R Einführung

Dr. Roman Kalkreuth

11.04.2022

R (Programmiersprache)

R lässt sich über https://www.r-project.org/ installieren. Wir werden Version 4.0.0 oder höher verwenden.

R Studio (IDE)

R Studio lässt sich über https://www.rstudio.com/ installieren.

Ich empfehle, R Studio als IDE zu verwenden.

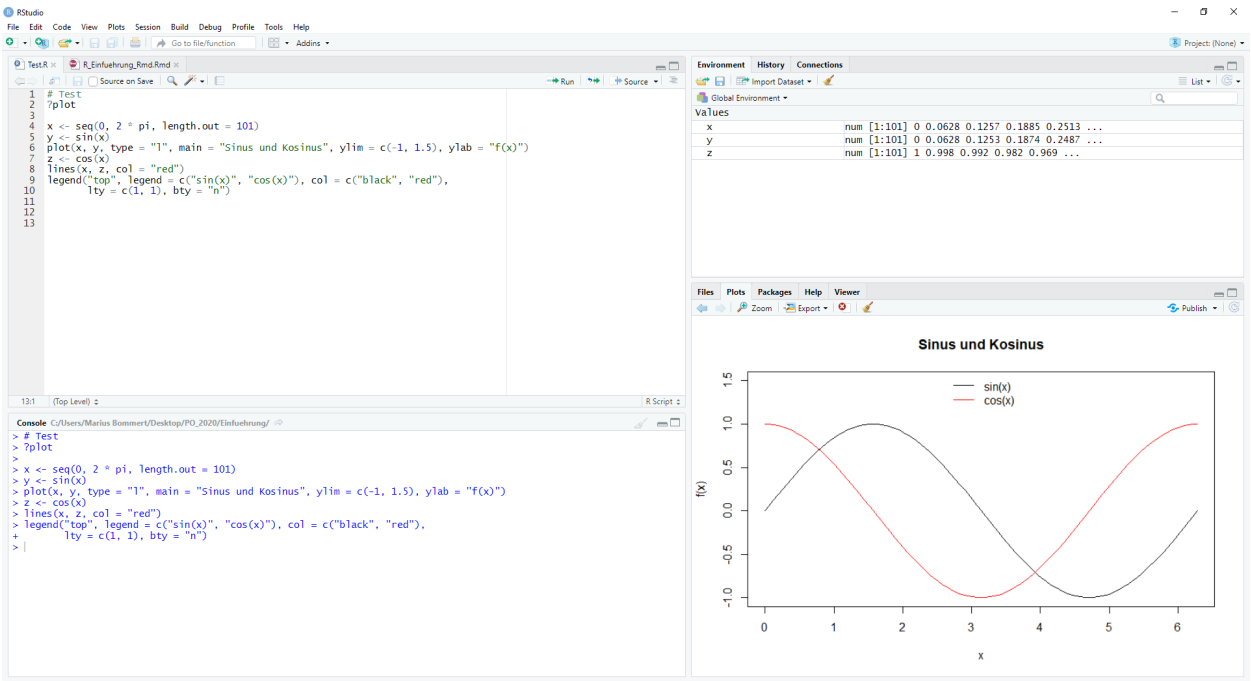


Figure 1: R Studio

In R Studio stehen 4 Fenster zur Verfügung. Oben links wird Code aus geöffneten Dateien angezeigt. Ist eine R-Datei geöffnet (Endung .R), so können über den Button **Run** einzelne Zeilen oder markierter Code ausgeführt werden. Über den Button **Source** kann der gesamte Code in der Datei ausgeführt werden. Ist eine R Markdown-Datei geöffnet, so steht statt **Source** ein Button für **Knit** zur Verfügung, mit dem ein Dokument erstellt werden kann.

Im Fenster unten links ist die Konsole zu finden, in der R-Code ausgeführt werden kann. Zum einen kann direkt in die Konsole R-Code eingegeben und mit **Enter** ausgeführt werden. Zum anderen erscheint in der Konsole auch der R-Code aus dem Fenster oben links, der mit **Run** ausgeführt wurde. Wichtig: Hier erscheinen auch Warnungen und Fehlermeldungen. Diese bitte nicht ignorieren!

Im Fenster oben rechts lässt sich insbesondere der Global Environment anschauen. Dort sind alle Variablen, Funktionen, etc. zu finden, die in der aktuellen R Session zur Verfügung stehen. Es ist wichtig, darauf zu achten, dass der R-Code, der bei den Übungsblättern abgegeben wird auch noch läuft, wenn der Global Environment vorher geleert wurde. Für nicht lauffähigen Code werden in der Regel 0 Punkte vergeben.

Im Fenster unten rechts sind insbesondere der Bereich **Plots** interessant, wo Plots angezeigt werden und der Bereich **Help**, wo es möglich ist, sich Hilfeseiten zu einzelnen Funktionen anzuschauen.

R Markdown (Berichte mit R Output)

Wenn eine Rmd-Datei in RStudio geöffnet ist und der **Knit**-Button geklickt wird, wird ein Dokument erzeugt, dass sowohl den geschriebenen Text als auch den Output des eingefügten R-Codes enthält. Wird als Output-Format pdf_document verwendet, so wird ein PDF-Dokument erstellt. Dazu wird TeX auf dem Computer benötigt. Dies ist eine sehr einfache Möglichkeit, Aufgaben mit R zu bearbeiten und die Ergebnisse zu erläutern. Teilweise kann es auch sinnvoll sein html_document als Output-Format zu verwenden. Dies ist beispielsweise sinnvoll, wenn interaktive Grafiken genutzt werden sollen, die sich in einer pdf-Datei nicht darstellen lassen.

Dieses Dokument wurde mit R Markdown erzeugt und dient als Beispiel für die Verwendung von R Markdown.

Weitere Informationen zu R Markdown sind unter https://rmarkdown.rstudio.com zu finden.

Komentare

Kommentare werden in R mit # begonnen. Der R-Code sollte **immer** kommentiert werden.

R Hilfe

Hilfe zu einzelnen Funktionen kann man am leichtesten über ?Funktionsname erhalten, wie zum Beispiel mit `?plot`.

Wichtigste Datentypen

Variablen sind in `R`, anders als in vielen anderen Programmiersprachen, ungetypt! Der Datentyp einer Variable kann sich ändern! Dies ist häufig hilfreich, aber man muss teilweise auch sehr aufpassen.

Beispiele für die wichtigsten Datentypen:

```
# Integer:
x = 10L # Zuweisungen mit = oder <-
x
```

```
## [1] 10
```

```
typeof(x = x) # R ist Case-sensitive
```

```
## [1] "integer"
```

```
# Double:
x = 10
x
```

```
## [1] 10
```

```
typeof(x = x)
```

```
## [1] "double"
```

```
# Character:
x = "Hallo"
x
```

```
## [1] "Hallo"
```

```
typeof(x = x) # In anderen Programmiersprachen waere dies haeufig ein String.
```

```
## [1] "character"
```

```
# Logical:
x = TRUE
x
```

```
## [1] TRUE
```

```
typeof(x = x)
```

```
## [1] "logical"
```

```
2 * x
```

```
## [1] 2
```

```
typeof(x = 2 * x) # Mit logicals kann auch gerechnet werden.
```

```
## [1] "double"
```

```
# Factor:
x = factor(x = c("a", "b", "b"), levels = c("a", "b")) # c: combine von Werten zu Vektor
x
```

```
## [1] a b b
## Levels: a b
```

```
typeof(x = x) # Wird als Integer abgespeichert!
```

```
## [1] "integer"
```

Wichtigste Datenstrukturen

Eindimensionale Datenstrukturen sind Vektoren und Listen. Vektoren sollten Elemente vom gleichen Typ enthalten und Listen sollten für Elemente von unterschiedlichen Typen genutzt werden. Eine Liste darf Einträge beliebiger Länge haben.

Beispiel:

```
# Vektoren:
x = c(1, 2, 3)
x
```

```
## [1] 1 2 3
```

```
x = 1:3 # Kurzschreibweise für c(1, 2, 3) bzw. seq(from = 1, to = 3, by = 1)
x
```

```
## [1] 1 2 3
```

```
typeof(x = x)
```

```
## [1] "integer"
```

```
x = c(1, "2", 3)
x
```

```
## [1] "1" "2" "3"
```

```r
typeof(x = x) # alle Elemente werden zum Typ character umgewandelt!
```

```
## [1] "character"
```

```r
# Laenge des Vektors:
length(x = x)
```

```
## [1] 3
```

```r
# Erstes Element extrahieren (Indexierung beginnt mit 1!!!):
x[1]
```

```
## [1] "1"
```

```r
# Letztes Element weglassen:
x[-3]
```

```
## [1] "1" "2"
```

```r
# Listen:
x = list(a = 1L, b = c(1, 2), c = "1", d = TRUE)
x
```

```
## $a
## [1] 1
##
## $b
## [1] 1 2
##
## $c
## [1] "1"
##
## $d
## [1] TRUE
```

```r
typeof(x = x)
```

```
## [1] "list"
```

```r
# Struktur der Liste anzeigen lassen (Beschreibung siehe unten):
str(object = x) # Hier werden double als num angegeben.
```

```
## List of 4
##  $ a: int 1
##  $ b: num [1:2] 1 2
##  $ c: chr "1"
##  $ d: logi TRUE
```

```r
# Auf erstes Element der Liste zugreifen:
x[[1]]
```

```
## [1] 1
```

Zweidimensionale Datenstrukturen sind Matrizen und Dataframes. Matrizen sollten wie Vektoren nur Elemente vom gleichen Typ enthalten und Dataframes sollten wie Listen für Elemente von unterschiedlichen Typen genutzt werden. Bei Dataframes ist es wichtig, dass jede Spalte die gleiche Länge hat.

Beispiel:

```
# Matrizen:
x = matrix(data = 1:4, nrow = 2)
x
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
y = matrix(data = 1:8, nrow = 2)
y
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    5    7
## [2,]    2    4    6    8
```

```
# Zugriff auf bestimmtes Element:
x[1, 1]
```

```
## [1] 1
```

```
# Zugriff auf erste Zeile:
x[1, ]
```

```
## [1] 1 3
```

```
# Zugriff auf erste Spalte:
x[, 1]
```

```
## [1] 1 2
```

```
# Addition von Matrizen:
x + y[, 1:2]
```

```
##      [,1] [,2]
## [1,]    2    6
## [2,]    4    8
```

```
# Multiplikation von Matrizen:
x %*% y
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    7   15   23   31
## [2,]   10   22   34   46
```

```
# Elementweise Multiplikation von Matrizen:
x * y[, 1:2]
```

```
##      [,1] [,2]
## [1,]    1    9
## [2,]    4   16
```

```r
# Dataframes:
x = data.frame(v1 = c(1, 2, 3), v2 = c("a", "b", "c"),
               v3 = factor(x = c("a", "b", "c"), levels = c("a", "b", "c")))
x
```

```
##   v1 v2 v3
## 1  1  a  a
## 2  2  b  b
## 3  3  c  c
```

```r
# Struktur des Dataframes (Beschreibung siehe unten):
str(x)
```

```
## 'data.frame':    3 obs. of  3 variables:
##  $ v1: num  1 2 3
##  $ v2: chr  "a" "b" "c"
##  $ v3: Factor w/ 3 levels "a","b","c": 1 2 3
```

```r
# character wurden vor Version 4.0.0 zu factor, wenn nicht das Argument
# stringsAsFactors = FALSE verwendet wurde. Ab R 4.0.0 ist dies der Default.

# Namen der Spalten:
colnames(x = x)
```

```
## [1] "v1" "v2" "v3"
```

```r
# Anzahl der Zeilen/Spalten:
nrow(x = x)
```

```
## [1] 3
```

```r
ncol(x = x)
```

```
## [1] 3
```

```r
# Liste?
is.list(x = x)
```

```
## [1] TRUE
```

```r
# Ein Dataframe ist eine Liste mit Eintraegen gleicher Laenge.
```

Verwendung der plot-Funktion

Das folgende Beispiel zeigt wie 101 äquidistante Punkte aus dem Intervall $[0, 2\pi]$ erzeugt werden und der Variable x zugewiesen werden. Danach werden die zugehörigen y-Werte berechnet mit $y = \sin(x)$ und die Punkte werden mithilfe der `plot`-Funktion geplotted. Der Kosinus wird mithilfe von `lines` hinzugefügt. Zum Schluss wird eine Legende ergänzt. Dies ist wichtig, damit der Plot verständlich wird.

```r
# 101 aequidistante Punkte in Intervall [0, 2*pi] erzeugen:
x = seq(from = 0, to = 2 * pi, length.out = 101)
head(x = x) # zeigt erste Elemente an
```

```
## [1] 0.00000000 0.06283185 0.12566371 0.18849556 0.25132741 0.31415927
```

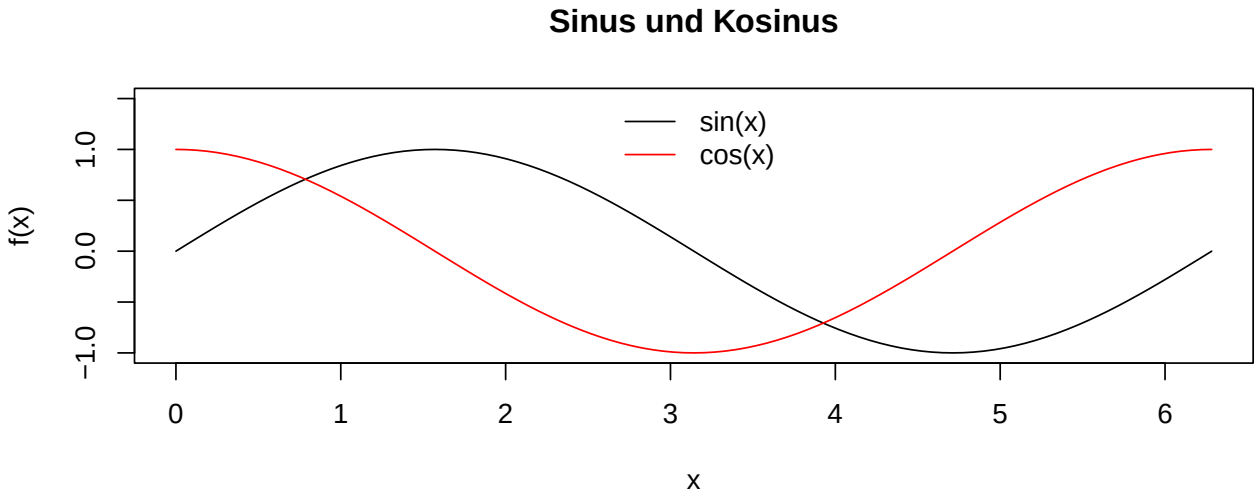
```r
# Sinus von x berechnen:
y = sin(x = x)
head(x = y)
```

```
## [1] 0.00000000 0.06279052 0.12533323 0.18738131 0.24868989 0.30901699
```

```r
# Plotten der Sinus-Werte mit Hilfe der Funktion plot:
plot(x = x, y = y, type = "l", main = "Sinus und Kosinus", ylim = c(-1, 1.5), ylab = "f(x)")
# Berechnung des Kosinus von x:
z = cos(x = x)
head(x = z)
```

```
## [1] 1.0000000 0.9980267 0.9921147 0.9822873 0.9685832 0.9510565
```

```r
# Hinzufügen des Kosinus mit Hilfe der Funktion lines:
lines(x = x, y = z, col = "red")
# Legende hinzufuegen, um Grafik verstaendlich zu machen:
legend(x = "top", legend = c("sin(x)", "cos(x)"), col = c("black", "red"),
       lty = c(1, 1), bty = "n")
```



Struktur von Objekten

Mithilfe der Funktion `str` lässt sich die Strukur von R-Objekten anschauen. Dies kann teilweise hilfreich sein, um zu sehen von welchem Typ ein R-Objekt ist und wie es aufgebaut ist. Bei dem nachfolgenden Beispiel lässt sich sehen, dass die Werte in x vom Typ num sind und dass x 101 Werte zugewiesen sind. Mit der Option `max.level` lässt sich die maximale Tiefe der Struktur begrenzen. Dies kann insbesondere bei großen Listen hilfreich sein. In diesem Beispiel hat die Option max.level keinen Einfluss auf das Ergebnis.

```r
str(x, max.level = 1)
```

```
##  num [1:101] 0 0.0628 0.1257 0.1885 0.2513 ...
```

Definition von Funktionen

Eine Funktion kann mithilfe von `function` definiert werden. Nachfolgend steht ein einfaches Beispiel für die Berechnung der Summe von x und y.

```r
summe = function(x, y) {
  return(x + y)
}
summe
```

```
## function(x, y) {
##   return(x + y)
## }
```

```r
# Summe von Zahlen:
summe(x = 1, y = 2)
```

```
## [1] 3
```

```r
# Elementweise Summe von Vektoren:
summe(x = c(1, 2), y = c(2, 3))
```

```
## [1] 3 5
```

```r
# Elementweise Summe von Matrizen:
summe(x = matrix(data = 1:4, nrow = 2), y = matrix(data = 5:8, nrow = 2))
```

```
##      [,1] [,2]
## [1,]    6   10
## [2,]    8   12
```

Paket installieren, laden und nutzen

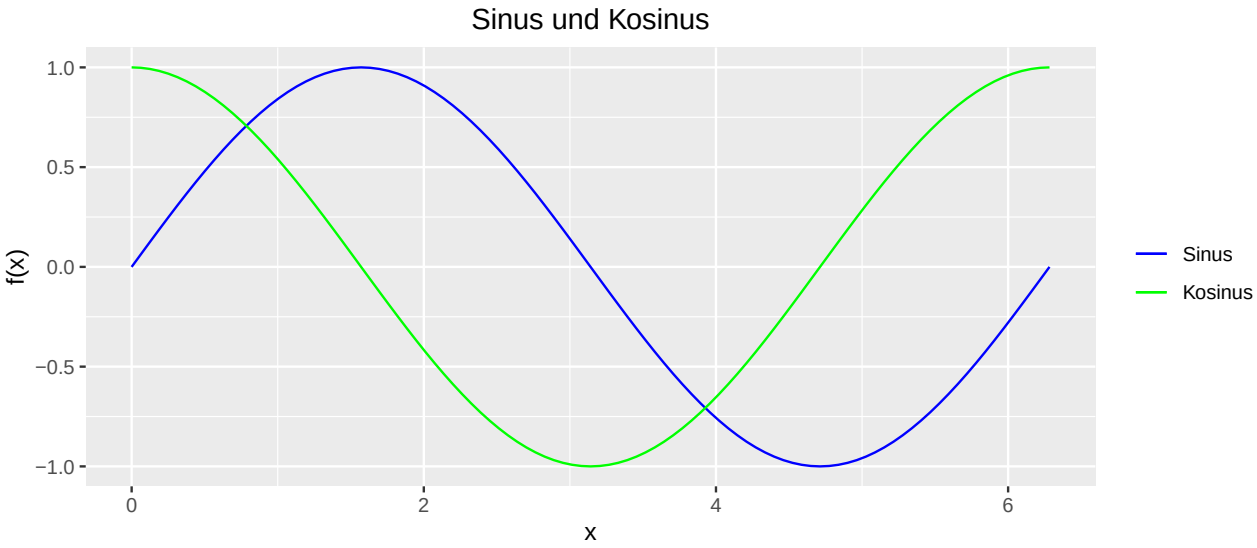
Ein Paket lässt sich mit der Funktion `install.packages` installieren. Wir werden überwiegend Pakete von https://cran.r-project.org/ verwenden. Dort ist auch eine Liste mit verfügbaren Paketen zu finden. Eine weitere Möglichkeit besteht darin Pakete mithilfe der Funktion `install_github` aus dem Paket `devtools` von GitHub zu installieren. Dort sind häufig aktuellere Versionen von CRAN-Paketen zu finden, bei denen beispielsweise Fehler behoben oder neue Funktionalitäten ergänzt wurden. Anschließend kann ein installiertes Paket mit `library` geladen werden. Im nachfolgenden Beispiel wird das Paket `ggplot2` mit der Funktion `ggplot` genutzt, um eine ähnliche Grafik wie im Beispiel weiter oben zu erzeugen. Die Daten müssen für die Funktion `ggplot` als Dataframe gespeichert werden.

```r
# install.packages("ggplot2")
library("ggplot2")

data = data.frame(x = x, y = y, z = z)
head(x = data) # zeigt erste Zeilen an
```

```
##            x          y         z
## 1 0.00000000 0.00000000 1.0000000
## 2 0.06283185 0.06279052 0.9980267
## 3 0.12566371 0.12533323 0.9921147
## 4 0.18849556 0.18738131 0.9822873
## 5 0.25132741 0.24868989 0.9685832
## 6 0.31415927 0.30901699 0.9510565
```

```r
print(ggplot(data = data, mapping = aes(x = x, y = y, col = "blue")) + # Plot von data
        geom_line() + # Linie zur Darstellung verwenden
        geom_line(mapping = aes(x = x, y = z, col = "green")) +  # Linie fuer Kosinus
        labs(title = "Sinus und Kosinus", x = "x", y = "f(x)") + # Beschriftung
        theme(plot.title = element_text(hjust = 0.5), # Plot anpassen
              legend.key = element_rect(fill = "transparent", colour = NA),
              legend.title = element_blank()) +
        scale_color_manual(values = c("blue", "green"), # Farbe und Label anpassen
                           labels = c("Sinus", "Kosinus")))
```



apply

Die Funktionen `apply` sowie `lapply`, `vapply` und `replicate` können hilfreich sein, um wiederholt Funktionen anzuwenden. Ein einfaches Beispiel der Berechnung der Summe ist nachfolgend gegeben.

```
apply(X = data, MARGIN = 2, FUN = sum)
```

```
##             x             y             z
##  3.173009e+02 -2.466641e-16  1.000000e+00
```

Die 2 gibt hierbei an, dass die Funktion (hier `sum`) auf die Spalten angewendet werden soll.

Auch die Verwendung von `sapply` ist möglich, wobei hier sehr darauf geachtet werden muss, dass das Ergebnis nicht anders vereinfacht wird als gewünscht.

3D-Plot

3D-Plots können beispielsweise mit der Funktion `plot3d` aus dem Paket `rgl` erzeugt werden. Nachfolgend ist der Code für $f(x, y) = x + y$ zu sehen.

```
library("rgl")
```

```
## This build of rgl does not include OpenGL functions.  Use
##  rglwidget() to display results, e.g. via options(rgl.printRglwidget = TRUE).
```

```
# Gitter zum Auswerten der Funktion erzeugen:
grid = expand.grid(x = seq(from = -10, to = 10, length.out = 101),
                   y = seq(from = -10, to = 10, length.out = 101))
head(x = grid, n = 5) # n gibt Anzahl Zeilen an
```

```
##       x   y
## 1 -10.0 -10
## 2  -9.8 -10
## 3  -9.6 -10
## 4  -9.4 -10
## 5  -9.2 -10
```

```
data = cbind(grid, z = apply(X = grid, MARGIN = 1,
                             FUN = function(x) x[1] + x[2]))
```

```r
plot3d(x = data$x, y = data$y, z = data$z)
# 3d-Plot kann nicht in pdf angezeigt werden.
```

For-Schleife, Zufallszahlen und summary

Werden Zufallszahlen verwendet, so sollte ein Seed verwendet werden, um die Ergebnisse reproduzieren zu können. Dies erfolgt mit `set.seed`. Dies wird für alle Abgaben mit R erwartet, bei denen die Ergebnisse stochastisch sind. Normalverteilte Zufallszahlen lassen sich beispielsweise mit der Funktion `rnorm` erzeugen.

Sollen Werte mithilfe einer for-Schleife berechnet werden, so ist es häufig sinnvoll zuerst einen Vektor der passenden Länge zu initialisieren, da dies in der Regel schneller sein sollte als in jedem Schritt ein weiteres Element hinzuzufügen.

Die Funktion `summary` gibt eine Zusammenfassung der Ergebnisse an, wobei die Art der Darstellung davon abhängt, was für eine Art von R-Objekt zusammengefasst wird.

Beispiel:

```r
set.seed(1)
result = numeric(length = 100)
head(x = result)
```

```
## [1] 0 0 0 0 0 0
```

```r
for(i in 2:100) {
  result[i] = summe(x = result[i - 1], y = rnorm(n = 1, mean = 2, sd = 1))
}
head(x = result)
```

```
## [1]  0.000000  1.373546  3.557190  4.721561  8.316842 10.646349
```

```r
summary(object = result)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00   53.56  103.58  104.77  158.60  209.36
```

Working directory

Mit dem Befehl `getwd` kann der Pfad zum aktuellen working directory angezeigt werden und mit `setwd` kann der Pfad zum aktuellen working directory gesetzt werden. Diese Befehle können hilfreich sein, um absolute Pfade zu vermeiden. Zudem kann mit `choose.dir` ein directory ausgewählt werden.

Einlesen von Daten und ändern der Spaltennamen

Datensätze können unter anderem mit der Funktion `read.table` eingelesen werden. Nachfolgend ist ein Beispiel zum Einlesen der Datei `data.txt` und dem Ändern der Spaltennamen gegeben.

```r
data = read.table(file = "data1.txt", header = TRUE, sep = "\t")
colnames(data) = c("a", "b", "c")
head(data, n = 2)
```

```
##           a        b c
## 1 0.8518534 1.308154 1
## 2 0.0201653 2.843631 1
```