IN DIESEM KAPITEL

- Einführung in die Statistik
- R und RStudio auf Ihrem Computer installieren
- Eine Sitzung mit R starten
- Mit Funktionen in R arbeiten
- Mit Strukturen in R arbeiten

Kapitel 1
R – Was es macht und wie es dies macht

Sie sind also bereit für eine Reise in die wunderbare Welt von R! R wurde von Statistikern und Datenwissenschaftlern für Statistiker und Datenwissenschaftler entwickelt und hat eine kurze, aber illustre Geschichte.

In den 1990er Jahren wurde R von Ross Ihaka und Robert Gentleman an der Universität von Auckland, Neuseeland, entwickelt. Das R-Core-Team und die R-Foundation for Statistical Computing unterstützen R, das weltweit eine große Nutzerbasis hat.

Bevor ich Ihnen jedoch von R erzähle, muss ich Ihnen die Welt vorstellen, in der sich R bewegt – die Welt der Daten und Statistiken.

Statistische (und verwandte) Konzepte, die man einfach kennen muss

Die in R bereitgestellten Analysewerkzeuge basieren auf statistischen Konzepten, die ich Ihnen in diesem Abschnitt genauer vorstellen werde. Wie Sie sehen werden, beruhen diese Konzepte auf gesundem Menschenverstand.

Stichproben und Grundgesamtheiten

Wenn Sie in der Wahlnacht fernsehen, wissen Sie, dass eines der wichtigsten Ereignisse die Hochrechnung des Ergebnisses unmittelbar nach Schließung der Wahllokale ist (noch bevor alle Stimmen ausgezählt sind). Wie kommt es, dass die Experten fast immer richtig liegen?

Die Idee ist, mit einer *Stichprobe* von Wählern direkt nach deren Stimmabgabe zu sprechen. Wenn die Befragten wahrheitsgemäß angeben, wie sie ihre Stimmzettel ausgefüllt haben, und wenn die Stichprobe repräsentativ für die *Grundgesamtheit* (auch als *Population* bezeichnet) der Wähler ist, können die Analysten anhand der Stichprobendaten Rückschlüsse auf die Gesamtheit ziehen.

Genau darum geht es in der Statistik – die Daten von Stichproben zu nutzen, um Rückschlüsse auf die Grundgesamtheit zu ziehen.

Ein weiteres Beispiel: Stellen Sie sich vor, Sie sollen die Durchschnittsgröße der 10-jährigen Kinder in Deutschland ermitteln. Da Sie wahrscheinlich nicht die Zeit oder die Mittel haben, jedes einzelne Kind zu messen, messen Sie die Größen innerhalb einer repräsentativen Stichprobe. Dann ermitteln Sie den Durchschnitt dieser Größen und verwenden diesen als Schätzung für den Durchschnitt der Grundgesamtheit.

Die Schätzung des Durchschnitts für eine Grundgesamtheit ist eine der *Schlussfolgerungen* oder *Inferenzen*, die Statistiker aus Stichprobendaten ziehen. Im späteren Abschnitt »Inferenzstatistik: Hypothesen testen« werde ich ausführlicher auf die Inferenz eingehen.

Wichtige Terminologie: Eigenschaften einer Grundgesamtheit (wie beispielsweise der Durchschnitt der Grundgesamtheit) werden als *Parameter* bezeichnet, und Eigenschaften einer Stichprobe (wie beispielsweise der Stichprobendurchschnitt) werden als *Statistiken* bezeichnet. Wenn es Ihnen nur um die Eigenschaften der Stichprobe geht (zum Beispiel die Größe der Kinder in Ihrer Stichprobe), sind die von Ihnen berechneten Statistiken *deskriptiv*. (Um deskriptive Statistiken geht es in Teil 2.) Wenn es Ihnen darum geht, die Eigenschaften der Grundgesamtheit zu schätzen, sind Ihre Statistiken *inferentiell*. (Um Inferenzstatistiken geht es in Teil 3.)

Und jetzt eine wichtige Konvention, was die Notation betrifft: Statistiker verwenden griechische Buchstaben (zum Beispiel μ, σ, ρ) für Parameter und lateinische Buchstaben (zum Beispiel $\overline{X}$, s, r) für Statistiken. Abbildung 1.1 veranschaulicht die Beziehung zwischen Grundgesamtheiten und Stichproben sowie zwischen Parametern und Statistiken.

Variablen: abhängige und unabhängige

Eine *Variable* ist ein Platzhalter, der unterschiedliche Werte annehmen kann – beispielsweise Ihr Alter, den Wert des Euros im Vergleich zu anderen Währungen oder die Anzahl der Spiele, die Ihre Lieblingsmannschaft gewonnen hat. Ein Platzhalter, der nur einen einzigen Wert haben kann, ist eine *Konstante*. Die Wissenschaftler sagen, die Lichtgeschwindigkeit

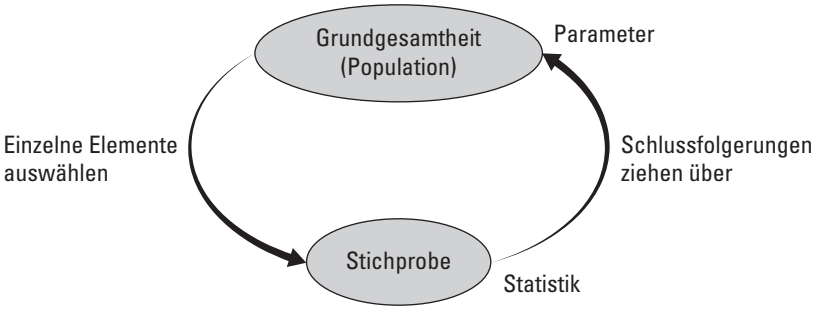

Abbildung 1.1: Beziehungen zwischen Grundgesamtheiten, Stichproben, Parametern und Statistiken.

ist eine Konstante, und wir verwenden die Konstante π, um die Fläche eines Kreises zu berechnen.

Statistiker arbeiten mit *unabhängigen* Variablen und mit *abhängigen* Variablen. In jeder Studie oder jedem Experiment finden Sie beide Arten. Statistiker bewerten die Beziehung zwischen ihnen.

Stellen Sie sich zum Beispiel eine computergestützte Trainingsmethode vor, die den IQ einer Person erhöhen soll. Wie würde ein Forscher herausfinden, ob diese Methode das tut, was sie tun soll? Zunächst würde der Forscher eine Stichprobe von Personen nach dem Zufallsprinzip in zwei Gruppen unterteilen. Für die eine Gruppe wird die untersuchte Trainingsmethode angewendet, während die andere eine andere Art von computergestützter Aktivität ausführt, zum Beispiel einen Text auf einer Website lesen. Bevor und nachdem jede der beiden Gruppen ihre Aktivitäten beendet hat, misst der Forscher den IQ der einzelnen Personen. Wie geht es weiter? Um dieses Thema geht es im späteren Abschnitt »Inferenzstatistik: Hypothesen testen«.

Stellen Sie sich hier einfach vor, dass die unabhängige Variable die Art der Aktivität darstellt. Die beiden möglichen Werte für diese Variable sind `IQ–Training` und `Text lesen`. Die abhängige Variable ist die Veränderung des IQ von vorher zu nachher.

Eine abhängige Variable ist das, was ein Forscher *misst*. In einem Experiment ist eine unabhängige Variable das, was ein Forscher *manipuliert*. In anderen Situationen kann es sein, dass ein Forscher eine unabhängige Variable nicht manipulieren kann. Stattdessen könnten beispielsweise natürlich vorkommende Werte der unabhängigen Variablen und ihre Auswirkungen auf eine abhängige Variable beobachtet werden.

Im Allgemeinen geht es darum, herauszufinden, ob Änderungen einer unabhängigen Variablen mit Änderungen einer abhängigen Variablen verknüpft sind.

In den Beispielen in diesem Buch zeige ich Ihnen, wie Sie R verwenden können, um die Merkmale von Gruppen von Ergebnissen zu berechnen oder Gruppen von Ergebnissen zu

vergleichen. Immer wenn ich Ihnen eine Gruppe von Ergebnissen zeige, spreche ich von den Werten einer abhängigen Variable.

Verschiedene Arten von Daten

Bei der Arbeit mit Statistiken können Sie auf vier Arten von Daten stoßen. Wie Sie mit Variablen arbeiten, hängt davon ab, um welche Art von Daten es sich handelt.

Die erste Art sind *nominale* Daten. Wenn es sich bei einer Reihe von Zahlen um nominale Daten handelt, sind die Zahlen nichts weiter als Beschriftungen – ihre Werte haben keine Bedeutung. Bei einer Sportmannschaft beispielsweise sind die Rückennummern nominal. Sie identifizieren lediglich die Spieler.

Die nächste Art sind die *ordinalen* Daten. Bei diesem Datentyp sind die Zahlen mehr als nur Beschriftungen. Wie der Name *ordinal* schon sagt, ist die Reihenfolge der Zahlen wichtig. Wenn ich Sie bitte, zehn Lebensmittel in eine Rangfolge zu bringen, von dem, das Sie am liebsten (1) bis zu dem, das Sie am wenigsten (10) mögen, dann erhalten wir eine Reihe ordinaler Daten.

Aber der Unterschied zwischen Ihrem drittliebsten und Ihrem viertliebsten Lebensmittel ist vielleicht nicht derselbe wie der zwischen Ihrem neuntliebsten und Ihrem zehntliebsten Lebensmittel. Bei dieser Art von Daten fehlt es also an gleichen Intervallen und gleichen Differenzen.

Intervalldaten liefern uns gleiche Differenzen. Die Fahrenheit-Skala für die Temperatur ist ein gutes Beispiel. Die Differenz zwischen 30 °F und 40 °F ist derselbe wie der zwischen 90 °F und 100 °F. Jedes Grad ist also ein Intervall.

Die Menschen sind manchmal überrascht, wenn sie feststellen, dass auf der Fahrenheit-Skala eine Temperatur von 80 °F nicht doppelt so heiß ist wie 40 °F. Damit Verhältnisangaben (»doppelt so viel wie«, »halb so viel wie«) einen Sinn ergeben, muss *Null* die völlige Abwesenheit der Sache bedeuten, die Sie messen. Eine Temperatur von 0 °F bedeutet jedoch nicht die völlige Abwesenheit von Wärme – es ist nur ein willkürlicher Punkt auf der Fahrenheit-Skala. (Das Gleiche gilt für Celsius.)

Die vierte Art von Daten, *Verhältnisdaten*, liefert einen aussagekräftigen Nullpunkt. Auf der Kelvin-Temperaturskala bedeutet der *Nullpunkt* den absoluten Nullpunkt, an dem alle Molekularbewegungen (die Grundlage der Wärme) aufhören. 200 Grad Kelvin ist also doppelt so heiß wie 100 Grad Kelvin. Ein weiteres Beispiel ist die Länge. Acht Zoll sind doppelt so lang wie 4 Zoll. *Null Zoll* bedeutet das völlige Fehlen von Länge.

Eine unabhängige oder abhängige Variable kann entweder nominale, ordinale, Intervall- oder Verhältnisdaten enthalten. Welche Analysetools Sie verwenden, hängt von der Art der Daten ab, mit denen Sie arbeiten.

Eine kleine Wahrscheinlichkeit

Wenn Statistiker Entscheidungen treffen, verwenden sie die Wahrscheinlichkeitsrechnung, um ihr Vertrauen in diese Entscheidungen auszudrücken. Sie können sich nie absolut sicher

sein, ob sie die richtige Entscheidung getroffen haben. Sie können Ihnen nur sagen, wie wahrscheinlich ihre Schlussfolgerungen sind.

Was verstehen wir unter *Wahrscheinlichkeit*? Mathematiker und Philosophen könnten Ihnen komplexe Definitionen nennen. Meiner Erfahrung nach lässt sich die Wahrscheinlichkeit jedoch am besten anhand von Beispielen verstehen.

Hier ein einfaches Beispiel: Angenommen, Sie werfen eine Münze. Wie groß ist die Wahrscheinlichkeit, dass sie Kopf zeigt? Wenn die Münze fair ist, könnte man annehmen, dass die Wahrscheinlichkeit für Kopf 50:50 und für Zahl 50:50 beträgt. Genau das ist der Fall. In den in der Wahrscheinlichkeitsrechnung verwendeten Zahlen ausgedrückt, ist das Ergebnis 1/2.

Stellen Sie sich vor, Sie würfeln mit einem fairen Würfel. Wie hoch ist die Wahrscheinlichkeit, dass Sie eine 4 würfeln? Ein Würfel hat sechs Seiten, eine davon zeigt die 4, also ist die Wahrscheinlichkeit 1/6.

Noch ein Beispiel: Wählen Sie eine Karte nach dem Zufallsprinzip aus einem Standardsatz von 52 Karten aus. Wie groß ist die Wahrscheinlichkeit, dass es sich um Karo handelt? Ein Kartensatz hat vier Farben, also ist die Wahrscheinlichkeit 1/4.

Diese Beispiele zeigen Ihnen, dass Sie, wenn Sie die Wahrscheinlichkeit des Eintretens eines Ereignisses wissen wollen, zählen müssen, auf wie viele Arten dieses Ereignis eintreten kann, und durch die Gesamtzahl der möglichen Ereignisse dividieren müssen. In den ersten beiden Beispielen (Kopf, 4) tritt das gewünschte Ereignis nur auf eine Weise ein. Für die Münze teilen wir 1 durch 2, für den Würfel teilen wir 1 durch 6. Im dritten Beispiel (Karo) kann das Ereignis auf 13 Arten eintreten (Ass bis König), also teilen wir 13 durch 52 (was gekürzt 1/4 ergibt).

Nun zu einem etwas komplizierteren Beispiel. Werfen Sie eine Münze und werfen Sie gleichzeitig einen Würfel. Wie hoch ist die Wahrscheinlichkeit für Zahl und eine 4? Denken Sie an alle möglichen Ereignisse, die eintreten können, wenn Sie gleichzeitig eine Münze werfen und einen Würfel werfen. Sie könnten Zahl und 1 bis 6 oder Kopf und 1 bis 6 haben. Das sind insgesamt 12 Möglichkeiten. Die Kombination aus Zahl und 4 kann nur auf eine Weise eintreten. Die Wahrscheinlichkeit ist also 1/12.

Im Allgemeinen lautet die Formel für die Wahrscheinlichkeit, dass ein bestimmtes Ereignis eintritt

$$Ws(Ereignis) = \frac{Anzahl\ der\ Möglichkeiten, wie\ das\ Ereignis\ auftreten\ kann}{Gesamtzahl\ der\ möglichen\ Ereignisse}$$

Zu Beginn dieses Abschnitts habe ich erklärt, dass Statistiker ihr Vertrauen in ihre Schlussfolgerungen in Form von Wahrscheinlichkeiten ausdrücken, weshalb ich das Thema überhaupt erst angesprochen habe. Diese Denkweise führt zur *bedingten* Wahrscheinlichkeit – der Wahrscheinlichkeit, dass ein Ereignis eintritt, wenn ein anderes Ereignis eintritt. Nehmen wir an, ich würfle mit einem Würfel, schaue ihn an (und zwar so, dass Sie ihn nicht sehen) und sage Ihnen, dass ich eine ungerade Zahl gewürfelt habe. Wie hoch ist die Wahrscheinlichkeit, dass ich eine 5 gewürfelt habe? Normalerweise ist die Wahrscheinlichkeit für eine 5 gleich 1/6, aber »ich habe eine ungerade Zahl gewürfelt« schränkt sie ein. Mit dieser Information scheiden die drei geraden Zahlen (2, 4, 6) als

Möglichkeiten aus. Nur die drei ungeraden Zahlen (1, 3, 5) sind möglich, also ist die Wahrscheinlichkeit 1/3.

Was hat es mit der bedingten Wahrscheinlichkeit auf sich? Welche Rolle spielt sie in der statistischen Analyse? Lesen Sie weiter!

Inferenzstatistik: Hypothesen testen

Bevor ein Statistiker eine Studie durchführt, formuliert er eine vorläufige Erklärung – eine *Hypothese*, die besagt, warum die Daten in einer bestimmten Weise ausfallen könnten. Nachdem alle Daten gesammelt wurden, muss der Statistiker entscheiden, ob er die Hypothese ablehnt.

Diese Entscheidung ist die Antwort auf eine Frage der bedingten Wahrscheinlichkeit – wie hoch ist die Wahrscheinlichkeit, die Daten zu erhalten, wenn die Hypothese richtig ist? Statistiker verfügen über Instrumente zur Berechnung dieser Wahrscheinlichkeit. Fällt die Wahrscheinlichkeit gering aus, lehnt der Statistiker die Hypothese ab.

Betrachten wir für ein Beispiel wieder den Münzwurf: Angenommen, Sie interessieren sich dafür, ob eine bestimmte Münze fair ist – also, ob sie bei jedem Wurf die gleiche Chance auf Kopf oder Zahl hat. Beginnen wir mit der Hypothese »Die Münze ist fair«.

Um die Hypothese zu testen, werfen Sie die Münze eine bestimmte Anzahl mal, beispielsweise 100-mal. Diese 100 Würfe sind die Stichprobendaten. Wenn die Münze fair ist (gemäß der Hypothese), würden Sie 50-mal Kopf und 50-mal Zahl erwarten.

Bei 99-mal Kopf und 1-mal Zahl würden Sie die Hypothese der fairen Münze sicher ablehnen: Die bedingte Wahrscheinlichkeit von 99-mal Kopf und 1-mal Zahl bei einer fairen Münze ist sehr gering. Natürlich könnte die Münze immer noch fair sein, und Sie könnten ganz zufällig ein Ergebnis von 99:1 erhalten, oder? Natürlich. Man weiß es nie so genau. Man muss die Stichprobendaten (die 100 Würfe) sammeln und dann entscheiden. Ihre Entscheidung kann richtig sein – oder auch nicht.

Geschworene treffen diese Art von Entscheidungen. In den Vereinigten Staaten geht man von der Hypothese aus, dass der Angeklagte nicht schuldig ist (»unschuldig, bis seine Schuld bewiesen ist«). Stellen Sie sich die Beweise als Daten vor. Die Geschworenen betrachten die Beweise und beantworten eine Frage zur bedingten Wahrscheinlichkeit: Wie groß ist die Wahrscheinlichkeit, dass die Beweise zutreffen, wenn der Angeklagte nicht schuldig ist? Ihre Antwort bestimmt das Urteil.

Null- und Alternativhypothesen

Denken Sie noch einmal an die Studie zum Münzwurf, die ich gerade beschrieben habe. Die Stichprobendaten sind die Ergebnisse von 100 Münzwürfen. Ich habe erklärt, dass wir mit der Hypothese beginnen können, dass die Münze fair ist. Dieser Ausgangspunkt wird als *Nullhypothese* bezeichnet. Die statistische Schreibweise für die Nullhypothese ist H_0. Nach dieser Hypothese ist jede Kopf-Zahl-Aufteilung in den Daten mit einer fairen Münze vereinbar. Man kann sich das so vorstellen, dass nichts in den Stichprobendaten ungewöhnlich ist.

Eine alternative Hypothese ist möglich, nämlich, dass die Münze nicht fair ist und eine ungleiche Anzahl von Kopf und Zahl produziert. Diese Hypothese besagt, dass jede Aufteilung in Kopf und Zahl mit einer nicht fairen Münze übereinstimmt. Dies wird als *Alternativhypothese* bezeichnet. Die statistische Schreibweise für die Alternativhypothese lautet H_1.

Werfen Sie nun die Münze 100-mal und notieren Sie die Anzahl der Kopf- und Zahlwürfe. Wenn die Ergebnisse etwa 90 Kopf und 10 Zahl sind, ist es naheliegend, H_0 zu verwerfen. Wenn die Ergebnisse etwa 50 Kopf und 50 Zahl sind, verwerfen Sie H_0 nicht.

Ähnliche Überlegungen gelten für das oben beschriebene IQ-Beispiel. Eine Gruppe erhält die computergestützte IQ-Trainingsmethode, die andere nimmt an einer anderen computergestützten Aktivität teil, zum Beispiel dem Lesen von Text auf einer Webseite. Bevor und nachdem jede Gruppe ihre Aktivitäten abgeschlossen hat, misst der Forscher den IQ der einzelnen Personen. Die Nullhypothese, H_0, besagt, dass sich die Verbesserung der einen Gruppe nicht von derjenigen der anderen unterscheidet. Wenn die Verbesserungen beim IQ-Training größer sind als bei der anderen Aktivität – so viel größer, dass es unwahrscheinlich ist, dass sich die beiden nicht voneinander unterscheiden –, verwerfen Sie H_0. Andernfalls verwerfen Sie H_0 nicht.

Beachten Sie, dass ich *nicht* gesagt habe: »Akzeptieren Sie H_0.« So wie die Logik funktioniert, akzeptiert man *niemals* eine Hypothese. Entweder lehnt man H_0 ab oder man lehnt H_0 nicht ab. Bei einem Geschworenenprozess lautet das Urteil entweder »schuldig« (Ablehnung der Nullhypothese »nicht schuldig«) oder »nicht schuldig« (keine Ablehnung von H_0). »Unschuldig« (Akzeptanz der Nullhypothese) ist keine mögliche Entscheidung.

Beachten Sie auch, dass ich in dem Beispiel mit dem Münzwurf gesagt habe: »etwa 50 Kopf und 50 Zahl«. Was bedeutet »etwa«? Außerdem habe ich gesagt, dass H_0 bei 90-10 abgelehnt wird. Wie sieht es mit 85-15 aus? 80-20? 70-30? Wie stark muss sich die Aufteilung von 50-50 unterscheiden, damit Sie H_0 ablehnen? Wie viel größer muss die IQ-Verbesserung im Beispiel des IQ-Trainings sein, um H_0 zu verwerfen?

Ich beantworte diese Fragen jetzt nicht. Statistiker haben für solche Situationen Entscheidungsregeln formuliert, die ich in Teil 3 erklären werde.

Zwei Arten von Fehlern

Wenn Sie Daten auswerten und sich entscheiden, H_0 abzulehnen oder H_0 nicht abzulehnen, können Sie sich nie absolut sicher sein. Man kennt nie wirklich den »wahren« Zustand der Welt. Im Beispiel des Münzwurfs bedeutet das, dass Sie nicht sicher sein können, ob die Münze fair ist. Sie können lediglich eine Entscheidung auf der Grundlage der Beispieldaten treffen. Wenn man sicher sein will, dass die Münze fair ist, muss man die Daten für die gesamte Population der Münzwürfe haben – was bedeutet, dass man die Münze bis ans Ende aller Zeiten werfen muss.

Da man sich seiner Entscheidungen nie sicher sein kann, kann man bei jeder Entscheidung einen Fehler machen. Wie ich bereits erwähnt habe, könnte die Münze fair sein, aber Sie erhalten zufällig 99-mal Kopf bei 100 Würfen. Das ist nicht wahrscheinlich, deshalb lehnen

Sie H_0 ab, wenn das passiert. Es ist auch möglich, dass die Münze verzerrt ist und man bei 100 Würfen zufällig 50-mal Kopf wirft. Auch das ist nicht wahrscheinlich, aber in diesem Fall lehnen Sie H_0 nicht ab.

Obwohl diese Fehler nicht wahrscheinlich sind, sind sie möglich. Sie lauern in jeder Studie, die Inferenzstatistiken verwendet. Statistiker bezeichnen sie als Fehler Typ I und Fehler Typ II.

Wenn Sie H_0 ablehnen, dies aber nicht tun sollten, ist das ein Fehler *Typ I*. Im Beispiel mit der Münze bedeutet dies, dass die Hypothese, dass die Münze fair ist, abgelehnt wird, obwohl es sich in Wirklichkeit um eine faire Münze handelt.

Wenn Sie H_0 nicht ablehnen, obwohl Sie es hätten tun müssen, ist das ein Fehler *Typ II*. Er tritt auf, wenn Sie die Hypothese, dass die Münze fair ist, nicht ablehnen, obwohl sie in Wirklichkeit tatsächlich verzerrt ist.

Wie können Sie wissen, ob Sie einen der beiden Fehler gemacht haben? Sie wissen es nicht – zumindest nicht direkt, nachdem Sie die Entscheidung getroffen haben, H_0 abzulehnen oder nicht abzulehnen. (Wenn es möglich wäre, das zu wissen, würden Sie den Fehler gar nicht erst machen!) Alles, was Sie tun können, ist, weitere Daten zu sammeln und zu prüfen, ob die zusätzlichen Daten mit Ihrer Entscheidung übereinstimmen.

Stellt man sich H_0 als Tendenz vor, den Status quo beizubehalten und nichts als ungewöhnlich zu interpretieren (egal, wie es aussieht), bedeutet ein Fehler vom Typ II, dass man etwas Großes verpasst hat. In der Tat sind einige legendäre Fehlentscheidungen Fehler vom Typ II.

Hier ein Beispiel. Am Neujahrstag 1962 sprach eine Rockgruppe, bestehend aus drei Gitarristen und einem Schlagzeuger, im Londoner Studio einer großen Plattenfirma vor. Die Legende besagt, dass den Verantwortlichen der Plattenfirma nicht gefiel, was sie hörten und sahen, und dass sie glaubten, Gitarrengruppen seien nicht mehr gefragt. Obwohl die Musiker ihr Bestes gaben, fiel die Gruppe beim Vorspielen durch.

Wer war diese Gruppe? Die Beatles!

Und *das war* ein Fehler vom Typ II.

R beschaffen

Nachdem ich Ihnen die Welt vorgestellt habe, in der sich R bewegt, wollen wir ganz in R eintauchen.

Wenn Sie R noch nicht auf Ihrem Computer haben, müssen Sie es zunächst herunterladen und installieren.

Sie finden die Software auf der Website des Comprehensive R Archive Network (CRAN). Geben Sie in Ihrem Browser diese Webadresse ein:

```
https://cran.rstudio.com
```

Klicken Sie auf den entsprechenden Link, um R für Ihren Computer herunterzuladen.

RStudio beschaffen

Die Arbeit mit R ist viel einfacher, wenn Sie RStudio dafür verwenden. Computerfachleute bezeichnen RStudio als IDE (Integrated Development Environment). Betrachten Sie es als ein Werkzeug, das Ihnen hilft, Ihren R-Code zu schreiben, zu bearbeiten, auszuführen und im Auge zu behalten, und als eine Umgebung, die Sie mit einer Welt von hilfreichen Hinweisen über R verbindet.

Hier die Webadresse für dieses großartige Tool:

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

Klicken Sie auf den Link für das Installationsprogramm für Ihren Computer und folgen Sie den üblichen Installationsverfahren. (Sie benötigen RStudio Desktop.)

In diesem Buch arbeite ich mit R Version 4.2.0 und RStudio Version 2022.02.3 Build 492. Wenn Sie dieses Buch lesen, sind vielleicht schon neuere Versionen von beidem verfügbar. Übrigens hat jede Version von R ihren eigenen skurrilen Spitznamen. Die Version 4.2.0 heißt *Vigorous Calisthenics* (»Kräftige Gymnastik«). Warum? Ich habe keine Ahnung. Vielleicht ist es eine Weiterentwicklung der Vorgängerversion, *One Push-Up*.

Nachdem Sie die Installation von R und RStudio abgeschlossen haben, klicken Sie auf Ihr brandneues RStudio-Symbol, um das Fenster zu öffnen, das dem in Abbildung 1.2 gezeigten Fenster sehr ähnlich sieht. Es wird nicht exakt übereinstimmen, denn mein Verlauf mit RStudio – der sich im oberen rechten Fenster widerspiegelt – ist wahrscheinlich anders als der Ihre.

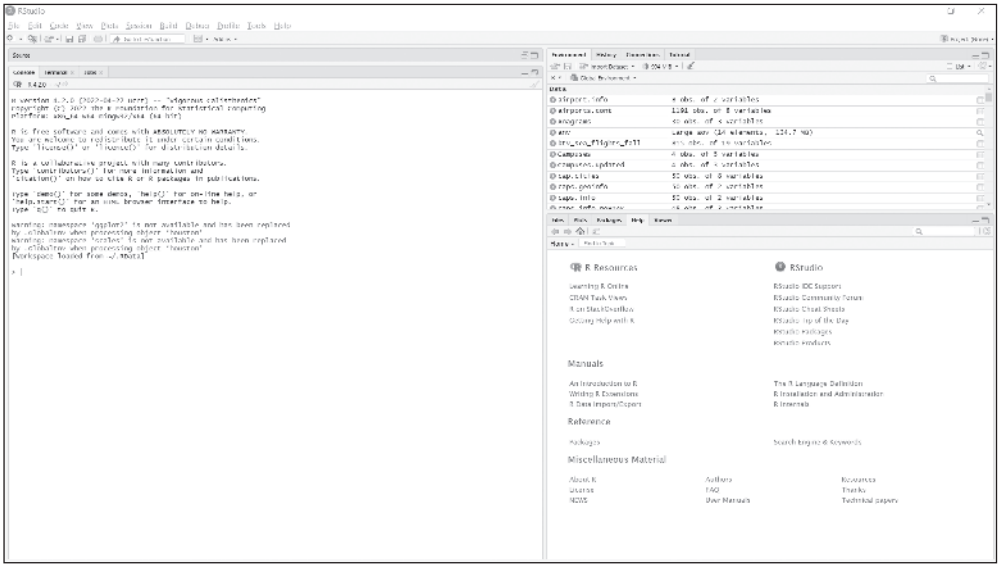

Abbildung 1.2: RStudio, unmittelbar nachdem Sie es installiert und auf das Symbol geklickt haben.

Der große Konsolenbereich auf der linken Seite führt R-Code aus. Eine Möglichkeit, R-Code auszuführen, besteht darin, ihn direkt in das Konsolenfenster einzugeben. Eine andere Möglichkeit zeige ich Ihnen gleich.

Die beiden anderen Fenster bieten hilfreiche Informationen für die Arbeit mit R. Das Fenster Environment and History (Umgebung und Verlauf) befindet sich oben rechts. Auf der Registerkarte Environment (Umgebung) werden die Dinge erfasst, die Sie bei der Arbeit mit R erstellen (in R als *Objekte* bezeichnet). Auf der Registerkarte History (Verlauf) wird der von Ihnen eingegebene R-Code verfolgt.

Gewöhnen Sie sich an das Wort *Objekt*. Alles in R ist ein Objekt.

Rechts unten befindet sich die Bereiche Files, Plots, Packages und Help (Dateien, Diagramme, Packages und Hilfe). Auf der Registerkarte Files (Dateien) werden die von Ihnen erstellten Dateien angezeigt. Die Registerkarte Plots (Diagramme) enthält Diagramme, die Sie aus Ihren Daten erstellen. Die Registerkarte Packages zeigt Add-ons (sogenannte *Packages*, manchmal auch als *Pakete* bezeichnet) an, die mit R heruntergeladen wurden. Um die Fähigkeiten eines Package nutzen zu können, ist ein weiterer Schritt erforderlich – und glauben Sie mir, Sie werden Packages nutzen wollen.

Abbildung 1.3 zeigt die Registerkarte Packages. Ich gehe später in diesem Teil auf Packages ein.

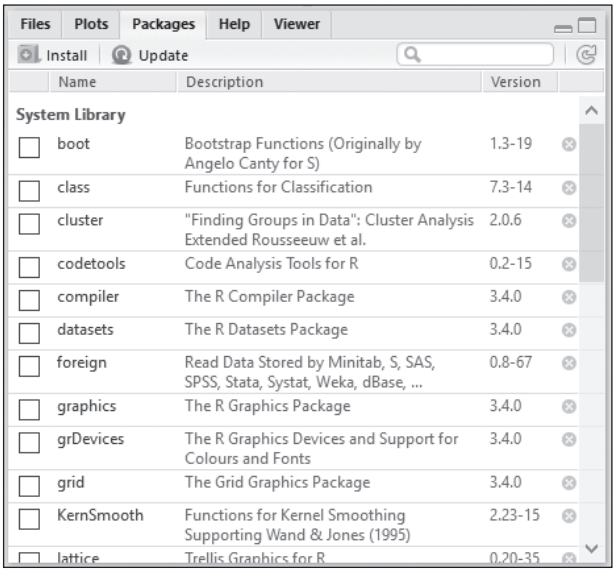

Abbildung 1.3: Die Registerkarte Packages in RStudio.

Die Registerkarte Help (Hilfe), die in Abbildung 1.4 dargestellt ist, bietet Ihnen eine Fülle von Informationen über R und RStudio.

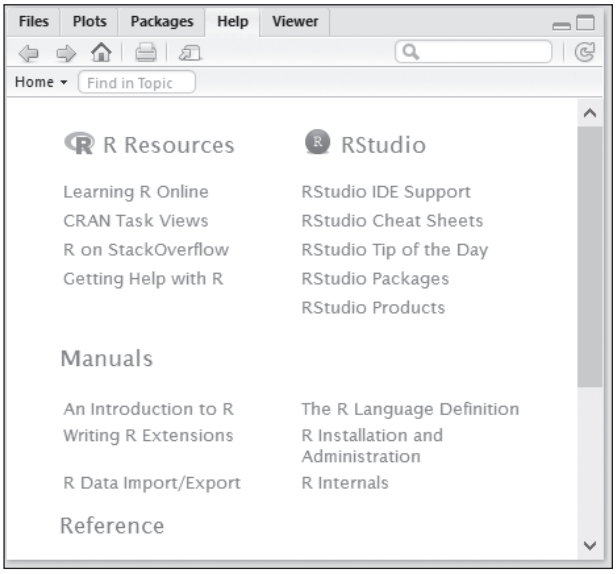

Abbildung 1.4: Die Registerkarte HELP (HILFE) in RStudio.

Um die volle Leistungsfähigkeit von RStudio als IDE zu nutzen, klicken Sie auf das Symbol in der rechten oberen Ecke des Konsolenfensters. (Es sieht aus wie ein großer Ordner mit einem grauen Band oben.) Dadurch ändert sich das Erscheinungsbild von RStudio, sodass es wie in Abbildung 1.5 gezeigt aussieht.

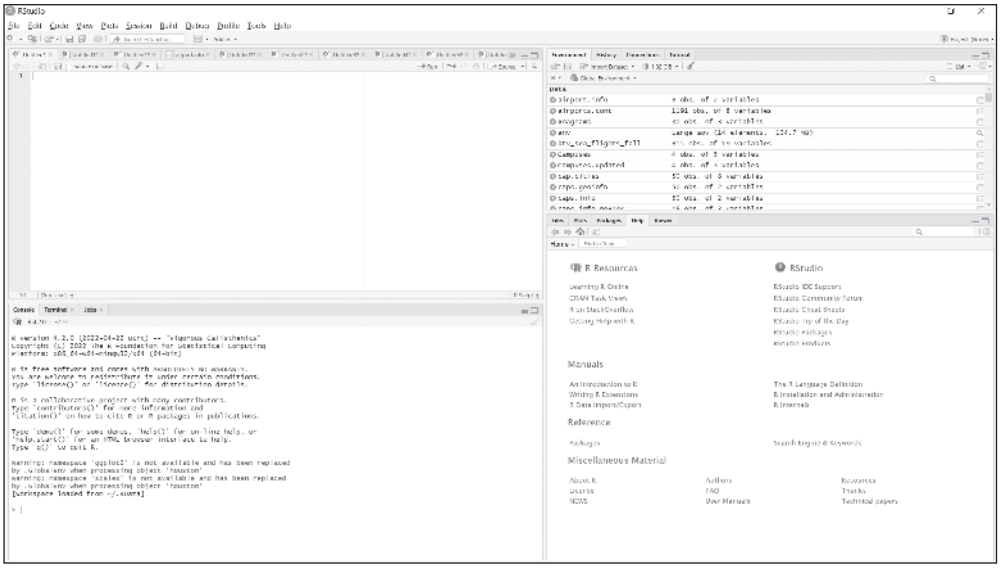

Abbildung 1.5: RStudio, nachdem Sie auf das Symbol in der oberen rechten Ecke des Konsolenfensters geklickt haben.

Der Konsolenbereich wird nach unten links verschoben. Der neue Bereich oben links ist der Bereich Scripts (Skripts). Sie geben den Code im Bereich Scripts ein und bearbeiten ihn, drücken Strg+← (cmd ⌘+← auf dem Mac) und der Code wird im Bereich Console (Konsole) ausgeführt.

Sie können auch Codezeilen im Script-Fenster markieren und Code | Run Selected Line(s) (Code | Ausgewählte Zeile(n) ausführen) aus dem Hauptmenü von RStudio wählen.

Eine Sitzung mit R

Bevor Sie mit der Arbeit beginnen, wählen Sie im Hauptmenü von RStudio File | Save As (Datei | Speichern unter) und speichern Sie das leere Fenster beispielsweise unter dem Namen *My First R Session*. Damit wird die Registerkarte im Bereich Scripts (Skripts) dem Namen der Datei zugeordnet und die Erweiterung `.R` wird hinzugefügt. Auch auf der Registerkarte Files (Dateien) wird der Dateiname (zusammen mit der Erweiterung `.R`) angezeigt.

Das Arbeitsverzeichnis

Wenn Sie meinem Rat folgen und etwas namens *My First R Session* speichern, was genau speichert R dann und wo speichert R es? Was R speichert, nennt sich *Workspace* (*Arbeitsbereich*), das heißt, die Umgebung, in der Sie arbeiten. R speichert den Arbeitsbereich im *Arbeitsverzeichnis*. Unter Windows ist das Standardarbeitsverzeichnis

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

Sollten Sie einmal den Pfad zu Ihrem Arbeitsverzeichnis vergessen, geben Sie

```
> getwd()
```

im Konsolenfenster ein und R zeigt den Pfad auf dem Bildschirm an.

Im Konsolenbereich müssen Sie die nach rechts zeigende Pfeilspitze am Anfang der Zeile nicht eingeben. Das ist eine Eingabeaufforderung, die standardmäßig vorhanden ist.

Mein Arbeitsverzeichnis sieht wie folgt aus:

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

Beachten Sie die Richtung, in der die Schrägstriche verlaufen, nämlich entgegengesetzt zu dem, was Sie normalerweise in Windows-Dateipfaden sehen. Das liegt daran, dass R den Backslash \ als *Escape-Zeichen* verwendet, was bedeutet, dass das, was auf das Zeichen \

folgt, etwas anderes bedeutet als das, was es normalerweise bedeutet. Zum Beispiel steht `\t` in R für die *Tabulatortaste*.

Sie können einen Windows-Dateipfad in R auch wie folgt schreiben:

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

Wenn Sie möchten, können Sie das Arbeitsverzeichnis ändern:

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

Eine andere Möglichkeit, das Arbeitsverzeichnis zu ändern, ist die Auswahl von Session | Set Working Directory | Choose Directory (Sitzung | Arbeitsverzeichnis festlegen | Verzeichnis auswählen) im Hauptmenü von RStudio.

Erste Schritte

Kommen wir zur Sache und schreiben wir R-Code. Geben Sie im Bereich Scripts (Skripts)

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

ein und drücken Sie dann Strg+←.

Dadurch wird diese Zeile in das Konsolenfenster eingefügt:

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

Wie in einem früheren Tipp erwähnt, ist die nach rechts zeigende Pfeilspitze (das Größer-als-Zeichen) eine Eingabeaufforderung, die R im Konsolenfenster anzeigt. Im Script-Fenster ist sie nicht zu sehen.

Und was hat R gerade gemacht? Das Pfeilzeichen besagt, dass *x* das zugewiesen wird, was rechts vom Pfeilzeichen steht. Betrachten Sie das Pfeilzeichen als den *Zuweisungsoperator* von R.

Die Zahlenmenge 5, 10, 15, 20 … 40 ist nun also *x* zugeordnet.

In der Sprache R ist ein solcher Satz Zahlen ein *Vektor*. Mehr zu diesem Thema erfahren Sie im späteren Abschnitt »R-Strukturen«. Das `c` vor der Klammer bewirkt, dass ein Vektor erstellt wird.

Sie können diese Codezeile als »x wird der Vektor 5, 10, 15, 20 … zugewiesen« lesen.

Geben Sie ***x*** in den Bereich Scripts (Skripts) ein und drücken Sie Strg+←. Im Konsolenbereich wird Folgendes angezeigt:

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

Die 1 in eckigen Klammern ist die Bezeichnung für die erste Zeile der Ausgabe. Das bedeutet also, dass 5 der erste Wert ist.

Hier haben Sie natürlich nur eine Zeile. Was passiert, wenn R viele Werte über viele Zeilen ausgibt? Jede Zeile erhält eine numerische Bezeichnung in eckigen Klammern, und die Zahl entspricht dem ersten Wert in der Zeile. Wenn die Ausgabe zum Beispiel aus 23 Werten besteht und der 18. Wert der erste in der zweiten Zeile ist, beginnt die zweite Zeile mit [18].

Durch das Erstellen des Vektors x wird die Zeile in Abbildung 1.6 auf der Registerkarte Environment (Umgebung) hinzugefügt.

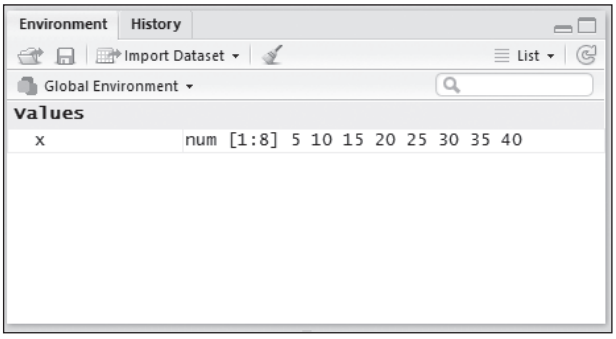

Abbildung 1.6: Eine Zeile auf der Registerkarte Environment (Umgebung) in RStudio nach der Erstellung des Vektors x.

Eine andere Möglichkeit, die Objekte in der Umgebung anzuzeigen, ist es,

```
ls()
```

in den Bereich Scripts (Skripte) einzugeben und dann Strg + ← zu drücken. Oder Sie können Folgendes

```
> ls()
```

direkt in den Konsolenbereich eingeben und ← drücken. In beiden Fällen ist das Ergebnis im Konsolenfenster

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

Jetzt können Sie mit x arbeiten. Addieren Sie zunächst alle Zahlen im Vektor. Durch die Eingabe von

```
sum(x)
```

im Bereich Scripts (Skripte) und anschließendes Drücken der Tastenkombination Strg + ← wird die folgende Zeile im Konsolenbereich ausgeführt:

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

Wie sieht es mit dem Durchschnitt der Zahlen im Vektor x aus?

Dazu verwenden Sie

```
mean(x)
```

im Bereich Scripts (Skripte) und anschließendes Drücken der Tastenkombination Strg + ← und erhalten

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

im Konsolenfenster.

Bei der Eingabe im Bereich Scripts (Skripts) oder im Konsolenbereich werden hilfreiche Informationen eingeblendet. Wenn Sie länger mit RStudio arbeiten, werden Sie schnell lernen, wie Sie diese Informationen nutzen können.

Die *Varianz* ist ein Maß dafür, wie stark sich eine Reihe von Zahlen von ihrem Mittelwert unterscheidet. So berechnen Sie mit R die Varianz:

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

Was genau ist Varianz und was bedeutet sie? Sie erfahren alles darüber in Teil 2.

Nachdem R alle diese Befehle ausgeführt hat, sieht die Registerkarte History (Verlauf) wie in Abbildung 1.7 gezeigt aus.

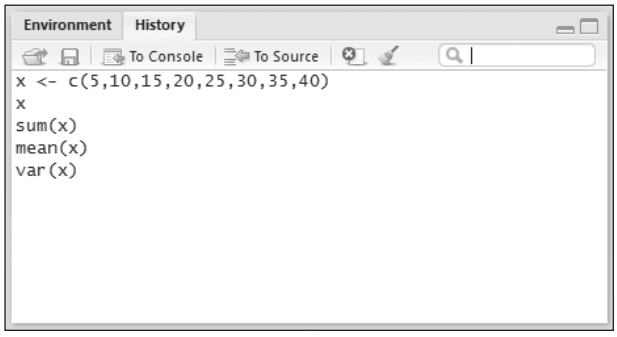

Abbildung 1.7: Die Registerkarte History (Verlauf) nach dem Erstellen und Bearbeiten eines Vektors.

Um eine Sitzung zu beenden, wählen Sie File | Quit Session (Datei | Sitzung beenden) im Hauptmenü von R Studio oder drücken Sie Strg + Q. Wie Abbildung 1.8 zeigt, öffnet sich ein Dialogfeld, in dem Sie gefragt werden, was Sie von der Sitzung speichern möchten. Wenn Sie die Optionen speichern, können Sie beim nächsten Öffnen von RStudio die Sitzung an der Stelle wieder öffnen, an der Sie aufgehört haben (obwohl das Konsolenfenster Ihre Arbeit nicht speichert).

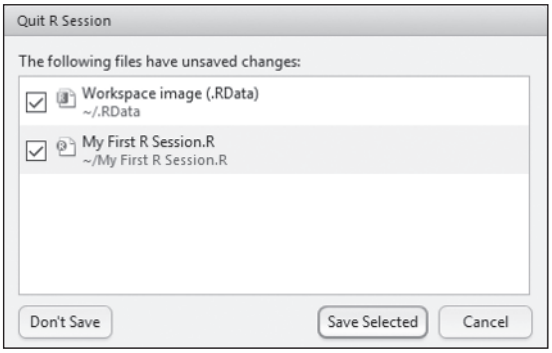

Abbildung 1.8: Das Dialogfeld QUIT R SESSION (R-SITZUNG BEENDEN).

In den weiteren Beispielen schreibe ich meistens nicht: »Geben Sie diesen Code in den Bereich Scripts (Skripts) ein und drücken Sie Strg+←«. Sie kennen diese Vorgehensweise jetzt, und ich zeige Ihnen einfach nur noch den Code und seine Ausgabe, wie im Beispiel `var()`.

Außerdem zeige ich den Code manchmal mit der Eingabeaufforderung > und manchmal ohne. Im Allgemeinen zeige ich die Eingabeaufforderung, wenn ich möchte, dass Sie den R-Code und seine Ergebnisse sehen. Ich zeige die Eingabeaufforderung nicht an, wenn ich Ihnen nur den R-Code zeigen möchte, den ich im Bereich Scripts (Skripts) erstellt habe.

Funktionen in R

Die Beispiele im vorangegangenen Abschnitt verwenden `c()`, `sum()` und `var()`. Dies sind drei in R eingebaute *Funktionen*. Jede dieser Funktionen besteht aus einem Funktionsnamen, der unmittelbar von Klammern gefolgt wird. Innerhalb der Klammern stehen die *Argumente*. Im Zusammenhang mit einer Funktion ist ein *Argument* nichts, was Sie in irgendwelchen Debatten vorbringen würden. Es ist der mathematische Begriff für das, worauf eine Funktion einwirkt.

Manchmal benötigt eine Funktion keine Argumente (wie im Fall von `ls()`). Die Klammern müssen trotzdem angegeben werden.

Die Funktionen in den zuvor gezeigten Beispielen sind ziemlich einfach: Sie geben ein Argument an, und die Funktion liefert ein Ergebnis. Einige Funktionen in R benötigen jedoch mehrere Argumente.

R bietet Ihnen mehrere Möglichkeiten, mit Funktionen mit mehreren Argumenten umzugehen. Eine Möglichkeit besteht darin, die Argumente in der Reihenfolge aufzulisten, in der sie in der Funktionsdefinition erscheinen. In R spricht man von »*positional mapping*«.

Hier ist ein Beispiel. Erinnern Sie sich an den Vektor x, den ich erstellt habe?

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

Eine andere Möglichkeit, einen Vektor aus diesen Zahlen zu erstellen, ist die Funktion `seq()`:

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

Stellen Sie sich `seq()` so vor, dass es eine »Sequenz« erzeugt. Das erste Argument für `seq()` ist die Zahl, mit der die Sequenz *beginnt* (5). Das zweite Argument ist die Zahl, die die Sequenz beendet – die Zahl, *bis zu der* die Sequenz *geht* (40). Das dritte Argument ist das Inkrement der Sequenz – der Betrag, um den sich die Sequenz jeweils *erhöht* (in diesem Fall 5).

Wenn Sie die Argumente *benennen*, ist es egal, wie Sie sie anordnen:

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

Wenn Sie also eine Funktion benennen und sie später verwenden, können Sie die Argumente der Funktion in einer beliebigen Reihenfolge platzieren. R nennt dies *Schlüsselwort-Matching*. Das ist praktisch, wenn Sie eine R-Funktion verwenden, die viele Argumente hat. Wenn Sie sich die Reihenfolge der Argumente nicht merken können, verwenden Sie ihre Namen, und die Funktion funktioniert.

Wenn Sie Hilfe zu einer bestimmten Funktion benötigen – zum Beispiel `seq()` –, geben Sie ***?seq*** ein und drücken Sie Strg + ←, um hilfreiche Informationen auf der Registerkarte Help (Hilfe) anzuzeigen.

Benutzerdefinierte Funktionen

R ermöglicht es Ihnen, Ihre eigenen Funktionen zu erstellen. Dieser Abschnitt beschreibt, wie das geht.

In R hat eine Funktion die folgende Form:

```
meineFunktion <- function(argument1, argument2, ...){
  Anweisungen
  return(Objekt)
}
```

Als Beispiel definieren wir hier eine Funktion für die Berechnung der Hypotenuse rechtwinkliger Dreiecke. Erinnern Sie sich? Ein rechtwinkliges Dreieck hat zwei Seiten, die einen rechten Winkel bilden, und eine dritte Seite, die *Hypotenuse*. Vielleicht erinnern Sie sich auch an den Satz des Pythagoras: Wenn eine Seite die Länge *a* und die andere Seite die Länge *b* hat, dann ist die Länge der Hypotenuse *c* gleich:

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

Hier folgt die einfache Funktion namens `hypotenuse()`, die zwei Zahlen ***a*** und ***b*** (die Längen der beiden Seiten eines rechtwinkligen Dreiecks) entgegennimmt und ***c***, die Länge der Hypotenuse, zurückgibt.

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

Geben Sie diesen Codeabschnitt in den Bereich Scripts (Skripts) ein und markieren Sie ihn. Drücken Sie dann Strg + ←. Im Konsolenbereich wird Folgendes angezeigt:

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

Jedes Pluszeichen ist eine *Fortsetzungsanzeige*. Es zeigt an, dass eine Zeile die vorangegangene Zeile fortsetzt.

Und so wird die Funktion verwendet:

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

Kommentare

Ein *Kommentar* ist eine Möglichkeit, Code zu kommentieren. Beginnen Sie einen Kommentar mit dem Rautensymbol, #. Es weist R an, alles rechts von ihm zu ignorieren.

Kommentare helfen allen, die den von Ihnen geschriebenen Code lesen und verstehen sollen. Ein Beispiel:

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

Ein Hinweis: Ich füge den Codezeilen in diesem Buch normalerweise keine Kommentare hinzu. Stattdessen gebe ich detaillierte Beschreibungen. In einem Buch wie diesem ist das meiner Meinung nach der beste Weg, um die Botschaft zu vermitteln.

Strukturen in R

Wie im Abschnitt »Funktionen in R« weiter oben in diesem Kapitel bereits erwähnt, kann eine Funktion in R viele Argumente haben. Und sie kann auch viele Ausgaben haben. Um

die möglichen Eingaben und Ausgaben zu verstehen, müssen Sie die Strukturen verstehen, mit denen R arbeitet.

Vektoren

Der *Vektor* ist die grundlegende Struktur in R. Ich habe Vektoren in früheren Beispielen bereits vorgestellt. Ein Vektor ist eine Anordnung von Elementen desselben Typs. Die Datenelemente in einem Vektor werden *Komponenten* genannt.

Um einen Vektor zu erstellen, verwenden Sie die Funktion `c()`, wie ich es im vorherigen Beispiel gemacht habe:

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

Bei dem Vektor `x` sind die Komponenten natürliche Zahlen.

In einem Zeichenkettenvektor sind die Komponenten in Anführungszeichen gesetzte Zeichenketten:

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

Darüber hinaus gibt es auch *logische Vektoren*, deren Komponenten `TRUE` und `FALSE` sind, oder die Abkürzungen `T` und `F`:

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

Um auf eine bestimmte Komponente eines Vektors zu verweisen, fügen Sie dem Vektornamen eine Zahl in Klammern an:

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

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

```
> beatles[2:3]
[1] "paul" "george"
```

Möchten Sie auf nicht aufeinanderfolgende Komponenten verweisen? Das ist ein bisschen komplizierter, aber mit `c()` machbar:

```
> beatles[c(2,4)]
[1] "paul" "ringo"
```

Numerische Vektoren

Zusätzlich zu `c()` bietet R zwei Abkürzungsfunktionen für die Erstellung numerischer Vektoren. Die eine, `seq()`, habe ich Ihnen bereits gezeigt:

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

Ohne das dritte Argument werden die Komponenten innerhalb der Folge um jeweils 1 inkrementiert:

```
> 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
   [20] 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
```

Auf meinem Bildschirm – und wahrscheinlich auch auf Ihrem – erscheinen alle Elemente aus y in einer Zeile. Die gedruckte Seite ist jedoch nicht so breit wie das Konsolenfenster. Dementsprechend habe ich die Ausgabe in zwei Zeilen aufgeteilt und die Zahl `[20]` ganz im Stil von R in Klammern hinzugefügt. Das mache ich im ganzen Buch dort, wo es nötig ist.

R hat eine spezielle Syntax für die Erstellung eines numerischen Vektors, dessen Elemente um jeweils 1 inkrementiert werden:

```
> y <- 5:40
> y
 [1] 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
   [20] 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:

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

Sie können auch einen Vektor als zweites Argument angeben:

```
> rep_vector <-c(1,2,3,4)
> repeated_quadrifecta <- rep(quadrifecta,rep_vector)
```

Der Vektor gibt die Anzahl der Wiederholungen für jedes Element an. Es passiert Folgendes:

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

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

Sie können `append()` verwenden, um ein Element am Ende eines Vektors hinzuzufügen:

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

Wie viele Elemente sind in einem Vektor enthalten? Das ermitteln Sie wie folgt:

```
> length(xx)
[1] 4
```

Matrizen

Eine *Matrix* ist eine zweidimensionale Anordnung von Datenelementen desselben Typs. Sie können eine Matrix aus Zahlen haben:

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

oder eine Matrix mit Zeichenketten:

```
"john"      "paul"     "george"  "ringo"
"groucho"   "harpo"    "chico"   "zeppo"
"levi"      "duke"     "larry"   "obie"
```

Die Zahlen sind eine 5 (Zeilen) x 4 (Spalten)-Matrix. Die Matrix der Zeichenketten ist eine 3 x 4-Matrix.

Um diese spezielle numerische 5 x 4-Matrix zu erstellen, erstellen Sie zunächst den Zahlenvektor von 5 bis 100 in 5er-Schritten:

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

Dann verwenden Sie die R-Funktion `dim()`, um den Vektor in eine zweidimensionale Matrix zu verwandeln:

```
> 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 die Zeilennummern in Klammern entlang der Seite und die Spaltennummern in Klammern in der oberen Zeile anzeigt.

Beim *Transponieren* einer Matrix werden die Zeilen mit den Spalten vertauscht. Die Funktion `t()` kümmert sich um diesen Vorgang:

```
> 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
```

Die Funktion `matrix()` bietet Ihnen eine weitere Möglichkeit, Matrizen zu erstellen:

```
> 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` hinzufügen, füllt R die Matrix zeilenweise, etwa so:

```
> 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 verweisen Sie auf eine bestimmte Matrixkomponente? Sie geben den Matrixnamen ein und dann in Klammern die Zeilennummer, ein Komma und die Spaltennummer:

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

Um auf eine ganze Zeile zu verweisen (wie die dritte Zeile), gehen Sie wie folgt vor:

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

Und auf eine ganze Spalte (zum Beispiel die zweite):

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

Obwohl es sich um eine Spalte handelt, zeigt R sie als Zeile im Konsolenfenster an.

ABER BEDENKEN SIE ...

Wie ich bereits erwähnt habe, ist eine Matrix ein zweidimensionales Array. In R kann ein Array jedoch mehr als zwei Dimensionen haben. Ein bekannter Datensatz (den ich in Teil 3 in Kapitel 11 als Beispiel verwende) hat drei Dimensionen: `Hair Color (Black, Brown, Red, Blond)`, `Eye Color (Brown, Blue, Hazel, Green)` und

`Gender (Male, Female)`. Dieses Array ist also 4 x 4 x 2 groß. Es heißt `HairEyeColor` und sieht wie folgt aus:

```
>
 HairEyeColor
, , Sex – Male

Eye
Hair        Brown          Blue    Hasel   Green
  Black        32            50       10       3
  Brown        53            11       26      15
  Red          10            10        7       8
  Blond         3            30        5       8

, , Sex = Female

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

Jede Zahl steht für die Anzahl der Personen in dieser Gruppe, die eine bestimmte Kombination von Haarfarbe, Augenfarbe und Geschlecht haben – zum Beispiel, 16 braunäugige rothaarige Frauen. (Warum habe ich braunäugige rothaarige Frauen ausgewählt? Weil ich das Vergnügen habe, jeden Tag eine besonders schöne Frau zu sehen.)

Wie würde ich auf alle Frauen verweisen? Das geht wie folgt:

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

Listen

In R ist eine *Liste* eine Sammlung von Objekten, die nicht unbedingt vom gleichen Typ sind. Nehmen wir an, Sie stellen einige Informationen über die Beatles zusammen:

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

Eine wichtige Information könnte das Alter der einzelnen Beatles sein, als sie der Gruppe beitraten. John und Paul begannen im Alter von 17 bzw. 15 Jahren gemeinsam zu singen, und der 14-jährige George stieß bald darauf dazu. Ringo, ein Spätberufener, wurde erst mit 22 Jahren ein Beatle. Wir haben also:

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

Um die Informationen in einer Liste zusammenzufassen, verwenden Sie die Funktion `list()`:

```
> beatles_info <-list(names=beatles,age_joined=ages)
```

Die Benennung der einzelnen Argumente (`names`, `age_joined`) veranlasst R, diese Namen als Namen für die Listenkomponenten zu verwenden.

Und so sieht die Liste aus:

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

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

R verwendet das Dollarzeichen ($), um jede Komponente der Liste zu kennzeichnen. Wenn Sie auf eine Listenkomponente verweisen möchten, geben Sie den Namen der Liste, das Dollarzeichen und den Namen der Komponente ein:

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

Und um auf einen ganz bestimmten Beatle zu verweisen, zum Beispiel auf den vierten Beatle? Ganz einfach:

```
> beatles_info$names[4]
[1] "Ringo"
```

R erlaubt es Ihnen auch, Kriterien innerhalb der Klammern zu verwenden. Um beispielsweise auf Mitglieder der Fab Four zu verweisen, die älter als 16 waren, als sie beitraten, schreiben Sie:

```
> beatles_info$names[beatles_info$age_joined > 16]
[1] "john" "ringo"
```

Datenrahmen

Eine Liste ist ganz praktisch, um Daten zu sammeln. Noch besser ist ein *Datenrahmen*. Und warum? Wenn Sie über die Daten einer Gruppe von Personen nachdenken, denken Sie normalerweise an Zeilen, die die Personen darstellen, und Spalten, die die Datenvariablen darstellen. Genau das ist ein Datenrahmen. Wenn Ihnen hier die Begriffe *Datensatz* oder *Datenmatrix* in den Sinn kommen, haben Sie die richtige Vorstellung.

Angenommen, ich habe eine Gruppe von sechs Personen:

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

und ich habe die Größe (in Zoll) und das Gewicht (in Pfund) der einzelnen Personen:

```
> heigth <- c(72,64,73,65,66,71)
> weight <- c(195,117,205,122,125,199)
```

Außerdem gebe auch das Geschlecht jeder der Personen an:

```
> gender <- c("M", "F", "M", "F", "F", "M")
```

Bevor ich Ihnen erkläre, wie Sie all diese Vektoren zu einem Datenrahmen kombinieren können, muss ich Ihnen noch eine weitere Sache zeigen. Die Komponenten des `gender`-Vektors sind Zeichenketten. Für die Zwecke der Datenzusammenfassung und -analyse ist es zu empfehlen, sie in Kategorien umzuwandeln – die Kategorie »Männlich« (*Male*) und die Kategorie »Weiblich« (*Female*). Zu diesem Zweck verwende ich die Funktion `factor()`:

```
> factor_gender <-factor(gender)
> factor_gender
[1] M F M F F M
Levels: F M
```

In der letzten Zeile der Ausgabe ist *Levels* der Begriff, den R für Kategorien verwendet.

Die Funktion `data.frame()` arbeitet mit den Vektoren, um einen Datenrahmen zu erstellen:

```
> d <- data.frame(name,factor_gender,height,weight)
> d
     name factor_gender heigth weight
1      al             M     72    195
2 barbara             F     64    117
3 charles             M     73    205
4   donna             F     65    122
5   ellen             F     66    125
6    fred             M     71    199
```

Möchten Sie wissen, wie groß die dritte Person ist?

```
> d[3,3]
[1] 73
```

Wie sieht es mit allen Informationen für die fünfte Person aus?

```
> d[5,]
   name factor_gender height weight
5 ellen             F     66    125
```

Wie bei Listen wird auch bei Datenrahmen das Dollarzeichen verwendet. In diesem Zusammenhang kennzeichnet das Dollarzeichen eine Spalte:

```
> d$height
[1] 72 64 73 65 66 71
```

Sie können Statistiken berechnen, zum Beispiel die durchschnittliche Körpergröße:

```
> mean(d$height)
[1] 68.5
```

Wie bei Listen können Sie auch hier Kriterien in die Klammern setzen. Dies wird häufig bei Datenrahmen gemacht, um Daten innerhalb von Kategorien zusammenzufassen und zu analysieren. Um die durchschnittliche Größe der Frauen zu ermitteln, gehen Sie wie folgt vor:

```
> mean(d$height[d$factor_gender == "F"])
[1] 65
```

Das doppelte Gleichheitszeichen (==) in den Klammern ist ein *logischer Operator*. Stellen Sie sich das so vor: »wenn d$factor_gender gleich 'F' ist«.

Das doppelte Gleichheitszeichen (`a == b`) unterscheidet den logischen Operator (»if a equals b«) vom Zuweisungsoperator (`a = b`; »a auf b setzen«).

Ja, ich weiß – ich bin ausführlich auf `factor()` eingegangen und habe erklärt, dass es besser ist, Kategorien (Levels) als Zeichenketten zu verwenden und dann musste ich Anführungszeichen um `F` innerhalb der Klammern setzen. R ist manchmal so.

Wenn Sie die $-Zeichen aus Ihrem R-Code entfernen möchten, können Sie die Funktion `with()` verwenden. Sie fügen Ihren Code innerhalb der Klammern nach dem ersten Argument ein, also den Daten, die Sie verwenden.

Beispiel:

```
> with(d,mean(height[factor_gender == "F"]))
```

ist gleichbedeutend mit

```
> mean(d$height[d$factor_gender == "F"])
```

Wie viele Zeilen enthält ein Datenrahmen?

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

Und wie viele Spalten?

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

Um einem Datenrahmen eine Spalte hinzuzufügen, verwenden Sie `cbind()`. Beginnen Sie mit einem Vektor von Bewertungen:

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

Dann fügen Sie diesen Vektor als Spalte hinzu:

```
> d.apt <- cbind(d,aptitude)
> d.apt
     name factor_gender height weight aptitude
1      al             M     72    195       35
2 barbara             F     64    117       20
3 charles             M     73    205       32
4   donna             F     65    122       22
5   ellen             F     66    125       18
6    fred             M     71    199       15
```

for-Schleifen und if-Anweisungen

Wie viele Programmiersprachen bietet auch R eine Möglichkeit, seine Strukturen zu durchlaufen, um bestimmte Dinge zu erledigen. Die Methode von R wird `for`-Schleife genannt. Und wie viele andere Sprachen bietet auch R eine Möglichkeit, ein Kriterium zu prüfen – die `if`-Anweisung.

Das allgemeine Format einer `for`-Schleife sieht wie folgt aus:

```
for counter in start:end{
          Anweisung 1
               .
               .
               .
          Anweisung n
}
```

Wie Sie sich vielleicht vorstellen können, verfolgt der `counter` die einzelnen Durchgänge, auch als *Iterationen* bezeichnet.

Das einfachste allgemeine Format für eine `if`-Anweisung ist

```
if(test){Anweisung, die ausgeführt wird, wenn test TRUE ergibt}
else{Anweisung, die ausgeführt wird, wenn test FALSE ergibt}
```

Hier ist ein Beispiel, das beides beinhaltet. Ich habe einen Vektor `xx`:

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

Und einen weiteren Vektor `yy`, der im Moment nichts enthält:

```
> yy <-NULL
```

Ich möchte, dass die Komponenten von `yy` die Komponenten von `xx` widerspiegeln: Wenn eine Zahl in `xx` eine ungerade Zahl ist, soll die entsprechende Komponente von `yy "ODD"` sein, und wenn die Zahl `xx` gerade ist, soll die Komponente von `yy "EVEN"` sein.

Wie prüfe ich, ob eine Zahl ungerade oder gerade ist? Mathematiker haben die *modulare Arithmetik* entwickelt, bei der es um den Rest einer Divisionsoperation geh. Dividiert man *a* durch *b* und hat das Ergebnis einen Rest von *r*, sagen die Mathematiker: »*a* modulo *b* ist *r*.« 10 dividiert durch 3 ergibt einen Rest von 1, also ist 10 modulo 3 gleich 1. Normalerweise wird *modulo* zu *mod* abgekürzt, also haben wir »10 mod 3 = 1«.

Die meisten Computersprachen schreiben 10 mod 3 als `mod(10,3)`. (Auch Excel.) R macht es anders: R verwendet das doppelte Prozentzeichen (`%%`) als *mod-Operator*:

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

Ich denke, Sie haben es verstanden: Wenn `xx[i] %% 2 == 0` ist, dann ist `xx[i]` gerade. Andernfalls ist es ungerade.

Hier ist also die `for`-Schleife und die `if`-Anweisung:

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

> yy
[1] "EVEN" "ODD" "EVEN" "ODD" "EVEN"
```