`. Schliesslich kann man die Variable `data$V1numrecod` in eine ordinalskalierte Variable um durch `data$V1neu=ordered (data$V1numrecod, label=c("Name der Zahl1","Name der Zahl 2", ..... "Name der Zahl n")`. (Die Zuordnung der Namen zu den Zahlen erfolgt in aufsteigender Reihenfolge der Zahlen) c). Zum Schluss mit `head(data,10)` überprüfen! Besser ist es allerdings, wenn immer möglich, mit der Variable `data$V1numrecod` zu arbeiten und bei Graphiken diese geeignet zu beschriften. Muss man die Variable als ordinale verwenden, kann man immer noch lokal ordered(`data$V1numrecod`) verwenden, ohne eine neue Variable zu schaffen. Die Variable `data$V1numrecod` bleibt dadurch erhalten. Ist eine num- oder int-Variable lokal als normaler, nicht-geordneter Faktor zu verwenden, schreibt man „`factor(data$Variable)`“. Die Variable selber bleibt unverändert, wird aber im entsprechenden Befehl als Faktor-Variable verwendet.

Umbenennen von bereits benannten Ausprägungen von (ordered) factors: `levels(data$V1) [levels (data$V1) == „zu_ersetzender_Name"]=„neuer_Name"` (Zum Verständnis des Befehls: `levels(data$V1)` ist ein Vektor. Durch die eckigen Klammern wird eine Stelle im Vektor angegeben, nämlich die Stelle, an der der Vektor mit dem zu ersetzenden Namen identisch ist.

4 R-Funktionen

R verwendet man im Allgemeinen, indem man in den Workspace oder in ein Skript Funktionen schreibt und diese ausführt. Funktionen haben einen Namen und eine Syntax, die nach dem Namen in Klammern steht. Die Syntax kann leer sein, besteht jedoch im allgemeinen aus Spezifikationen der Funktion: worauf soll die Funktion angewendet werden? Wie soll sie anwendet werden? Was soll ausgegeben werden? usw. So ist in `log(5)` „log“ der Name der Funktion und in „()“ stehen die Spezfikationen, im vorliegenden Falle, wovon der natürliche Logithmus zu berechnen ist (nämlich von 5). So ist in

$$\texttt{plot}(\texttt{c}(5,6,7,8),\texttt{c}(8,9,10,11),\texttt{xlab}=\text{„}\texttt{Time}\text{”})$$

„`plot`“ die Funktion, die auf die Vektoren (Datensätze) $\texttt{c}(5,6,7,8), \texttt{c}(8,9,10,11)$ angewendet wird. In `xlab = „Time"` drückt das Gleichheitszeichen aus, dass für die Beschriftung der x-Achse „Time“ steht. xlab ist eine vorgegebene Spezifikation, die man selber verändern kann oder nicht. Für die meisten Spezifikationen gibt es nämlich eine Voreinstellung. Es ist dann nicht nötig, die entsprechenden Spezifikation anzugeben. Sie muss nur angegeben werden, wenn man eine andere Spezifikation vornehmen will, also z.B. „Zeit“ statt „Time“. R-Funktionen können sehr viele Spezifikationen enthalten. Auf die Angabe des Ausdrucks vor dem Gleichheitszeichens und auf dieses selbst kann verzichtet werden, wenn man genau die Reihenfolge der vorgegebenen Spezifikationen

einhält und keine auslässt. In der Funktion „plot“ könnte man z.B. auf „xlab = “ nicht verzichten, da die plot-Funktion vor xlab noch viele weitere Spezifikationen enthält, die wir im obigen Befehl nicht verändern wollten und deshalb nicht erwähnten.

Hilfe zu vorgegebenen Funktionen erhält man durch die Eingabe von

```
help(Funktionsname)
```

enter. Dies ist nur erfolgreich, falls man den Namen der Funktion kennt. Alle vom Core-Team vorgegebenen Funktion findet man, indem man in der Programminstallation unter „library“ die Namen der Pakete einsieht und deren Name in die Hilfefunktion eingibt: mit `help(stats)` findet man z.B. alle Befehle für statistische Tests, die vom Core-Team zur Verfügung gestellt werden. Kommt man so nicht zum Ziel, kann man es mit dem `help.search(„Ausdruck")`-Befehl versuchen. Es werden dann Fundstellen des Wortes im Hilfeprogramm angegeben. Wenn dies nicht hilft, sucht man am besten auf der R-Homepage, auf Google oder einer anderen Suchmaschine. Die Quellcode von Funktionen kann man oft durch `edit(Name_der_Funktion)` ausgeben. Damit kann man - falls nötig - überprüfen, wie die Funktion etwas berechnet.

5 Häufigkeitsverteilungen und Verteilungsfunktion

Häufigkeitsverteilungen: Sei $V1$ eine Variable der Datenmatrix „dataframename“ mit drei Ausprägungen, mit 1, 2 und 3 kodiert (für grün, rot und blau). Dann wird die *absolute Häufigkeitsverteilung* durch

```
table(dataframename$V1)
```

erstellt. Will man diese weiterverwenden, wählt man einen Namen dafür, z.B.

```
hauefig=table(dataframename$V1)
```

Wir haben in **R** ein Objekt geschaffen, das „haeufig“ heisst und das wir anschauen können, indem man „`hauefig`“ schreibt und durch „enter“ bestätigt. Man erhält die Häufigkeitsverteilung. Diese kann man beschriften mit `names(hauefig)=c(„grün", „rot", „blau")`. Führt man wieder `haeufig` aus, erscheint die Tabelle mit den Benennungen.
Graphische Darstellung: `barplot(hauefig,names.arg=c(„grün",„rot",„blau"),`
`main=„Absolute Häufigkeitsverteilung", xlab=„Zeit",ylab=„absolute Häufigkeiten")`
names.arg: Beschreibung der Ausprägungen (ist nicht nötig, wenn man die Ausprägungen in der Tabelle beschriftet hat); main = Haupttitel; xlab= Beschreibung der x-Achse, ylab = Beschreibung der y-Achse.
Für die Beschriftung mit den Häufigkeiten speichert man den obigen barplot-Befehl unter z.B „pb“ ab. Unter „pb“ wird dadurch ein Vektor, der die Lage der Balken auf der x-Achse angibt, abgespeichert. Man kann dann z.B. schreiben:
`text(x=pb,y=hauefig+1,labels=haeufig)` (die Häufigkeiten werden knapp über den Balken hingeschrieben) oder `text(x=bp,y=hauefig-1,labels=haeufig)` (die Häufigkeiten werden knapp unter der oberen Balkenbegrenzung hingesetzt). Statt 1 kann man andere Zahlen wählen.

Verwandlung einer metrischen Variable in eine Faktorvariable mit klassierten Daten: `b=cut(df$Variable, c(Grenze_1,...,Grenze_m))` - es werden mit `table(b)` die absoluten Häufigkeiten für rechts-geschlossenen Intervalle berechnet. Man kann die neue Variable b auch in das bearbeitete Dataframe aufnehmen, indem man für das Dataframe Df z.B. schreibt: `Df$VariableF=cut()` (F für Faktor, die Namengebung ist beliebig; die erste und die letzte Grenze müssen die Daten umfassen. Das erste und das letzte Datum kann man mit `range(df$variable)` ermitteln. Gibt man diese Werte dann als erste und letzte Grenze ein, wird das erste Datum nicht berücksichtigt (da links offen). Dies kann man beheben durch `cut(df$Variable, c(Grenze_1,...,Grenze_m),include.lowest=T)`. Rechtsoffene Intervalle erhält man mit dem zusätzlichen Parameter `right=F` im cut-Befehl. Das letzte Datum erhält man wiederum mit `include.lowest=T` berücksichtigt.

Die relative Häufigkeitsverteilung berechnen wir aus der absoluten Häufigkeitsverteilung:

```
relhauefig=hauefig/sum(hauefig)
```

Kumulierte Häufigkeitsverteilungen erhält man durch

```
kumabsHaeuf=cumsum(hauefig)
kumrelHaeuf=cumsum(relhauefig)
```

Die graphischen Darstellungen erhält man mit den selben Befehlen wie für die absolute Häufigkeitsverteilung.
Alle vier Häufigkeitsverteilungen kann man auf einmal durch den Befehl „`hverteilung()`“ auf dem R-Paket „varia“ gewinnen.
Stamm-Blatt-Darstellung: `stem(dataframename$variable)`

Histogramme: `hist(dataframename$variable)`. Daten werden automatisch klassifiziert. Man kann die Voreinstellungen ändern durch:
(a) „`hist(dataframename$variable,breaks= c(5,10,20,40))`“. (Angabe der Klassengrenzen; dabei muss der ganze Datenbereich durch die Grenzen abgedeckt sein; findet man durch `max()` und `min()`) oder (b) durch
„`hist(dataframename$variable,breaks=7)`“ (Angabe der Anzahl Klassen). Durch „`freq=F`“ werden die relativen Häufigkeiten angegeben, sonst die absoluten.

Eindimensionales Streudiagramm:
„`plot(dataframename$variable,rep(1,length(dataframename$variable)))`“
Durch „`rep(1,5)`“ wird ein Vektor der Länge 5 von Einsen produziert.
„`length(dataframename$variable)`“ gibt die Anzahl der Komponenten des Datenvektors „`dataframename$variable`“ an (Excel: =anzahl()).

Empirische Verteilungsfunktion: Treppenfunktion mittels

```
plot(ecdf(dataframename$variable))
```

Punkteform: Im R-Paket „varia“ durch „EmpirVerteilFunktion(dataframename$variable)“ erhält man die Verteilungsfunktion in Punkteform, durch

```
EmpirVerteilFunktion(dataframename$variable)
```

6 Kennwerte

Will man im Output mehr Stellen sehen, kann man z.B. `options(digits=10)` eingeben (für 10 Ziffern). Die Voreinstellung ist 7. Bei fehlenden Daten, muss dem Programm oft gesagt werden, dass fehlende Werte nicht berücksichtigt werden sollen - dies merkt man daran, dass sonst eine Fehlermeldung erscheint. Dies geschieht bei vielen Befehlen durch den Befehl „`na.rm=T`“ (na = not available; rm = remove, T = TRUE; im ersten folgenden Befehl wird das gezeigt)
Maximum: `max(dataframename$variable,na.rm=T)`
Minimum: `min(dataframename$variable)`
Arithmetische Mittel: `mean(dataframename$variable)`
Median: `median(dataframename$variable)`
Modalwert: `modalwert(dataframename$variable)` (zu finden im R-Paket „varia“)
Spannweite: `max(datenframename$variable)-min(dataframename$variable)`
oder mit `range(datenframename$variable)`

Mittlerer Abweichungsbetrag: `MAB()` (zu finden im R-Paket „varia“)
Varianz: `var()`
Standardabweichung: `sd()`
Normierte Lorenzkonzentration: `Lorenz()` (zu finden im R-Paket „varia“, fehlende Werte werden automatisch entfernt); es können statt der Daten auch Häufigkeitsverteilungen eingegeben werden. In diesem Fall muss man zwei Vektoren in `Lorenz()` eingeben. z.B.
`Lorenz(c(5,6,7,8,2),c(2.5,7.5,12.5,17.5))`. Der zweite Vektor gibt die Klassenmitten an, der erste die absolute Häufigkeitsverteilung. Statt der absoluten Häufigkeitsverteilung kann an derselben Stelle auch die relative Häufigkeitsverteilung eingegeben werden.
Quantile: `quantile(daten, probs = seq(0, 1, 0.25), names = TRUE, type = 7)`. Mit „seq“ gibt man eine arithmetische Folge zwischen 0 und 1 mit dem angegebenen Abstand (hier 0.25) an. Man kann andere Abstände wählen oder einen Vektor $c(a, b, c, d, ...)$ mit frei gewählten Abständen eingeben. „na.rm=T“ muss stehen, wenn es fehlende Werte gibt. **R** unterscheidet 9 verschiedene Arten, Quantile zu berechnen. Die im Skript eingeführte Methode findet man mit `type=7` (= Excel-Typ), wobei `type=7` die Voreinstellung ist und nicht spezifiziert werden muss)
Quartilsabstand: `quantile(daten,probs=0.75)-quantile(daten,probs=0.25)`
Simpsonindex: `Simpson(dataframename$V1)`; (zu finden im R-Paket „varia“); Es kann auch eine absolute oder relative Häufigkeitsverteilung eingegeben werden: absolute Häufigkeitsverteilung: `Simpson(c(5,3,4,8),AHVerteilung=T)`
relative Häufigkeitsverteilung: `Simpson(c(0.4,0.2,0.1,0.3),rel=T)`
Schiefe: `schiefe()`; (zu finden auf dem R-Paket „varia“);
Mit `summary(dataframename$V1)` kann man ein paar Kennwerte auf einmal berechnen lassen.

7 Bivariate beschreibende Statistik

Kreuztabellen:
`tab= table(dataframename$variablenname1, dataframename$variablenname2)`; will man die Ausprägungen benannt haben, kann man die Variablen in Faktoren verwandeln und die Zahlen durch Beschriftung ersetzen (s.o., nicht empfehlenswert). Besser ist es, die Kreuztabelle direkt zu beschriften mit `colnames(tab)=c(„Name1",..., „NameI")` und `rownames (tab)= c(„Name1" ,..., „NameJ")`. Mit dem Ausführen von `tab` sieht man das Resultat.

Säulendiagramme: `barplot(tab)` (mit den Daten der Kreuztabelle). Es werden automatisch die Beschriftungen der Spalten der Tabelle auf die x-Achse gesetzt. Will man die Ausprägungen der y-Achse benannt haben muss man eine Legende erstellen: (s. `help(legend)`und das folgende Beispiel). Will man die Säulen nebeneinander haben, gibt man im Barplot-Befehl zusätzlich `beside=T` ein.

Beispiel: (2X3 Kreuztabelle „tab“ (Körpergrösse* Augenfarbebeispiel): (im Beispiel wurden in der Tabelle vorgängig keine Beschriftung der Spalten vorgenommen)

```
   barplot(tab, main = c(„Körpergrösse * Augenfarbe"),
ylim = c(0,10),
ylab=„Häufigkeiten",
names.arg = c(„klein",„gross"),
col = c(„green", „blue",„gray90"))

   legend(„topright",c(„grün",„blau", „andere"),
fill=c(„green",„blue",„gray90"))
```

Hat man die Spalten in der Tabelle bereits beschriftet, erübrigt sich der names.arg-Befehl. Eine Tabelle mit den möglichen Farben findet man unter:
http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf. Ohne `ylim=c(0,10)` würde die Legende in die Balken hineinkommen. „`xlim`“ und „`ylim`“ legen die Länge der Achsen fest. Ohne Spezifikation werden sie aus der ersten graphischen Darstellung berechnet (z.B. Längster Balken, Anzahl

Balken). Erfolgen mehrere graphische Darstellungen in einer Graphik, hat das zur Folge, dass man dann nicht alles sieht oder dass sich Graphiken überlagern. Entsprechend muss man in diesem Fall spezifizieren.

Will man die Balken mit der Häufigkeiten beschriften, speichert man den barplot-Befehl, z.B. unter „pb“ ab. Dadurch wird unter „pb“ ein Vektor der Lage der Balken auf der x-Achse abgespeichert. Für die Beschriftung im Balken oder knapp oberhalb der Balken kann man die Häufigkeiten der Kreuztabelle verwenden. Im folgenden Beispiel werden die Häufigkeiten jeweils in die Mitte der Streifen gesetzt:

text(x=pb,y=tab[1,]/2, labels = tab[1,])
text(x=pb,y=tab[1,]+tab[2,]/2, labels = tab[2,])
text(x=pb,y=tab[1,]+tab[2,]+tab[3,]/2, labels = tab[3,1])

Setzt man die Balken nebeneinander, so wird unter pb eine Matrix mit der Lage der Balken abgespeichert.

Mit `margin.table (Tabelle,1)` wird die Zeilenrandverteilung berechnet mit
`margin.table(Tabelle,2)` die Spaltenrandverteilung einer Kreuztabelle.

Mit `prop.table(Tabelle)` werden die relativen Häufigkeiten für eine Kreuztabelle berechnet (die Summe der Zellen gibt nachher 1).

Zweidimensionales Streudiagramm (für zwei Variablen, die metrisch skaliert sind, oder für den Fall, dass die x-Achsen-Variable nominal und die y-Achsen-Variable metrisch skaliert ist): `plot(v1,v2,main=„Haupttitel", xlab=„x-Achsenbeschriftung",`
`ylab=„y-Achsenbeschriftung")`). Ist im zweiten Fall die nominalskalierte Variable als Faktor eingegeben, so erscheint automatisch ein Boxplot. Die Ausgabe eines Streudiagramms kann man erzwingen durch den Zusatz „`as.numeric(v1)`“, besser ist es aber `stripchart (metrische_variable~ Faktorvariable, vert=T)` zu verwenden.

Boxplot (für Daten, die auf der x-Achse nominal und auf der y-Achse metrisch skaliert sind): `boxplot(v1~v2)` (an erster Stelle muss die metrisch skalierte Variable stehen, an zweiter Stelle die Gruppenvariable).
Beschriftung: boxplot(v1~v2, names=c("Gruppe 1", "Gruppe 2", "Gruppe 3"), ylab="Einkommen", main="Boxplot Einkommen pro Gruppe"). Ist die Gruppenvariable eine num- oder int-Variable, schreiben wir `boxplot(v1~factor(v2)).`

Gamma: Durch die Funktion `gamma.test(v1,v2)` im **R**-Paket „varia“.

Quotenverhältnis (engl. Odds-Ratio) durch „fisher.test()“ auf zwei dichotome Variable (oder auf eine 2*2-Tabelle); Es wird allerdings eine weitere Berechnungsart des Quotenverhältnisses verwendet, die auf theoretischen Grundlagen ruht, die wir nicht behandeln können (Maximum-Likelihood-Schätzer)

Pearson-Korrelation: Durch die Funktion `cor(v1,v2)`.

Spearman-Korrelation: Durch die Funktion `cor(v1,v2,method=„spearman")`.

Partielle Korrelation: Durch die Funktion `PartielKor(v1, v2,v3)` im **R**-Paket „varia“ (v3 ist die Kontrollvariable).

Kontingenz-Koeffizient im varia-Paket (`kontingenz.koeff(v1,v2)`)

Regression: `a=lm(Zielvariable~Unabhängige_Variable, data=NameDataframe)`
`a` (es werden die Koeffizienten geliefert)
`plot(Zielvariable~Unabhängige_Variable)` oder
`plot(Unabhängige_Variable,Zielvariable)`
`abline(a)` oder `abline(coef(a))` oder `abline(coefficients(a))` oder
`abline(a$coefficients[1],a$coefficients[2])` (Es wird die Regressionsgerade in den Punkteplot eingefügt; mit `abline(h=0)` und `abline(v=0)` kann man eine horizontale und vertikale Achse im Punkt (0,0) einfügen.
predict (mit Hilfe der Regressionsgeraden Voraussagen machen; dazu muss man zuerst ein Dataframe mit den Argumenten erstellen, für die die Voraussagen gemacht werden sollen - mit dem Variablennamen der unabhängigen Variable, z.B. `b=data.frame(Unabhängig_Variable=c(4,5))`.
Dann schreibt man:
`predict(a,b))`. Es erscheint ein Vektor mit den Voraussagen für 4 und 5 (für `a` wie oben bei

Regression, also Objekt, das mit `lm` erstellt wurde). Man kann für einzelne Voraussagewerte an der Stelle x aber auch `c(1,x)%*%coef(a)` rechnen, für mehrere Werte, z.B. für die Werte 4,6,7,11 `cbind(rep(1,4),c(4,6,7,11))%*%coef(a)`

Bei fehlenden Daten kann man unvollständige Zeilen löschen mit `red_data_frame=complete.cases (data.frame.name)` und man führt dann die Regressions-Analyse auf `red_data_frame` durch. Dies ist bei mehreren Variablen nicht empfehlenswert, da man dann auch Zeilen löscht, die bezüglich einer spezifische Analyse vollständig sind. R berücksichtigt bei der Berechnung der Regression Zeilen, die bezüglich der in der Regression verwendeten Analyse nicht vollständig sind, nicht.

Einem Vektor oder einer Variablen eines Dataframes Rangzahlen zuordnen: `order(vektor)` oder `order(dataframe$variable)`. Durch den Befehl werden durchgängig ganze Zahlen zugeordnet. Liegen gleiche Ausprägungen vor, werden also verschiedene Ränge zugeordnet. Der Befehl wird vor allem für das Sortieren von Dataframes nach einer Variable verwendet. Entsprechend ist die Art des Zuordnens von Rängen angemessen.

Ordnen eines Dataframes nach einer Variable: Der Variable wird zuerst mit `order` eine Rangordnung zugeordnet. Die Rangordnungszahlen geben die Zeilen an, welche die Zeilen des alten Dataframes einnehmen sollen. Man schreibt also
`dataframe=dataframe[order(dataframe[,4]), ]` (es wird im Beispiel nach der 4. Zeile geordnet). oder
`dataframe=dataframe[order(dataframe[,dataframe$Variablenname]), ]` (es wird nach der Spalte mit dem Variablennamen "Variablenname"geordnet). Zum Verständis des Befehls: Man betrachte den Vektor `a=c(4.1,3.5,1.4,4.6,6.2,7.2,9.8)`. Mit `order(a)` erhält man: [1] 3 2 1 4 5 6 7 (die Ränge der Komponenten von a in aufsteigender Ordnung). Bei a[3] erscheint die 3. Komponente von a, nämlich die kleinste. Durch a[c(3,2,1,4,5,6,7)] erscheinen entsprechend die geordneten Zahlen des Vektors a.

Ordnen eines Vektors: `sort(vektor)`.

`lapply(dataframe,mean,na.rm=T)` berechnet die Mittelwerte für jede metrische Variable des Dataframes und gibt diese in einer Liste aus (es können auch andere Funktionen wie var, sd, etc. verwendet werden).

`sapply(dataframe,mean,na.rm=T)` berechnet die Mittelwerte für jede metrische Variable des Dataframes und gibt diese in einem Vektor aus.

`apply(meexp, 1, mean,na.rm=T)` berechnet die Mittelwerte für jede Zeile des Dataframes und gibt diese in einem Vektor aus. Für 2 werden die Mittelwerte pro Variable berechnet (also wie sapply).

8 Kombinatorik

`factorial(5)` = 5! (bei grossen Zahlen kann man mit `lfactorial(n)` arbeiten. Es wird der Logarithmus der entsprechenden Zahl geliefert)

`combn(5,3)` zählt alle $\binom{5}{3}$ Stichproben der Grösse drei aus den ersten 5 natürlichen Zahlen 1,...,5 auf (Ziehen ohne Zurücklegen ohne Reihenfolge).

Die Anzahl $\binom{n}{m}$ der Kombinationen durch `choose(n,m)` (bei grossen Zahlen kann man mit `lchoose(n,m)` arbeiten. Es wird der Logarithmus der entsprechenden Zahl geliefert).

9 Wahrscheinlichkeitstheorie

Die Funktionen für Wahrscheinlichkeiten sind alle gleich aufgebaut. Vor den Namen kann man ein „r“, „p“, „q“ oder „d“ setzen. Im Falle von diskreten Verteilungen, ergibt „d“ den Wert der Wahrscheinlichkeitsfunktion, im Fall von stetigen Verteilungen den Wert der Dichtefunktion. „p“ gibt den Wert der Verteilungsfunktion aus. „q“ gibt das Quantil der Verteilung. „r“ erzeugt gemäss der gewählten Verteilung verteilte Zufallszahlen („r“ für „random“, „d“ für „density“, „p“ für „probability“ und „q“ für „quantile“).

Binomialverteilung: `dbinom(x, size, prob)` (Funktionswert: Wert der Wahrscheinlichkeitsfunktion; Argumente: „x“ für die Anzahl der ausgezeichneten Ergebnisse, „size“ für die Stichprobengrösse und „prob“ für die Wahrscheinlichkeit)

`pbinom(q, size, prob)` (Funktionswert: Wert der Verteilungsfunktion; Argumente: „q“ für die Anzahl der ausgezeichneten Ergebnisse (= Argument der Verteilungsfunktion), „size“ für Stichprobengrösse und „prob“ für die Wahrscheinlichkeit)

`qbinom(p, size, prob)` (Funktionswert: p-Quantil; Argumente: „p“ für das p-Quantil, „size“ für Stichprobengrösse und „prob“ für die Wahrscheinlichkeit)

`rbinom(n, size, prob)` (Funktionswert: Vektor mit Zufallszahlen; Argumente: „n“ = Anzahl zu schaffender Zufallszahlen, „size“ = Stichprobengrösse, „prob“ = Wahrscheinlichkeit)

Poissonverteilung:

`dpois(x, lambda)` (x = Argument der Wahrscheinlichkeitsfunktion)
`ppois(q, lambda)` (q = Argument der Verteilungsfunktion)
`qpois(p, lambda)` (p = p-Quantil)
`rpois(n, lambda)` (n = Anzahl zu schaffender Zufallszahlen).

Hypergeometrische Verteilung

`dhyper(x, m, n, k)` (x = Argument der Wahrscheinlichkeitsfunktion, m Anzahl der ausgezeichneten Ergebnisse, n = Anzahl Grundgesamtheit minus m, k = Stichprobengrösse)
`phyper(q, m, n, k)` (q = Argument der Verteilungsfunktion, sonst wie dhyper)
`qhyper(p, m, n, k)` (p für p-Quantil, sonst wie dhyper)
`rhyper(nn, m, n, k)` (nn für Anzahl zu schaffender Zufallszahlen, sonst wie dhyper).
Achtung: in **R** ist `dhyper(3, 50, 150, 20)` dasselbe wie in Excel hypgeomvert(3,20,50,200)

Stetige Uniforme Verteilung

dunif(x,a,b), x = Wert der Zufallsvariable, liefert Wert der Dichtefunktion, a und b sind die Intervallgrenzen
punif(q,a,b), q = Quantil, liefert Wert der Verteilungsfunktion, a und b sind die Intervallgrenzen
qunif(p,a,b), p = Wahrscheinlichkeit, liefert Quantil einer spezifiscshen Wahrscheinlichkeit p, a und b sind die Intervallgrenzen.
runif(n,a,b), liefert n uniform verteilte Zufallszahlen zwischen a und b.

Exponentialverteilung

`dexp(x)` gibt Wert der Dichtefunktion der Exponentialverteilung in x mit Lamda = 1 an. dexp(x, 5) gibt den Wert der Dichtefunktion der Exponentialverteilung mit dem Parameter 5 an.
`pexp(q)` gibt den Wert der Verteilungsfunktion an. Für die Gegenwahrscheinlichkeit kann man lower.tail = F eingeben.
`qexp(p)` gibt das p-Quantil an.
`rexp(n,`λ`)` liefert n exponentialverteilte Zufallszahlen mit Parameter λ.

Normalverteilung

`dnorm(x)` gibt Wert der Dichtefuntion für Standardnormalverteilung in x an. dnorm(x, mean = 5, sd = 2) gibt den Wert der Dichtefunktion der Normalverteilung mit den Parametern 5 und 4 an.
`pnorm(q, mean = 0, sd = 1, lower.tail = TRUE)` gibt den Wert der Verteilungsfunktion an. Für die Gegenwahrscheinlichkeit kann man lower.tail = F eingeben.
`qnorm(p, mean = 0, sd = 1, lower.tail = TRUE)` gibt das p-Quantil an.
`rnorm(n, mean = 0, sd = 1)` liefert n normalverteilte Zufallszahlen.

Chi-quadratverteilung

```
dchisq(x, df)
pchisq(q, df, lower.tail = TRUE)
qchisq(p, df, lower.tail = TRUE)
rchisq(n, df)
```

t-Verteilung

```
dt(x, df)
pt(q, df, lower.tail = TRUE)
qt(p, df, lower.tail = TRUE)
rt(n, df)
```

F-Verteilung

```
df(x, df1, df2)
pf(q, df1, df2, lower.tail = TRUE)
qf(p, df1, df2, lower.tail = TRUE)
rf(n, df1, df2)
```

10 Analysis; Lineare Algebra

Nullstellen von Polynomen: `polyroot(c(a0, a1, a2, a3, ...., an))`. Es müssen die Koeffizienten eingegeben werden (in aufsteigender Reihenfolge). Achtung. Koeffizienten, die Null oder 1 sind, müssen ebenfalls angeben werden. So müsste man für

$$f(x) = 3x^5 - 2x^3 + x^2 - 5x + 5$$

schreiben: `polyroot(c(5, −5, 1, −2, 0, 3))`

Nullstellen von weiteren stetigen Funktionen: Zuerst die Funktion definieren, z.B. „f(x) =ln(1.5*xˆ2-3)“ durch `f=function(x) log(1.5*xˆ2-3)`. Dann diese zeichnen, um Intervalle für Nullstellen zu finden: `curve(f)` (f muss im einzugebenden Intervall definiert sein). Im Beispiel wird eine Fehlermeldung ausgegeben. Deshalb Bereich verändern durch `curve(f,xlim=c(a,b), ylim=c(d,e))`. Im Beispiel etwa `curve(f,xlim=c(1.5,5))`. Dann die Nullstelle berechnen durch `uniroot(f,c(a,b))`, wobei a und b die Grenzen des Intervalls sind, in dem sich eine Nullstelle befindet (im Beispiel: `uniroot(f,c(1.5,5));` im Intervall darf nur eine Nullstelle vorkommen). Es finden Iterationen statt, solange sich durch die Interationen die x-Werte weniger als der Wert verändern, der nach `tol` angegeben ist (Voreinstellung `tol=.Machine$double.epsˆ0.25` (= 0.0001220703)), oder sobald $f(x) = 0$. Gibt man z.B. `tol=10ˆ-10` ein, erhält man eine bessere Näherung und kann bei gewünschter höherer Genauigkeit die Hochzahl erhöhen.
Funktionswerte x der definierten Funktion können durch `f(x)` berechnet werden.

Möchte man eine Funktionkurve g in den gleichen Plot malen wie eine bereits gezeichnete, so schreibt man `curve(g, add=T)`. Verschiedene Farben für die Kurven kann man wählen, indem man für eine Kurve eingibt `col = „blue"`, „green“, oder eine andere Farbe „red“, etc. Man kann für die Farbe auch Zahlen eingeben.

Exponential- und Logarithmusfunktionen: `exp(5)` ergibt e^5
`log(5)` ergibt ln(5).
`log(5,3)` ergibt $\log_3(5)$.
`log10(5)` (=`log(5,10)`) ergibt $\log_{10}(5)$.
`log2(5)` (=`log(5,2)`) ergibt $\log_2(5)$.

Matrizen: Eine Matrix gibt man ein, indem man die Koeffizienten spaltenweise in einen Vektor gibt. Dann gibt man die Anzahl der Spalten an. Für die Matrix

$$\begin{pmatrix} 5 & 9 & 9 \\ 6 & 10 & 10 \\ 7 & 7 & 11 \\ 8 & 18 & 5 \end{pmatrix}$$

würde man eingeben: `m=matrix(c(5,6,7,8,9,10,7,18,9,10,11,5),ncol=3).` Möchte man diese Matrix mit der Matrix

$$\begin{pmatrix} 4 & 8 & 9 & 10 \\ 1 & 11 & 21 & 22 \\ 3 & 3 & 33 & 4 \end{pmatrix}$$

multiplizieren, würde man `m2=matrix(c(4,1,3,8,11,3,9,21,33,10,22,4),ncol=4)` schreiben und dann `m%*%m2`, und enter. Man erhält das Produkt der Matrizen.

Inverse einer Matrix: mit `solve(m)`.

Determinante: `det(m)`.

n×n-Einheitsmatrix: `diag(n)`

Diagonalmatrix mit den Komponenten des Vektors p: `diag(p)`

11 Statistische Tests mit R

11.1 Einstichproben z-Test

- `z.test(daten, mu, stabw, alternative = „two.sided", conf.level = 0.95)` im Paket „Varia“ (s. `help(z.test)`). Für `daten` setzt man eine metrisch skalierte Variable, für `mu` den Mittelwert der Grundgesamtheit, für `stabw` die Standardabweichung der Grundgesamtheit.

- Beispiele

  - `z.test(brems$Bremswege,5,3)` (zweiseitiger Test)
  - `z.test(brems$Bremswege,5,3,„l")` (linksseitiger Test)
  - `z.test(brems$Bremswege,5,3,„g")` (rechtsseitiger Test)
  - `z.test(brems$Bremswege,5,3,„l",0.99)` (linksseitiger Test; Vertrauensintervall 0.99)
  - `z.test(brems$Bremswege,5,2,conf.level =0.99)`; (zweiseitiger Test, Vertrauensintervall = 0.99)

11.2 Einstichproben-t-Test

- `t.test()`

- Syntax: `help(t.test)`

- Beispiele:

  - `t.test(brems$Bremswege,mu=55)` (zweiseitiger Test, 55 Mittelwert der Grundgesamtheit)
  - `t.test(brems$Bremswege,mu=55,alternative="l")` (linksseitiger Test)
  - `t.test(brems$Bremswege,mu=5,alternative="g",conf.level=0.99)` (rechtsseitiger Test mit 0.99-Vertrauensintervall. Alter Mittelwert 5).

11.3 Einstichproben Wilcoxon-Test

- `wilcox.test()`

- Syntax: `help(wilcox.test)`

- Die Rangsumme wird unter V= ausgegeben. Es wird bei der asymptotischen Berechnung des p-Wertes eine Kontinuitätskorrektur berechnet, wodurch sich die Ergebnisse oft von den mit Excel berechneten ein wenig unterscheiden. Es gibt auch sogenannte exakte Tests. Das entsprechende R-Paket „exactRankTests“ wird nicht mehr weiterentwickelt. Es wird auf das R-Paket „coin“ verwiesen, das mit neueren Methoden arbeitet. Nach der Installation des Pakete wird dieses mit library(coin) eingebunden. Man kann dann rechnen:
  `wilcoxsign_test(brems$Bremswege~rep(55,length(brems$Bremswege)),distribution="exact")`

- Beispiele:

  - `wilcox.test(brems$Bremswege,mu=55)` (zweiseitiger Test, 55 Mittelwert der Grundgesamtheit)

11.4 Einstichproben Vorzeichen-Test (Binomialtest)

- `vorzeichen.test()` im R-Paket „varia“.
- `vorzeichen.test(daten, AlterMedian, alternative = „two.sided")`
- Bemerkung: Der Vorzeichentest kann auch mit dem R-Test „`binom.test`“ druchgeführt werden: Dazu wird zuerst z.B. „`minus=V1<AlterMedian`“ gerechnet. Dann `binom.test(table(minus))`
- Beispiele:
  - `vorzeichen.test(brems$Bremswege,55)` (zweiseitiger Test; alter Median 55)
  - `vorzeichenTest(brems$Bremswege,55,"l")` (für den linksseitigen Test)
  - `binom.test(table(brems$Bremswege<55))`

11.5 Einstichprobentest auf abweichende Varianz

- `vartest()` im R-Paket „varia“
- Syntax: `vartest(daten, varianz, alternative = „two.sided", conf.level = 0.95)`
- Beispiele
  - `vartest(brems$Bremswege,3)` (zweiseitiger Test)
  - `vartest(brems$Bremswege,3,conf.level=0.98)` (zweiseitiger Test, 98%-Vertrauenintervall)
  - `vartest(brems$Bremswege,3,„l")` (linksseitiger Test)
  - `vartest(brems$Bremswege,3,„g")` (rechtsseitiger Test)

11.6 Einstichproben Anteilstest (zwei Ausprägungen) (Binomialtest)

- `binom.test()`
- Syntax: `help(binom.test)`; es ist `binom.test(x,n,p)` oder `binom(c(x,y),p)` einzugeben. x ist der ausgezeichnete Wert, n = x+y. p ist die Wahrscheinlichkeit der Binomialverteilung unter der Nullhypothese.
- Beispiele:
  - zuerst 1X2-Tabelle berechnen: `tab=table(geschlecht$geschlecht)` und dann: `binom.test(tab,p=0.3)`
    oder ohne vorgängige Definition von `tab=table(geschlecht$geschlecht)`:
    `binom.test(table(geschlecht$geschlecht),p=0.3)`
  - rechtsseitiger Test: `binom.test(tab,p=0.3,"g")`
  - linksseitiger Test mit 0.98-Vertrauensintervall:
    `binom.test(tab,p=0.3,alternative="l",conf.level=0.98)`

11.7 Einstichproben Anteilstest (mehr als 2 Ausprägungen) (Chiquadrat-Test)

- `chisq.test()`
- Syntax: `help(chisq.test)`

- obligatorisch: `chisq.test(Häufigkeitsverteilung)`
  Die Häufigkeitsverteilung wird auf die Daten durch „`table(V1)`“ gebildet.
  Soll nicht auf uniforme Verteilung getestet werden, ist ein Anteilsvektor `p=c(`p_1`,`p_2`,...,`p_n`)` einzugegeben, wobei p_i die Anteile sind.

- fakultativ: Wird die Bedingung für χ^2-Tests „Erwartete Häufigkeiten dürfen höchsten sind 20% der Fälle kleiner als 5 sein“ verletzt, so kann man mit einem Computer Experiment (Monte Carlo Verfahren) einen p-Wert berechnen: `simulate.p.value=T; B=10000` (B gibt die Anzahl Durchgänge an).

- **R** bietet die sogenannten Pearson Residuen an: (observed - expected) / sqrt(expected) (sqrt = square root = zweite Wurzel aus). Dazu muss man die Resultate des Testes speichern, indem an z.B. schreibt
  `test=chisq.test(table(V1))`
  Ausgabe der Residuen mit: `test$residuals`
  Zudem werden die Standardisierten Pearson-Residuen geliefert (test$stdres). Letztere sind näherungsweise standardnormalverteilt.
  Will man die unstandardisierten Residuen, schreibt man: `test$observed-test$expected`
  Will man alles in einer Tabelle haben, kann man schreiben
  `ueber=cbind(test$expected, test$observed, test$residuals, test$expected-test$observed,test$stdres)`
  `colnames(ueber)=c(ërw. Haeufig.","Beob. Haeufig.", "Pearson Resid", "Residuen", "Stand. Resid"); ueber`

- Beispiele:

  - Testen auf uniforme Verteilung: zuerst Tabelle berechnen: `tab=table(Verkauf)` und dann: `chisq.test(tab)`
    oder ohne vorgängige Definition von `tab=table(beidaten)`:
    `chisq.test(table(Verkauf))`
  - Testen auf nicht-uniforme Verteilung mit vier Ausprägungen:
    `chisq.test(table(Verkauf),p=c(0.15,0.15,0.3,0.4))`
  - Monte Carlo Verfahren mit 1000 Durchgängen:
    `chisq.test(tab,p=c(0.15,0.15,0.3,0.4),simulate.p.value=T,B=1000)`

11.8 Montecarlo-Verfahren

- Bei bekannter Verteiltung der Zufallsvariablen unter der Nullhypothese. Es werden z.B. 1000 Datensätze der Grösse der Stichprobe erstellt (im Beispiel exponentialverteilte Daten mit $\lambda = 1$ und $n = 50$); im Beispiel experimentell zu schätzende Verteilung der Statistik „Varianz“ bei exponentialverteilten Daten):
  `vek=rep(NA,1000)` (erstellt Vektor für die Aufnahme der Varianzen der 1000 Stichproben).
  for (i in (1:1000)) vek[i]=var(rexp(50,1)) #rexp(50,1) erstellt 50 exponentiell verteilte Zufallszahlen (Lamda = 1). Von diesen 50 Zahlen wird die Varianz berechnet und im Vektor vek abgespeichert. Man erhält 1000 Varianzen von 1000 Datensätzen.
  Berechnung kritische Grenzen:
  `quantile(vek,0.05)` bei einem linksseitigen Test;
  `quantile(vek,0.95)` bei einem rechtsseitigen Test
  `quantile(vek,c(0.025,0.975))` bei einem zweiseitigen Test (jeweils $\alpha = 0.05$)

  Berechnung näherungsweiser $p-$Wert für den linkseitigen Test (t ist der Testwert):
  `v1=vek[vek<=t]`
  p1=`length(v1)/length(vek)`
  Für einen rechtsseitigen Test: `v1=vek[vek>=t]`

p2=`length(v1)/length(vek)`
Für den zweiseitigen Test: `2*min(p1,p2)`.
Fürs Beispiel im Skript Testwert $t = 0.7$ (man ersetzt also in den obigen Formeln t durch 0.7).

- Bootstrap: Sei eine Stichprobe `dat` der Grösse n gegeben und es sollen die $\frac{\alpha}{2}$- und $1 - \frac{a}{2}$-Quantile der Statistik T bestimmt werden. Kennen wir die Verteilung der Statistik nicht oder haben wir für die Geltung einer asymptotischen Verteilung zu wenig Daten, kann man ein Bootstrap-Montecarlo-Verfahren anwenden. Dann werden B Stichproben der Grösse n (= `length(dat)`) aus `dat` gezogen (mit Zurücklegen) und darauf jeweils der Wert der Statistik T für die gezogene Stichprobe berechnet mit
  `vek=rep(NA,B)`
  `for (i in 1:B) vek[i]=T(sample(dat,n,replace=TRUE))`
  `vek` enhält nun B Werte der Statistik T für die B Stichproben aus *dat* (mit Zurücklegen!).
  Darauf berechnen wir `quantile(vek,c(`$\frac{\alpha}{2}, 1 - \frac{a}{2}$`))`.
  In der Literatur wird angegeben,dass $B \geq 200$ zu sein hat.

- Beispiel für Bootstrap: Wir möchten für die Statistik Median die 0.025- und 0.975-Quantile berechnen und haben die (allzu kleine) Stichprobe $dat = c(5, 3, 1.2, 8, 9.7, 6.3, 4)$ zur Verfügung. Da wir die Verteilung der Statistik Median nicht kennen (im Gegensatz zur Statistik Mittelwert, die ja näherungsweise normalverteilt ist - bei so wenig Daten könnten wir wir allerdings auch bei der Statistik Mittelwert nicht auf die asymptotische Geltung der Normalverteilung zählen, ausser die Daten wären normalverteilt), sind wir auf ein Montecarlo-Verfahren angewiesen. Wir schreiben:
  `vek=rep(NA,200)`
  `for (i in 1:200) vek[i]=median(sample(dat,7,replace=TRUE))` #es werden aus dat 7 Daten mit Zurücklegen gezogen, der Median dieser Daten berechnet und dieser im vek-Vektor abgespeichert. Dies wird 200 Male durchgeführt.
  `quantile(vek,c(`$0.025, 0.975$`))` (Berechnet die entsprechenden Quantile der 200 Mediane)
  Liegt nun der Median unter der Nullhypothese links oder rechts der Quantile, so weicht der Median der Stichprobe signifikant vom Median unter der Nullhypothese ab (zweiseitiger Fall, Alpha = 0.05). Liegt der Median unter der Nullhypothese links des 0.05-Quantils, so wird die Nullhypothese beim linksseitigen Testen verworfen, liegt der Median unter der Nullhypothese rechts des 0.95-Quantils, so wird die Nullhypothese beim rechtsseitigen Teste verworfen (Alpha = 0.05).

11.9 Verbundener Zweistichproben t-Test

- `t.test()`
- Syntax: `help(t.test)`
- obligatorisch: `t.test(V1,V2, paired=T)`
- Bemerkung: statt `t.test(V1,V2,paired=T)` kann man auch `t.test(V1-V2,mu=0)` rechnen.
- Beispiele:
  - `t.test(vorher,nachher,paired=T)`
  - `t.test(vorher,nachher,paired=T,alternative="g")`
  - `t.test(vorher, nachher, paired=T,`
    `alternative="two.sided",conf.level=0.99)`

11.10 Verbundener Wilcoxon-ZweistichprobenTest (Rangtest)

- `wilcox.test()`
- Syntax: `help(wilcox.test)`
- obligatorisch: `wilcox.test(V1, V2, paired = T)`
- Bemerkung: Es wird auch ein exakter Test angeboten, der sich vor allem bei kleinen Stichproben aufdrängt - mit `exakt =T`. Vertrauensintervalle durch `conf.int=T`. Statt `wilcox.test(V1, V2, paired = T)` kann auch `wilcox.test(V1-V2)` gerechnet werden. Mit dem R-Paket „coin“ rechnet man wilcoxsign_test(vorher~nachher,distribution=”exact”).
- Beispiele:
  - `wilcox.test(vorher, nachher, paired = T)`
  - `wilcox.test(vorher-nachher, exact=T,conf.int=T,conf.level=0.98)`

11.11 Verbundener Zweistichproben Vorzeichentest (Binomialtest)

- `vorzeichen.test` im R-Paket „varia“
- Syntax: `vorzeichen.test(V2-V1, 0, alternative = „two.sided")`
- Bemerkung: (1) Differenzen, die 0 ergeben sollte man vorgängig aus dem Datensatz löschen (Vorgehen: sei dat der Dataframe-Name. Man löscht Zeilen mit Null-Differenzen aus dem Dataframe mit datneu=dat[V2-V1 !=0,]. (2) Der Vorzeichentest kann auch mit dem R-Befehl „`binom.test`“ berechnet werden: `binom.test(table(V1-V2<0))`. Durch den Befehl `V1-V2<0` wird ein Vektor mit FALSE und TRUE erstellt (TRUE genau dann, wenn `V1-V2<0`, sonst steht FALSE. Dann wird eine Tabelle erstellt (Anzahl Falsche und Anzahl Wahre. Die Falschen stehen am Anfang, werden vom Befehl `binom.test` also als die Ausgezeichneten behandelt. Wenn nur negative oder nur positive Vorzeichen auftauchen, ergibt sich eine Fehlermeldung. Dies kann man vorgängig überprüfen (`table((V1-V2)<0)`.
- Beispiele:
  - `vorzeichen.test(vorher-nachher,0)`
  - `vorzeichen.test(vorher-nachher,0,„g")` (für den rechtsseitigen Test)

11.12 Verbundener ZweistichprobenTest bei zwei Ausprägungen (McNemar-Test)

- `mcnemar.test()`
- Syntax: `help(mcnemar.test)`
- obligatorisch: `mcnemar.test(V1, V2)` oder `mcnemar.test(table(dat$V1, dat$V2))`
- `correct = TRUE` (rechnet bei asymptotischer Berechnung mit Kontinuitätskorrektion)
- Bemerkung: Da der McNemar-Test von der Anlage her ein Binomialtest ist, ist es besser, diesen mit binom.test zu berechnen, da dann mit der Binomialverteilung gerechnet wird und nicht mit einer Chi-Quadrat-Näherung. Man kann rechnen: `dat=V1-V2`; `dat=dat[dat!=0]`; `binom.test(table(dat))`. Mit `dat` kann man sehen, was als „success“ behandelt wird. (wir berechnen ja V1-V2. Gleiche Ausprägungen ergeben 0 - wenn es solche gibt, werden sie mit `dat[dat!=0]` weggelassen, ungleiche ergeben eine negative Zahl -x oder die positive Zahl x (V1,V2 müssen gleich kodiert sein).

Alternative: Man berechnet die Kreuztabelle von V1 und V2 und übernimmt aus dieser „von Hand“ die Häufigkeiten der Nebendiagonale. Diese Häufigkeiten seien a und b. Damit kann man rechnen: `binom.test(c(a,b))`. Statt die Häufigkeiten von Hand aus der Tabelle zu lesen, dann man auch die Tabelle z.B. unter `tab` abspeichern und dann `c(tab[2,1],tab[1,2])` verwenden. Der erste Eintrag im Vektor c(a,b) wird als die Anzahl der Ausgezeichneten betrachtet (= Testwert). Bei einem rechtseitigen Test wird also $P(X \geq a)$ berechnet, bei einem linksseitigen Test $P(X \leq a)$, bei einem zweiseitigen Test $2\min\{P(X \leq a), P(X \geq a)\}$).

- Beispiele:
  - `mcnemar.test(vorher, nachher)` (zweiseitig)
  - `mcnemar.test(table(vorher, nachher),alternative="l")` (linksseitig)
  - `dat=vorher-nachher; dat=dat[dat!=0]; binom.test(table(dat))` (zweiseitig)
  - `tab=table(vorher,nachher); binom.test(c(tab[2,1],tab[1,2]),alternative="g")` (rechtsseitig).

11.13 Unverbundener Zweistichprobentest auf unterschiedliche Varianzen

- `var.test()`
- Syntax: `help(var.test)`
- obligatorisch: `var.test(V1~V2)` (V2 ist die Gruppenvariable, die angibt, zu welcher Stichprobe die Daten gehören).
  Graphische Überprüfung der Daten auf Normalverteilung mit: Sei D der Name des Dataframes. Man erhält den ersten Datensatz (Vektor) mit `d1=D$V1[D$V2==1]` (wenn die erste Stichprobe mit 1 kodiert ist). Analog für die Zweite Stichprobe: `d2=D$V1[D$V2==2]`. Auf d1 und d2 kann man direkt ppplotNorm() des varia-Paketes anwenden, da es sich bei d1 und d2 um Vektoren handelt.
  Alternative 1: Man kann auch das Dataframe in zweie neue, getrennte Dataframes aufteilen mit d1=D[D$V2==1,] für die erste Gruppe und d2=D[D$V2==2,] für die zweite Gruppe. Der Zugriff für den Befehl ppplotNorm() erfolgt dann durch d1$V1 und d2$V1.
  Alternative 2: d=split(D,V2). Dadurch wird eine Liste mit zwei Dataframes geschaffen. Auf die Variablen der Listenelemente hat man Zugriff durch d$'1'$V1 und d$'2'$V1 (wobei für V1 die entsprechenden Variablen stehen müssen). Auf diese Objekte kann ppplotNorm() angewendet werden.
- Beispiele:
  - `var.test(Produktion~Stichprobe)`
  - `var.test(Produktion~Stichprobe,alternative="g")`

11.14 Unverbundener Zweistichprobentest auf unterschiedliche Mittelwerte (t-Test)

- `t.test()`
- Syntax: `help(t.test)`
- obligatorisch: `t.test(V1~V2,var.equal=T)` (V2 ist die Gruppenvariable, die angibt, zu welcher Stichprobe die Daten gehören).
- Beispiele:
  - `t.test(zweivar$Produktion~zweivar$Stichprobe,var.equal=T)`

  - `t.test(zweivar$Produktion~zweivar$Stichprobe,var.equal=T, alternative="l", conf.level=0.99)`

11.15 Unverbundener Zweistichprobentest auf unterschiedliche Mittelwerte (Welch-t-Test)

- `t.test()`
- Syntax: `help(t.test)`
- obligatorisch: `t.test(V1~V2)` (V2 ist die Gruppenvariable, die angibt, zu welcher Stichprobe die Daten gehören).
- Beispiele:
  - `t.test(zweivar$Produktion~zweivar$Stichprobe)`
  - `t.test(zweivar$Produktion~zweivar$Stichprobe, alternative="l", conf.level=0.99)`

11.16 Rangtest von Mann-Witney

- `wilcox.test()`
- Syntax: `help(wilcox.test)`
- obligatorisch: `wilcox.test(V1~V2)` (V2 ist die Gruppenvariable, die angibt, zu welcher Stichprobe die Daten gehören).
- `exact = NULL` (bei TRUE wird exakter Test gerechnet), `correct = TRUE` (bei asymptotischer Berechnung wird Kontinutitätskorrektur verwendet). Für den exakten Test man auch das coin-Paket verwenden: z.B.
  `wilcox_test(zwei$Prod~zwei$gruppen, data = asat,distribution = "exact", alternative = "less",conf.int = TRUE)`
  Falls die Gruppenvariable metrisch ist, muss `as.factor(zwei$gruppen)` stehen.
- Beispiele:
  - `wilcox.test(zweivar$Produktion~zweivar$Stichprobe)`
  - `wilcox.test(zweivar$Produktion~zweivar$Stichprobe, alternative="l", conf.level=0.99,exact=T)`

11.17 Zweistichproben-Anteilstest (je zwei Ausprägungen in jeder Stichprobe)

- `AnteilZweistichTest()` im R-Paket „varia“.
- Syntax: `AnteilZweistichTest(daten, stichvar, alternative = "two.sided", conf.level = 0.95)`
- Beispiele:
  - `AnteilZweistichTest(V1,V2,"l")` (linksseitiger Test)
  - `AnteilZweistichTest(V1,V2,"g",conf.level=0.99)` (rechtsseitiger Test)

11.18 Bivariater Einstichprobentest: Chi-Quadrat-Test auf Unabhängigkeit

- `chisq.test()`
- Syntax: `help(chisq.test)`
- obligatorisch: `chisq.test(V1,V2)` oder `chisq.test(table(V1,V2))`. Statt `table(V1,V2)` kann man auch eine Matrix mit der zweidimensionalen Häufigkeitsverteilung eingeben.
- `simulate.p.value =T` berechnet Monte Carlo Experiment für Berechnung des p-Wertes (nützlich, wenn mehr als 20% der erwarteten Häufigkeiten kleiner als 5). Anzahl Durchläufe bei Monte Carlo Experiment durch B=natürliche Zahl (Voreinstellung 2000).
- Beispiele:
  - `chisq.test(V1,V2,simulate.p.value =T,B=10000)`
  - `chisq.test(V1,V2,simulate.p.value =T)`
- Um eine Kreuztabelle zu haben, welche die erwarteten und die faktischen Häufigkeiten samt Testergebnissen ausgibt , kann man das Paket `gmodels` laden, mit dem Befehl `library(gmodels)`. Dann gibt man `CrossTable(V1, V2, expected=T)`ein. Standardmässig wird zusätzlich der Chi-Quadrat-Unabhängigkeitstest ausgegeben. Man kann weitere Tests ausgeben lassen (s. Syntax mit `help(CrossTable)`).

11.19 Bivariater Einstichprobentest: Fisher-Test

- `fisher.test()`
- Syntax:`help(fisher.test)`
- obligatorisch: `fisher.test(V1,V2)` oder Häufigkeitstabelle in Matrixform (table(V1,V2) funktioniert nicht).
- fakultativ: `simulate.p.value =T` berechnet Monte Carlo Experiment für Berechnung des p-Wertes. Anzahl Durchläufe bei Monte Carlo Experiment durch B=natürliche Zahl (Voreinstellung 2000).
- Ausgabe: Im zweiseitigen Fall wird als p-Wert die Summe der Wahrscheinlichkeiten der Tabellen geliefert, die kleiner gleich der Wahrscheinlichkeit der faktischen Tabelle sind. Zudem wird im 2X2-Fall das sogenannte Quotenverhältnis (engl. Odds-Ratio) und deren Vertrauensintervall ausgegeben. Dieses ist das folgende Verhältnis: $\frac{a_{11}a_{22}}{a_{21}a_{12}} = \frac{\frac{a_{11}}{a_{12}}}{\frac{a_{21}}{a_{22}}}$. Es drückt das Verhältnis der Quoten aus, in der ersten Zeile in der Zelle a_{11} zu sein, zur Quote, in der zweiten Zeile in der Zelle a_{21} zu sein. Haben wir eine Kreuztabelle Geschlecht (m,w) X Rauchverhalten (Raucher, Nichtraucher), so wäre das Quotenverhältnis das Verhältnis der Quoten, als Mann Raucher zu sein zur Quote, als Frau Raucherin zu sein. Ist das Quotenverhältnis z.B. 4, so wäre die Quote als Mann Raucher zu sein, 4 mal grösser als als Frau Raucherin zu sein. Die Teststatistik drückt aus, ob das Quotenverhältnis signifikant von 0 verschieden ist oder nicht. Das Quotenverhältnis wird oft in der medizinischen Statistik verwendet, spielt aber auch in der logistischen Regression eine grosse Rolle.
- Beispiel:
  - `fisher.test(V1,V2,simulate.p.value =T,`
    `B=10000,alternative="t",conf.level=0.99)`

11.20 Bivariater Einstichprobentest: Chi-Quadrat-Test auf Anpassung

- wie Einstichproben Anteilstest (mehr als 2 Ausprägungen) (Chiquadrat-Test)
- `chisq.test()`
- Syntax: `help(chisq.test)`
- obligatorisch: `chisq.test(as.vector(table(V1,V2)),p=c(p1,p2,...,pn))`
  Die Häufigkeitsverteilung wird auf die Daten durch „`table(V1,V2)`“ gebildet.
  durch „`as.vector`“ wird die Kreuztabelle in einen Vektor verwandelt. p_i sind die Anteile der Grundgesamtheit. Die relativen Häufigkeiten der Grundgesamtheit sind spaltenweise einzugeben.
- fakultativ: Wird die Bedingung für χ^2-Tests „Erwartete Häufigkeiten dürfen höchsten sind 20% der Fälle kleiner als 5 sein“ verletzt, so kann man mit einem Computer Experiment (Monte Carlo Verfahren) einen p-Wert berechnen: `simulate.p.value=T; B=10000` (B gibt die Anzahl Durchgänge an).
- **R** bietet die sogenannten Pearson Residuen an: (observed - expected) / sqrt(expected) (sqrt = square root = zweite Wurzel aus). Dazu muss man die Resultate des Testes speichern, indem an z.B. schreibt
  `test=chisq.test(table(V1,V2),p=c(p1,p2,...,pn))`
  Ausgabe der Residuen mit: `test$residuals`
  Will man die unstandardisierten Residuen, schreibt man: `test$observed-test$expected`
  Die standardisierten Residuen erhält man mit: test$stdres
- Beispiele:
  - Testen bei 2X2-Tabelle:
    `chisq.test(table(Verkauf1,Verkauf2),p=c(0.15,0.15,0.3,0.4))`
  - Monte Carlo Verfahren mit 1000 Durchgängen:
    `chisq.test(table(Verkauf1,Verkauf2),p=c(0.15,0.15,0.3,0.4), simulate.p.value=T,B=1000)`

11.21 Bivariater Einstichprobentest: Gamma

- `gamma.test()` im R-Paket „varia“
- Syntax:`gamma.test(v1, v2 = NULL, alternative = „two.sided", conf.level = 0.95, MonteCarlo = FALSE, Stichprobe = NULL, Runden = 1000)`
- obligatorisch: `gamma.test(V1,V2)`
- Mit `bootstrap=T` wird ein Test mit Hilfe eines Zufallsexperimentes berechnet. „Stichprobe“ sollte nur anders spezifiziert werden, wenn man überprüfen will, wie sich die Quantile bei Stichproben unterschiedlicher Grösse verhalten. Die Voreinstellung ist die Grösse der empirischen Stichprobe.
- Beispiel:
  - `gamma.test(V1,V2, bootstrap=T,Runden=1000)`

11.22 Bivariater Einstichprobentest: Einfaktorielle ANOVA

- `aov()`
- Syntax: `help(aov)`
- obligatorisch: `aov(Einkommen~Kaeufergruppe,data=ErfundDat)` oder `aov(ErfundDat$Einkommen~ErfundDat$Kaeufergruppe)`. Die Gruppenvariable muss als Faktor-Variable vorliegen (allenfalls konvertieren durch `ErfundDat$KaeufergruppeF=factor(ErfundDat$Kaeufergruppe)` oder direkt in Befehl `factor (ErfundDat$ Kaeufergruppe)` verwenden.
- Überprüfung der Voraussetzungen:
  - Varianzgleichheit kann mit graphisch mit `boxplot(V1~V2)` oder mit Streudiagramm: `plot(as.numeric(V2),V1)` (oder letzteres noch besser mit `stripchart(V1~V2,vert=T))` oder rechnerisch mit Mehrgruppenvarianztest überprüft werden, z.B. mit dem Levenetest (Kruskal-Wallis auf die Residuen (s. unten) oder mit `bartlett.test(V1~V2)`; Berechnung Residuen mit `b=aov(V1~V2)` und `b$residuals`
  - Normalverteilung der Residuen: `ppplotNorm(b$residuals)` für `b=aov(V1~V2)`
  - Speichert man das Ergebnis von `aov()` ab, z.B. mit `b=aov()`, so findet man mit `summary(b)` die klassische ANOVA-Tabelle.
- Beispiel: `b=aov(Einkommen~Kaeufergruppe,data=ErfundDat)` `summary(b).`
- Um die Unterschiede zwischen den Mitteln zu begutachten, empfiehlt es sich, die Gruppenmittel zu berechnen. Dies erfolgt z.B. durch `mean(ErfundDat$ Einkommen [ErfundDat$ Kaeufergruppe==1])` für die Gruppe, die mit 1 kodiert ist oder `mean(ErfundDat$ Einkommen [ErfundDat$ Kaeufergruppe == „reich"])` für die Gruppe, die mit "reich"kodiert ist. Einfacher geht es mit `tapply (metrische_Variable, Faktorvariable,mean)` Ein Vergleich ist auch mittels `summary(lm(Einkommen~Kaeufergruppe,data=ErfundDat))` möglich. R betrachtet dabei den Mittelwert der erste Gruppe als Referenz (= intercept) und vergleicht die Abweichungen der Mittelwerte der anderen Gruppen mit diesem Mittelwert. Ein Test auf paarweise Unterschiede der Gruppen liefert `pairwise.t.test()`. Im Test wird eine Korrektur vorgenommen, weil mehrfach getestet wird und dadurch die Fehlerwahrscheinlichkeit erster Art steigt. Dabei gibt es verschiedene Mölgichkeiten (s. Hilfe zum Befehl). Die Korrektur kann man durch `p.adj=„none"` ausschalten.

11.23 Bivariater Einstichprobentest: Kruskal-Wallis

- `kruskal.test()`
- Syntax: `help(kruskal.test)`
- obligatorisch: `kruskal.test(V1,V2)`, V2 ist Gruppenvariable, V1 metrisch oder ordinal skalierte Variable; Gruppenvariable kann Faktor sein oder nicht, Faktor ist eine Gruppenvariable, die im System als nomalskalierte Variable gekennzeichnet ist, Transformation durch `dateiname$V2=as.factor(dateiname$V2))`.
- Bemerkung: die Ausgabe zum Kruskal-Wallis-Test ist leider etwas rudimentär (Rangsummenmittel fehlen). Im Pakte „varia" findet man die Funktion `summary.kruskal(V1,V2)`, wobei V2 die Gruppenvariable ist. Es werden die Rangmittel und die Anzahl der Daten pro Gruppe ausgegeben.
- Beispiel:
  - `kruskal.test(v1,v2)`

11.24 Bivariater Einstichprobentest: Korrelationen

- `cor.test()`
- Syntax: `help(cor.test)`
- obligatorisch: `cor.test(v1,v2)`, es wird der Test für die Pearson-Korrelation geliefert
- fakulativ: `cor.test(x, y, alternative = c(„two.sided", „less", „greater"),method = c(„pearson", „kendall", „spearman"), conf.level = 0.95, ...)`. Mit den Methoden kann die zu berechnende Korrelationsart festgelegt werden.
- Beispiel:
  - `cor.test(v1,v2,alternative=„t",method=„spearman", con.level=0.98)`

11.25 Trivariater Einstichprobentest: Partielle Korrelation

- `PartielKor()` im R-Paket „varia“
- Syntax: `PartielKor(v1, v2, kontroll, alternative = „two.sided", conf.level = 0.95, complete = F)`
- obligatorisch: `PartielKor=function(v1,v2, kontroll)`. Kontroll ist die Kontrollvariable.
- fakultativ: bei `complete=T` werden Zeilen mit fehlenden Daten nicht weggelassen.
- Beispiel:
  - `PartielKor(v1,v2,v3,alternative=„l")`

11.26 Bivariater Einstichprobentest: Regression

- `lm()`
- Syntax: `help(lm)`
- obligatorisch: `lm(V1~V2)` (V1 ist die Zielvariable, V2 ist die unabhängige Variable. Testen der Koeffizienten und ANOVA: `summary(lm(V1~V2))` oder `summary(m)` für `m=lm(V1~V2)`
  0.98-Vertrauensintervall durch `confint(m,level=0.98)`
  Um das Dataframe in den Variablen nicht erwähnen zu müssen, kann man schreiben: `lm(V1~V2,data="dataframename")`. Diese Version ist auch zu empfehlen, da manche Befehle auf das lm-Objekt nur dann funktionieren (s. unten).
- Prüfen der Voraussetzungen
  - zweidimensionales Streudiagramm durch `plot(V2,V1)`
  - Hinzuzeichnen der Regressiongeraden durch `abline(m)`
  - Normalverteilung der Residuen: `ppplotNorm(m$residuals)` (nach Einbindung des Varia-Paketes)
  - Varianzgleichheit via Plot der Residuen: `plot(V2,m$residuals));` Die 0-Linie kann durch `abline(0,0)` hinzugefügt werden.
- Schätzungen für beliebige x durch `c(1,x)%*%m$coefficients;` für mehrere x, z.B. 3 `v=matrix(c(1,1,1,x1,x2,x3),ncol=2)` und `v%*%m$coefficients`
  (Etwas kompliziert durch `predict(m, dataframe)`, wobei dataframe ein Dataframe von Daten ist, bezüglich derer die Voraussage gemacht werden soll. Im Dataframe sind die Namen der Variablen ohne Anführung einzugeben, also z.B. `voraus=data.frame(groesse=c(5,6,9))`.

Es sind dieselben Variablen wie im Modell zu verwenden, wobei im Modell nicht mit $ zusammengesetzte Namen verwendet werden dürfen. Man muss mit `m=lm(V1~V2,data="dataframename")` arbeiten. Es werden die Voraussagen für die Zahlen 5, 6 und 9 geliefert. 95%-Vertrauensintervalle für Voraussagen werden durch Hinzufügen von interval=„confidence“ im predict-Befehl geliefert).

- Zeichnen des 0.95-Vertrauensbandes um die Regressiongerade: für
`m=lm(V1~V2,data="dataframename")`
`pre=predict.lm(m,interval = "confidence", level = 0.95,type="response")` #erzeugt Matrix (nicht Dataframe); Voraussagen (Schätzungen) für alle x der Daten.
`pre=cbind(pre,V2=dataframe$V2)` #unabhaengige Variable zu Matrix hinzufuegen
`pre=pre[order(pre[,4]), ]` #nach unabhaengiger Variable (= 4. Spalte) ordnen
`plot(V1~V2)`
`abline(m)`
`lines(pre[,4],pre[,2],col="red",lty=2)` #zeichnen, Punkte, die durch 2. und 4. Spalte gegeben sind
`lines(pre[,4],pre[,3],col="red",lty=2)`#zeichnen, Punkte, die durch 3. und 4. Spalte gegeben sind
- ANOVA-Tabelle durch `anova.lm(m)`

11.27 Multivariater Einstichprobentest: Multiple Regression

- `lm()`
- Syntax: `help(lm)`
- obligatorisch: `lm(V1~V2+V3+V4+...+Vn)` (V1 ist die Zielvariable, Vi mit $i > 1$ sind die unabhängigen Variablen. Am besten speichert man das Resultat unter einem Namen ab: `bei=lm(V1~V2+V3+V4+...+Vn)` .
Testen der Koeffizienten und ANOVA: `summary(bei)`
Vertrauensintervalle durch `confint(bei)`
- Prüfen der Voraussetzungen
  - Matrix von Streudiagrammen: `pairs(dateiname[c(„V1",„V2",„V3",...,„Vn")])`
  - Normalverteilung der Residuen: `ppplotNorm(residuals(bei))`
  - Varianzgleichheit via Plot der Residuen gegen die erwarteten Werte:
    `plot(predict(bei),residuals(bei))`
    Manches davon und weiteres kann auch durch den Befehl
    `par(mfrow=c(2,2))`
    `plot(bei,cook.levels=4/nrow(dateiname))`
    gezeichnet werden: Die Graphik Residuals vs Leverage trägt die standardisierten Residuen gegen das Gewicht der Daten in der Berechnung der Koeffizienten ab. Man kann so sehen, ob gewisse Daten die Resultate verfälschen können. Ungünstig sind Daten ausserhalb der eingezeichneten gestrichelten Linien.
- Voraussage für bei einem Modell mit 4 unabhängigen Variablen, z.B. für $(5, 6, 7, 8)$:
c(1,5,6,7,8)%*%bei$coefficients
(oder komplizierter durch `predict(bei, dataframe)`, wobei `dataframe` ein Dataframe von Daten ist, bezüglich derer die Voraussage gemacht werden soll. Im Dataframe sind die Namen der Variablen ohne Anführung einzugeben, also z.B.
`voraus=data.frame(groesse=c(5,6,9),Geschlecht=(1,1,1),`
`Frequenz=(50,60,65))`. Es werden die Voraussagen für die Zahlen Tripel (5,1,50), (6,1,65) und (9, 1, 65) gemacht. Die Variablennamen sind ohne den Dataframenamen einzugeben. Das lm-Objekt muss mit dem Unterbefehl "data=ërzeugt werden (s. oben unter Bivariater Einstichprobentest: Regression).

12 Stichprobentheorie

Dazu gibt es ein **R**-Paket, das im Internet zu finden ist. Das Paket kann geladen werden, indem man **R** öffnet und dann auf „Paket“, „Installiere Pakete“ geht. Ist man on-line, erscheint ein Fenster, das „CRAN mirror“ heisst. Dort wählt man einen Server aus, am besten „Switzerland“. Es erscheint eine Liste mit allen im Augenblick offiziell zur Verfügung stehenden Paketen (das Paket „varia“ finden Sie dort nicht, da für den schulischen Gebrauch in Siders bestimmt!). Paket „survey“ suchen und anklicken. Das Paket wird installiert. Nach der Installation muss man das Paket noch für die konkrete **R**-Session einbinden mit dem Befehl `library(survey).`

Einfache Zufallstichprobe: Daten in ein Dataframe `X` einlesen. Eine `Variable X$groesse=N` bilden, wobei `N` die Grösse der Grundgesamtheit ist. Ein sogenanntes Survey-Design-Objekt bilden und benennen z.B. durch
`des=svydesign(id=~1,fpc=~groesse,data=X)`
(durch `id=~1` wird angegeben, dass keine Klumpen (cluster) vorliegen; durch `fpc=~groesse` wird angegeben, dass die für die Berücksichtigung der Endlichkeitskorrektur fpc (= finite population correction) nötigen Grössen in der Variable „groesse“ der unter „data“ angegeben Datei `X` gefunden werden. Sei `V1` die interessierende Variable. Dann berechnet man geschätztes Mittel und Total (samt entsprechenden Standardfehlern) bezüglich dieser Variable mit
`svymean(~V1,des)`
`svytotal(~V1,des).`
Will man Vertrauensintervall haben, so speichert man z.B. `a=svymean(~V1,des)` und rechnet 95%-Vertrauensintervalle durch `confint(a)`, 98%-Vertrauensintervalle durch
`confint(a,level=0.98)`

Anteile werden mit `svymean` berechnet, wobei die Variable als Faktor (mit `dataframename$alteVariable= as.factor(dataframename$alteVariable)`) zu definieren ist. Will man die alte Variable belassen und eine neue Faktorvariable schaffen, so kann man `dataframename$neueVariable= as.factor(dataframename$alteVariable)`) schreiben.

Geschichtete Zufallsstichproben: Die Schichtzugehörigkeit ist durch eine Variable anzugeben. Um die Endlichkeitskorrektur zu berechnen muss für jedes Datum die Grösse N_h der Schicht h in Grundgesamtheit angegeben werden. Dies wird durch eine Variable „groesse“ (hier so genannt, Name beliebig) gemacht. Für die verschiedenen Ausprägungen x_h der Schichtvariable wird gerechnet:
`X$groesse[X$schichtvariable==`x_h`]=`N_h.
Das Survey-Design-Objekt wird nun wie folgt definiert (vor dem ersten Identitätszeichen steht der beliebig gewählte Name des Survey-Design-Objektes):

`des1=svydesign(id=~1,strata= ~schichtvariable,fpc=~groesse,data=X)`

`strata` gibt die Schichtvariable an.

Vom Verfasser des Pakets wurde für dessen Verwendung ein Buch herausgegeben (?, ?).

13 Umkodieren und Berechnung von Variablen

Umkodieren kann man die mit y kodierte Ausprägung der Variable V1 des Dataframes X in die Kodierung z durch `X$V1[X$V1==y]=z`. Dabei ist zu beachten, dass man in der richtigen Reihenfolge rekodiert, wenn `z` als Code in der Variable schon vorkommt! Wenn man die Codierung 1 und 2 in 0 und 1 verwandeln will, muss man also zuerst 1 in 0 verwandeln und dann erst 2 in 1. In diesem einfachen Fall kann man aber einfach auch von allen Werten der Variable 1 abzählen durch `X$V1=X$V1-1.`

Will man die alte Variable belassen und eine neue rekodierte Variable erzeugen, so gibt man `X$V2[X$V1==y]=z` ein. Durch den Befehl werden in der neuen Variable für Zellen, denen nichts zugeordnet wird, NAs erzeugt.

Das Beispiel `X$V1=X$V1-1` führt über zur Berechnung von Variablen, die in **R** sehr einfach ist. Die Spalten in Dataframes sind Vektoren und man kann die von **R** bereitsgestellten Vektoroperationen auf sie ausführen. So muss man bei der Addition von Vektoren diese nicht zeilenweise

addieren, es genügt `X$sum=X$V1+X$V2` zu schreiben, wodurch eine Variable `sum` zum Dataframe hinzugesetzt wird, welche die Summe der Werte der Variablen V1 und V2 enthält. Die Multiplikation eines Vektors mit einer reellen Zahl a erfolgt durch `X$V1a=`$a*$`X$V1,` wodurch eine Variable V1a dem Dataframe zugefügt wird, welche das Produkt von a und den Zellen des Vektors $V1$ enthält.

14 Programmieren mit R

Ein Programm besteht aus einer Folge von Befehlen. Diese Befehle müssen in der Sprache einer Programmiersprache verfasst sein. Je nach Programmiersprache sehen die Befehle etwas anders aus. Die Grundideen sind jedoch in allen Programmiersprachen dieselben. In modernen Programmiersprachen liegen jeweils bestimmte Befehle schon vor, die für die Programmierung anderer Befehle verwendet werden können. Befehle werden auf Objekte ausgeführt, die zu verschiedenen Typen gehören. In R gibt es als Objekttypen etwa Listen, Tabellen, Vektoren, Matrizen, Dataframes, reelle Zahlen, ganze Zahlen, die Wahrheitswerte „wahr“ und „falsch“ sowie Funktionen. Man kann auch selber Objekttypen definieren. In der Folge geht es darum, ein paar Grundideen des Programmierens mit Hilfe von R darzustellen. Befehle sind in **R** sogenannte Funktionen, die man wiederum in andere Funktionen anwenden kann. Man kann selber geschriebene Funktionen in eine Bibliothek ablegen und immer wieder verwenden. Am besten schreibt man Programme in **R** in einem Skript. Skripten sind im Grunde Text-Dateien, wobei diese mit der Endung *.r abgespeichert werden. Man kann Skripten abspeichern und später wieder holen. Nimmt man in **R** beruflich statistische Auswertungen vor, so sollte man immer mit Skripten arbeiten, da man damit die Arbeitsschritte dokumentieren und damit später wieder rekonstruieren kann.

Beispiel 2. *Wir möchten einen Befehl erstellen, der für ein Kapital $K0$ das Kapital Kn berechnet (Zinseszins). Als Eingabe sind $K0$, der Zinssatz zs und die Anzahl Jahre n einzugeben. Die Ausgabe soll Kn sein. Das folgende Miniprogramm leistet das Gewünschte:*

```
zinszins=function(K0,zs,n) {Kn=K0*(1+zs)^n
Kn}
# Kn gibt an, dass am Schluss Kn ausgegegeben werden soll.
```

Texte nach dem Zeichen # werden von R überlesen. Man gibt also nach # Beschreibungen des gemachten an. Es ist wichtig, Programme detailliert zu kommentieren.

Wir beleuchten das obige Mini-Programm und lassen es leer laufen (durch ctrl + r). Dann können wir im Workspace z.B. die folgende Berechnung durchführen:
`zinszins(5000,0.05,10).`
Wir drücken o.k. und erhalten 8144.473.
*Statt neue Befehle auf neue Zeilen zu setzen, kann man sie in **R** durch Strichpunkt voneinander abgrenzen. man würde fürs Beispiel erhalten:*
`zinszins=function(K0,zs,n) {Kn=K0*(1+zs)^n; Kn}`

Übung 3. *Schreiben Sie entsprechende Miniprogramm für den Endwert einer Rente (Eingabe: Rente, Zinssatz, Anzahl Jahre; Ausgabe Endwert der Rente), die Berechnung der Annuität (Eingabe: Anfangsschuld, Anzahl Jahre, Zinssatz, Ausgabe: Annuität)*

14.1 if (){}, &, | (wenn, dann; und; oder)

Für if() wird in die Klammer die Bedingung gesetzt, nach der Klammer folgt in geschweifter Klammer der Befehle, der auszuführen ist, wenn die Bedingung erfüllt wird.

Beispiel 4. `x=5; if (x > 5) {x = 7} ;x`
führt zur Ausgabe 5,
`x=10; if (x > 5) {x = 7} ;x`
führt zur Ausgabe 7,

Ist bei der Erfüllung der Bedingung nur ein Befehl auszuführen, können die geschwungenen Klammern weggelassen werden. Man kann auch einen „else“-Befehle hinzufügen (der angibt, was zu tun ist, wenn die Bedingung nicht erfüllt ist).

Beispiel 5. `x=5; if (x > 5) x = 7 else x = 8;x`
führt zur Ausgabe 8
`x=12; if (x > 5) x = 7 else x = 8;x`
führt zur Ausgabe 7

In Klammern können wir „und“ und „oder“ brauchen, um Bedingungen zu formulieren und zwar mit & für „und“ und | für „oder“ (Alt Gr + 7).

Beispiel 6. `x=7; if (x > 5 & x < 10) x = 8 else x = 20;x`
ergibt 8
`x=12; if (x > 5 & x < 10) x = 8 else x = 20;x`
ergibt 20

Beispiel 7. `x=7; if (x < 5 | x > 10) x = 8 else x = 20;x`
ergibt 8
`x=3; if (x < 5 | x > 10) x = 8 else x = 20;x`
ergibt 20

14.2 Kontrollstukturen

for (i in 1:n){} Für i von 1 bis n wird der nachfolgende (in geschwungenen Klammern) Befehl durchgeführt (für 1 und n kann man beliebige, definierte Zahlenobjekte oder Zahlenvariablen einsetzen, wobei die erste kleiner als die zweite sein muss). Man vergleicht diese Kontrollstruktur am besten mit dem Summen- oder Produktzeichen ($\sum$; $\prod$). Auf die geschwungene Klammer kann verzichtet werden, wenn nur ein Befehl auszuführen ist.

Beispiel 8. *Es soll die Summe der ersten 100 natürlichen Zahlen gebildet werden. Mit dem Summenzeichen ist dies* $\sum_{i=1}^{100} i = 5050$.
```
x=1
n=100
for (i in 2:n) {x = x + i}
x
```
1. Zeile: Variablen müssen initialisiert werden, damit sie eine Bedeutung haben. Hier wird x mit 1 initialisiert.
3. Zeile: für i von $i = 2$ *bis n soll jeweils zu x die Zahl i hinzugezählt werden. Man erhält jeweils ein neues x.*
Es wird 5050 *ausgegeben. (statt* $n = 100$ *und dann n zu verwenden, könnte man auch direkt (i in* $2 : 100$*) schreiben. Der Vorteil der obigen Schreibweise bei mehrmaliger Verwendung von* 100 *ist, dass man das n nur einmal verändern muss!*
Eine Funktion, welche die Summe der ersten n Zahlen bildet ist:
`GaussSumme=function(n){x=1;for (i in 2:n) x = x + i;x}`
(Schneller zu berechen wäre $\frac{n(n+1)}{2} = \frac{100 \cdot 101}{2} = 5050$ *oder mit einer R-Funktion:*
GS=function(n){n(n+1)/2}).*

Beispiel 9. *Man bilde die Summe der Zahlen eines Vektors, die grösser als* 5 *und kleiner als* 10 *sind. Wir betrachten dazu den Vektor* $\mathtt{a} = \mathrm{seq}(1, 100, 1)$ *(es handelt sich um die Folge der ersten 100 natürlichen Zahlen ohne die 0):*
```
s=0; n=length(a)
for (i in 1:n) {if (a[i]>5 & a[i]<10) s=s+a[i]}
s
```
1. Zeile: die Summe ist zuerst 0, da noch nichts dazugezählt wurde; length(a) bestimmt die Länge

eines Vektors; length(a[,1]) bestimmt die Anzahl der Zeilen einer Matrix. length(a[1,]) bestimmt die Anzahl der Spalten einer Matrix.
3. Zeile: a[i] ist die $i-te$ Komponente von a.
(Kontrolle: Es sind die Zahlen $6+7+8+9=30$).

Beispiel 10. *Man bilde die Summe der Zahlen eines Vektors, die kleiner als 5 oder grösser als 10 sind. Wir betrachten dazu den obigen Vektor* `a=`seq`(1,100,1)`*:*

```
s=0; n=length(a)
for (i in 1:n) {if (a[i]<5|a[i]>10) s=s+a[i]}
s
```

1. Zeile: die Summe ist zuerst 0, da noch nichts dazugezählt wurde.
(Kontrolle: Es sind die Zahen $1+2+3+4+10+11+...+100$
$\frac{100\cdot 101}{2}-\frac{10\cdot 11}{2}+\frac{4\cdot 5}{2}=5005$)

Beispiel 11. *Es sollten in einem Vektor a die Zahlen summiert werden, die grösser sind als 5. Um ein konkrets a zu haben, schreiben wir* `a=`seq`(1,100,1)`

```
s=0; n=length(a)
for (i in 1:n) {if (a[i]>5) s=s+a[i]}
s
```

Kontrolle für den oben definierten Vektor a: $\frac{100\cdot 101}{2}-\frac{5\cdot 6}{2}=5035$.

Beispiel 12. *Soll z.B. immer, wenn die Zahl im Vektor grösser als 5 ist, diese zur Summe der bisherigen Zahlen, die grösser als 5 waren hinzugezählt werden und bei Zahlen, die kleiner als 5 sind, jeweils 100 hinzugezählt werden, schreiben wir:*

```
s=0; n=length(a)
for (i in 1:n) {if (a[i] > 5) s=s+a[i] else s = s + 100}
s
```

Übung 13. *Schreiben Sie ein Programm, dass die Summe der ersten 150 natürlichen Zahlen bildet (Kontrolle: Diese Summe ist* $\frac{150\cdot 151}{2}=11325$).

Lösung 14. `x=0; n=150; for (i in 1:n) {x=x+i};x`
Die geschwungenen Klammern bedeuten, dass die Befehle in geschwungenen Klammern für i = 1 bis n ausgeführt werden sollen. Im Beispiel wären sie nicht nötig, da nur ein Befehl folgt. Bei mehreren Befehlen sind sie aber nötig.

Repeat{ if () break } Der nach Repeat in der Klammer stehende Befehl wird so lange ausgeführt, bis die Bedingung nach if () vor break erfüllt ist. Wird die Bedingung nie erfüllt, wird der Computer in eine Endlosschlaufe geschickt, die man mir **R** aber durch die escape-Taste abbrechen kann.

Beispiel 15. `x=1;repeat { x=1+x ; if (x > 20) break };x`
x wird auf 1 gesetzt. Dann wird wiederholter Massen zu x 1 hinzugezählt, bis $x>20$ *ist. Ergibt die Zahl 21. Steht* $x=1$ *in der Klammer nach repeat erfolgt eine Endlosschlaufe. Dann wird nämlich jedesmal x auf 1 gesetzt, dann 1 hinzugezählt, dann wieder x auf 1 gesetzt, etc. Die Bedingung* $x>20$ *wird nie erfüllt.*

While(){} Solange die Bedingung in der runden Klammer erfüllt ist, wird der Befehl in der geschweiften Klammer ausgeführt. Der Computer wird in eine Endlosschlaufe geschickt, wenn die While-Bedingung immer erfüllt bleibt.

Beispiel 16. `x=1;while(x<21) { x=1+x };x`
x wird mit 1 initialisiert. Solang $x<21$ *ist, wird zu x jeweils 1 hinzugezählt. Als Resultat ergibt sich 21. Im Beispiel erfolgt dieselbe Berechnung wie im Beispiel für repeat, aber anders formuliert.*

Rekursive Funktionen Rekursive Funktionen brauchen bei der zu definierenden Funktion die Funktion selber. Man versteht sie am Besten in Analogie zu rekursiven Definitionen.

Beispiel 17. *So kann man z.B. die Potenzfunktion a^n (für $a > 0$ und $n > 0$) wie folgt rekursiv definieren. $a^1 := a$; $a^n := a^{n-1} \cdot a$. Wir brauchen also Potenzen, um Potenzen zu definieren - wobei eben rekursiv. In letzter Instanz sind Potenzen mit Hilfe der Multiplikation definiert. Eine entsprechende R-Funktion sieht wie folgt aus:*

```
pot=function(a,n) {
if (n ==1) a
else pot(a,n-1)*a}
```

pot(5,4) ergibt dann $5^4 = 625$.
*Um die „Logik“ durch die Nähe zur obigen Definition besser zu verstehen, kann man sich den (so in **R** nicht funktionierenden) Befehl wie folgt vorstellen:*
pot=function(a,n) {
if (n ==1) pot(a,1)=a
*else pot(a,n)=pot(a,n-1)*a}*
lässt man nun „pot(a,1)=“ und „pot(a,n)=“ weg, erhält man die funktionierende rekursive R-Funktion.

Beispiel 18. *Fakultäten kann man rekursiv wie folgt definieren: $0! = 1$; $n! = (n-1)!n$. Analog erhält man die rekursive R-Funktion:*

```
fak = function(n) {
if( n == 0 ) 1
else n * fak( n - 1 )}
```

fak(5) ergibt 125

14.3 Ein statistisches Beispiel

Beispiel 19. *Es soll in R der Einstichproben-z-Test implementiert werden. Eingabe: Daten; Mittelwert unter der Nullhypothese; Standardabweichung unter der Nullhypothese, Seitigkeit (Voreinstellung Zweiseitig). Es soll der p-Wert ausgegeben werden:*
Man schreibt z.B.

```
z.test=function(daten,mu0,s0,seite='zweiseitig') {
n=length(daten)
m=mean(daten)
testwert=(m-mu0)/(s0/sqrt(n)) # sqrt für Quadratwurzel
if (seite=='links') pWert=pnorm(testwert)  # Doppelte Gleichheitszeichen
#bedeutet Überprüfung auf Gleichheit,
# einfaches Gleichheit durch Definition.
if(seite=='rechts') pWert=1-pnorm(testwert)
if(seite=='zweiseitig') pWert=2*min(pnorm(testwert),1-pnorm(testwert))
pWert}
```

Steht in der Funktionsdefinition `seite='zweiseitig'`, so bedeutet das, dass „zweiseitig“ die Voreinstellung ist. Gibt man nichts ein für `seite`, wird automatisch ein zweiseitiger Test berechnet (sofern im folgenden definiert!).
Der p-Wert kann nun für einen rechtsseitigen Test durch folgenden Befehl berechnet werden:
`z.test(a,5,2,seite='rechts')`

Übung 20. *Erstellen Sie eine analoge R-Funktion, die den p-Wert des Einstichproben-t-Testes berechnet.*

14.4 Ein paar weitere, nützliche Befehle

Operatoren

+	Addition von Zahlen, Vektoren und Matrizen
−	Subtraktion von Zahlen, Vektoren und Matrizen
∗	Multiplikation von Zahlen, von Zahlen mit Vektoren und von Zahlen mit Matrizen
/	Division durch Zahlen
% ∗ %	Matrixmultplikation
abs	Absolutbetrag
exp(x)	e^x
log(x)	ln(x)=$\log_e(x)$
sqrt(x)	$\sqrt{x}$
length	Länge eines Vektors
sum	Summe eines Vektors oder einer Matrix
cumsum	Kumulierte Summen eines Vektors
rank	Rang einer Zahl in einem Vektor
==	Überprüfung auf Gleichheit
identical()	Überprüfung auf Gleichheit, wenn NA vorkommt
!=	nicht identisch
>	grösser als
<	kleiner als
>=	grösser-gleich
<=	kleiner-gleich
&	und
|	oder
!	nicht

Vektoren:

- `x=c(5.1,6.3,7.7,10.3,15.2); x` ergibt: > 5.1,6.3,7.7,10.3,15.2
- `x[2]` ergibt: > 6.3
- `1:5` ergibt: > 1,2,3,4,5
- `2*1:5` ergibt: > 2,4,6,8,10
- `seq(0,5,0.1)` ergibt die arithmetische Folge $a_n = a_{n-1} + b$ von $a_0 = 0$ bis 5 und $b = 0.1$ (Abstand zwischen den Folgegliedern)
- `rep(0.5,15)` ergibt einen Vektor der Länge 15 mit den Komponenten 0.5.
- `rep(1:3,4)` ergibt den Vektor 1,2,3,1,2,3,1,2,3,1,2,3
- `rep(1:3,1:3)` ergibt den Vektor 1, 2,2, 3,3,3
- für den Vektor v mit Zahlen ergibt v>7 einen Vektor mit Wahrheitswerten: falsch, wenn die Komponente kleiner-gleich 7 ist, wahr, wenn die Komponente > 7 ist.
- für den Vektor v und mit `groesser7=v>7` ergibt `v[groesser7]` den Vektor der Zahlen, die grösser als 7 sind. (analog für ==, > <=, >=). Auch Junktoren (&, |) können verwendet werden, um Vektoren von Wahrheitswerten zu erzeugen, z.B. `v>7&v<7`.
- `is.vector(v)` überprüft, ob v ein Vektor ist (TRUE oder FALSE).

Matrizen:

- Eine (nXm)- Matrix kann aus einem Vektor v der Länge $n * m$ mit `matrix(v,ncol=m)` oder `matrix(v,nrow=n)` erstellt werden. Dabei werden die Zahlen des Vektors spaltenweise von oben nach unten und von links nach rechts eingetragen. Will man eine andere Anordnung kann man `matrix(v,ncol=m, byrow=T)` schreiben.

- Durch `r.bind` kann man Vektoren derselben Länge zeilenweise zu einer Matrix verbinden, z.B. `r.bind(1:5,4:9, rep(1,5))`; analog spaltenweise mit `c.bind`.

- Zugriff auf i-te Zeile der Matrix m: `m[i,]`, auf j-te Spalte der Matrix m: `m[,j]`. Zugriff auf die Zelle (i, j) der Matrix m : `m[i,j]`.

- Die i-te Zeile der Matrix m kann man entfernen durch `m[-i,]`, die $j-$ te Spalte durch `m[,-j]`. (statt i und j können auch Vektoren stehen).

- Transponierte: `t(m)`

- Inverse: `solve(m)`

- Benennung der m Spalten durch `dimnames(matrixname)=list(NULL, c("name1","name2",....,"namem"))`

- `dimnames` gibt die eingegeben Namen aus.

- `is.matric(m)` überprüft, ob m eine Matrix ist.

Listen

- Listen können Objekte verschiedenen Typs umfassen. Zugriff auf i-tes Listenobjekt durch `listenname[[i]]` oder wenn der Name des Listenobjektes bekannt ist durch `listenname$objektname`.

- Zugriff auf das j-te Objekt des i-ten Listenobjektes durch `listenname[[i]][j]` oder durch `listenname$objektname[j]`.

- `is.list(listenname)` überprüft, ob `listenname` eine Liste ist.

Data frames

- Zugriff erfolgt wie bei Matrizen.

- `names(dataframe)` = Namen der Variablen.

- Umwandlung eines Dataframes in eine Matrix durch `as.matrix(Dataframename)`

- Umwandlung einer Matrix in ein Dataframe durch `as.data.frame(Matrixname)`.

Einlesen von externen Objekten Mit `source("Pfad/Dateiname.r")` können **R**-Skripten eingelesen werden (Achtung Schrägstrich, nicht rückwärtiger Schrägstrich). Diese können anschliessend verwendet werden. Man würde die obige fak-Funktion (s. Beispiel 18) in ein Skript eingeben und die Datei unter fak.r ins Verzeichnis c: abspeichern. Dann wird diese Funktion geladen mit source("c:/fak.r"). Man kann mehrere Funktionen auf einem Skript ablagern. Mit ls() erhält man eine Liste der eingelesenen Objekte (die allerdings bereits im Workspace befindliche Objekte mit auflistet).

Exportieren von Matrizen, Vektoren und Dataframes Mit `write.table(A, "c:/name.txt")` wird die Matrix, der Vektor oder das Dataframe A unter dem Namen „name“ als Textdatei „name.txt“ auf das Verzeichnis c:/ abgeschpeichert. Text-Dateien können dann z.B. von Excel eingelesen werden (wobei die Variablennamen um eine Zelle nach links verschoben sind!).

Kontakt mit Datenbanken Es gibt R-Pakete, um Daten aus Datenbanken zu verwenden (z.B. RSQLite für SQLite, oder RODBC für ODBC).

Graphik Um die diesbezüglichen Möglichkeiten von **R** zu sehen: s. http://addictedtor.free.fr/graphiques/

Graphische Oberfläche Es gibt verschiedene Pakete, die graphische Oberflächen für **R** bereitstellen. Damit lässt sich **R** ähnlich bedienen wie kommerzielle Statistikprogramme (Beispiel für ein solche Paket: `Rcmdr`; man kann es herunterladen, indem man in **R** auf Pakete geht, „installiere Pakete“, „Switzerland“. Dann erscheint eine Liste mit allen Pakten, die auf Cran vorliegen. Man klickt Rcmdr an. Das Paket wird heruntergeladen. Man bindet diese ein durch library(Rcmdr). Es erscheint eine Warnung, dass Pakete fehlen, um das Paket laufen zu lassen. Man gibt den Befehl, die fehlenden Pakete herunterzuladen. Das Herunterladen und Entpacken der Pakete kann eine Weile dauern. Anschliessend wird Rcmdr geöffnet und kann verwendet werden).

R mit Excel (nur für Windows, aber auch für Calc von Open Office). Es gibt die Möglichkeit, R (Rcmdr kann dabei so eingebunden werden, dass man mit Excel-Items Befehle ausführen kann) innerhalb Excel zu brauchen. Für eine Darstellung der Möglichkeiten siehe das folgende Video: http://rcom.univie.ac.at/RExcelDemo/. Für eine Beschreibung des Installationsvorgangs:
http://learnserver.csd.univie.ac.at/rcomwiki/doku.php?id=wiki:how_to_install
Speicher- und Ladeort: setwd()

Hinweis: Diesen Schnelleinstieg finden Sie als pdf auf http://math.logik.ch unter Statistik.