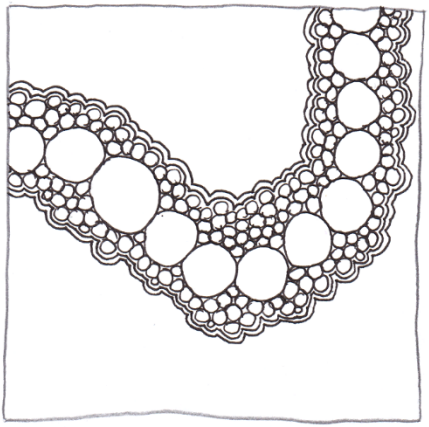
Kapitel 5: **Programmierung**



In diesem Kapitel lernen Sie…

- wie Sie mit R Daten auswerten
- wie Sie mit Python Daten auswerten
- wie Sie Laufzeitumgebungen und Cloud Computing verwenden

Computer sind aus wissenschaftlichen Projekten kaum wegzudenken. Sie helfen bei der Verwaltung von Daten, nehmen klaglos Routineaufgaben der Datenanalyse ab und sind meist auch bei der Kommunikation von Ergebnissen beteiligt. Ohne die Anweisungen von Menschen sind Computer aber nutzlos. Anweisungen werden in der Form von Programmen gegeben, die Eingaben der Hardware verarbeiten und wieder ausgeben. Bei der Benutzung aktueller PCs oder Smartphones treten die Programme in der Regel in den Hintergrund, sie verbergen sich hinter der Benutzeroberfläche. In diesem Kapitel schauen geht es darum, hinter die Oberfläche zu schauen und selbst Programme zu schreiben. Das ist besonders dann sinnvoll, wenn es für ein bestimmtes Problem entweder keine vorkonfektionierte Software gibt oder wenn der Prozess der Datenerhebung und -analyse möglichst transparent und kontrolliert sein soll. Denn die Programme dokumentieren im besten Fall jeden einzelnen Schritt der eigenen Vorgehensweise. Das gilt insbesondere für sogenannte Skripte, die eine Abfolge von Befehlen in einem Textdokument festhalten. Diese Skripte sind von Menschen und von Maschinen lesbar. Auf einem Computer wird dazu ein Interpreter – das

ist ebenfalls ein Programm – eingesetzt, der die Befehle Zeile für Zeile abarbeitet. Programme führen also Programme aus.

Eine solche Interpretersprache für statistische Berechnungen ist R. Demgegenüber stehen Compilersprachen wie C, bei denen der Quelltext vor der Ausführung des Programms erst in Maschinensprache übersetzt wird. Die Kompilierung des Programms braucht dann einen Moment, dafür laufen die fertigen Programme aber schneller ab. Zwischen Interpreter- und Compilersprachen stehen Sprachen wie Python oder Java. Hier wird erst ein sogenannter Bytecode erzeugt, der dann von einem Interpreter abgearbeitet wird.

Programmiersprachen lassen sich also danach einteilen, wie aus dem Quelltext ein lauffähiges Programm entsteht. Sie unterscheiden sich aber auch in ihrer Art und Weise der Problemlösung. Werden einfach nur Befehle nacheinander abgearbeitet, so handelt es sich um prozedurale bzw. imperative Programmierung. Dagegen wird das Problem bei funktionaler Programmierung so formuliert, dass es auf eine Formel gebracht wird. Hier sind in einer Codezeile häufig mehrere Funktionsaufrufe ineinander verschachtelt. Bei objektorientierter Programmierung wiederum werden die Probleme als Klassen abgebildet, die Eigenschaften und Methoden besitzen. Von einer solchen Klasse – zum Beispiel einem Balkendiagramm – lassen sich dann Instanzen erzeugen, also etwa ein konkretes Balkendiagramm mit blauen und roten Balken.[1] Aktuelle Programmiersprachen verbinden in der Regel mehrere dieser Konzepte. Das gilt auch für die im Folgenden besprochenen Sprachen R und Python. Prozedurale Elemente sind in fast jedem Skript vorhanden. Besonders in R sind immer wieder auch funktionale Elemente offensichtlich und in Python kommt man häufig mit Klassen und Objekten in Berührung.

Wichtig für die Programmierung sind auch die Werkzeuge, mit denen man Programme erstellt, die Entwicklungsumgebungen genannt werden. Im Zuge der Einführung zu R und Python werden Entwicklungsumgebungen vorgestellt, die sich besonders für die wissenschaftliche Datenanalyse eignen. Mitunter sollen die fertigen Programme dann aber nicht nur auf dem eigenen Computer laufen, sondern benötigen spezielle Server-Umgebungen. Am Ende des Kapitels wird auf verschiedene Möglichkeiten eingegangen, solche Laufzeitumgebungen

[1] Zu den Vorzügen objektorientierter Programmierung bei der Erstelluing von Grafiken siehe Wilkinson (1999: 2-19)

einzurichten. Mit virtuellen Maschinen lassen sich etwa auch unter Windows Laufzeitumgebungen mit Linux einrichten und umgekehrt.

Zudem basiert Cloud Computing darauf, dass besonders rechenintensive Programme innerhalb virtualisierter Laufzeitumgebungen auf verteilten Systemen ausgeführt werden.

Abbildung 1: Wenn statistische Programme/-sprachen Autos wären. https://www.reddit.com/r/rstats/comments/dn7fa7/i_thought_the_if_stats_programs_were_cars_meme/

5.1 Einführung in die Datenanalyse mit R

R ist eine Programmiersprache, die für die Datenanalyse optimiert ist. Im Gegensatz zu vielen kommerziellen Produkten wie Stata oder SPSS können Sie R kostenlos verwenden – und Sie profitieren darüber hinaus von einer aktiven

Entwicklungsgemeinschaft, in die Sie sich nach einiger Zeit sogar selbst einbringen können.

Während sich klassische Statistikprogramme durch eine Benutzeroberfläche auszeichnen und man erst bei speziellen Problemstellungen auf die Programmierung (in der Regel über eine so genannte Syntax) ausweicht, dreht sich dieses Verständnis bei R um. R ist eine Programmiersprache ohne Benutzeroberfläche, die primär über eine Kommandozeile bedient wird. Grafische Programme binden dann R ein, um die Arbeit zu erleichtern. Das klingt vielleicht im ersten Moment kompliziert, diese Arbeitsweise hat aber drei ganz wesentliche Vorzüge. Sie können den Stand eines Projekts inklusive der Daten und der Schritte für die Datenanalyse jederzeit abspeichern, weitergeben und wiederholen (replizieren). Zudem können Sie klein anfangen und das Projekt nach und nach erweitern, ohne immer wieder von vorne zu beginnen (inkrementell). Und schließlich können Sie bei Bedarf Teile der Datenauswertung automatisieren, zum Beispiel die gleichen Schritte mit wenig Aufwand auf weitere Datensätze anwenden (skalieren).

Tatsächlich wird die Arbeit mit R erheblich erleichtert, wenn man dafür eine Entwicklungsumgebung verwendet. Eine für die Datenanalyse häufig eingesetzte Entwicklungsumgebung ist RStudio. Die kostenlose Version von RStudio steht unter einer Open-Source-Lizenz (AGPL) und bringt mehr Funktionen mit als für die meisten Einsatzzwecke im sozialwissenschaftlichen Kontext benötigt werden.

Installieren Sie beides auf Ihrem Computer:

- Die base-Distribution von R: https://www.r-project.org/ (Startseite) bzw. https://cloud.r-project.org/ (Downloadseite)

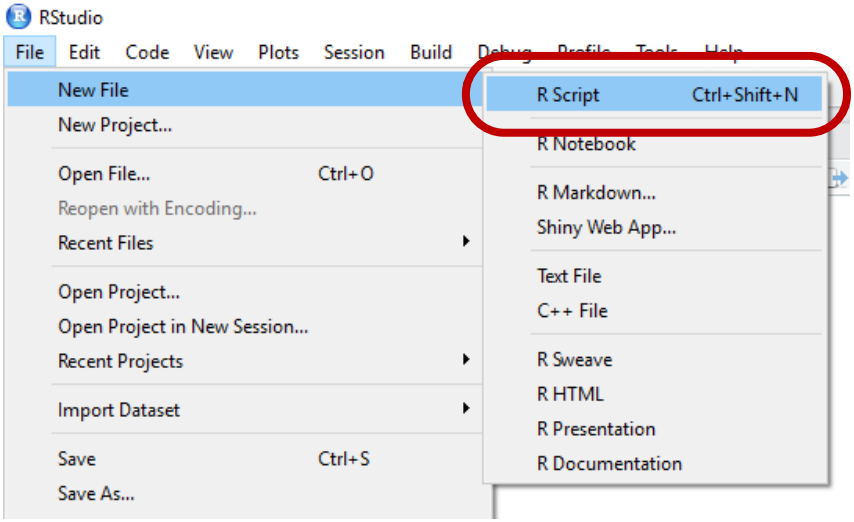
- RStudio Desktop: https://www.rstudio.com/products/rstudio/download/

5.1.1 Die Oberfläche von RStudio

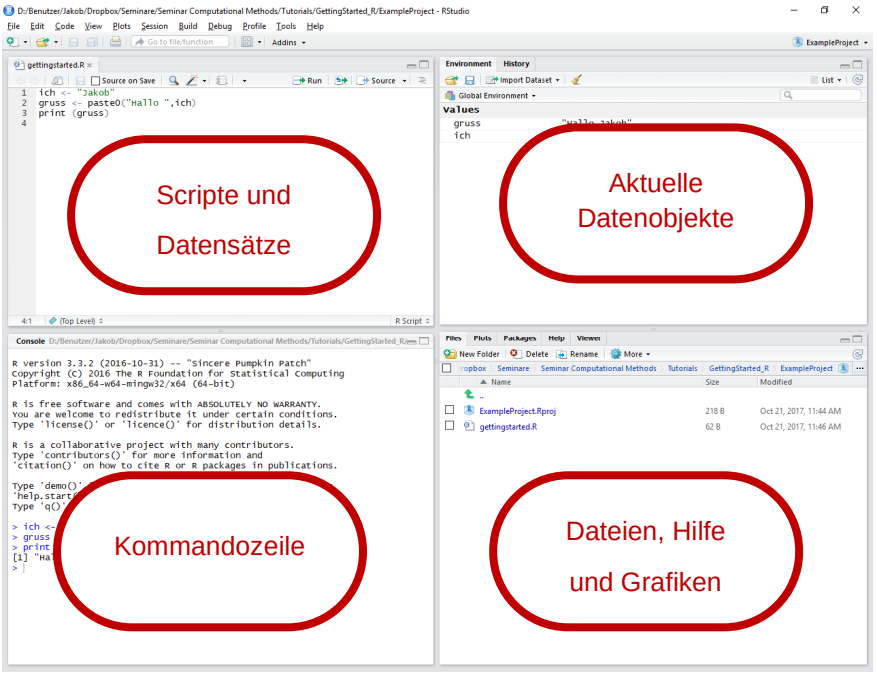
Mit RStudio können Sie die verschiedenen Dateien eines Projekts in einem gemeinsamen Ordner auf der Festplatte verwalten. Legen Sie dazu über das Files-Menü ein neues leeres Projekt in einem Ordner Ihrer Wahl an.

Der wichtigste Dateityp innerhalb eines Projekts sind R-Scripte, in denen die Programmierbefehle gespeichert werden. Legen Sie innerhalb des Projekts ein

neues Script an, Sie finden den Befehl dazu ebenfalls im File-Menü (siehe Abbildung).



Anschließend sollten Sie eine Oberfläche mit vier Bereichen sehen, die bei Bedarf umsortiert werden kann.



- **Links oben** werden Script-Dateien angezeigt. Mit der Schaltfläche „Source“ wird die gesamte Datei ausgeführt. Das ist aber meistens erst am Ende der Entwicklung sinnvoll, wenn alles fertig geschrieben ist. Bis dahin schreiben Sie hier einzelne Codezeilen, markieren Sie mit der Maus oder Tastatur und führen nur den markierten Bereich über die Schaltfläche „Run“ oder schneller noch mit der Tastenkombination Strg+Enter aus. Wenn Sie nichts markieren, wird die aktuelle Zeile ausgeführt. Der Cursor springt anschließend auf die nächste Zeile, so dass Sie sich Stück für Stück mit Strg+Enter durch den Text arbeiten können.

  Beim Ausführen wird der Programmiercode an die Kommandozeile im unteren Bereich übergeben. Probieren Sie es aus, indem Sie die folgenden Zeilen in das neue Script schreiben, den Cursor auf die erste Zeile setzen, die

Strg-Taste gedrückt halten und dreimal nacheinander die Enter-Taste betätigen:

```
ich <- "Leo"
gruss <- paste0("Hallo ",ich)
print(gruss)
```

- **Links unten** befindet sich die Kommandozeile. Hier wird der Inhalt des Scripts ausgeführt, Fehler und zurückgegebene Werte werden angezeigt. Sie können Befehle auch direkt in die Kommandozeile statt in die Script-Datei eingeben. Wenn Sie den Namen eines Objekts eingeben, wird wie mit dem print-Befehl der Inhalt des Objekts angezeigt:

```
gruss
```

- **Rechts oben** wird der Arbeitsbereich (Environment) angezeigt. Dies umfasst alle Objekte und Funktionen, die Sie mit einem Script geladen oder erzeugt haben. Nach dem Ausführen der obigen Zeilen, sehen Sie hier zwei Objekte mit dem Namen „gruss“ und „ich“. Diese Objekte haben Sie mit dem Zuweisungsoperator `<-` erzeugt. Hiermit wird dem Symbol vor dem Operator der Wert dahinter zugewiesen. Mit „Symbol“ ist der Name eines Objekts gemeint.[2]

  Über der Liste finden Sie auch eine Schaltfläche, um Datensätze (CSV, Excel etc.) einzulesen. Später werden Sie Programmierbefehle kennenlernen, mit denen Datensätze eingelesen werden. Das ist in der Regel der bessere Weg, weil die Scripte dann auch von anderen oder einige Zeit später besser nachvollzogen werden können, weil die verwendeten Datensätze aus dem Script ersichtlich sind. Sobald Datensätze aufgelistet sind, können Sie diese anklicken und sie werden in einem neuen Bereich links oben angezeigt.

  Sie können alle Objekte mit der Besen-Schaltfläche löschen. Einzelne Objekte werden über den rm-Befehl im Script oder in der Kommandozeile gelöscht. Probieren Sie es aus:

```
rm(ich)
```

2 Sie könnten alternativ auch das =-Zeichen verwenden, in R ist dafür aber der nach links gerichtete Pfeil üblich.

- **Rechts unten** können Sie zwischen verschiedenen nützlichen Bereichen umschalten. In der Files-Ansicht sehen Sie die Dateien des Projekts. In der Plots-Ansicht werden Grafiken ausgegeben. Im Packages-Bereich können Sie zusätzliche Programmbibliotheken verwalten und vor allem installieren. Zwei Packages, die für viele Projekte sehr hilfreich sind, können Sie gleich installieren, damit sie für die anderen Beispiele in diesem Buch bereitstehen:[3]

  ```
  tidyverse, psych
  ```

  Der Viewer zeigt den Inhalt von Textdateien oder HTML-Dateien an. Ganz besonders wichtig ist aber der Hilfebereich. Die eingebaute Hilfe ist eine große Stärke von R. Sie können zu jedem Befehl die Dokumentation aufrufen, indem Sie dem Befehl ein Fragezeichen voranstellen. In der Hilfe werden die möglichen Argumente beschrieben und auch Beispiele gegeben. Probieren Sie es aus:

  ```
  ?paste0
  ```

  Noch schneller geht das, wenn Sie den Cursor auf dem Befehl platzieren und die F1-Taste drücken. Die Hilfe wird nur für die Befehle angezeigt, die in aktuell geladenen Packages enthalten sind. Nach der Installation werden Packages nicht automatisch geladen, sondern bei Bedarf über den Befehl „library“, zum Beispiel:

  ```
  library(tibble)
  ```

  Sie können aber auch die gesamte Hilfe durchsuchen, inklusive noch nicht geladener Packages, indem Sie zwei Fragezeichen verwenden. So können Sie zum Beispiel herausfinden, in welchem Package ein Befehl enthalten ist:

  ```
  ??describe
  ```

3 Das Package tidyverse enthält eine Reihe weiterer Packages, die weiter unten besprochen werden.

5.1.2 Datentypen

In vielen Programmierumgebungen lassen sich grundlegend drei Datensorten unterscheiden:

1. Skalare[4] oder **elementare Objekte** stellen einzelne Werte dar. Hierzu zählen insbesondere verschiedene Arten von Zahlen, Wahrheitswerte und Zeichenketten. Bei Zeichenketten, also Text, ist zu beachten, dass sie in Anführungszeichen gesetzt werden. Andernfalls wäre nicht klar, ob nicht doch der Name eines Objekts statt der Zeichenkette gemeint ist.

	Ausdruck	**Beispiel**
Zahlen	integer	1884
Kommazahlen	double	1.6
Wahrheitswerte	logical	TRUE, FALSE
Zeichenketten	character	"Anna "

Einige Werte haben besondere Bedeutungen, wichtig ist vor allem die Kennzeichnung fehlender Werte durch den Ausdruck NA.

	Ausdruck
Fehlende Werte	NA
Unendlich	Inf, -Inf
Keine Zahl	NaN
Kein Wert	NULL

Darüber hinaus gibt es in R mit factor einen besonderen Typ für kategoriale Daten. Statt etwa zur Kennzeichnung der Haarfarbe einer Person nur Zahlen (Dummy-Kodierung) oder nur Zeichenketten (die Namen der Farben) zu verwenden, verbinden Faktoren beides. Jede Haarfarbe bzw. jede Ausprägung einer Variablen erhält intern eine Nummer, die für die Ausgabe mit einer Bezeichnung verbunden wird. Damit kann die Reihenfolge der Kategorien in

4 Genau genommen gibt es in R keine Skalare, sondern nur atomare Objekte. Einem der Entwickler von R wird die Äußerung zugeschrieben: „Everything that exists is an object. Everything that happens is a function call“ (Chambers 2014: 170).

Grafiken oder Tabellen leichter verändert werden als bei reinen Zeichenketten. Zudem wird auf diese Weise meistens Speicherplatz gespart.

2. **Mehrere elementare Werte** bilden einen Vektor oder eine Liste. Diese listenartigen Strukturen sind in R sehr häufig anzutreffen, denn viele Funktionen können nicht nur einzelne Werte verarbeiten, sondern die Funktion gleich auf alle Werte eines Vektors anwenden. Diese vektorisierten Funktionen sind ein Unterschied zu vielen anderen Programmiersprachen.[5]

	Ausdruck	**Beispiel**
Vektor	c	c(1,5,9)
Liste	list	list(1,5,9)

In einem Vektor können nur Elemente mit dem gleichen Typ enthalten sein, also nur Zahlen oder nur Zeichenketten, während in Listen verschiedene Typen gleichzeitig möglich sind. Die Anzahl der Elemente in einem Vektor oder einer Liste kann mit der length-Funktion ermittelt werden. Mit der c-Funktion lassen sich Vektoren nicht nur erstellen, sondern auch um weitere Elemente erweitern:

```
personen <- c("Bea", "Leo")
personen <- c(personen, "Niska")
print(personen)
```

Elemente in Listen und Vektoren können auch bezeichnet werden, zum Beispiel:

```
person <- list(vorname="Inger",nachname = "Engmann")
```

3. Für die Datenanalyse werden in der Regel **tabellenartige Datenstrukturen** verwendet. Die Standardstruktur für Datensätze ist in R der Typ data.frame, der aus Spalten und Zeilen besteht. Diese data frames sind intern als Liste von Vektoren organisiert. Jede Spalte ist ein Vektor mit einem einheitlichen Typ und die gesamte Tabelle besteht dann aus einer Liste dieser Vektoren.

5 In anderen Programmiersprachen werden für den gleichen Zweck häufiger Schleifen (foreach, for in etc.) und ähnliche Kontrollstrukturen eingesetzt.

Die Spalten haben Namen. Auch die Zeilen können benannt werden, normalerweise sind die Zeilen aber einfach durchnummeriert.

```
personen <- data.frame (
  name  = c("Bea","Leo","Niska","Inger","Tobbe"),
  alter = c(24,26,25,52,17),
  typ   = c("Bot","Bot","Bot", "Mensch","Mensch")
)
```

Es gibt verschiedene Möglichkeiten, um auf Teile eines data frames zuzugreifen. Einzelne Spalten können mit dem Dollarzeichen über den Spaltennamen ausgegeben werden:

```
personen$name
```

Eine weitere Möglichkeit zum Erreichen einer Spalte besteht über die Nummer der Spalte, die in doppelten eckigen Klammern angegeben wird:

```
personen[[1]]
```

Diese Notation mit doppelten eckigen Klammern funktioniert für alle Listen. Da ein data frame eine Liste aus Vektoren ist, wird im Beispiel das erste Element der Liste, das heißt der erste Vektor, angesprochen.

Wenn mehrere Spalten adressiert werden sollen, dann wird dafür eine Notation mit einfachen rechteckigen Klammern verwendet:

```
personen[,c("name","alter")]
```

Der erste Parameter innerhalb der eckigen Klammern identifiziert die Zeilen und ist hier leer, so dass alle Zeilen ausgewählt werden. Der zweite Parameter identifiziert die Spalten. Die Spaltennamen werden hier als Vektor angegeben, sie können aber ebenso als Zahlen angegeben werden, wobei die Zahlen die Nummer der Spalte oder Zeile bezeichnen. So werden mit dem folgenden Ausdruck die Zeilen 1 bis 3 (der Doppelpunkt kennzeichnet einen Bereich), daraus aber nur Spalte 1 ausgegeben.

```
personen[c(1:3),1]
```

Darüber hinaus kommen in einigen Fällen zwei ganz ähnliche Datentypen zum Einsatz. Erstens: Wenn in allen Spalten einer Tabelle der gleiche Datentyp enthalten ist – zum Beispiel nur Zahlen – dann kann dafür auch der Datentyp matrix eingesetzt werden. Dieser Datentyp hat den Vorteil, dass er sich schnell umformen lässt, zum Beispiel wenn die Werte auf andere Spalten und Zeilen verteilt werden sollen. Zweitens: Aus dem package tibble stammt der Datentyp tbl_df. Dieser wird auch als tibble bezeichnet und ist in den meisten Fällen eine etwas komfortablere Variante von data frames. Beispielsweise ist die Ausgabe über den print-Befehl kompakter. Diese (und auch andere) Datentypen können meistens ineinander überführt werden:

```
as.matrix(personen)
as_tibble(personen)
as.data.frame(personen)
```

Der Typ eines Objekts kann mit der str-Funktion festgestellt werden.

```
str(personen)
```

Diese Funktion hilft immer dann weiter, wenn man es mit unbekannten Objekten zu tun, etwas nicht funktioniert oder wenn man den Überblick verloren hat.

5.1.3 Loops und andere Kontrollstrukturen

```
# for in Kontrollstruktur
for (name in personen) {
  print(paste0(name," is a bot"))
}

# Apply-Funktion
lapply(personen,paste0," is a bot")

# Vektorisierte Funktionen
paste0(personen," is a bot")
```

5.1.4 Aufbau eines Scripts

Ein R-Script besteht aus einer Abfolge von Zuweisungen, Funktionsaufrufen und Kommentaren. Mit Zuweisungen werden Daten sozusagen in verschiedene Schachteln, Dosen und Tüten wegsortiert. Funktionen verarbeiten die Daten, formen sie um, zerlegen oder kombinieren sie. Kommentare sind Notizzettel, die man an Daten und Funktionen anheftet. So kann man sich notieren, was in welcher Schachtel enthalten ist und wozu eine Funktion dient.

Genauer gesagt werden bei Zuweisungen Werte in einem Symbol gespeichert, dazu wird der linksgerichtete Pfeil eingesetzt. Dieses Zuordnungszeichen kann in RStudio über die Tastenkombination Alt + - (Alt-Taste und Minus-Taste) erzeugt werden, es kann aber auch mit dem Kleinerzeichen und einem Minuszeichen eingetippt werden.

```
name <- "Bea"
```

Zur Verarbeitung von Daten werden Funktionen eingesetzt, die in Klammern Argumente entgegennehmen. Diese Argumente sind der Input der Funktion. In der Funktion werden die Daten verwendet und als Output zurückgeben. Dieser Output kann einer Variablen zugewiesen werden. Die paste0-Funktion nimmt beispielsweise mehrere Zeichenketten als Input entgegen und verbindet sie zu einer einzigen Zeichenkette, die dann als Ergebnis in einem neuen Objekt abgespeichert werden kann:

```
name <- paste0("Inger","Engmann")
```

Ähnlich funktionieren Operatoren, etwa die Grundrechenarten. Im Unterschied zu einem normalen Funktionsaufruf werden die Argumente links und rechts vom Operator angegeben:[6]

```
alter <- 2017 – 1989
jahr <- 1989 + alter
```

Beliebige Funktionen können mit dem folgenden Schema selbst definiert werden:

```
CalculateAge <- function(year.now,year.birth) {
  alter <- year.now – year.birth
  return (alter)
```

6 Wenn Argumente nicht in Klammern dahinter angegeben werden, sondern links und rechts von einer Funktion, dann spricht man von Infixnotation, ansonsten von Präfixnotation.

```
}
```

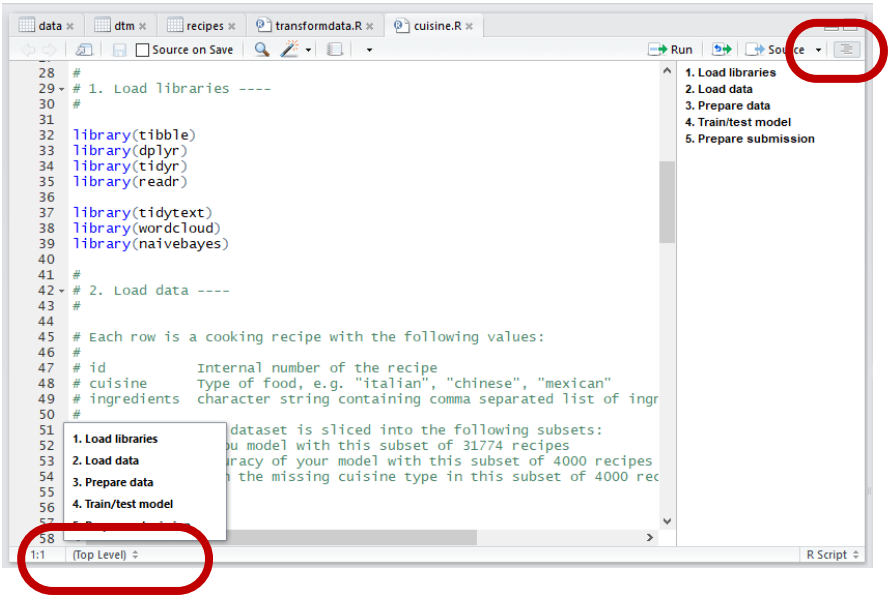
Der Name der Funktion steht links vom Zuweisungszeichen – Sie können sich einen beliebigen Namen bestehend aus Buchstaben und Ziffern ausdenken. Häufig kommen auch der Punkt oder der Unterstrich in Funktionsnamen vor. Die Parameter werden in Klammern angegeben. Innerhalb der geschweiften Klammern folgt der Code, welcher in der Funktion ausgeführt werden soll. Am Ende wird ein Wert zurückgegeben (return).[7] Diese Funktion kann nun über den Namen verwendet werden:

```
alter <- CalculateAge(2017, 1989)
```

Kommentare sind ein weiteres wichtiges Element von Skripten. Sie beginnen mit einem Rautezeichen und helfen dabei, den Code zu beschreiben, so dass man ihn später besser versteht:

```
# Dies ist ein Kommentar
```

In RStudio lässt sich ein Script mit Kommentaren in einzelne Abschnitte einteilen, indem an eine Kommentarzeile mindestens vier Bindestriche angehängt werden



7 Ohne return wird das letzte Objekt zurückgegeben.

(siehe Abbildung). Die so gekennzeichneten Zeilen werden in ein Inhaltsverzeichnis aufgenommen. Dieses Inhaltsverzeichnis lässt sich über die Outline-Buttons rechts oben oder unterhalb des Scripts einblenden, einzelne Abschnitte können so schnell angesprungen werden.

Die Dokumentation von Code ist wichtig, denn kaum jemand kann sich noch Monate später genau erinnern, warum welcher Befehl wo eingefügt wurde. Zusätzlich ist wie beim Verfassen von Romanen oder Aufsätzen auch beim Programmieren ein konsistenter Schreibstil (coding style) für die Verständlichkeit sehr hilfreich. Sie sollten sich beispielsweise entscheiden, ob Sie einzelne Teile von Bezeichnern mit Punkten, Unterstrichen oder CamelCase trennen wollen. Alle diese Varianten und noch einige mehr sind in R möglich. Sie können aber auch schnell zu Verwirrung oder Fehlern führen, denn bei unterschiedlicher Schreibweise sind es auch unterschiedliche Bezeichner:

```
calculate.age
calculate_age
calculateAge
CalculateAge
```

Sie können sich zum Beispiel an Google's R Style Guide orientieren (https://google.github.io/styleguide/Rguide.xml). Dort wird für Variablen die Schreibweise mit Punkten („calculate.age“) und für Funktionen CamelCase mit einem großen Anfangsbuchstaben empfohlen („CalculateAge“).

5.1.5 Die Ausgabe in der Kommandozeile

Beim Programmieren macht man Fehler, das ist nahezu unvermeidlich. Dazu kommt, dass jeder einzelne Computer seine Eigenarten hat, die schnell zu Stolpersteinen werden. Deshalb besteht ein wesentlicher Teil der Entwicklung in der Fehlersuche und -behebung. Das ist im ersten Moment ärgerlich – es führt aber gleichzeitig immer wieder zu kleinen Erfolgserlebnissen, wenn ein Fehler behoben ist.

Ein Programmierfehler wird auch bug genannt und die Fehlersuche debugging. Programmierumgebunden wie RStudio stellen dafür verschiedene Werkzeuge bereit. Insbesondere kann man sich Schritt für Schritt an die fehlerhafte Stelle

herantasten und nicht nur tief in den Quellcode der eigenen Skripte eintauchen, sondern auch die Pakete von anderen erkunden. Die Arbeit mit diesen Werkzeugen ist zum Beispiel in der Dokumentation von RStudio erläutert.[8]

Bevor man richtiges Debugging lernt, ist es sehr hilfreich, sich mit den Ausgaben auf der Kommandozeile näher zu beschäftigen. Hier werden nicht nur die Befehle eingegeben und die Ergebnisse ausgegeben. Auf der Kommandozeile werden auch Fehler angezeigt. Grundsätzlich muss man zwei Sorten von Meldungen unterscheiden:

1. **Warnungen** treten auf, wenn die aktuellen Umstände eines Funktionsaufrufs nicht ganz dem entsprechen, was der Entwickler bzw. die Entwicklerin erwartet hat. Das Programm läuft trotzdem weiter. Sie sollten aber versuchen, die Warnung zu verstehen und abschätzen, welche Folgen sich ergeben. Denn möglicherweise entsprechen die Ergebnisse der Datenanalyse nicht dem, was Sie erwarten. Unkritische Warnungen ergeben sich häufig beim Einbinden von packages:

```
> library(dplyr)
Attache Paket: 'dplyr'
The following objects are masked from 'package:stats':
    filter, lag
The following objects are masked from 'package:base':
    intersect, setdiff, setequal, union
Warning message:
Paket 'dplyr' wurde unter R Version 3.4.4 erstellt
```

   Die Warnung in der letzten Zeile ist in der Regel unkritisch und kann meistens ignoriert werden. Sie deutet lediglich darauf hin, dass sich Ihre R-Version von der für das Package verwendeten R-Version unterscheidet. Diese Meldung werden Sie gegebenenfalls durch ein Aktualisierung Ihrer R-Version los.

   Praktisch relevant kann aber der Hinweis über die Maskierung von Objekten sein. Dieser Hinweis entsteht dadurch, dass in verschiedenen Packages Funktionen oder Objekte mit gleichem Namen enthalten sind. Es wird dann

8 Siehe https://support.rstudio.com/hc/en-us/articles/205612627-Debugging-with-RStudio

immer auf das Objekt aus demjenigen Package zurückgegriffen, das zuletzt geladen wurde. Meistens ist das auch gewünscht. Im Beispiel würde ein Aufruf der filter-Funktionen die Funktion aus dem gerade geladenen dplyr-Package aufrufen. Falls dagegen die Funktion aus dem stats-Package aufgerufen werden soll, dann muss der Name des Packages getrennt durch einen *doppelten* Doppelpunkt explizit angegeben werden: `stats::filter()`

Warnungen entstehen auch bei der Datenanalyse, zum Beispiel bei der Visualisierung:

```
> ggplot(data,aes(x=day,y=news)) +
+   geom_point()
Warning message:
Removed 8 rows containing missing values (geom_point).
```

Bei einer solchen Warnung sollten Sie überlegen, ob sie nachvollziehbar ist und Ihren Erwartungen entspricht. Wenn Ihnen nicht bewusst war, dass in den Daten acht Werte gefehlt haben, dann schauen Sie sich die Daten noch einmal an und gehen Sie der Ursache auf den Grund!

2. **Fehler** führen dazu, dass das Programm abbricht. Hier bleibt Ihnen nichts anderes übrig, als sich auf die Suche nach der Ursache zu machen:

```
> ggplot(data,aes(x=day,y=Fatalities)) +
+   geom_point()
Error in FUN(X[[i]], ...) : object 'Fatalities' not found
```

Die Meldung im Beispiel weist darauf hin, dass ein Objekt nicht gefunden wurde. Im besten Fall liegt das einfach an einem Tippfehler, etwa weil Sie Groß- und Kleinschreibung verwechselt haben. Wenn Sie keine Idee haben, woher der Fehler stammen könnte, dann kopieren Sie die Meldung in die Suchmaschine Ihrer Wahl. Häufig sind Sie nicht der oder die erste mit einer solchen Meldung und können auf Hilfe hoffen. Eine gute Quelle für Hilfestellungen ist die Seite stackoverflow.com. Hier können Sie auch selbst Fragen stellen, wenn Sie nicht weiterkommen.

Geben Sie also auf rote Texte acht. Nicht immer bedeutet das ein Problem. Aber häufig hilft es bei der Behebung und Vermeidung von Problemen. Ein wesentlicher Teil von Entwicklungsarbeit besteht in der Bearbeitung und Lösung von solchen Problemen.

5.1.6 Funktionen für die Datenanalyse

Schon in der Basisausstattung bringt R eine Vielzahl von Funktionen für die Datenanalyse mit, zum Beispiel:

Funktion	**Beschreibung**
table	Zum Auszählen von Häufigkeiten und für das Erstellen von Kreuztabellen.
plot	Zum Ausgeben von Grafiken, beispielsweise Streudiagrammen.
boxplot	Zur Ausgabe von Boxplots.
summary	Gibt eine Übersicht über die Werte aus, bei Vektoren mit Zahlen die Fünf-Punkte-Zusammenfassung.

Versuchen Sie herauszufinden, wie diese Funktionen funktionieren: geben Sie die Namen in ein R-Script ein, setzen Sie den Cursor auf den Namen und rufen Sie dann in RStudio mit der Taste F1 die Hilfe auf.

Weiterhin sind verschiedene Arten statistischer Tests und Modellierungswerkzeuge eingebaut, zum Beispiel:

Funktion	**Beschreibung**
chisq.test	Chi-Quadrat-Test
t.test	T-Test
lm	Lineares Modell (Regression)
anova	Varianzanalyse
glm	Verallgemeinertes Lineares Modell

Viele nützliche Funktionen für die Datenanalyse werden darüber hinaus in Packages bereitgestellt, die von verschiedenen Nutzern erstellt wurden. Im

Kontext sozialwissenschaftlicher Datenanalyse ist zum Beispiel die Funktion describe aus dem Package psych sehr nützlich. Eine gute Übersicht über die gängigsten Szenarien der Datenanalyse und wie sie mit R gelöst werden, finden Sie im Web unter dem Stichwort Quick-R auf der Seite https://www.statmethods.net.

Eine besondere Sammlung von Funktionen beinhaltet das Package tidyverse (https://www.tidyverse.org/). Es enthält eine Reihe anderer Packages, die auch einzeln geladen werden können.

- tibble: Führt einen Datentyp tbl_df ein, der die data.frames von R verbessert.
- readr und readxl: Erleichtern das Einlesen von Datensätzen mit Funktionen wie read_csv und read_xlsx.
- dplyr: Stellt Funktionen für das Filtern, Zusammenführen und Aggregieren von Datensätzen bereit.
- tidyr: Dient der Umformung von Datensätzen zwischen long und wide-Format (siehe Kapitel 4.2.4).
- ggplot2: Das ggplot2-Package stellt eine Vielzahl mächtiger Funktionen zum Erzeugen von Grafiken bereit.
- purr: Erweitert die Möglichkeiten für funktionale Programmierung.

Weitere nützliche Packages aus dem TidyVerse beziehen sich auf spezielle Datentypen:

- stringr für Zeichenketten (z. B. Reguläre Ausdrücke).
- lubridate für Datums- und Zeitangaben.
- DBI für den Zugriff auf Datenbanken wie MySQL.

Um zu erkunden, was in einem Package enthalten ist, gehen Sie in die Hilfe des Packages. Dazu klicken Sie in RStudio in der Liste aller Packages auf das Package. Sie können sich dann zunächst einen Überblick über die Funktionen verschaffen. Viele Packages stellen zudem Einführungen zur Verfügung, die sogenannten

Vignetten. Klicken Sie dazu in der Dokumentation auf den Punkt „User guides, package vignettes and other documentation“.

Das wichtigste Konzept des TidyVerse besteht darin, die Welt durch eine viereckige Brille zu betrachten: alle Daten werden möglichst in der Form von Tibbles verarbeitet. Tibbles sind nichts anderes als ganz normale Tabellen, für die gilt:

1. Jede Beobachtungseinheit wird in einer eigenen Tabelle abgespeichert. Wenn es beispielsweise um Zeitungsartikel, Medien, Sätze und Journalisten geht, wird für jeden dieser Typen in der Regel eine eigene Tabelle verwendet.

2. Jede Variable wird in einer eigenen Spalte vorgehalten. Eine Variable wäre zum Beispiel das Alter von Befragten. Im Kopf der Tabelle steht dann der Name der Variablen und die Werte werden darunter aufgeführt.

3. Jede Beobachtung hat eine eigene Zeile. Beobachtungen oder Fälle sind zum Beispiel die Befragten einer Studie. Die Antworten einer oder eines Befragten befinden sich möglichst alle in einer einzigen Zeile.

4. Jeder Wert ist in einer eigenen Zelle abgelegt. Eine Zelle ist dabei der Ort, an dem sich Spalte (vertikal) und Zeile (horizontal) treffen. Das Alter jeder einzelnen Person liegt also in einzelnen Zellen, es sollten nicht mehrere Angaben in einer gemeinsamen Zelle liegen.

Wer viel Datenanalyse betreibt, für den sind diese Regeln nahezu selbstverständlich. Es ist vor allem für die Zusammenarbeit mit anderen sehr hilfreich, sich an diese Regeln zu halten. Denn mit diesen Regeln lassen sich auch fremde Daten schnell verstehen. Selbst die eigenen Daten werden nach einiger Zeit fremd. Das gilt auch außerhalb von R: vermeiden Sie zum Beispiel in Excel verbundene Zellen, reservieren Sie immer die erste Zeile für Variablennamen, alle anderen Zeilen sind für Fälle da. Anmerkungen sollten Sie nicht in die Zellen schreiben, sondern die Anmerkungsfunktion verwenden oder ein eigenes Tabellenblatt beginnen.

Im TidyVerse stehen Tibbles über den Datentyp tibble zur Verfügung, der als Ersatz für die in R eingebauten data.frames funktioniert. Die Funktionen aus dem TidyVerse sind meistens sehr ähnlich aufgebaut und verlangen fast immer als

ersten Parameter ein Tibble. Sie geben auch ein Tibble zurück. Dadurch lassen sich verschiedene Funktionen sehr gut mit dem Pipe-Operator %>% aneinanderketten:

```
auswahl <- personen %>%
             filter(alter > 22) %>%
             select(name)
```

Jedem Befehl hinter einer Pipe wird als erster Parameter automatisch das vorherige Ergebnis übergeben. Dadurch werden Zuweisungen eingespart, was sich insbesondere bei längeren Operationen durch eine bessere Übersichtlichkeit auszahlt. Ohne die Pipe würde der gleiche Code wie folgt funktionieren:

```
auswahl <- filter(personen,alter > 22)
auswahl <- select(auswahl,name)
```

Vergleichen Sie hier die select- und filter-Befehle; mit der Pipe wird der erste Parameter jeweils übersprungen, da er aus der Zeile vorher stammt. Diese Aneinanderkettung funktioniert insbesondere mit den Funktionen aus dem Package dplyr sehr gut:

Funktion	**Beschreibung**
filter	Auswählen von Datensätzen.
select	Auswählen oder Umbenennen von Spalten.
arrange	Sortieren von Datensätzen
mutate	Neuberechnen von Spalten.
count	Zählen von Datensätzen, ggf. für einzelne Kategorien
group_by	Gruppieren von Datensätze
summarize	Aggregieren von Datensätzen, zum Beispiel zur Berechnung von Summen oder zur Bestimmung der Anzahl innerhalb einer Gruppe.
ungroup	Auflösen einer Gruppierung

Mit den letzten drei Funktionen lässt sich ein typisches Szenario der Datenanalyse umsetzen, das als Split-Apply-Combine bezeichnet wird. Hierbei wird ein Datensatz zunächst in Teilgruppen aufgeteilt (group_by), dann auf diese

Teilgruppen eine Funktion angewendet (summarize) und schließlich das Ergebnis zusammengefügt (ungroup). Auf diese Weise können Sie zum Beispiel in einem Experiment den Altersdurchschnitt von Personen innerhalb jeder einzelnen Experimentalgruppe berechnen. Wichtig ist nur, dass die Gruppenzugehörigkeit in einer eigenen Spalte erfasst ist, z.B.:

```
alter <- personen %>%
          group_by(typ) %>%
          summarize(durchschnitt=mean(alter)) %>%
          ungroup()
```

Wenn es um kategoriale Daten geht, dann ist die count-Funktion sehr hilfreich. Ohne Parameter wird einfach die Anzahl der Datensätze gezählt:

```
personen %>%
  count()
```

Die Anzahl lässt sich für einzelne Gruppe ermitteln, indem die Spalte angegeben wird, in der die Gruppe verzeichnet ist:

```
personen %>%
  count(typ)
```

Letztendlich handelt es sich dabei um eine Kurzfassung von Split-Apply-Combine. Die Langfassung sähe wie folgt aus:

```
personen %>%
   group_by(typ) %>%
   summarize(n=n()) %>%
   ungroup()
```

Verwirrend ist möglicherweise das doppelte Vorkommen von „n“ innerhalb der summarize-Funktion. Vor dem Gleichheitszeichen wird der Name der Spalte angegeben, in dem das Ergebnis gespeichert werden soll. Sie können das einfach umbenennen, z. B. zu „anzahl“. Nach dem Gleichheitszeichen wird die Funktion mit dem Namen „n“ mit leeren Klammern aufgerufen, das heißt ohne Argumente. Diese Funktion zählt die Anzahl der Zeilen.

Man kann sowohl in der count-Funktion als auch beim Gruppieren mit group_by mehrere Spaltennamen angeben und so die Kombination von Merkmalen auszählen. Das Ergebnis ist dann vergleichbar mit einer Kreuztabelle.

5.1.7 Datensätze einlesen und wieder exportieren

Die bisherigen Beispiele bezogen sich auf einen sehr kleinen Datensatz, der als data.frame direkt im Script eingegeben wurde. In der Regel liegen die Daten aber in eigenen Dateien vor, zum Beispiel in CSV-Dateien. Mit dem CSV-Format können nahezu alle Programme umgehen, die zur Erfassung oder Analyse von Daten eingesetzt werden (siehe Kapitel **Fehler! Verweisquelle konnte nicht gefunden werden.**). Das Einlesen und Speichern solcher Dateien gehört zu den Basisfunktionen von R, zum Beispiel die Funktion „read.csv“. Günstiger sind aber in der Regel die Funktionen aus dem Tidyverse, die mit einem Unterstrich statt Punkt geschrieben werden. Ein Beispiel ist die Funktion read_csv aus dem readr-Package:

```
personen <- read_csv("personen.csv")
```

Diese Funktion nimmt als ersten Parameter den Dateinamen in Anführungszeichen entgegen und gibt den Datensatz als Tibble zurück. Je nach Aufbau der Datendatei werden andere Funktionen oder weitere Parameter benötigt. Wenn die Daten nicht mit einem Komma (comma separated values), sondern mit einem Tabulator getrennt sind, dann hilft die Funktion read_tsv (tab separated values). Mit Semikolon getrennte Daten liest die Funktion read_csv2 ein.

Die Funktionen im Tidyverse haben meistens passendere Voreinstellungen als die Basisfunktionen, beispielsweise behandeln sie Text als Text und nicht als Faktor. Trotzdem kann mit weiteren Parametern auf nahezu beliebige Arten solcher Dateien zugegriffen werden. Insbesondere die Funktion read_delim lässt sich umfangreich konfigurieren. Der folgende Aufruf wäre identisch mit read_csv, da ein Komma als Trennzeichen definiert wird:

```
personen <- read_delim("personen.csv", delim=",")
```

Analog stehen Funktionen zum Speichern von CSV-Dateien zur Verfügung. Als erster Parameter wird dann der Datensatz angegeben und der zweite Parameter bestimmt den Dateinamen:

```
write_csv(personen,"personen.csv")
```

Wenn man konsequent mit Funktionen aus dem Tidyverse arbeitet, dann lassen sich die meisten Ergebnisse als solche Tabellen abspeichern, zum Beispiel das Ergebnis der oben besprochenen count-Funktion. Im folgenden Beispiel wird das Ergebnis zunächst in einem eigenen Objekt abgelegt und dann als csv-Datei abgespeichert:

```
personen.jetyp <- personen %>%
  count(typ)

write_csv(personen.jetyp,"personen.jetyp.csv")
```

Auf diese Weise lässt sich der Output der Datenanalyse gut dokumentieren und weiterverwenden. Statt in Dateien, können Daten auch in die Zwischenablage geschrieben werden. Dazu gibt man statt eines Dateinamens das Stichwort „clipboard“ an. Wenn man dabei Tabulatoren als Trennzeichen verwendet, lässt sich der Inhalt gut in Excel oder ähnlichen Programmen wieder einfügen:

```
write_tsv(personen.jetyp,"clipboard")
```

Die folgende Funktion verwendet als Trennzeichen ein Semikolon, als Dezimalzeichen das Komma und markiert die Zeichenkodierung als UTF-8 (Byte Order Mark am Anfang, siehe Kapitel 2.1). Diese Voreinstellungen sind besonders für den Austausch mit Excel geeignet:

```
write_excel_csv2(personen.jetyp,"clipboard")
```

Für den Austausch mit Excel gibt es in den Pakteten readxl und writexl auch eigene Funktionen:[9]

```
write_xlsx(personen.jetyp,"personen.jetyp.xlsx")

personen <- read_xlsx("personen.xlsx")
```

Die Dateienamen beziehen sich in allen Fällen auf das aktuelle Arbeitsverzeichnis. Wenn man mit RStudio ein Projekt angelegt hat, dann ist das Arbeitsverzeichnis zu Beginn immer das Projektverzeichnis. Es können zum Einlesen der Dateien

9 Bei Dateien aus anderen Statistikprogrammen wie SPSS, Stata oder SAS hilft das Paket „haven“ weiter.

und zum Speichern auch relative Pfade verwendet werden. Es empfiehlt sich, Daten und Ergebnisse in Unterverzeichnissen abzulegen, zum Beispiel in einem Unterverzeichnis mit dem Namen „daten“:

```
write_xlsx(personen,"daten/personen.xlsx")

personen <- read_xlsx("daten/personen.xlsx")
```

Für die Arbeit mit Textkorpora müssen manchmal Textdateien oder auch Worddokumente eingelesen werden. Beispiele dafür finden Sie im Kapitel **Fehler! Verweisquelle konnte nicht gefunden werden.** Textanalyse.

5.1.8 Grafiken

Ein wichtiger Teil der Datenauswertung besteht in der Visualisierung von Ergebnissen. Inspiration und Lösungsvorschläge für die Visualisierung mit R finden Sie in der R Graph Gallery.[10] Sehr häufig wird das Paket ggplot2 verwendet, das nicht nur eine beeindruckende Menge an Funktionen bereitstellt, sondern die Funktionalität in ein aufgeräumtes Gesamtkonzept integriert und damit eine Art Grafiksprache etabliert.[11] Die Grundidee besteht darin, dass eine Grafik aus mehreren übereinandergelegten Schichten besteht. Eine solche Schicht enthält zum Beispiel ein Säulendiagramm (=geom_col). Das Erscheinungsbild (=aesthetics) des Säulendiagramms wird dann durch die Daten bestimmt. Hierzu werden die Spalten einer Tabelle den visuellen Merkmalen zugeordnet (=mapping).

Eine solche Grafik beginnt zunächst mit dem Aufruf der Funktion ggplot, bei dem die für alle Schichten gemeinsamen Merkmale angegeben werden. Das erste Argument enthält den data.frame. In dieser Tabelle ist das Ergebnis der Datenauswertung enthalten, etwa das Ergebnis der count-Funktion. Das zweite Argument ist ein Aufruf der Funktion aes und definiert die Zuordnung zwischen grafischen Elementen und Daten. Im folgenden Beispiel wird die x-Achse der Grafik durch die Spalte „typ“ und die y-Achse durch die Spalte „n“ erzeugt. Zu diesem Grundobjekt wird dann durch das Plus-Zeichen eine Schicht mit einem Säulendiagrammm hinzugefügt:

10 https://www.r-graph-gallery.com/

11 Das Konzept geht zurück auf das Buch „The Grammar of Graphics“ von Leland Wiklinson (2005).

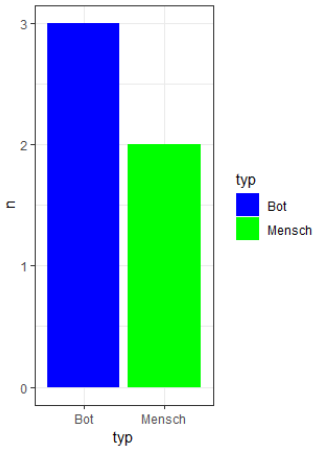
```
library(ggplot2)

ggplot(personen.jetyp,aes(x=typ,y=n)) +
  geom_col()
```

Dieser Grundaufbau lässt sich nun erweitern, beispielsweise mit Farben für die beiden Säulen (siehe Abbildung 2, links). Die Füllfarbe wird ebenfalls durch die Spalte „typ“ erzeugt. Um die Farben selbst festzulegen, kann eine Skala für die Füllung hinzugefügt werden. In dieser Skale werden für die beiden möglichen Typen manuell Farben festgelegt (scale_fill_manual). Das allgemeine Erscheinungsbild von Grafiken lässt sich zudem durch *themes* steuern. Im folgenden Beispiel wird ein einfaches Schwarz-Weiß-Thema (bw=black and white) verwendet:

```
ggplot(personen.jetyp,aes(x=typ,y=n,fill=typ)) +
  geom_col() +
  scale_fill_manual(values=c("blue","green")) +
  theme_bw()
```

Säulendiagramm



Boxplot

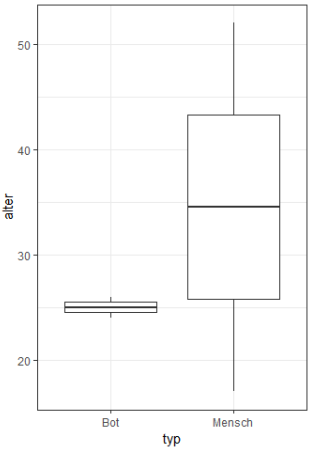


Abbildung 2: Grafiken mit ggplot2

Beim Erzeugen von Grafiken ist es häufig sinnvoll, die Daten zunächst selbst auszuwerten und dann die fertigen Ergebnisse mit ggplot darzustellen. So setzt das vorangegangene Beispiel voraus, dass die Anzahl der Zeilen je Typ in der Spalte „n" abgelegt wurde. Dieses Ergebnis wurde mit der count-Funktion berechnet. Mitunter ist die Auswertung aber auch in die Grafikfunktionen integriert. Vor allem bei der Erzeugung von Boxplots lässt sich der Datensatz in der ursprünglichen, nicht aggregierten Form einsetzen, die notwendigen Parameter berechnet die Funktion geom_boxplot selbst (siehe Abbildung 2, rechts):

```
ggplot(personen,aes(x=typ,y=alter)) +
  geom_boxplot() +
  theme_bw()
```

Die beiden Beispiele sind nur ein kleiner Ausschnitt aus den vielen Möglichkeiten der Visualisierung. Mit ggplot2 lassen sich genauso Punktdiagramme, Liniendiagramme und sogar Netzwerkgrafiken erzeugen. Hinweise zu Tutorials finden Sie auf der Webseite des Pakets: https://ggplot2.tidyverse.org.

Das Ergebnis kann als Bilddatei abgespeichert werden, zum Beispiel als PNG-Datei. Für manuelle Nacharbeiten empfiehlt sich das SVG-Format, das mit kostenlosen Programmen wie Inkscape[12] weiterbearbeitet werden kann.

5.1.9 Weiterführende Literatur

Durch die starke Community findet sich eine Vielzahl sehr guter Hilfestellungen rund um R im Web. Die zentrale Anlaufstelle ist hierbei The Comprehensive R Archive Network (CRAN): https://cran.r-project.org. Dort werden auch die meisten Packages verwaltet. Die Dokumentation der Packages kann nicht nur über RStudio, sondern etwa in der Form von PDF-Dateien auch von CRAN bezogen werden.

Typische Lösungen für typische Probleme aber auch eine übersichtliche Einführung finden Sie bei Quick-R unter https://www.statmethods.net. Spezielle

12 Inkscape ist ein Programm zur Bearbeitung von Vektorgrafiken: https://inkscape.org/

Probleme werden häufig bei Stackoverflow besprochen, hier können Sie auch selbst Fragen stellen: https://stackoverflow.com. Gerade in der ersten Lernphase einer Sprache sind Cheat Sheets hilfreich, auf denen die wichtigsten Funktionen knapp zusammengefasst sind. Eine Zusammenstellung solcher Blätter bietet die Webseite von RStudio an: https://www.rstudio.com/resources/cheatsheets/

Eine umfassende Einführung in die Datenanalyse mit R mit einem Fokus auf das TidyVerse finden Sie in:

> Wickham, Hadley; Grolemund, Garrett (2017): R for Data Science. Sebastopol, CA: O'Reilly UK Ltd. http://r4ds.had.co.nz.

Darüber hinaus gibt es einige spezialisierte Bücher für die Analyse mit R, zum Beispiel:

> Silge, Julia; Robinson, David (2017): Text mining with R. A tidy approach. Beijing, Boston, Farnham: O´Reilly. http://tidytextmining.com.

Diese beiden Bücher stehen unter einer Creative Commons-Lizenz, sind sowohl kostenlos online verfügbar als auch in gedruckter Form im Buchhandel käuflich.