Teil I
Einführung in die Arbeit mit R und RStudio

Im ersten Teil dieses Buches geht es primär darum, Grundlagen im Umgang mit R und RStudio zu schaffen.

Gute Gründe, R für statistische Analysen zu nutzen, werden in **Kapitel 1** kurz dargelegt.

In **Kapitel 2** stehen die Grundprinzipien der R-Programmierung (Abschnitt 2.1) sowie die zur Verfügung stehenden Objekttypen im Fokus (Abschnitt 2.2). Hieran schließt sich das Management von Analysepaketen an (Abschnitt 2.3), bevor die für die in diesem Buch gezeigten Analyseverfahren notwendigen Analyseformate und die gegenseitige Überführung (Abschnitt 2.4) gezeigt werden. Den Abschluss des zweiten Kapitels bilden die zunächst noch etwas abstrakt anmutenden Pipe-Operatoren (Abschnitt 2.5). Diesen Abschnitt können Sie zunächst getrost überspringen und erst nach Verweis durch einen konkreten Anwendungsfall durcharbeiten.

Den Abschluss des ersten Teils dieses Buches bildet die Einführung in RStudio in **Kapitel 3**. Speziell wird das Layout erklärt (Abschnitt 3.1) und empfohlene Einstellungen gezeigt (Abschnitt 3.2).

Warum gerade R für statistische Analysen?

Die Frage nach dem »Warum« ist auch in der Datenanalyse allgegenwärtig. Damit dieses Buch nicht zu philosophisch wird und seinem Versprechen eines anwendungsorientierten Nachschlagewerkes gerecht wird, werde ich hier nicht zu ausschweifend sein. So viel sei aber gesagt: Jede Person hat andere Präferenzen, **warum** gerade dieses eine Analyseprogramm das für sie beste ist. Zu den Kriterien zählen Einsteigerfreundlichkeit, Bedienbarkeit, Leistungsumfang, Updates, Preis – um nur ein paar zu nennen.

In den meisten o.g. Kategorien schneidet R sehr gut ab. Eigentlich in allen, außer der Einsteigerfreundlichkeit – aber dieses Buch ist ja dafür da, genau diesen Malus zu beheben. Eine gewisse Grundkenntnis statistischer Begriffe ist ohnehin bei allen Analyseprogrammen von Vorteil.

Zur Bedienung von R wird eine sog. *Syntax* verwendet. Sie beschreibt vereinfacht ausgedrückt das korrekte Kombinieren von Befehlen mit Objekten. Objekte können Variablen, Dataframes usw. sein. Diese Arbeitsweise zeichnet alle statistischen Analyseprogramme aus. Allerdings wurden im Laufe der Jahre aus Gründen der einfacheren Bedienbarkeit von manchen Herstellern (z.B. SPSS, inzwischen IBM) grafische Benutzeroberflächen mit Dialogfeldern aufgesetzt. Diese nehmen dem Nutzer das Eingeben der Syntax ab. Dies hat den Vorteil, dass man die Befehle nicht auswendig kennen muss und es nicht zu Tippfehlern kommen kann – allerdings zum Teil auf Kosten der Nachvollziehbarkeit und Reproduzierbarkeit der Analyseschritte.

Im Hinblick auf den Leistungsumfang ist R das »mächtigste« Analyseprogramm. Es werden standardmäßig sog. *Base packages* mitgeliefert, die aber nur einen Bruchteil der 19.000 existierenden Pakete darstellen. Diese Pakete beinhalten die von Nutzern verwendeten Analysefunktionen. Diese enorme Anzahl von Paketen wird größtenteils von Wissenschaftlern mit statistischem

Hintergrundwissen freiwillig erstellt und beständig mit Updates versorgt. Für jedes dieser R-Pakete existiert eine umfangreiche auf CRAN (Comprehensive R Archive Network) zugängliche Dokumentation.

Abschließend kann noch kurz der Preis erwähnt werden. R und sämtliche Pakete sind vollständig kostenlos herunterladbar. Es gibt auch kostenlose Zusatzprogramme, allen voran RStudio Desktop in der Open Source Edition. RStudio vereinfacht das Arbeiten erheblich, indem es die Übersichtlichkeit stark erhöht. Daher steht bereits an dieser Stelle meine klare Empfehlung, dieses Programm zu nutzen. Zu RStudio, dessen Installationen sowie Nutzung komme ich in Kapitel 3.

2 R-Grundlagen in Kurzform

In den Grundlagen geht es nur um die rudimentärsten Dinge, die in R möglich sind und uns eine einfachere Auswertung ermöglichen. Dazu gehört das Verständnis der Syntax und deren Aufbau (Abschnitt 2.1), die Variablenformate (sog. *Objekttypen*, Abschnitt 2.2) sowie das Management der bereits erwähnten Pakete (Abschnitt 2.3). Dazu kommt das je nach Analysemethode unterschiedliche Datenformat (Abschnitt 2.4) und das Prinzip einer sehr eleganten Art der Schachtelung von Befehlen mittels Pipe-Operatoren (Abschnitt 2.5), die Ihnen später häufiger begegnen wird.

2.1 Syntax

Im vorangegangenen Kapitel wurde bereits kurz auf die Syntax eingegangen. Bisweilen liest man auch den Begriff »R-Programmiersprache«. An dieser Stelle werde ich mit den Begrifflichkeiten nicht zu genau sein – die kann man bei Bedarf (erneut) bei Wikipedia oder in diversen Büchern (à la »Einführung in R«) sehr detailliert nachlesen. Da es der Zweck des Buches ist, ein anwendungsorientiertes Nachschlagewerk zu sein, sei zum Thema Syntax nur so viel erwähnt, dass die Kombination von Befehlen mit Objekten für die Datenanalyse im Mittelpunkt steht. Der vom Nutzer eingegebene Quelltext wird nicht extra an einen Compiler übergeben, der dies dann in Maschinensprache übersetzen müsste, und dann erst zur Ausführung gebracht. Vielmehr wird durch die Enter-Taste die Ausführung direkt angestoßen.

Wichtige zu verinnerlichende Prinzipien beim »Programmieren« mit R sind die folgenden:

- R unterscheidet **Klein- und Großbuchstaben** (»case sensitive«).
- Das **Dezimaltrennzeichen** in R ist ein Punkt (z.B. `3.45` in R bedeutet 3,45).

- **Zuweisungen** (dazu später mehr) erfolgen über `<-`.
  In vielen Funktionen ist auch = nutzbar.
- Die **Bezeichnung** von Variablen bzw. Objekten allgemein darf nur alphanumerische Zeichen (A–Z, 0–9), Punkte und Unterstriche beinhalten, darf aber nicht mit einer Zahl beginnen (`data.2` wäre okay, `2.data` hingegen nicht).
- **Zeilenumbrüche** zur besseren Lesbarkeit sind mit + am Zeilenende möglich.
- **Abhängigkeiten** in Formeln werden mit ~ dargestellt. `y~x+z` bedeutet, dass die links stehende abhängige Variable »y« aus den rechts stehenden unabhängigen Variablen »x« und »z« geschätzt werden soll. Das + ist hier jedoch kein arithmetischer Operator und wird hier nur für die Aufnahme der Variablen verwendet.

2.2 Objekttypen in R

Die Arbeit in und mit R dreht sich um sog. **Objekte** bzw. **Objekttypen**. Dies sind Vektoren, Faktoren und Data Frames.

Im Gegensatz zur mathematischen Definition repräsentieren **Vektoren** in R *numerische* Variablen. Numerisch bedeutet Ordinal-, Intervall- und Verhältnisskalenniveau. Beispiel: Hat man die Körpergröße von Befragten (Vehältnisskalenniveau) erhoben, wird diese in einem beliebigen Vektor entsprechend der o.g. Namenskonvention gespeichert. Jede weitere *numerische* Variable (z.B. Alter, Einkommen) wird in einem extra Vektor gespeichert. Dies sind sog. **numeric-Vektoren**.

Ein Spezialfall eines Vektors ist der sog. **Faktor**. Faktoren enthalten Variablen auf Nominal- bzw. Kategorialskalenniveau. Hierzu zählen z.B. das Geschlecht von Befragten oder deren Lieblingsfarbe. Diese können entweder als Zahlen mit zusätzlicher Identifikation hinterlegt sein (z.B. 0-männlich, 1-weiblich) oder direkt als Wort (sog. **character-Vektoren**).

Eine Menge von Vektoren und Faktoren sind in einem sog. **Data Frame** zusammenfassbar. Sie können sich dies wie eine große Datentabelle (aus Excel oder SPSS) vorstellen, die zeilenweise die Befragten und spaltenweise die Variablen enthält:

ID	Geschlecht	Alter	Körpergröße	Einkommen
1	W	20	1,62	2100
2	M	21	1,78	2200
3	W	22	1,94	2300
4	…	…	…	…

In R ist die Arbeit mit Data Frames alltäglich, weil nach einem Datenimport die Speicherung der Daten in der Regel in einem Data Frame vorgenommen wird.

Der Vollständigkeit halber sei noch erwähnt, dass es drei weitere Objekte gibt, die aber im Rahmen der in diesem Buch gezeigten Analysemethoden praktisch keine Relevanz besitzen. **Matrizen** beinhalten wie Data Frames Objekte, allerdings können sie nur *entweder* numerische *oder* Textdaten beinhalten.

Arrays umfassen mehrere Matrizen und sind mehrdimensional. Sie können sie sich also wie eine Stapelung von Matrizen vorstellen.

Listen umfassen, ähnlich wie Data Frames oder Matrizen, mehrere Objekte. Der Unterschied ist, dass Listen Vektoren mit unterschiedlichen Längen (= Anzahl von Elementen) und Eigenschaften repräsentieren können.

2.3 R-Pakete finden und verwenden

2.3.1 Pakete installieren und laden

Es wäre logischer, an dieser Stelle mit dem Auffinden von Paketen zu beginnen. Allerdings findet man bestenfalls per Hörensagen heraus, welche Pakete für die eigenen Vorhaben taugen. Das Orientieren an Paketnamen schlägt leider ebenfalls fehl, da es z.T. sehr generische Namen sind und kaum Konventionen zu existieren scheinen. Und R wäre nicht R, wenn es kein Paket für das Finden von Paketen geben würde. ;-) Daher zeige ich zunächst anhand des Pakets **`packagefinder`** die Installation und das Aktivieren von Paketen.

Ein Paket wird stets mit der `install.packages()`-Funktion installiert. Hierbei ist es zwingend notwendig, das Paket mit exaktem Namen in Anführungszeichen in die Klammer zu setzen. Nach Ausführung dieser Codezeile werden benötigte Dateien bzw. Pakete, auf die das aktuell zu installierende Paket zugreift, automatisch heruntergeladen und installiert.

```
install.packages("packagefinder")
```

Nach erfolgreicher Installation (und bei jedem Start von RStudio, sofern kein Startskript existiert) muss das Paket zwingend geladen werden. Zum Laden wird die `library()`-Funktion verwendet. Hier ist das Paket erneut mit exaktem Namen, allerdings OHNE Anführungszeichen einzugeben. Zum Entladen wird die `detach()`-Funktion verwendet.

```
# Laden des Pakets packagefinder
library(packagefinder)

# Entladen des Pakets packagefinder
detach("package:packagefinder", unload = TRUE)
```

2.3.2 Finden von Paketen

Neben den Erfahrungen anderer, die Auswertungen vornehmen und hierfür für sie gut funktionierende Pakete gefunden haben, oder diesem Buch, wo ich auch diverse Pakete vorstelle, gibt es noch die Möglichkeit, über das Paket »packagefinder« Stichworte einzugeben.

Die `fp()`-Funktion erlaubt das gezielte Suchen nach Stichwörtern, die sich im Namen, der Kurz- und Langbeschreibung des Pakets befinden. Speziell in beiden Letzteren ist die Chance sehr gut, Treffer zu erzielen. Hierzu muss lediglich in Anführungszeichen das entsprechende Stichwort in die Klammer gesetzt und diese Codezeile ausgeführt werden.

Bei mehr als einem Stichwort wird in `fp()` zusätzlich `c()` eingefügt und die Stichwörter per Komma getrennt. Das Argument `mode=""` gibt mit einem logischen Operator an, ob ein oder mehrere der Stichwörter in der Suche vorkommen müssen. `or` verlangt *mindestens eins* der Stichwörter, `and` verlangt zwingend das Vorkommen *aller* Stichwörter.

```
001 fp("regression")
002 fp(c("regression", "interaction"), mode = "or")
```

Nachfolgend erhalten Sie im *Viewer* von RStudio (das Fenster unten rechts) eine Ergebnisübersicht (vgl. Abbildung 2.1), die einen kleinen Score am Anfang der jeweiligen Zeile hat, der als Indikator des Suchmatchings fungiert. Daneben stehen Paketname, die Kurzbeschreibung und der sog. **GO-Code**.

Score	Name	Short Description	GO
100.0	SIMPLE.REGRESSION	Multiple Regression and Moderated Regression Made Simple	15891
90.6	fRegression	Rmetrics - Regression Based Decision and Prediction	5637
85.6	iRegression	Regression Methods for Interval-Valued Variables	7878
84.4	quickregression	Quick Linear Regression	13120
83.6	AnchorRegression	Perform AnchorRegression	377
82.8	mrregression	Regression Analysis for Very Large Data Sets via Merge and Reduce	10284
74.1	deepregression	Fitting Deep Distributional Regression	3492
73.3	safeBinaryRegression	Safe Binary Regression	15104
71.9	TwoRegression	Process Data from Wearable Research Devices Using Two-Regression Algorithms	18106
69.8	UnilsoRegression	Unimodal and Isotonic L1, L2 and Linf Regression	18192
68.1	MultipleRegression	Multiple Regression Analysis	10453
67.7	riskRegression	Risk Regression Models and Prediction Scores for Survival Analysis with Competing Risks	14221

Abb. 2.1: Suchergebnisübersicht für das Stichwort »regression« im RStudio Viewer

Mit dem GO-Code, am Beispiel des SIMPLE.REGRESSION-Pakets 15891 arbeitet man wie folgt:

- `go(15891)` gibt die Kurzbeschreibung des Pakets in die R-Konsole aus.
- `go(15891,"manual")` ruft das Handbuch des Pakets im PDF-Format auf, während
- `go(15891,"website")` zur Homepage des Pakets führt, die in einem separaten Browserfenster geöffnet wird.

Wenn Sie die Arbeit im Browser bevorzugen, arbeiten Sie mit dem Zusatzargument `display = "browser"`.

```
fp("regression", display = "browser")
```

Im Ergebnis wird im Browser (vgl. Abbildung 2.2) zusätzlich eine Langbeschreibung, die Anzahl an Downloads, Links zur Beschreibung und zum Handbuch auf CRAN sowie der Installationscode angezeigt. Letzterer kann per Klick in die Zwischenablage kopiert und im R-Skript oder der R-Konsole eingefügt werden.

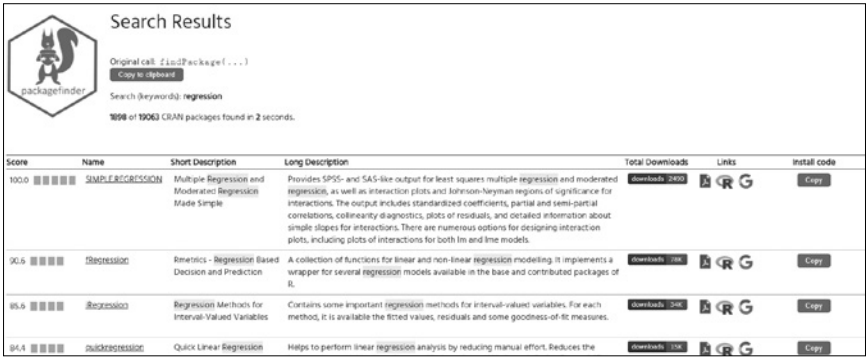

Abb. 2.2: Suchergebnisübersicht für das Stichwort »regression« im Browser

2.4 Datenformate in R

Die aus der Statistik bekannten Datenformate *wide* und *long* können natürlich auch in R verwendet werden. Diese Unterscheidung ist essenziell, da je nach Analyseziel und anzuwendender Methode das eine oder andere Datenformat notwendig ist. Daher wird an dieser Stelle eine kurze Einordnung vorgenommen.

2.4.1 Wide-Format

Das Wide-Format ist das in den meisten Disziplinen häufiger anzutreffende Format. Es wird auch »ungestapelt« genannt und zeichnet sich dadurch aus, dass jedes Untersuchungsobjekt in einer separaten Zeile steht. Gleichzeitig stehen in den Spalten die Variablen, die für die Untersuchungsobjekte erhoben wurden. Ähnlich der Darstellung in Tabelle 2.1.

Sollte beispielsweise der BMI für die Probanden zu verschiedenen Zeitpunkten erhoben werden, wird für jeden Messzeitpunkt eine separate Variable angelegt, z. B. `BMI_t0`, `BMI_t1` usw. Das ist zwar prinzipiell möglich und auch deutlich übersichtlicher, z.B. verlangt aber eine ANOVA mit Messwiederholung, dass die Daten im Long-Format vorliegen.

ID	Geschlecht	BMI_t0	BMI_t1	BMI_t2
1	w	21,72	21,48	21,65
2	m	30,41	30,00	29,48
3	w	24,05	24,18	23,82
…	…	…	…	…

Tab. 2.1: Beispiel für das Wide-Format

2.4.2 Long-Format

Im Long-Format (auch »gestapelt«) wird, um im Beispiel des BMI zu bleiben, für jede Messung des BMI eine separate Zeile erstellt. Zusätzlich bedarf es zweier Variablen: Zum einen muss erkennbar sein, um welche Messung bzw. welchen Zeitpunkt es sich handelt. Zum anderen ist das Untersuchungsobjekt mit einem **Identifier** (ID) eindeutig zuzuordnen.

ID	Geschlecht	Zeitpunkt	BMI
1	w	1	21,72
2	m	1	30,41
3	w	1	24,05
1	w	2	21,48
2	m	2	30,00
3	w	2	24,18
1	w	3	21,65
2	m	3	29,48
3	w	3	23,82
…	…	…	…

Tab. 2.2: Beispiel für das Long-Format

2.4.3 Transformation der Formate

Für die Transformationen vom einem zum anderen der beiden o.g. Formate kann das sog. **tidyr**-Paket verwendet werden. Die Installation und das Laden von Paketen kennen Sie bereits aus Abschnitt 2.3 und wenden dieses Wissen direkt an. Mit `install.packages()` wird es installiert und mit `library()` geladen:

```
001 install.packages("tidyr")
002 library(tidyr)
```

Transformation Wide-Format zu Long-Format

Aus dem `tidyr`-Paket wird die `pivot_longer()`-Funktion verwendet.

In die Funktion ist zu Beginn der Data Frame einzugeben, der transformiert werden soll. Anschließend sind die Variablen, in denen die Messwiederholungen stehen, anzugeben. Schließlich werden die Namen, die den Zeitpunkt (`names_to`) sowie die Messwerte (`values_to`) bezeichnen, vergeben.

Im Beispiel heißt der zu transformierende Data Frame `data_wide` und die Variablen `t0` bis `t20` aus ihm sollen transformiert werden. Die neue Zeitpunktvariable wird schlicht mit `t` und die Wertevariable mit `v` abgekürzt und benannt.

```
001 data_long <- pivot_longer(data_wide, t0:t20,
002                           names_to = "t",
003                           values_to = "v")
```

Transformation Long-Format zu Wide-Format

Für das umgekehrte Prinzip wird die `pivot_wider()`-Funktion aus dem `tidyr`-Paket angewandt. Mit (`names_from`) werden die Variablennamen für die Variable, die die x Zeitpunkte (hier `t`) ausdrückt, erfasst. Mit (`values_from`) wird die Variable (hier `v`) benannt, aus der die Werte in die neuen x Spalten verschoben werden.

```
001 data_wide <- pivot_wider(data_long,
002                          names_from = t,
003                          values_from = v)
```

Proband	t0	t10	t20
A	28	42	48
B	5	10	17
C	23	26	28
D	20	30	25
E	16	24	27

Proband	t	v
A	t0	28
B	t0	5
C	t0	23
D	t0	20
E	t0	16
A	t10	42
B	t10	10
C	t10	26
D	t10	30
E	t10	24
A	t20	48
B	t20	17
C	t20	28
D	t20	25
E	t20	27

Abb. 2.3: Beispiel des Wide-Formats (links) und des Long-Formats (rechts) und die jeweilige Überführung in das andere Format

2.5 Pipe-Operatoren

Dieser Abschnitt behandelt ein recht kompliziertes Vorgehen und ist erst für spätere Kapitel relevant und kann und sollte daher zunächst übersprungen und erst im konkreten Anwendungsfall durchgearbeitet werden – für den Umgang mit R erachte ich ihn als unbedingt grundlegend. Speziell in späteren Kapiteln dient es zur erheblichen Reduktion des Arbeitsaufwands. Unscheinbar, aber sehr hilfreich, ist der sog. Pipe-Operator `%>%`, der ursprünglich aus dem `magrittr`-Paket stammt und inzwischen fester Bestandteil der Paketsammlung `tidyverse` ist.

Im Rahmen dieses Buches findet der Pipe-Operator vor allem in den Pakten `dplyr` und `rstatix` seine Anwendung, weil er eine sehr einfache Möglichkeit bietet, Funktionen zu schachteln, ohne sie paradoxerweise schachteln zu müssen. Beim Piping werden Befehle aneinandergereiht und mit dem Pipe-Operator `%>%` verbunden. Das Ergebnis sind flexiblere und weniger geschachtelte Funktionen. Dieses Vorgehen mutet zunächst kryptisch und kompliziert an, ist aber der einfachste Weg.

Dieses Vorgehen ist v.a. bei Auswertungen von Datensätzen im Long-Format (speziell bei mehr als zwei Beobachtungszeitpunkten, vgl. Abschnitt 10.2) sehr hilfreich. Die Simplizität kann an einem einfachen Beispiel dargelegt werden:

Anstelle der `aggregate()`-Funktion und einer Schachtelung diverser Argumente und Dopplungen:

```
001 aggregate(data.a_l[, 3], list(data.a_l$Zeitpunkt), mean)
```

kann eine viel intuitivere Verkettung vorgenommen werden:

```
001 library(dplyr)
002 data.a_l %>%
003   group_by(Zeitpunkt) %>%
004   summarize(M = mean(Wert))
```

Die Ergebnisse sind identisch:

```
# aggregate()-Funktion
  Group.1     Wert
1      T0 24.48333
2      T1 27.40000
3      T2 29.01667
# Pipe Operator >%>
  Zeitpunkt        M
1        T0 24.48333
2        T1 27.40000
3        T2 29.01667
```

Mit einer zunehmend komplexeren Datenstruktur, speziell von im Long-Format vorliegenden Daten, ist die Auswertung mittels Pipe-Operatoren bequemer, teilweise auch nahezu alternativlos. Dies wird uns v.a. in den Kapiteln zur Untersuchung von Unterschieden und Veränderungen begegnen.