

Applied Statistics for Data Science

Serie 1

Bemerkungen zu dieser Serie:

- Es geht in dieser Serie hauptsächlich darum, dass Sie die Funktionsweise von **R** kennenlernen. Es geht *nicht* darum, dass Sie jeden Befehl verstanden haben oder gar auswendig können müssen.
- Auch wird *nicht* verlangt, dass Sie jeden Schritt in der Musterlösung verstehen.
- Es geht auch nicht darum, dass Sie alle Aufgaben lösen.

Aufgabe 1.1

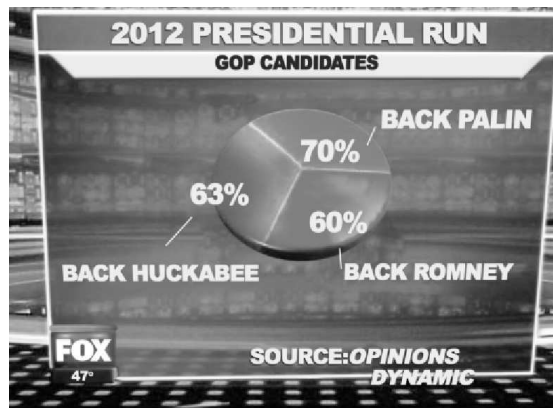
- a) In der folgenden Tabelle sind die zivilen Toten im 1. und 2. Weltkrieg aufgeführt. Wir betrachten die Toten der Alliierten im 2. Weltkrieg. Was ist *sehr* problematisch an diesem Teil der Tabelle?

Hinweis: Betrachten Sie die Gesamtzahl der Toten.

<u>CIVILIANS</u>					
(a)	<u>World War I</u>	-	Not known		
(b)	<u>World War II</u>				
	<u>Allied</u>				
	United Kingdom	60,595	
	Belgium	90,000	
	China	An enormous number	
	Denmark	Unknown	
	France	152,000	
	Netherlands	242,000	
	Norway	5,638	
	U.S.S.R.	6,000,000	
				<u>6,548,233</u>	
	<u>Enemy</u>				
	Germany	800,000	
	Austria	125,000	
	Italy	180,000	
	Japan	600,000	
	Poland	5,000,000	
	Yugoslavia	Large number	
				<u>6,705,000</u>	

- b) In der folgenden Abbildung ist in einem sogenannten *Kuchendiagramm* oder *pie chart* eine Präsidentenwahlprognose in den USA dargestellt.

Was ist problematisch an dieser Darstellung?



Aufgabe 1.2

In dieser Aufgabe werden einige einfache **R**-Befehle besprochen. Schauen Sie im Zweifelsfall im Dokument **R_Intro_en.pdf** auf ILIAS nach.

- Bilden Sie einen Vektor **x** mit den Zahlen 4, 2, 1, 3, 3, 5, 7.
- Wählen Sie mit **R** den dritten Wert aus.
- Wählen Sie mit **R** den ersten und vierten Wert aus.
- Bestimmen Sie die Länge des Vektors **x**.
- Was macht der Befehl **x+2**? Stellen Sie zuerst eine Vermutung auf und führen dann den Befehl aus.
- Was macht der Befehl **sum(x+2)**? Stellen Sie zuerst eine Vermutung auf und führen dann den Befehl aus.
- Was macht der Befehl **x <= 3**? Stellen Sie zuerst eine Vermutung auf und führen dann den Befehl aus.
- Was macht der Befehl **x[x <= 3]**? Stellen Sie zuerst eine Vermutung auf und führen dann den Befehl aus.
- Was macht der Befehl **sort(x)**? Stellen Sie zuerst eine Vermutung auf und führen dann den Befehl aus.

- j) Was macht der Befehl `order(x)`? Stellen Sie zuerst eine Vermutung auf und führen dann den Befehl aus. Vergleichen Sie dabei die Werte von `order(x)` mit den Werten von `x`.
- k) Sie möchten den Wert des 4. Eintrages durch die Zahl 8 ersetzen. Wie machen Sie das?

Aufgabe 1.3

Gegeben sind folgende Temperaturen in Grad Fahrenheit (°F)

51.9, 51.8, 51.9, 53

- a) Bilden Sie einen Vektor `fahrenheit` mit diesen Werten.
- b) Berechnen Sie diese Temperaturen in Grad Celsius (°C) um. Die Umrechnungsformel lautet

$$C = \frac{5}{9}(F - 32)$$

Bilden Sie dazu einen Vektor `celsius`.

- c) Gegeben sind weitere Temperaturen

48, 48.2, 48, 48.7

Bestimmen Sie die Differenz zu den ursprünglichen Temperaturen. Benützen Sie wieder Vektoren.

Aufgabe 1.4

Wir haben von 6 Personen die Körpergewicht (kg)

60, 72, 57, 90, 95, 72

und das Körpergrösse (in m)

1.75, 1.80, 1.65, 1.90, 1.74, 1.91

gegeben.

Nun wollen wir den Body Mass Index (BMI) berechnen. Dieser berechnet sich wie folgt

$$\text{BMI} = \frac{\text{Gewicht}}{\text{Grösse}^2}$$

- Erzeugen Sie zwei Vektoren `weight` und `height`.
- Berechnen Sie den BMI dieser 6 Personen gleichzeitig. Erzeugen Sie dazu einen Vektor `bmi`.

Aufgabe 1.5

In Aufgabe 1.2 sind wir dem Befehl `order(...)` begegnet, dessen Output auf den ersten Blick nicht offensichtlich ist. Wollen wir wissen, was dieser Befehl macht, so können wir die R-Hilfefunktion konsultieren. Diese Hilfefunktion in R ist leider *nicht* sehr hilfreich. Man *kann* mit beispielweise `?...` einen Befehl abfragen, zum Beispiel `?order` oder `help(order)`.

```
help(order)
```

```
order {base}
```

R Documentation

Ordering Permutation

Description

`order` returns a permutation which rearranges its first argument into ascending or descending order, breaking ties by further arguments. `sort.list` is the same, using only one argument. See the examples for how to use these functions to sort data frames, etc.

Usage

```
order(..., na.last = TRUE, decreasing = FALSE,
       method = c("auto", "shell", "radix"))

sort.list(x, partial = NULL, na.last = TRUE, decreasing = FALSE,
          method = c("auto", "shell", "quick", "radix"))
```

Arguments

<code>...</code>	a sequence of numeric, complex, character or logical vectors, all of the same length, or a classed R object.
<code>x</code>	an atomic vector.
<code>partial</code>	vector of indices for partial sorting. (Non-NULL values are not implemented.)
<code>decreasing</code>	logical. Should the sort order be increasing or decreasing? For the "radix" method, this can be a vector of length equal to the number of arguments in <code>...</code> . For the other methods, it must be length one.
<code>na.last</code>	for controlling the treatment of NAs. If TRUE, missing values in the data are put last; if FALSE, they are put first; if NA, they are removed (see 'Note').
<code>method</code>	the method to be used: partial matches are allowed. The default ("auto") implies "radix" for short numeric vectors, integer vectors, logical vectors and factors. Otherwise, it implies "shell". For details of methods "shell", "quick", and "radix", see the help for sort .

Dies ist ein Teil des Outputs (weiter unten wird es nicht verständlicher, versuchen Sie es). Das Resultat ist für Anfänger aber meist mehr verwirrend als helfend.

Weitaus hilfreicher ist die Verwendung von **google**. Geben Sie den Suchbegriff **r order** ein, so finden Sie schnell eine Erklärung die auch brauchbar ist, zum Beispiel

https://www.datacamp.com/community/tutorials/sorting-in-r?utm_source=adwords_ppc&utm_campaignid=898687156&utm_medium=referral

Allgemein ist www.datacamp.com eine Fundgrube für R und Python.

- a) Ein häufig vorkommender Befehl, ist der `seq(...)`-Befehl.

Googeln Sie diesen Befehl mit den Suchworten wie `r seq examples`. Erklären Sie die Funktionsweise dieses Befehles mit den Optionen `by` und `length.out`

- b) Wir haben die folgende Liste gegeben

```
x <- c(4, 10, 3, NA, NA, 1, 8)
```

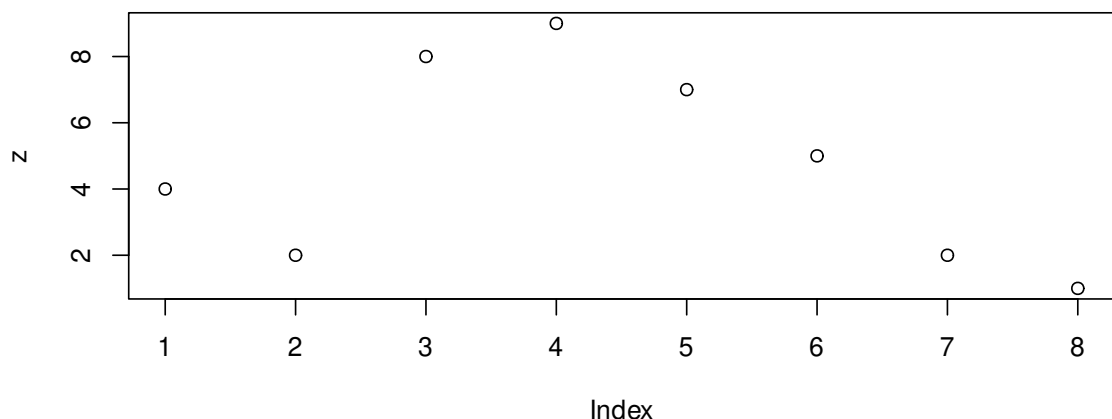
Zuerst eine Bemerkung zu den Wert `NA` (not available). Diese stehen für fehlende Daten, die aus irgendeinem Grund nicht vorhanden sind. Dies kommt in Statistiken recht häufig vor.

- Wenn wir den Mittelwert von `x` bilden (`mean(x)`), so ist das Resultat `NA`. Können Sie erklären, warum dies so ist?
- Wie können Sie den Mittelwert aller vorkommenden Werte bilden? Googeln Sie wieder.
- Wenden Sie die Befehle `sort(...)` und `order(...)` auf die Liste `x` an. Was machen diese Befehle?

In beiden kommen die Optionen `na.last = ...` und `decreasing = ...`, die man `TRUE` (oder `T`) oder `FALSE` setzen kann. Was bewirken diese Optionen.

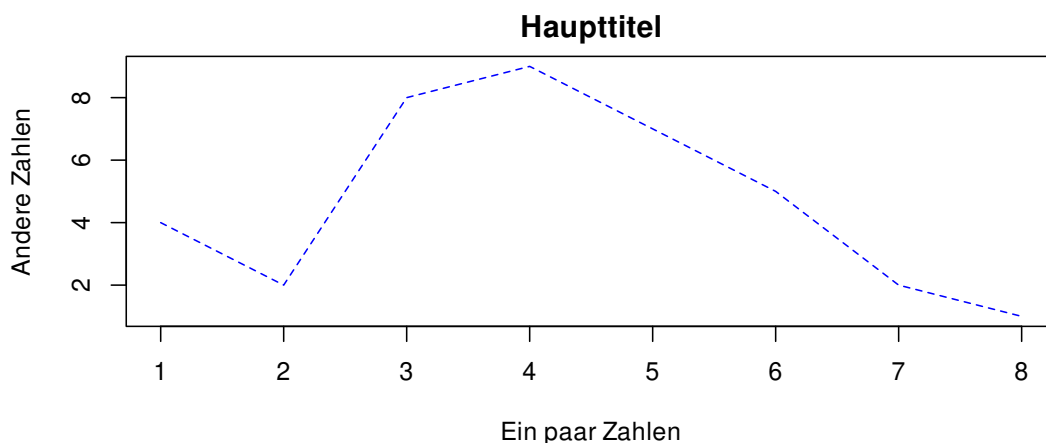
- c) Plots spielen in der Statistik eine wichtige Rolle. Der folgende Plot ist zwar sehr einfach zu erstellen, sieht aber auch etwas gar schmucklos aus.

```
z <- c(4, 2, 8, 9, 7, 5, 2, 1)
plot(z)
```



- i) Ändern Sie im folgenden Befehl die Parameter der Optionen ab und beschreiben Sie, was diese Optionen bewirken (vor allem `type` und `lty`, die anderen sollten klar sein). Googeln Sie.

```
plot(z,
     type = "l",
     col = "blue",
     lty = 2,
     main = "Hauptttitel",
     xlab = "Ein paar Zahlen",
     ylab = "Andere Zahlen"
)
```



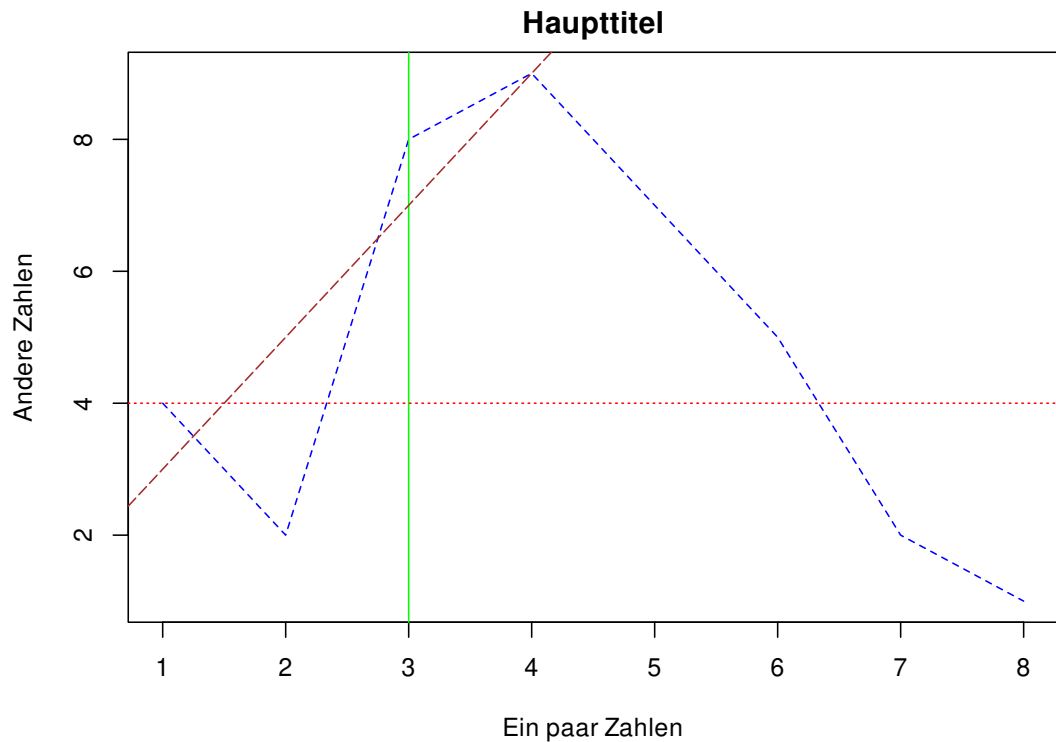
- ii) Fügen Sie mit dem Befehl `abline(...)` drei Linien zur Graphik oben hinzu (siehe Graphik unten).

- Eine senkrechte Gerade $x = 3$, durchgezogen, grün.
- Eine waagrechte Gerade $y = 4$, gepunktet, rot.
- Eine Gerade $y = 2x + 1$, gestrichelt mit langen Strichen, braun.

```
plot(z,
     type = "l",
     col = "blue",
     lty = 2,
     main = "Hauptttitel",
     xlab = "Ein paar Zahlen",
     ylab = "Andere Zahlen"
)

abline(...)
abline(...)
abline(...)
```

Wichtig ist hier, dass der `plot()` und `abline()` zusammen ausgeführt werden.



Aufgabe 1.6

Diese Aufgabe befasst sich mit dem Datensatz `weather.csv`, den wir in der Einführung kennengelernt haben.

Schauen Sie im Zweifelsfall im Dokument `R_Intro_en.pdf` auf ILIAS nach.

- Laden Sie den Datensatz und speichern Sie diesen unter der Variable `data` ab.
- Wählen Sie den Wert der zweiten Zeile und dritten Spalte aus.
- Wählen Sie die 4. Zeile aus?
- Wählen Sie die 1. und die 4. Spalte aus. Verwenden Sie dazu die Spaltennamen.
- Speichern Sie obige Data unter dem Namen `data1` ab und speichern Sie dies unter dem Namen `weather2.csv`.
- Wie können Sie herausfinden (mit `R` natürlich), welches der Name der 3. Spalte ist?

- g) Wir möchten den Spalten **Basel** durch **Genf** ersetzen. Wie würden Sie vorgehen?
- h) Wir betrachten den Befehl

```
data3 <- data[order(data[, "Zurich"]), ]
```

Dieser erzeugt

```
data3 <- data[order(data[, 'Zurich']), ]
data3

##      Luzern Basel Chur Zurich
## Feb         5     6    1      0
## Jan         2     5   -3      4
## Mar        10    11   13      8
## Apr        16    12   14     17
## May        21    23   21     20
## Jun        25    21   23     27
```

- i) Wenn Sie die Tabelle anschauen, was macht dieser Befehl?
- ii) Erklären Sie, warum dieser Befehl diese Wirkung hat.

Aufgabe 1.7

Das Dataframe **d.fuel** enthält die Daten verschiedener Fahrzeuge aus einer amerikanischen Untersuchung der 80er-Jahre. Jede Zeile (row) enthält die Daten eines Fahrzeuges (ein Fahrzeug entspricht einer Beobachtung).

- a) Lesen Sie die auf Ilias abgelegte Datei **d.fuel.dat** ein mit dem folgenden R-Befehl:

```
d.fuel <- read.table(file = "./d.fuel.dat", header = T, sep = ",")
```

Das Argument **sep = ","** braucht es, weil die Kolonnen im File **d.fuel.dat** durch Kommata getrennt sind. Im File **d.fuel.dat** wurden die Zeilen durchnummeriert und daher steht in der ersten Spalte die Nummer der Zeile. Die Spalten (columns) enthalten die folgenden Variablen:

weight: Gewicht in Pounds (1 Pound = 0.453 59 kg)
 mpg: Reichweite in Miles Per Gallon (1 gallon = 3.789 l; 1 mile = 1.6093 km)
 type: Autotyp

- b) Wählen Sie nur die fünfte Zeile des Dataframe **d.fuel** aus. Welche Werte stehen in der fünften Zeile?

- c) Wählen Sie nun die erste bis fünfte Beobachtung des Datensatzes aus. So lässt sich übrigens bei einem unbekannten Datensatz ein schneller Überblick über die Art des Dataframe gewinnen.
- d) Zeigen Sie gleichzeitig die 1. bis 3. und die 57. bis 60. Beobachtung des Datensatzes an.
- e) Berechnen Sie den Mittelwert der Reichweiten aller Autos in Miles/Gallon.

R-Hinweis:

```
mean(...)
```

- f) Berechnen Sie den Mittelwert der Reichweite der Autos 7 bis 22.
- g) Erzeugen Sie einen neuen Vektor `t.km1`, der alle Reichweiten in km/l, und einen Vektor `t.kg`, der alle Gewichte in kg enthält.
- h) Berechnen Sie den Mittelwert der Reichweiten in km/l und denjenigen der Fahrzeuggewichte in kg.

Applied Statistics for Data Science

Musterlösungen zu Serie 1

Lösung 1.1

- a) Das Hauptproblem ist, dass die Gesamtzahl der Toten *exakt* angegeben wurde, aber nur der *bekannten* Zahlen. China mit „An enormous number“ und Dänemark mit „Unknown“ wurden nicht berücksichtigt. Dementsprechend kann die Gesamtzahl auch nicht stimmen, vor allem nicht auf einzelne Tote genau.

Ein weiteres Problem ist die Angabe der Zahlen. Norwegen hatte 3638 Tote, also eine exakte Zahl, während die Sowietunion 6 000 000 Tote hatte, was wohl eine Schätzung ist.

Die Angabe der Gesamtzahl auf einzelne Tote genau macht keinen Sinn. Hier wird eine Genauigkeit vorgespielt, die nicht vorhanden ist.

- b) Die Prozentzahlen addieren sich nicht auf 100 % auf, sondern fast 200 %. Der Grund dafür ist wohl, dass die Befragten mehrere Kandidaten oder Kandidatin (Sarah Palin) angeben konnten. Dadurch entsteht dann aber das Problem, dass die Kandidaten und die Kandidatin sehr schlecht miteinander vergleichbar sind.

Lösung 1.2

- a) Ein Vektor wird mit dem Befehl `c(...)` gebildet.

```
x <- c(4, 2, 1, 3, 3, 5, 7)
```

- b) `x[3]`

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

- c) `x[c(1, 4)]`

```
## [1] 4 3
```

- d) `length(x)`

```
## [1] 7
```

- e) `x+2`

```
## [1] 6 4 3 5 5 7 9
```

f) `sum(x+2)`

```
## [1] 39
```

Hier werden alle Werte in `x+2` aufaddiert.

g) `x <= 3`

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

Der Befehl erzeugt einen Vektor der Länge von `x`. Für alle Werte die kleiner als 3 sind, wird **TRUE**, für die anderen **FALSE**.

h) `x[x <= 3]`

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

Die Konstruktion `x[...]` wählt Elemente aus dem Vektor `x` aus. Die Auswahl geschieht nun mit `x <= 3` aus g). Es werden alle Werte ausgewählt, die den Wert **TRUE** haben.

i) `sort(x)`

```
## [1] 1 2 3 3 4 5 7
```

Die Werte von `x` werden der Grösse nach aufsteigend geordnet.

j) `order(x)`

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

```
x
```

```
## [1] 4 2 1 3 3 5 7
```

Es ist auf den ersten Blick nicht ganz offensichtlich, was dieser Befehl macht. So kommt in `order(x)` der Wert 6 vor, aber nicht in `x`. Wenn wir die Vektoren lange genug anschauen, so erkennen wir (vielleicht) folgendes Muster:

- Der erste Wert von `order(x)` ist 3. Betrachten wir nun der 3. Eintrag in `x`, so ist dieser 1.
- Der zweite Wert von `order(x)` ist 2. Der 2. Eintrag von `x` ist 2.
- Der dritte Wert von `order(x)` ist 4. Der 3. Eintrag von `x` ist 3.
- Der vierte Wert von `order(x)` ist 5. Der 4. Eintrag von `x` ist 3.
- usw. ...

Der Befehl `order(x)` gibt also die Stellen an, *wo* sich die Werte von `x` der Grösse nach befinden. Oder anders gesagt: Er erzeugt die *Rangliste* der Werte von `x`: Der 3. Wert ist der kleinste von `x`, der 2. der zweitkleinste usw.

Was machen wir, wenn wir dieses Muster nicht erkennen? Siehe Aufgabe 1.5.

```
k) x[4] <- 8
x
## [1] 4 2 1 8 3 5 7
```

Lösung 1.3

a) Vektor `fahrenheit`

```
fahrenheit <- c(51.9, 51.8, 51.9, 53)
fahrenheit
## [1] 51.9 51.8 51.9 53.0
```

b) Temperaturen in Grad Celsius (°C)

```
celsius <- 5/9 * (fahrenheit - 32)
celsius
## [1] 11.05556 11.00000 11.05556 11.66667
```

c) Weitere Temperaturen

```
fahrenheit_2 <- c(48, 48.2, 48, 48.7)
fahrenheit_3 <- fahrenheit - fahrenheit_2
fahrenheit_3
## [1] 3.9 3.6 3.9 4.3
```

Lösung 1.4

```
a) weight <- c(60, 72, 57, 90, 95, 72)
height <- c(1.75, 1.80, 1.65, 1.90, 1.74, 1.91)
```

b) In `R` erreichen wir dies durch

```
bmi <- weight / height^2
bmi
## [1] 19.59184 22.22222 20.93664 24.93075 31.37799 19.73630
```

Somit haben wir den BMI für alle 6 Personen gleichzeitig berechnet!

Lösung 1.5

- a) Der Befehl `seq(...)` bildet eine Folge von Zahlen, die mit der ersten Zahl (`from=...`) beginnt und der 2. Zahl (`to = ...`) aufhört (sofern das geht). Die Schrittlänge wird durch `by = ...` angegeben.

```
seq(from = 3, to = 10, by = 2)

## [1] 3 5 7 9
```

oder in diesem Fall einfacher:

```
seq(3, 10, 2)

## [1] 3 5 7 9
```

Die Zahl 10 ist hier nicht dabei, da sie in der Aufzählung gar nicht vorkommt.

Mit der Option `length.out` wird angegeben, wieviele Zahlen zwischen der Anfangs- und der Endzahl gebildet werden, alle im gleichen Abstand.

```
seq(from = 3, to = 10, length.out = 10)

## [1] 3.000000 3.777778 4.555556 5.333333 6.111111 6.888889
## [7] 7.666667 8.444444 9.222222 10.000000
```

Oder

```
seq(3, 10, length.out = 10)

## [1] 3.000000 3.777778 4.555556 5.333333 6.111111 6.888889
## [7] 7.666667 8.444444 9.222222 10.000000
```

Die Optionen `by` und `length.out` *nicht* miteinander verwendet werden, da die Angaben meist widersprüchlich sind.

```
seq(from = 3, to = 10, length.out = 10, by = 2)

## Error in seq.default(from = 3, to = 10, length.out = 10, by = 2): too
many arguments
```

- b) i) Der `mean`-Befehl macht hier keinen Sinn, da `R` versucht aus *allen* Werten den Mittelwert zu ziehen und das geht mit `NA`'s natürlich nicht.
- ii) Wir können allerdings mit der Option `na.rm=TRUE` (default ist `FALSE`) erreichen, dass die `NA`'s ignoriert werden (`.rm` steht für remove).

```
mean(x, na.rm = TRUE)

## [1] 5.2
```

- iii) Der `sort`-Befehl ist der einfachere der beiden.

```
sort(x)

## [1] 1 3 4 8 10
```

Der sortiert die vorhandenen Zahlen aufsteigend.

Wollen wir die Sortierung aber absteigend, so verwenden wir die Option `decreasing = TRUE` (default ist `FALSE`).

```
sort(x, decreasing = TRUE)

## [1] 10  8  4  3  1
```

Die `NA`-Werte wurden allerdings ignoriert. Wollen wir die auch noch dabei haben, so wählen wir die Option `na.last = TRUE`

```
sort(x, decreasing = TRUE, na.last = TRUE)

## [1] 10  8  4  3  1 NA NA
```

Wollen wir die `NA`'s am Anfang, so setzen wir diese Option `FALSE`

```
sort(x, decreasing = TRUE, na.last = FALSE)

## [1] NA NA 10  8  4  3  1
```

Wir betrachten nun noch den `order`-Befehl.

```
order(x)

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

Betrachten wir die ursprüngliche Liste

```
x

## [1]  4 10  3 NA NA  1  8
```

Hier sehen wir, dass die 6. Zahl die kleinste ist, die 3. Zahl die zweitkleinste usw. Die `NA`'s (4. und 5. Zahl) sind dabei am Schluss.

Die Optionen `decreasing = ...` und `na.last = ...` funktionieren hier gleich, wie beim `sort`-Befehl.

- c) i) Die Optionen `main = "...", col = "...", xlab = "..."` und `ylab = "..."` dürften klar sein.

Die Option `type = "..."` gibt den Linientyp an. Siehe auch

<https://www.dummies.com/programming/r/how-to-create-different-plot-types-in-r/>

Die Option `lty = "..."` gibt den Linientyp für „durchgezogene“ Linien vor. Siehe auch

<http://www.sthda.com/english/wiki/line-types-in-r-lty>

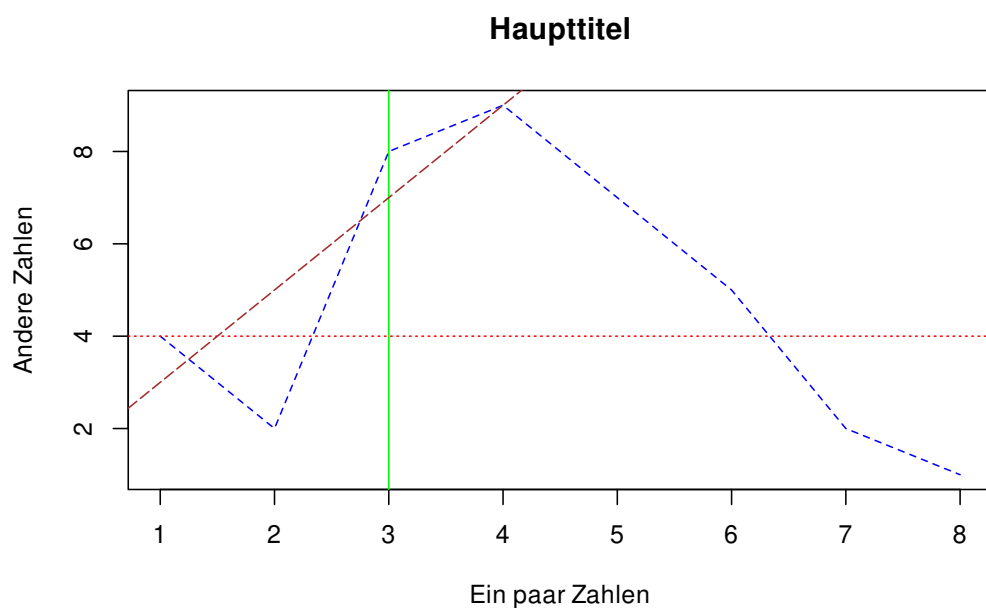
Für die Farbpalette in `R` siehe

<http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf>

ii) Code:

```
plot(z,
     type = "l",
     col = "blue",
     lty = 2,
     main = "Haupttitel",
     xlab = "Ein paar Zahlen",
     ylab = "Andere Zahlen"
)

abline(v = 3, col = "green", lty = 1)
abline(h = 4, col = "red", lty = 3)
abline(a = 1, b = 2, col = "brown", lty = 5)
```



Lösung 1.6

a) `data <- read.csv("../..../Software_R_Python/R/weather.csv")`
`data`

```
##      Luzern Basel Chur Zurich
## Jan      2      5    -3      4
## Feb      5      6      1      0
## Mar     10     11     13      8
## Apr     16     12     14     17
## May     21     23     21     20
## Jun     25     21     23     27
```

Ihr Pfad wird natürlich anders lauten. Für Windows-User: Sie müssen die \

durch / ersetzen.

b) `data[2, 3]`

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

Nochmals: Der erste Wert von `data[... , ...]` bezieht sich *immer* auf die Zeile und der zweite Wert auf die Spalte.

c) `data[4,]`

```
##      Luzern Basel Chur Zurich
## Apr      16     12    14     17
```

d) `data[, c("Luzern", "Zurich")]`

```
##      Luzern Zurich
## Jan         2      4
## Feb         5      0
## Mar        10      8
## Apr        16     17
## May        21     20
## Jun        25     27
```

e) `data1 <- data[, c("Luzern", "Zurich")]`
`write.csv(data1, "../../../Software_R_Python/R/weather2.csv", row.names=F)`

```
data2 <- read.csv("../..../Software_R_Python/R/weather2.csv")
data2
```

```
##      Luzern Zurich
## 1         2      4
## 2         5      0
## 3        10      8
## 4        16     17
## 5        21     20
## 6        25     27
```

f) `colnames(data)[3]`

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

Der Befehl `colnames(data)` erzeugt einen Vektor mit den Spaltennamen der Datei `data`. Mit `...[3]` wird der dritte Wert ausgewählt.

g) `colnames(data)[2] <- "Genf"`

```
data
##      Luzern Genf Chur Zurich
## Jan         2     5   -3      4
## Feb         5     6    1      0
## Mar        10    11   13      8
```



```
## Apr      16    12    14     17
## May      21    23    21     20
## Jun      25    21    23     27
```

Wenn wir nicht Spalten nachschauen wollen, bietet sich die folgende Variante an.

```
data <- read.csv("../..../Software_R_Python/R/weather.csv")
data

##      Luzern Basel Chur Zurich
## Jan       2     5   -3      4
## Feb       5     6    1      0
## Mar      10    11   13      8
## Apr      16    12   14     17
## May      21    23   21     20
## Jun      25    21   23     27

colnames(data)[which("Basel" == colnames(data))] <- "Genf"
data

##      Luzern Genf Chur Zurich
## Jan       2     5   -3      4
## Feb       5     6    1      0
## Mar      10    11   13      8
## Apr      16    12   14     17
## May      21    23   21     20
## Jun      25    21   23     27
```

Hier würden *alle* Einträge **Basel** durch **Genf** ersetzt.

"Basel == colnames(data) gibt eine Vektor mit TRUE, FALSE Werten aus. An jeder Stelle, wo "Basel" vorkommt, wird **TRUE** gesetzt, ansonsten **FALSE**.

Der Befehl **which("Basel" == colnames(data))** bestimmt alle Stellen, wo **Basel** in **colnames(data)** vorkommt. Dies sind die Stellen, wo **Basel** durch **Genf** ersetzt wird

== ist das logische gleich und keine Zuweisung. Hier wird nur geprüft, ob die Werte gleich sind.

h)

```
data3 <- data[order(data[, 'Zurich']), ]
data3
```

```
##      Luzern Genf Chur Zurich
## Feb       5     6    1      0
## Jan       2     5   -3      4
## Mar      10    11   13      8
## Apr      16    12   14     17
## May      21    23   21     20
## Jun      25    21   23     27
```

- i) i) Die Tabelle wird nach den Werten von Zürich ansteigend geordnet.
- ii) Im ersten Eintrag für `data` steht `order(data[, "Zurich"])`. Dieser gibt die Ordnung der Spalte `Zurich` an.

```
order(data[, 'Zurich'])
```

```
## [1] 2 1 3 4 5 6
```

Damit werden die Zeilen von `data` werden nach dieser Ordnung geordnet.

Lösung 1.7

- a) Siehe Aufgabenstellung.

Um die Daten in Tabellenform zu sehen, tippt man den Namen des Objektes ein

```
d.fuel
```

```
##      X weight mpg    type
## 1    1  2560  33   Small
## 2    2  2345  33   Small
## 3    3  1845  37   Small
## 4    4  2260  32   Small
## 5    5  2440  32   Small
## 6    6  2285  26   Small
## 7    7  2275  33   Small
## 8    8  2350  28   Small
## 9    9  2295  25   Small
## 10  10  1900  34   Small
## 11  11  2390  29   Small
## 12  12  2075  35   Small
## 13  13  2330  26   Small
## 14  14  3320  20  Sporty
## 15  15  2885  27  Sporty
## 16  16  3310  19  Sporty
## 17  17  2695  30  Sporty
## 18  18  2170  33  Sporty
## 19  19  2710  27  Sporty
## 20  20  2775  24  Sporty
## 21  21  2840  26  Sporty
## 22  22  2485  28  Sporty
## 23  23  2670  27 Compact
## 24  24  2640  23 Compact
## 25  25  2655  26 Compact
## 26  26  3065  25 Compact
## 27  27  2750  24 Compact
## 28  28  2920  26 Compact
## 29  29  2780  24 Compact
## 30  30  2745  25 Compact
## 31  31  3110  21 Compact
```

```
## 32 32 2920 21 Compact
## 33 33 2645 23 Compact
## 34 34 2575 24 Compact
## 35 35 2935 23 Compact
## 36 36 2920 27 Compact
## 37 37 2985 23 Compact
## 38 38 3265 20 Medium
## 39 39 2880 21 Medium
## 40 40 2975 22 Medium
## 41 41 3450 22 Medium
## 42 42 3145 22 Medium
## 43 43 3190 22 Medium
## 44 44 3610 23 Medium
## 45 45 2885 23 Medium
## 46 46 3480 21 Medium
## 47 47 3200 22 Medium
## 48 48 2765 21 Medium
## 49 49 3220 21 Medium
## 50 50 3480 23 Medium
## 51 51 3325 23 Large
## 52 52 3855 18 Large
## 53 53 3850 20 Large
## 54 54 3195 18 Van
## 55 55 3735 18 Van
## 56 56 3665 18 Van
## 57 57 3735 19 Van
## 58 58 3415 20 Van
## 59 59 3185 20 Van
## 60 60 3690 19 Van
```

b) Auswählen der fünften Beobachtung:

```
d.fuel[5, ]

##   X weight mpg  type
## 5 5   2440  32 Small
```

c) Auswählen der 1. bis 5. Beobachtung:

```
d.fuel[1:5, ]

##   X weight mpg  type
## 1 1   2560  33 Small
## 2 2   2345  33 Small
## 3 3   1845  37 Small
## 4 4   2260  32 Small
## 5 5   2440  32 Small
```

Alternativ kann man sich eine Übersicht verschaffen mit Hilfe der R-Funktion `head(...)`

```
head(d.fuel)

##      X weight mpg  type
## 1 1    2560  33 Small
## 2 2    2345  33 Small
## 3 3    1845  37 Small
## 4 4    2260  32 Small
## 5 5    2440  32 Small
## 6 6    2285  26 Small
```

d) Auswählen der 1. bis 3. und 57. bis 60. Beobachtung:

```
d.fuel[c(1:3, 57:60), ]

##      X weight mpg  type
## 1    1    2560  33 Small
## 2    2    2345  33 Small
## 3    3    1845  37 Small
## 57 57    3735  19  Van
## 58 58    3415  20  Van
## 59 59    3185  20  Van
## 60 60    3690  19  Van
```

e) Die Werte der Reichweiten stehen in der dritten Spalte, die **mpg** heisst. Zur Berechnung des Mittelwertes gibt es verschiedene Möglichkeiten, welche sich in der Art der Datenselektion unterscheiden:

```
mean(d.fuel[, 3])

## [1] 24.58333

mean(d.fuel[, "mpg"])

## [1] 24.58333

mean(d.fuel$mpg)

## [1] 24.58333
```

f) Auch hier gibt es wieder verschiedene Möglichkeiten. Eine davon ist:

```
mean(d.fuel[7:22, "mpg"])

## [1] 27.75
```

g) Umrechnung der Miles Per Gallon in Kilometer pro Liter und der Pounds in Kilogramm:

```
t.kml <- d.fuel[, "mpg"] * 1.6093 / 3.789
t.kg <- d.fuel[, "weight"] * 0.45359
```

h) Mittelwert der Reichweite und des Gewichtes:

```
mean(t.kml)
```

```
## [1] 10.44127
```

```
mean(t.kg)
```

```
## [1] 1315.789
```