IN DIESEM KAPITEL

- R und RStudio auf Ihren Computer holen
- In eine R-Session eintauchen
- Mit R-Funktionen arbeiten
- Mit R-Strukturen arbeiten

Kapitel 1
R: Was R kann und wie R das macht

Sie sind also bereit, die wunderbare Welt von R zu bereisen? R ist von und für Statistiker und Datenwissenschaftler entwickelt und besitzt eine kurze, jedoch glanzvolle Geschichte.

In den 1990er Jahren entwickelten Ross Ikaha und Robert Gentlemen R an der University of Auckland, Neuseeland. R wird vom gemeinnützigen Verein »The R Foundation for Statistical Computing« unterstützt und erfreut sich mit jedem Tag größerer Beliebtheit.

R herunterladen

Falls R noch nicht auf Ihrem Computer installiert ist, müssen Sie als Allererstes R herunterladen und installieren.

Sie finden die geeignete Software auf der Website des Comprehensive R Archive Network (CRAN). Falls Sie Windows verwenden, geben in Ihrem Browser diese Adresse ein:

```
cran.r-project.org/bin/windows/base
```

Falls Sie auf dem Mac arbeiten, ist dies die Downloadseite:

```
cran.r-project.org/bin/macosx
```

Klicken Sie den Link an, um R herunterzuladen. Hierdurch wird eine ausführbare `.exe`-Datei auf Ihren Windows-PC heruntergeladen beziehungsweise eine `.pkg`-Datei auf Ihren Mac. Führen Sie in beiden Fällen die gewohnten Schritte zur Installation eines Programms

durch. Wenn die Installation fertiggestellt wurde, sehen Windows-Anwender auf ihrem Desktop zwei R-Programmsymbole, eines für 32-Bit-Betriebssysteme und eines für 64-Bit-Betriebssysteme (verwenden Sie das Symbol für Ihre Windows-Variante). Mac-Benutzer finden R im Ordner *Programme*.

Beide URLs enthalten auch nützliche Links zu »Häufig gestellten Fragen, FAQs«. Auf der Downloadseite der Windows-Version finden Sie auch einen Link zu »Installation and other instructions«.

RStudio herunterladen

Die Verwendung von R ist sehr viel einfacher, wenn Sie hierfür eine Anwendung mit dem Namen *RStudio* verwenden. Computerfreaks nennen diese Art von Programm Entwicklungsumgebung oder kurz IDE für den englischen Fachbegriff *I*ntegrated *D*evelopment *E*nvironment. Hierbei handelt es sich um ein Werkzeug, dass Sie beim Erstellen, Bearbeiten, Ausführen und Nachverfolgen Ihres R-Codes unterstützt. Gleichzeitig bietet die Entwicklungsumgebung Zugriff auf hilfreiche Tipps über R.

Hier ist die Webadresse dieses fantastischen Tools:

`www.rstudio.com/products/rstudio/download`

Klicken Sie den Link des Installationsprogramms für Ihr Betriebssystem – Windows, Mac oder eine der Linux-Varianten – an und führen Sie die üblichen Schritte zur Programminstallation durch.

In diesem Buch verwende ich die R-Version 3.4.3 und die RStudio-Version 1.1.419. Zu dem Zeitpunkt, zu dem Sie diese Zeilen lesen, stehen möglicherweise neuere Versionen zur Verfügung.

Nachdem die Installation abgeschlossen ist, klicken Sie das nagelneue Programmsymbol von RStudio an, damit Sie das Fenster in Abbildung 1.1 sehen.

In der großen Konsole auf der linken Seite wird der R-Code ausgeführt. Eine Möglichkeit, um R-Code auszuführen, besteht darin, ihn direkt in die Konsole einzugeben. Eine andere Variante zeige ich Ihnen in Kürze.

In den beiden Bereichen auf der rechten Seite des Fensters finden Sie hilfreiche Informationen, während Sie mit R arbeiten. Oben rechts finden Sie den Arbeitsbereich Environment (Umgebung) und History (Historie). Auf der Registerkarte Environment werden die Dinge mitgehalten, die Sie beim Arbeiten mit R erstellen. (Diese werden in R Objekte genannt.) Auf der Registerkarte History wird der R-Code mitgehalten, den Sie eingeben.

Gewöhnen Sie sich schon mal an das Wort *Objekt*. In R ist alles ein Objekt.

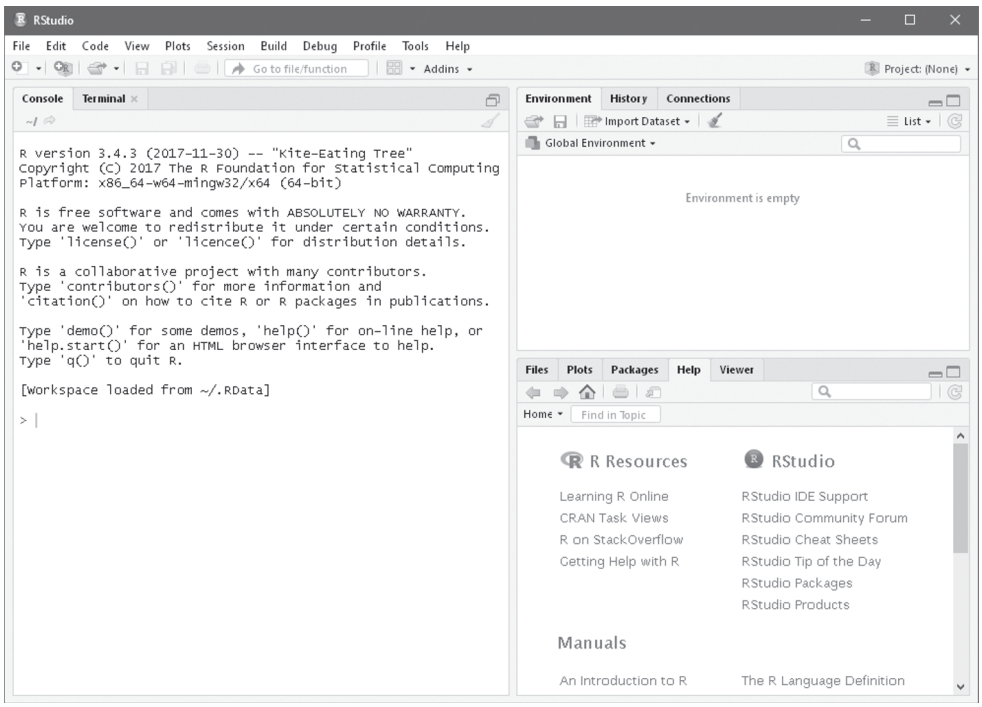

Abbildung 1.1: RStudio, wie es beim ersten Start nach der Installation aussieht

Im Panel unten rechts finden Sie die Registerkarten Files, Plots, Packages und Help. Auf der Registerkarte Files werden die von Ihnen erstellten Dateien angezeigt. Die Registerkarte Plots enthält die Diagramme, die Sie mit Ihren Daten erstellt haben. Auf der Registerkarte Packages sehen Sie die Add-Ons (die in R Packages, Pakete, genannt werden), die Sie zusammen mit R heruntergeladen haben. Beachten Sie, dass *heruntergeladen* nicht bedeutet, dass die Packages bereits genutzt werden können. Hierfür ist ein weiterer Schritt erforderlich. Und glauben Sie mir, Sie wollen Packages verwenden.

Die Registerkarte Packages sehen Sie in Abbildung 1.2. Packages stelle ich weiter hinten in diesem Kapitel vor.

Die Registerkarte Help, die Sie in Abbildung 1.3 sehen, enthält Links zu einer Fülle von englischsprachigen Informationen über R und RStudio.

Um alle Möglichkeiten der grafischen Entwicklungsumgebung RStudio zu nutzen, klicken Sie auf das größere der beiden Symbole, die Sie in der oberen rechten Ecke der Konsole sehen. Das Aussehen von RStudio ändert wie in Abbildung 1.4 gezeigt.

Das Konsolenfenster wird verkleinert und befindet sich nun links unten. Darüber sehen Sie ein neues Fenster, das Skriptfenster. Das Skriptfenster ist der Editor für Ihren R-Code. Sie können dort Code eingeben und bearbeiten und dann Strg + ↵ drücken, um den Code im Konsolenfenster ausführen zu lassen. Auf dem Mac drücken Sie ⌘ + ↵.

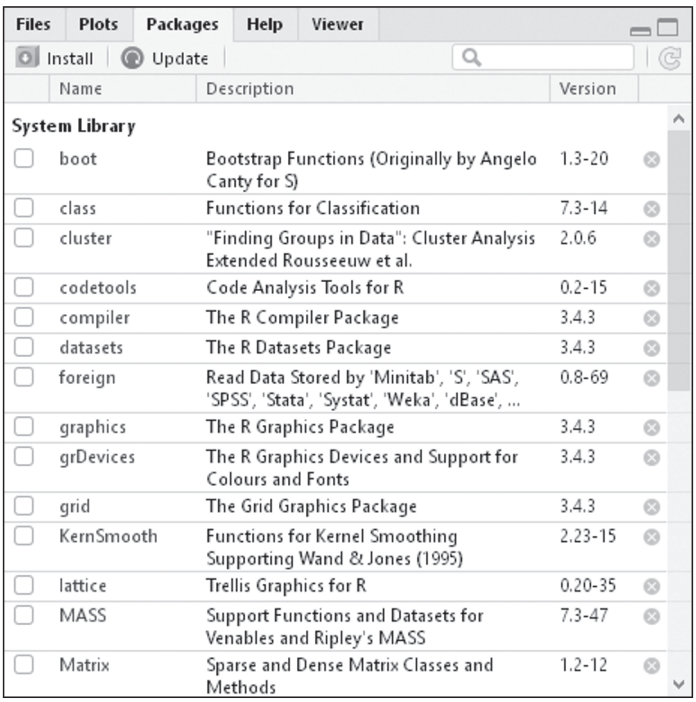

Abbildung 1.2: Die Registerkarte PACKAGES in RStudio

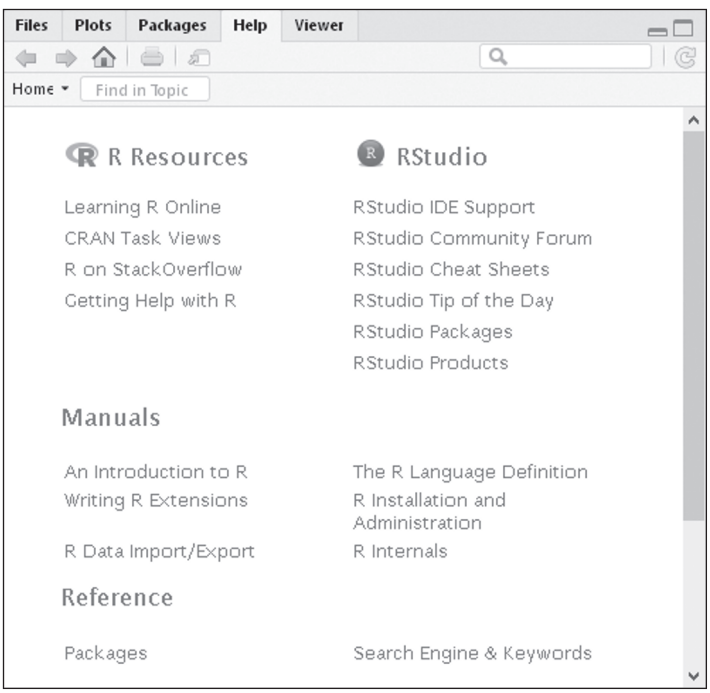

Abbildung 1.3: Die Registerkarte HELP in RStudio

Sie können auch im Skriptfenster die gewünschten Codezeilen markieren und dann im Menü von RStudio den Befehl Code | Run Selected Line(s) verwenden.

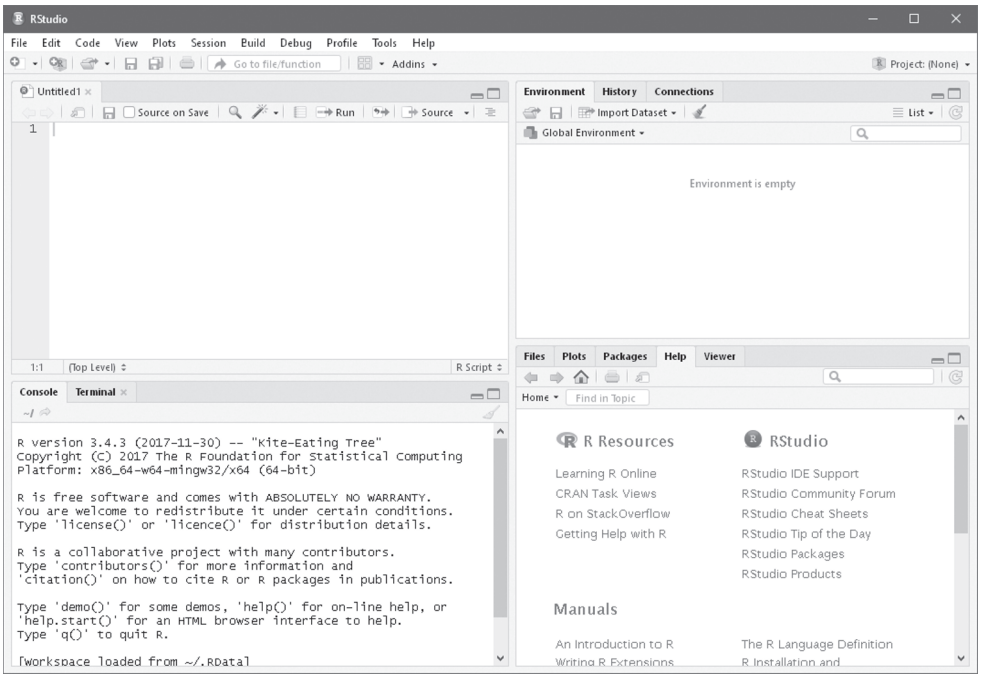

Abbildung 1.4: RStudio, nachdem Sie an der rechten Seite der Titelleiste der Konsole das große Symbol angeklickt haben

Eine Session mit R

Bevor Sie loslegen, wählen Sie den Befehl File | Save As und verwenden Sie als Dateiname **Meine erste R-Session**. Die Registerkarte im Skriptfenster wird umbenannt und Sie sehen dort den eingegebenen Dateinamen mit der Erweiterung `.R`. Außerdem wird der Dateiname (ebenfalls mit der Erweiterung `.R`) auf der Registerkarte Files angezeigt.

Das Arbeitsverzeichnis

Was genau speichert R und wo wird gespeichert? R speichert den sogenannten Workspace; das ist die Umgebung, in der Sie arbeiten. Der Workspace wird von R im *Arbeitsverzeichnis* gespeichert. In Windows ist dies das Standardarbeitsverzeichnis:

```
C:\Benutzer\<Benutzername>\Dokumente
```

Auf dem Mac ist es der folgende Ordner:

```
/Benutzer/<Benutzername>
```

Falls Sie jemals den Pfad zu Ihrem Arbeitsverzeichnis vergessen sollten, geben Sie in der Konsole den folgenden Befehl ein. R zeigt dann das Arbeitsverzeichnis auf dem Bildschirm an:

```
> getwd()
```

In der Konsole brauchen Sie das Größerzeichen nicht einzugeben. Es befindet sich bereits am Anfang der Zeile und stellt die Eingabeaufforderung dar.

Mein Arbeitsverzeichnis sieht folgendermaßen aus:

```
> getwd()
[1] "C:/Users/Joseph Schmuller/Documents/R"
```

Der Befehl `getwd()` zeigt den englischen Ordnernamen an. In den deutschsprachigen Windows-Versionen kümmert sich das Betriebssystem darum, dass der Ordnername als Benutzer angezeigt wird. Die Pfadangaben `C:\Users` und `C:\Benutzer` verweisen also auf den gleichen Ordner.

Beachten Sie, dass in der Ausgabe von `getwd()` die normalen Schrägstriche und keine Backslashs verwendet werden. Dies ist anders, als Sie es von Windows her gewohnt sind. Der Grund ist, dass R den Backslash (\) als Escape-Zeichen verwendet. Zeichen, die nach dem Escape-Zeichen eingegeben werden, erhalten so eine andere Bedeutung. So entspricht `\t` in R dem Tabulatorzeichen.

Wenn Sie lieber den Backslash verwenden, müssen Sie ihn für Pfadnamen in R doppelt eingeben, beispielsweise `C:\\Benutzer\\<Benutzername>\\Dokumente`.

Wenn Sie möchten, können Sie das Arbeitsverzeichnis mit diesem Befehl ändern:

```
> setwd(<Dateipfad>)
```

Eine andere Möglichkeit zum Ändern des Arbeitsverzeichnisses ist der Befehl Session | Set Working Directory | Choose Directory.

Jetzt geht es richtig los

Jetzt kümmern wir uns um R. Geben Sie Folgendes in das Skriptfenster ein:

```
x <- c(5,10,15,20,25,30,35,40)
```

Drücken Sie dann Strg + ↵.

Hierdurch wird die folgende Zeile in das Konsolenfenster eingefügt:

```
> x <- c(5,10,15,20,25,30,35,40)
```

Wie ich bereits in einem Tipp weiter vorne erwähnte, ist das Größerzeichen, das Sie in der Konsole sehen, der Prompt, die Eingabeaufforderung in R. Im Skriptfenster ist es nicht zu sehen.

Was hat R gerade gemacht? Die Zeichen <– geben an, dass x das zugewiesen wird, was auf der rechten Seite von <– steht. Sie können sich die Zeichen <– als Zuweisungsoperator von R vorstellen. Die Werte 5, 10, 15, 20 ... 40 sind also nun x zugewiesen.

In R wird ein Satz von Zahlen *Vektor* genannt. Ich komme hierauf im Abschnitt »R-Strukturen« weiter hinten in diesem Kapitel zurück.

Sie können diese Zeile folgendermaßen lesen: »x wird der Vektor 5, 10, 15, 20 usw. zugewiesen«.

Geben Sie im Skriptbereich **x** ein und drücken Sie Strg + ↵. In der Konsole sehen Sie dann die folgende Ausgabe:

```
>x
[1]  5 10 15 20 25 30 35 40
```

Die 1 in den eckigen Klammern ist das Label für den ersten Wert in der Ausgabezeile, in unserem Beispiel also der Wert 5.

In diesem Beispiel gibt es lediglich eine Zeile mit Werten. Was passiert, wenn R mehrere Werte in mehreren Zeilen ausgibt? Jede Zeile beginnt mit einem numerischen Label, das von eckigen Klammern eingeschlossen ist. Die Zahl in den Klammern korrespondiert mit dem Index des ersten Werts der Zeile. Wenn die Ausgabe beispielsweise aus 23 Werten besteht und der 18te Wert der erste Wert in der zweiten Zeile ist, dann beginnt die zweite Zeile mit [18].

Durch die Erstellung des Vektors ändert sich auch der Inhalt der Registerkarte ENVIRONMENT, wie es Abbildung 1.5 zeigt.

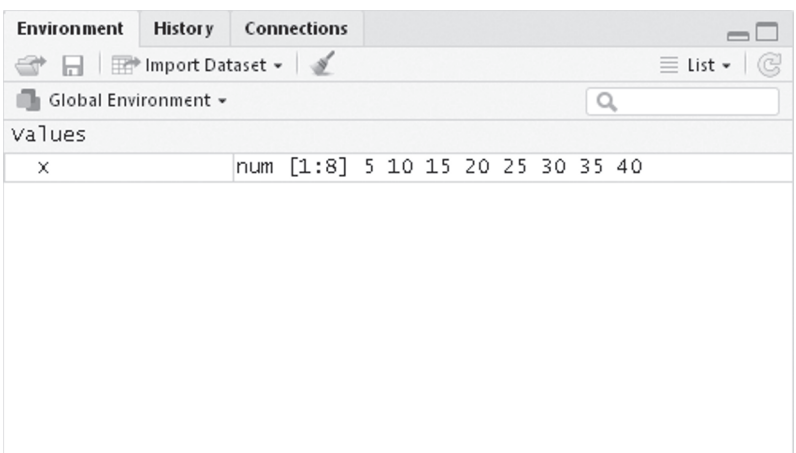

Abbildung 1.5: Die Registerkarte ENVIRONMENT nach der Erzeugung des Vektors x

Eine andere Möglichkeit, um sich die Objekte im Environment anzusehen, besteht darin, in das Skriptfenster `ls()` einzugeben und dann Strg + ↵ zu drücken. Sie können `ls()` auch direkt in das Konsolenfenster eingeben und ↵ drücken. In beiden Fällen wird im Konsolenfenster folgendes Ergebnis angezeigt:

```
[1] "x"
```

Nun können Sie mit `x` arbeiten. Addieren Sie zuerst alle Zahlen im Vektor. Geben Sie

```
sum(x)
```

im Skriptfenster ein (denken Sie daran, Strg + ↵ zu drücken). Hierdurch wird in der Konsole diese Zeile ausgeführt:

```
> sum(x)
[1] 180
```

Wie wäre es mit dem Mittelwert der Zahlen im Vektor x? Tippen Sie

```
mean(x)
```

im Skriptfenster ein, drücken Sie Strg + ↵ und Sie erhalten in der Konsole diese Ausgabe:

```
> mean(x)
[1] 22.5
```

Während Sie im Skriptfenster oder in der Konsole Anweisungen eingeben, werden hilfreiche Popupfenster angezeigt. Wenn Sie mehr Erfahrung mit RStudio haben, werden Sie lernen, wie Sie diese Informationen nutzen können.

Die Varianz gibt an, wie sehr sich eine Gruppe von Zahlen von deren Mittelwert unterscheidet. Um mit R die Varianz zu berechnen, verwenden Sie die folgende Anweisung:

```
> var(x)
[1] 150
```

Was genau ist Varianz und was bedeutet es? (Achtung: Es folgt ein schamloser Werbehinweis.) Antworten auf diese und viele weitere Fragen über Statistik und Analyse finden Sie in dem Klassiker *Statistik mit R für Dummies* (der ebenfalls von mir verfasst und bei Wiley-VCH veröffentlicht wurde).

Nachdem R all diese Befehle ausgeführt hat, sieht die Registerkarte HISTORY so aus wie in Abbildung 1.6.

Um eine Session zu beenden, wählen Sie FILE | QUIT SESSION oder drücken Sie Strg + Q. Wie in Abbildung 1.7 zu sehen, wird ein Dialogfeld angezeigt und Sie werden gefragt, welche Elemente der Session gespeichert werden sollen. Wenn Sie die Auswahl unverändert lassen und auf SAVE SELECTED klicken, können Sie beim nächsten Start von RStudio die Session erneut öffnen und an der Stelle weitermachen, an der Sie aufgehört haben. (Beachten Sie, dass die Inhalte der Konsole nicht gespeichert werden.)

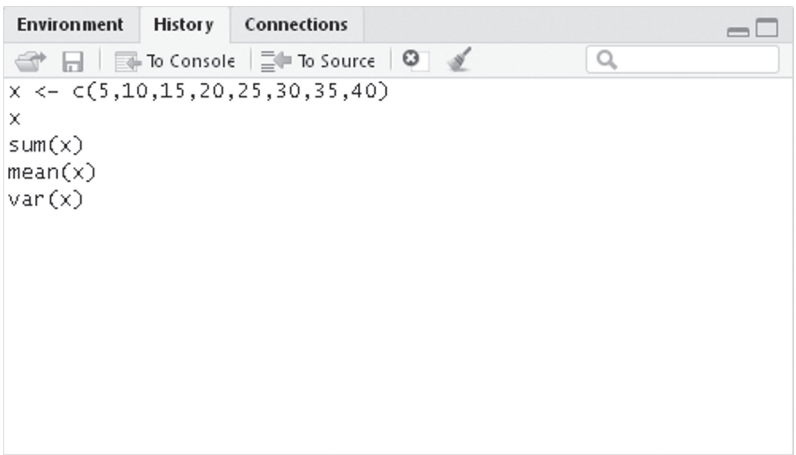

Abbildung 1.6: Die Registerkarte HISTORY nach der Erstellung des Vektors und dessen Verwendung

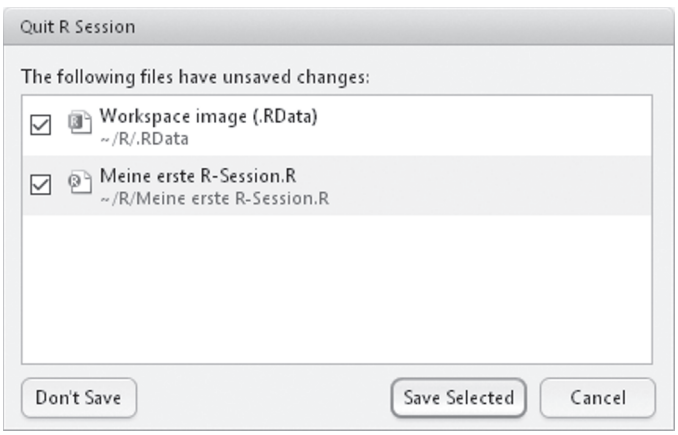

Abbildung 1.7: Das Dialogfeld QUIT R SESSION

Im weiteren Verlauf des Buches werde ich beim Vorstellen eines Beispiels nicht immer schreiben: »Geben Sie diesen R-Code in das Skriptfenster ein und drücken Sie Strg + ↵.« Stattdessen zeige ich Ihnen wie im Beispiel mit `var()` einfach den Code und dessen Ausgabe.

Manchmal ist der Code mit dem Prompt › dargestellt und manchmal ohne ihn. Im Prinzip zeige ich Ihnen den Prompt immer dann, wenn ich Ihnen den R-Code mitsamt den Ergebnissen zeigen will. Falls ich lediglich Code darstellen möchte, den ich im Skriptbereich eingegeben habe, lasse ich den Prompt weg.

R-Funktionen

Im vorigen Abschnitt habe ich `c()`, `sum()`, `mean()` und `var()` verwendet. Dies sind Beispiele für die in R eingebauten Funktionen. Die Funktion besteht aus einem Namen, dem ein Paar runde Klammern folgt. Innerhalb der runden Klammern stehen die *Argumente*. In diesem Zusammenhang ist mit »Argument« nicht der Austausch über eine Meinungsverschiedenheit gemeint. Argument, manchmal auch *Parameter* genannt, ist der mathematische Ausdruck für die Daten, die eine Funktion verwendet.

Auch wenn eine Funktion keine Argumente verwendet (wie im Fall der Funktion `ls()`), müssen Sie die runden Klammern angeben.

Die Funktionen in den vorherigen Beispielen sind sehr einfach: Geben Sie ein Argument an und die Funktion liefert ein Ergebnis. Einige der R-Funktionen nehmen jedoch mehr als ein Argument entgegen.

R stellt für den Einsatz von Funktionen, die mehrere Argumente besitzen, verschiedene Varianten zur Verfügung. Eine Möglichkeit besteht darin, die Argumente in der gleichen Reihenfolge anzugeben, die bei der Definition der Funktion verwendet wurde. R nennt dies *Stellungsparameter* (positional matching).

Hier ein Beispiel dafür, was damit gemeint ist. Erinnern Sie sich noch daran, wie ich den Vektor x erstellte?

```
x <- c(5,10,15,20,25,30,35,40)
```

Eine andere Möglichkeit, einen Vektor dieser Zahlen zu erstellen, ist die Verwendung der Funktion `seq()`:

```
> y <- seq(5,40,5)
> y
[1]  5 10 15 20 25 30 35 40
```

Mit der Funktion `seq()` können Sie eine Sequenz erstellen. Das erste Argument für `seq()` ist die Zahl, ab der die Sequenz beginnen soll; in unserem Beispiel die Zahl 5. Der Name dieses Arguments lautet *from*. Das zweite Argument ist die Zahl, bis zu der die Sequenz gehen soll; in unserem Beispiel die Zahl 40. Der Name dieses Arguments lautet *to*. Das dritte Argument gibt das Inkrement der Sequenz an, also den Wert, um den sich die Sequenz erhöht (5). Der Name dieses Arguments lautet *by*.

Wenn Sie beim Funktionsaufruf die Namen der Argumente angeben, spielt deren Reihenfolge keine Rolle:

```
> z <- seq(to=40,by=5,from=5)
> z
[1]  5 10 15 20 25 30 35 40
```

Wenn Sie also eine Funktion verwenden, können Sie deren Parameter in beliebiger Reihenfolge angeben, solange Sie die Parameter benennen. Diese Art der Parameterübergabe wird in R *Schlüsselwortparameter* (keyword matching) genannt. Dies ist nützlich, wenn Sie eine R-Funktion mit zahlreichen Argumenten verwenden wollen. Wenn Sie deren Reihenfolge nicht genau kennen, geben Sie einfach die Namen an und die Funktion funktioniert.

Falls Sie jemals Hilfe zu einer bestimmten Funktion benötigen, beispielsweise für `seq()`, geben Sie **?seq** ein und schauen Sie sich auf der Registerkarte HELP die nützlichen Informationen an. Während Sie **?seq** eingeben, werden neben dem eingegebenen Text in einem kleinen Popup-Fenster Kurzinfos zu der Funktion angezeigt.

Benutzerdefinierte Funktionen

R erlaubt es Ihnen, eigene Funktionen zu erstellen. Nachfolgend erläutere ich hierzu die Grundlagen.

Eine R-Funktion ist folgendermaßen aufgebaut:

```
meinefunktion <- function(argument1, argument2, ... ){
   Anweisungen
   return(object)
}
```

Hier ist eine Funktion für rechtwinklige Dreiecke. Erinnern Sie sich noch? Ein rechtwinkliges Dreieck hat zwei Seiten, die einen rechten Winkel bilden; die dritte Seite wird Hypotenuse genannt. Vielleicht erinnern Sie sich auch daran, dass ein Typ namens Pythagoras gezeigt hat, dass die Länge der Hypotenuse *c* folgendermaßen berechnet wird, wenn eine Seite die Länge *a* und die andere Seite die Länge *b* besitzt:

$$c = \sqrt{a^2 + b^2}$$

Hier ist eine einfache Funktion mit dem Namen `hypotenuse()`, der zwei Zahlen übergeben werden (die Länge der beiden Seites des rechtwinkligen Dreiecks) und die `c`, die Länge der Hypotenuse, zurückgibt:

```
hypotenuse <- function(a,b){
   hyp <- sqrt(a^2+b^2)
   return(hyp)
}
```

Geben Sie diesen Code in das Skriptfenster ein, markieren Sie ihn und drücken Sie Strg+↵. In der Konsole erscheint die folgende Ausgabe:

```
> hypotenuse <- function(a,b){
+   hyp <- sqrt(a^2+b^2)
+   return(hyp)
+}
```

Das Pluszeichen ist das Zeilenfortsetzungszeichen. Es gibt an, dass die aktuelle Zeile von der vorherigen Zeile fortgesetzt wird.

Und so können Sie diese Funktion verwenden:

```
> hypotenuse(3,4)
[1] 5
```

Wie Sie zu Recht vermuten, können Sie mit R-Funktionen viel mehr tun als das, was ich kurz angerissen habe. Wenn Sie an weiterführenden Informationen interessiert sind, sollten Sie einen Blick in das Buch *R für Dummies* von Andrie de Vries und Joris Meys werfen.

Kommentare

Kommentare sind Anmerkungen für Ihren Code. Leiten Sie einen Kommentar mit dem Raute-Zeichen ein (#), heute eher als Hashtag bekannt. R ignoriert alles, was rechts vom #, dem Kommentarzeichen, steht.

Kommentare sind für diejenigen nützlich, die den Code, den Sie geschrieben haben, lesen und verstehen wollen. Hier ein Beispiel:

```
hypotenuse <- function(a,b){  # Liste der Argumente
   hyp <- sqrt(a^2+b^2)       # Berechnung durchführen
   return(hyp)                # Den Wert zurückgeben
}
```

Nur zur Info: Im Beispielcode dieses Buchs verwende ich keine Kommentare. Stattdessen stelle ich detaillierte Beschreibungen zur Verfügung. Ich glaube, dass das in einem Buch wie diesem der beste Weg ist, das rüberzubringen, was ich vermitteln möchte.

R-Strukturen

Im Abschnitt »R-Funktionen« weiter vorne in diesem Kapitel habe ich erwähnt, dass eine R-Funktion viele Argumente besitzen kann. Außerdem kann eine R-Funktion auch mehrere Werte, Rückgabewerte genannt, ausgeben. Um die verschiedenen Ausgaben (und auch Eingaben) zu verstehen, müssen Sie die Strukturen verstehen, mit denen R arbeitet.

Vektoren

Der Vektor ist die fundamentale Datenstruktur in R. Vektoren haben Sie bereits in vorherigen Beispielen gesehen. Ein Vektor ist ein eindimensionales Array von Datenelementen, die den gleichen Typ besitzen. Diese Datenelemente werden auch Komponenten genannt.

Um einen Vektor zu erzeugen, verwenden Sie die Funktion `c()`, wie im Beispiel weiter vorne:

```
> x <- c(5,10,15,20,25,30,35,30)
```

Im Beispiel des Vektors `x` handelt es sich bei den Datenelementen um Zahlen.

In einem Zeichenvektor besitzen die Elemente den Typ Zeichenfolge; sie werden jeweils in doppelte Anführungszeichen eingefasst:

```
> beatles <- c("John","Paul","George","Ringo")
```

Es gibt auch logische Vektoren, deren Elemente die Werte `TRUE` oder `FALSE` (wahr oder falsch) besitzen. Diese Wahrheitswerte können mit `T` und `F` abgekürzt werden:

```
> w <- c(T,F,T,F,T,T)
```

Um auf ein bestimmtes Element eines Vektors zuzugreifen, geben Sie nach dem Vektornamen eckige Klammern an und fügen Sie in die eckigen Klammern den gewünschten Index ein:

```
> beatles[2]
[1] "Paul"
```

Innerhalb der eckigen Klammern können Sie einen Doppelpunkt verwenden, um auf zwei aufeinanderfolgende Komponenten zu verweisen:

```
> beatles[2:3]
[1] "Paul"   "George"
```

Wollen Sie auf nicht aufeinanderfolgende Komponenten verweisen? Dies ist ein wenig komplizierter, aber mit `c()` machbar:

```
> beatles[c(2,4)]
[1] "Paul"   "Ringo"
```

Numerische Vektoren

Zusätzlich zur Funktion `c()` stellt Ihnen R zwei weitere Funktionen zur Verfügung, mit denen Sie numerische Vektoren erstellen können. Eine davon, `seq()`, habe ich weiter vorne gezeigt:

```
> y <- seq(5,40,5)
> y
[1]  5 10 15 20 25 30 35 40
```

Wenn Sie das dritte Argument weglassen, wird die Sequenz um 1 erhöht:

```
> y <- seq(5,40)
> y
[1]  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
[2] 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
```

Auf meinem Bildschirm, und vermutlich auch auf Ihrem, werden alle Elemente in `y` in einer Zeile angezeigt, jedoch ist die gedruckte Seite nicht so breit wie die Konsole. Daher habe ich die Ausgabe auf zwei Zeilen aufgeteilt und am Anfang der zweiten Zeile im Stil von R die 20 in eckigen Klammern ergänzt.

```
20 25 30 35 40
```

R unterstützt eine bestimmte Syntax für das Erstellen eines numerischen Vektors, dessen Werte jeweils um 1 erhöht werden:

```
> y <- seq(5:40)
> y
[1]  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
[2] 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
```

Eine andere Funktion, `rep()`, erzeugt einen Vektor mit sich wiederholenden Werten:

```
> viererwette <- c(7,8,4,3)
> wiederholte_viererwette <- rep(viererwette,4)
> wiederholte_viererwette
[1] 7 8 4 3 7 8 4 3 7 8 4 3 7 8 4 3
```

Sie können `rep()` auch so verwenden, dass Sie als zweites Argument einen Vektor angeben:

```
> rep_vektor <-c(1,2,3,4)
> wiederholte_viererwette <- rep(viererwette,rep_vektor)
```

Der Vektor gibt die Anzahl der Wiederholungen jedes Elements an. Daher erhalten Sie dieses Ergebnis:

```
> wiederholte_viererwette
[1] 7 8 8 4 4 4 3 3 3 3
```

Das erste Element wird einmal wiederholt, das zweite zweimal, das dritte dreimal und das vierte Element viermal.

Sie können `append()` verwenden, um am Ende des Vektors ein Element anzuhängen:

```
> xx <- c(3,4,5)
> xx
[1] 3 4 5
> xx <- append(xx,6)
> xx
[1] 3 4 5 6
```

Außerdem können Sie `prepend()` einsetzen, um am Anfang eines Vektors ein Element einzufügen:

```
> xx <- prepend(xx,2)
> xx
[1] 2 3 4 5 6
```

Wie viele Elemente befinden sich in einem Vektor? Dies ermitteln Sie mit `length()`:

```
> length(xx)
[1] 5
```

Matrizen

Eine Matrix ist ein zweidimensionales Array von Datenelementen des gleichen Typs. Sie können eine Matrix mit numerischen Werten erstellen:

```
5    30    55    80
10   35    60    85
15   40    65    90
20   45    70    95
25   50    75    100
```

Oder eine Matrix aus Zeichenfolgen:

```
"John"      "Paul"      "George"    "Ringo"
"Groucho"   "Harpo"     "Chico"     "Zeppo"
"Karo"      "Herz"      "Pik"       "Kreuz"
```

Die Zahlen ergeben eine 5×4-Matrix (fünf Zeilen, vier Spalten); die Zeichenfolgenmatrix besitzt drei Zeilen und vier Spalten.

Um die numerische Matrix zu erstellen, erstellen Sie zuerst einen Vektor der Zahlen von 5 bis 100 in 5er-Schritten:

```
> num_matrix <- seq(5,100,5)
```

Anschließend verwenden Sie die Funktion `dim()` und konvertieren den Vektor in eine zweidimensionale Matrix:

```
> dim(num_matrix) <- c(5,4)
> num_matrix
     [,1] [,2] [,3] [,4]
[1,]    5   30   55   80
[2,]   10   35   60   85
[3,]   15   40   65   90
[4,]   20   45   70   95
[5,]   25   50   75  100
```

Beachten Sie, dass R links die Zeilennummer und oben die Spaltennummer in eckigen Klammern anzeigt.

Wenn Sie eine Matrix *transponieren*, werden die Zeilen mit den Spalten vertauscht. In R erledigen Sie das mit der Funktion t():

```
> t(num_matrix)
     [,1] [,2] [,3] [,4] [,5]
[1,]    5   10   15   20   25
[2,]   30   35   40   45   50
[3,]   55   60   65   70   75
[4,]   80   85   90   95  100
```

Eine weitere Möglichkeit zum Erstellen von Matrizen bietet die Funktion matrix():

```
> num_matrix <- matrix(seq(5,100,5),nrow=5)
> num_matrix
     [,1] [,2] [,3] [,4]
[1,]    5   30   55   80
[2,]   10   35   60   85
[3,]   15   40   65   90
[4,]   20   45   70   95
[5,]   25   50   75  100
```

Wenn Sie das Argument byrow=T angeben, füllt R die Matrix zeilenweise und nicht spaltenweise:

```
> num_matrix <- matrix(seq(5,100,5),nrow=5,byrow=T)
> num_matrix
     [,1] [,2] [,3] [,4]
[1,]    5   10   15   20
[2,]   25   30   35   40
[3,]   45   50   55   60
[4,]   65   70   75   80
[5,]   85   90   95  100
```

Wie greifen Sie auf ein bestimmtes Element der Matrix zu? Geben Sie den Namen der Matrix ein, gefolgt von eckigen Klammern, in denen die Nummer der Zeile, ein Komma und die Nummer der Spalte stehen:

```
> num_matrix[5,4]
[1] 100
```

Gehen Sie folgendermaßen vor, um auf eine ganze Zeile, beispielsweise die dritte, zu verweisen:

```
> num_matrix[3,]
[1,]   45   50   55   60
```

Das geht auch mit ganzen Spalten, hier am Beispiel der zweiten Spalte gezeigt:

```
> num_matrix[,2]
[1,]   10   30   50   70   90
```

Bedenken Sie jedoch das Folgende...

Wie ich oben erwähne, ist eine Matrix ein zweidimensionales Array. In R kann ein Array jedoch mehr als zwei Dimensionen besitzen. Ein bekannter Datensatz (den ich als Beispiel im dritten Kapitel verwende), besitzt drei Dimensionen: Haarfarbe (Black, Brown, Red, Blond), Augenfarbe (Brown, Blue, Hazel, Green) und Geschlecht (Male, Female). Es handelt sich also um ein 4×4×2-Array. Der Name des Datensatzes lautet `HairEyeColor` und er sieht so aus:

```
> HairEyeColor
, , Sex = Male

       Eye
Hair     Brown Blue Hazel Green
Black       32   11    10     3
Brown       53   50    25    15
Red         10   10     7     7
Blond        3   30     5     8

, , Sex = Female

       Eye
Hair     Brown Blue Hazel Green
Black       36    9     5     2
Brown       66   34    29    14
Red         16    7     7     7
Blond        4   64     5     8
```

Jede Zahl repräsentiert die Anzahl der Personen in der Gruppe, die eine bestimmte Kombination aus Haarfarbe, Augenfarbe und Geschlecht aufweisen, beispielswiese 14 Frauen mit braunen Augen und roten Haaren. (Warum habe ich Frauen mit braunen Augen und roten Haaren ausgewählt? Weil ich das Vergnügen habe, jeden Tag eine solche extreme Schönheit anschauen zu dürfen.)

Wie würden Sie auf all diese Frauen verweisen? Das geht so:

```
HairEyeColor[,,2]
```

Listen

In R ist eine Liste eine Sammlung von Objekten, die nicht notwendigerweise den gleichen Typ besitzen müssen. Angenommen, Sie tragen verschiedene Informationen über die Beatles zusammen:

```
> beatles <- c("john","paul","george","ringo")
```

Eine weitere Information könnte das Alter der einzelnen Beatles sein, in dem sie Bandmitglied wurden. John und Paul haben angefangen, gemeinsam zu singen, als sie 17 beziehungsweise 15 Jahre alt waren. Der 14 Jahre alte George schloss sich ihnen kurz danach an. Ringo, ein Nachzügler, wurde ein Beatle, als er 22 war. Diese Daten stehen in diesem Vektor:

```
> alter <- c(17,15,14,22)
```

Um diese Informationen in einer Liste zu kombinieren, verwenden Sie die Funktion `list()`:

```
> beatles_info <-list(namen=beatles,alter_eintritt=alter)
```

Indem Sie die Argumente benennen (`name`, `alter_eintritt`) bringen Sie R dazu, diese Namen als Namen der Listenkomponenten zu verwenden.

Und so sieht diese Liste aus:

```
> beatles_info
$namen
[1] "john"   "paul"   "george" "ringo"

$alter_eintritt
[1] 17 15 14 22
```

Wie Sie sehen können, verwendet R das Dollarzeichen (`$`), um die einzelnen Komponenten der Liste anzugeben. Um auf eine Listenkomponente zuzugreifen, verwenden Sie daher den Namen der Liste, dann das Dollarzeichen und abschließend den Namen der Komponente:

```
> beatles_info$namen
[1] "john"   "paul"   "george" "ringo"
```

Und wie greifen Sie nun auf einen bestimmten Beatle zu, beispielsweise den vierten? Ich vermute, Sie wissen bereits, wie Sie vorgehen müssen:

```
> beatles_info$namen[4]
[1] "ringo"
```

R erlaubt es außerdem, innerhalb der eckigen Klammern Kriterien anzugeben. Um sich beispielsweise die Namen der Fab Four anzeigen zu lassen, die beim Bandeintritt älter als 16 waren, gehen Sie so vor:

```
> beatles_info$namen[beatles_info$alter_eintritt > 16]
[1] "john"  "ringo"
```

Datensätze (Data Frames)

Eine Liste ist eine gute Möglichkeit, Daten zu sammeln. Ein Data Frame ist noch besser? Warum? Wenn Sie sich die Daten für eine Gruppe von Individuen vorstellen, dann sehen Sie vor Ihrem inneren Auge vermutlich eine Tabelle, bei der die Zeilen die Daten der einzelnen Personen enthalten und die Spalten die Datenvariablen. Wenn Ihnen die Begriffe Datentabelle oder Datenmatrix in den Sinn kommen, dann wissen Sie, was ich meine.

Hier folgt ein Beispiel. Angenommen, ich habe einen Datensatz mit sechs Personen:

```
name <- c("al","barbara","charles","donna","ellen","fred")
```

Außerdem kenne ich für jede Person deren Länge (in Zentimeter) und deren Gewicht (in Kilogramm):

```
laenge <- c(182,162,185,165,167,180)
gewicht <- c(88,53,93,55, 56, 90)
```

Zusätzlich ergänze ich das Geschlecht jeder Person:

```
geschlecht <- c("M","W","M","W","W","M")
```

Bevor ich Ihnen zeige, wie Sie diese Vektoren in einem Datensatz kombinieren können, muss ich Ihnen etwas anderes zeigen. Bei den Komponenten des Vektors `geschlecht` handelt es sich um Zeichenketten. Im Hinblick auf die Zusammenfassung der Daten und die Analyse ist es eine gute Idee, diese in Kategorien umzuwandeln, und zwar in die Kategorie Männlich und die Kategorie Weiblich. Um dies zu bewerkstelligen, verwende ich die Funktion `factor()`:

```
> faktor_geschlecht <-factor(geschlecht)
> faktor_geschlecht
[1] M W M W W M
Levels: M W
```

In der letzten Zeile der Ausgabe wird `Levels` verwendet; dies ist der Begriff, den R für Kategorien benutzt.

Die Funktion `data.frame()` erstellt aus vorhandenen Vektoren einen Datensatz:

```
> d <- data.frame(name,faktor_geschlecht,laenge,gewicht)
> d
     name faktor_geschlecht laenge gewicht
1      al                M    182      88
2 barbara                W    162      53
3 charles                M    185      93
4   donna                W    165      55
5   ellen                W    167      56
6    fred                M    180      90
```

Wollen Sie die Länge der dritten Person ermitteln?

```
> d[3,3]
[1] 185
```

Und wie erhalten Sie alle Informationen zu dritten Person?

```
> d[5,]
   name faktor_geschlecht laenge gewicht
5 ellen                W    167      56
```

Wie Listen, so verwenden auch Datensätze das Dollarzeichen. In diesem Kontext kennzeichnet das Dollarzeichen eine Spalte:

```
> d$laenge
[1] 182 162 185 165 167 180
```

Sie können auch Statistiken berechnen, wie beispielsweise den Mittelwert der Länge:

```
> mean(d$laenge)
[1] 173.5
```

Wie Sie es bereits bei den Listen gesehen haben, können Sie die Filterkriterien in eckige Klammern einfassen. Dies wird oft bei Datensätzen gemacht, um Daten einer bestimmten Kategorie zusammenzufassen und zu analysieren. So berechnen Sie die durchschnittliche Länge der Frauen:

```
> mean(d$laenge[d$faktor_geschlecht == "W"])
[1] 164.6667
```

Bei den doppelten Gleichheitszeichen in den eckigen Klammern (==) handelt es sich um einen *logischen Operator*. In diesem Fall lautet die Bedingung »wenn d$faktor_geschlecht ist gleich W«.

Das doppelte Gleichheitszeichen (a==b) unterscheidet den logischen Operator (wenn a gleich b) vom Zuweisungsoperator (a=b; weise a den Wert von b zu).

Ja ich weiß – ich habe ausführlich die Funktion `factor()` beschrieben und erläutert, warum es besser ist, Kategorien (Levels) zu verwenden als Zeichenfolgen. Dennoch musste ich das `W` in den eckigen Klammern mit Anführungszeichen einfassen. R ist, was das betrifft, etwas sonderbar.

Wenn Sie die Dollarzeichen aus Ihrem R-Code eliminieren wollen, können Sie die Funktion `with()` verwenden. Sie geben Ihren Code in die runden Klammern nach dem ersten Argument ein, der die Daten angibt, die Sie verwenden.

```
> with(d,mean(laenge[faktor_geschlecht == "W"]))
```

ist mit dieser identisch:

```
> mean(d$laenge[d$faktor_geschlecht == "W"])
```

Wie viele Zeilen befinden sich im Datensatz?

```
> nrow(d)
[1] 6
```

Und wie viele Spalten?

```
> ncol(d)
[1] 4
```

Um in den Datensatz eine weitere Spalte einzufügen, verwende ich `cbind()`. Beginnen Sie mit einem Vektor aus Werten:

```
> begabung <- c(35,20,32,22,18,15)
```

Fügen Sie dann diesen Vektor als weitere Spalte hinzu:

```
> d.beg <- cbind(d,begabung)
> d.beg
     name faktor_geschlecht laenge gewicht begabung
1      al                 M    182      88       35
2 barbara                 W    162      53       20
3 charles                 M    185      93       32
4   donna                 W    165      55       22
5   ellen                 W    167      56       18
6    fred                 M    180      90       15
```

for-Schleifen und if-Anweisungen

Wie viele andere Programmiersprachen besitzt auch R die Möglichkeit, über die Strukturen zu iterieren, um Dinge zu erledigen. In R wird dies `for`-Schleife genannt. Und wie in vielen Programmiersprachen können Sie auch in R einen Test gegen eine Bedingung durchführen, und zwar mit der `if`-Anweisung.

Das allgemeine Format einer `for`-Schleife lautet:

```
For zaehler in start:ende{
   Anweisung 1


   Anweisung n
}
```

Die Syntax der einfachsten, allgemeinen `if`-Anweisung lautet:

```
If(test) {Anweisung, die ausgeführt wird, wenn test WAHR ist}
else {Anweisung, die ausgeführt wird, wenn test FALSCH ist}
```

Hier folgt ein Beispiel, in dem beides verwendet wird. Ich habe den Vektor `xx`:

```
> xx
[1] 2 3 4 5 6
```

Und einen zweiten Vektor `yy`, der derzeit nichts enthält:

```
> yy <- NULL
```

Ich möchte, dass die Komponenten in `yy` die Komponenten in `xx` widerspiegeln: Wenn eine Zahl in `xx` eine ungerade Zahl ist, soll die entsprechende Komponente in `yy` den Text `"UNGLEICH"` enthalten. Wenn die Zahl in `xx` gerade ist, soll die betreffende Komponente in `yy` den Text `"GERADE"` enthalten.

Wie teste ich eine Zahl darauf, ob sie gerade oder ungerade ist? Mathematiker haben die Berechnung des Modulo entwickelt, der den Restwert einer Division ermittelt. Wenn Sie *a* durch *b* dividieren und das Ergebnis der Rest *r* ist, sagen Mathematiker »a modulo b ist r«. Wenn Sie beispielsweise 10 durch 3 dividieren, erhalten Sie den Restwert 1; 10 modulo 3 ist also 1. Üblicherweise wird Modulo mit *mod* abgekürzt. R besitzt einen eigenen Modulo-Operator, der durch zwei Prozentzeichen (`%%`) symbolisiert wird.

```
> 10 %% 3
[1] 1
> 5 %% 2
[1] 1
> 4 %% 2
[1] 0
```

Ich vermute, Sie wissen, was ich meine: wenn `xx[i] %% 2 == 0`, dann ist der Wert in `xx[i]` gerade und anderenfalls ungerade.

Hier ist dann die `for`-Schleife mit eingebetteter `if`-Anweisung:

```
for (i in 1:length(xx)){
  if (xx[i] %% 2== 0) {yy[i] <- "GERADE"}
  else {yy[i] <- "UNGERADE"}
}

> yy
[1] "GERADE"   "UNGERADE" "GERADE"   "UNGERADE" "GERADE"
```