Einführung in R für SBWL Tourismusanalyse und Freizeitmarketing VK 2

Regina Tüchler & Thomas Rusch

October 9, 2008

1 Dokumentation

Diese Einführung in R ist nicht als eine generelle Einführung gedacht. Sie enthält gemeinsam mit den anderen R-Beschreibungen von der LV-Seite nur jene Dinge, die für diese LV relevant sind.

Um sich darüber hinausgehend über R zu informieren stehen Manuals auf

`http:/www.R-project.org`

zur Verfügung.

2 Installation und Start

Fertige R Distributionen sind auf CRAN (Comprehensive R Achive Network) erhältlich:

`http://CRAN.R-project.org.`

Unter Windows startet man durch Anklicken des R-Symbols. Unter Unix wird `$ R` aufgerufen.

Das Arbeitsverzeichnis: Beim Aufruf von R befindet man sich immer in einem bestimmten Arbeitsverzeichnis. Man kann abfragen, in welchem Verzeichnis gerade gearbeitet wird,

```
> getwd()
```

Dieses Verzeichnis kann geändert werden, z.B.

```
> setwd("C:/R/tourism_vk2")
```

Es empfiehlt sich, ein eigenes Verzeichnis für die Berechnungen mit R in dieser LV anzulegen, in das dann alle Daten gespeichert werden können. Der Inhalt des Arbeitsverzeichnisses kann abgefragt werden:

```
> dir()
```

3 Hilfesystem

In R steht ein ausführliches Hilfesystem zur Verfügung. Um z.B. Hilfe über die Funktion `dir()` zu bekommen, schreibt man `help("dir")` bzw. `?dir`.

4 Rechnen mit R

R kann einfach als Rechenmaschine verwendet werden.

```
> 2
[1] 2
> 2 + 4.5
[1] 6.5
> 4.5 - 2
[1] 2.5
> 2 * 4.5
[1] 9
> 4/2
[1] 2
> 4^2
[1] 16
```

Es stehen auch besondere Funktionen zur Verfügung:

```
> sin(0)
[1] 0
> cos(pi)
[1] -1
> exp(1)
[1] 2.718282
> log(1)
[1] 0
```

Wenn das Ergebnis einer Rechnung für den Computer zu klein/groß wird, wird `-Inf` bzw.`Inf` ausgegeben. Wenn das Ergebnis nicht wohldefiniert ist, erhält man `NaN` (not a number).

Man kann Ergebnisse auch Variablen zuweisen und damit rechnen. Dazu wird in R das Zeichen `'<-'` verwendet.

```
> x <- 2 + 4
[1] 6
> y <- 4^2
[1] 16
> z <- y - x
[1] 10
```

5 Der Suchpfad

Alle Elemente, die gerade zur Verfügung stehen, findet man im sogenannten Suchpfad, der abgefragt werden kann:

```
> search()
```

In diesen Elementen sind jene Objekte enthalten, die jetzt gerade zum Arbeiten zur Verfügung stehen. Dabei ist das erste Element des Suchpfades immer ".GlobalEnv". Das ist der Workspace, in dem standardmäßig Objekte erzeugt werden. Wenn man also die obigen Berechnungen von `x, y, z` durchführt, werden diese drei Objekte erzeugt und sind dann unter ihrem Namen im Workspace vorhanden.

Zusätzlich werden einige Packages (Objektbibliotheken) beim Starten von R automatisch in den Suchpfad geladen. Diese werden für diese LV ausreichend sein. Alle weiteren Packages müssen mit `library` extra geladen werden: z.B. `library("stats4")`.

Um zu sehen welche Objekte in einem Element des Suchpfads enthalten sind, verwendet man `objects`. In dieser LV werden wir zumeist Objekte aus der Programmbibliothek `stats` verwenden, die mit folgendem Befehl angezeigt werden:

```
> objects("package:stats")
```

Mit `objects()` oder `ls()` kann man sehen, welche Objekte im Workspace ".GlobalEnv" stehen. Objekte können mit `remove` bzw. `rm` aus dem Workspace gelöscht werden.

```
> ls()
> remove(x)
```

6 Laden und Speichern von Daten

In den einzelnen Packages stellt R schon eine Vielzahl von fertigen Datensätzen zur Verfügung. Diese können mit dem Befehl `data` geladen werden. Z.B. werden die im "package:stats" vorhandenen `UKgas`-Daten wie folgt geladen:

```
> data("UKgas")
```

Mit

```
> help("UKgas")
```

erhält man eine Beschreibung des Datensatzes.

Um einen nicht in R vorgegebenen Datensatz in einer R-Datei abzuspeichern, verwendet man den Befehl `save`. Z.B. würde mit folgendem Befehl ein Datensatz der im Workspace unter dem Namen "zeitreihe" steht, als File "zr.rda" im Arbeitsverzeichnis gespeichert werden:

```
> save(zeitreihe, file = "zr.rda")
```

Dieses File kann dann mit

```
> load("zr.rda")
```

wieder geladen werden, sodass der Datensatz "zeitreihe" wieder zur Verfügung steht.

Die in dieser LV benötigten Daten werden auf der LV-Seite schon in diesem R-Format zur Verfügung gestellt. Für das Laden und Speichern von Daten in anderen Formaten (z.B. Text, SPSS, Excel,...) stehen geeignete Funktionen zur Verfügung (s. Manuals).

7 Datenstrukturen in R

In R gibt es verschiedene Datenstrukturen. Abhängig von der Datenstruktur werden Operationen verschieden interpretiert bzw. sind bestimmte Operationen überhaupt möglich. Die genaue Struktur eines Objekts kann mit mit `str` abgefragt werden.

Im folgenden geben wir eine Zusammenstellung jener Datenstrukturen, die wir im ersten Teil dieser LV verwenden werden. Die genaue Anwendung wird dann im Laufe der LV erklärt bzw. kann über das Help-Menü abgefragt werden.

7.1 Vektoren

R-Vektoren bestehen aus Elementen *gleichen* Typs (auch nicht-numerisch möglich, wir benötigen im Moment aber nur numerische Vektoren). Die einfachste Art solche Vektoren zu definieren, ist über die Funktion `c`. Sie enthält hier als Argumente numerische Werte und setzt diese zu einem Vektor zusammen.

```
> x <- c(2, 3, 4, 5, 6, 7)

[1] 2 3 4 5 6 7

> x2 <- c(x, c(8, 9, 10))

[1]  2  3  4  5  6  7  8  9 10

> str(x)

 num [1:6] 2 3 4 5 6 7
NULL
```

Die Funktion `seq` erzeugt eine Folge:

```
> x <- seq(2, 7)

[1] 2 3 4 5 6 7

> x <- 2:7

[1] 2 3 4 5 6 7

> seq(2, 10, by = 2)

[1]  2  4  6  8 10

> seq(2, 10, length = 5)

[1]  2  4  6  8 10
```

Um die Schreibarbeit zu reduzieren, kann `rep` zur Wiederholung eingesetzt werden:

```
> rep(3, 4)

[1] 3 3 3 3

> rep(1:2, 3)

[1] 1 2 1 2 1 2

> rep((1:2), c(2, 4))
```

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

Mit `is.vector(x)` kann man prüfen, ob `x` ein R-Vektor ist.

```
> is.vector(x)

[1] TRUE
```

Der Zugriff auf einzelne Elemente eines Vektors erfolgt durch die Angabe des jeweiligen Index:

```
> x[2]

[1] 3

> x[c(1, 3, 6)]

[1] 2 4 7

> x[-3]

[1] 2 3 5 6 7
```

Das Minus bedeutet hier, alle Elemente bis auf die mit '-' angegebenen.
Den kleinsten/größten Wert eines Vektors kann man ausgeben lassen:

```
> min(x)

[1] 2

> max(x)

[1] 7
```

Die aus Statistik 1 bekannten Maßzahlen Mittelwert, Varianz und Standardabweichung (Wurzel aus der Varianz) erhält man mit Befehlen `mean()`, `var()` und `sd()`. Die Fünf-Punkte-Zusammenfassung, d.h. Minimum, unteres Quartil, Mittelwert, oberes Quartil und Maximum, wird auf den Befehl `summary` geliefert.

7.2 Matrizen

Eine Matrix ist eine Liste von Elementen *gleichen* Typs, die in einem Rechteck angeordnet sind. Die Funktion `matrix` erzeugt eine Matrix. Dabei müssen als Argumente die Daten (`data`), die Anzahl der Zeilen (`nrow`) und die Anzahl der Spalten (`ncol`) angegeben werden:

```
> m <- matrix(data = seq(1:6), nrow = 2, ncol = 3)

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

> str(m)

 int [1:2, 1:3] 1 2 3 4 5 6
NULL
```

bzw. kürzer

```
> m <- matrix(seq(1, 6), 2, 3)
```

In R sind auch die aus der Mathematik bekannten Zeilen- und Spaltenvektoren Matrizen:

```
> ze <- matrix(seq(1, 6), 1, 6)

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

Die Operationen '+' und '-' werden elementweise ausgeführt (für Matrizen mit gleichen Dimensionen):

```
> m2 <- matrix(c(2, 4, 3, 5, 1, 2), 2, 3)

     [,1] [,2] [,3]
[1,]    2    3    1
[2,]    4    5    2

> dim(m2)

[1] 2 3

> dim(m)

[1] 2 3

> m + m2

     [,1] [,2] [,3]
[1,]    3    6    6
[2,]    6    9    8

> m - m2

     [,1] [,2] [,3]
[1,]   -1    0    4
[2,]   -2   -1    4
```

Die Zeilensumme bzw. Spaltensumme berechnet in einer Matrix die Summe aus allen Elementen eine Zeile bzw. Spalte. Die Befehle dazu sind `colSums` (column=Spalte), `rowSums` (row=Zeile) und ergeben einen R-Vektor:

```
> colSums(m)

[1]  3  7 11

> rowSums(m)

[1]  9 12
```

Der Zugriff auf einzelne Elemente einer Matrix erfolgt über die Angabe der Indices:

```
> m[2, 3]

[1] 6
```

Der Zugriff auf gesamte Zeilen bzw. Spalten ergibt einen R-Vektor und funktioniert so:

```
> m[, 3]
```

```
[1] 5 6

> m[2, ]

[1] 2 4 6

> m[, c(1, 3)]

     [,1] [,2]
[1,]    1    5
[2,]    2    6

> m[1, -2]

[1] 1 5
```

Mit `is.matrix(m)` kann man prüfen, ob `m` eine Matrix ist.

7.3 Zeitreihen: `ts` (time series)

Eine ZR enthält neben den Daten auch eine Information über den Start- und Endzeitpunkt. Weiters ist festgelegt, wieviele Zeitpunkte pro Zeiteinheit vorhanden sind. Z.B. enthält die ZR `UKgas` eine ZR mit Quartalsdaten (also 4 Datenpunkte pro Jahr) über den Erdgaskonsum in GB im Zeitraum 1959-1997:

```
> data("UKgas")

> UKgas
```

Mit `ts` kann man eine R-Zeitreihe erzeugen. Dabei muss der Start- und der Endzeitpunkt und die Anzahl der Zeitpunkte pro Zeiteinheit angegeben werden. Bei den obigen `UKgas`-Daten wären das die Optionen: `start = 1960`, `end = 1986`, `freq = 4`.

Mit `is.ts(x)` kann man prüfen, ob x eine ZR ist. Mit der Funktion `window` kann man einen Ausschnitt aus einer ZR herausnehmen. Wenn man z.B. die Zeitpunkte 2. Quartal 1975 bis 1. Quartal 1980 aus den `UKgas`-Daten auswählen will, schreibt man:

```
> window(UKgas, start = c(1975, 2), end = c(1980, 1))
```

Grafische Darstellung von Zeitreihen

Der Befehl `plot(x)` ist kontextabhängig und erzeugt je nach Struktur von `x` eine unterschiedliche zu `x` passende Grafik. Für Objekte, die Zeitreihen sind, wird daher ein Zeitreihenplot gemacht (Fig. 1).

Will man in den schon bestehenden Plot noch eine Linie hinzufügen, muss nach dem Plot-Befehl der Befehl `lines` verwendet werden. In Fig. 1 haben wir den Wert 50 von der ZR abgezogen und diese neue Reihe dazugezeichnet. Mit der Option `col="red"` zeichnen wir die Linie in roter Farbe ein. (Außer dieser Option stehen noch eine Vielzahl von Optionen bei der Erzeugung der Plots zur Verfügung. Diese könnnen unter `help(par)` nachgeschaut werden.)

D.h. zuerst erzeugt `plot` die Grafik, dann fügt `lines` Linien hinzu (Reihenfolge!)

8 Editoren für R

Um längere Befehlsfolgen zu erstellen und für den späteren Gebrauch abzuspeichern, sollte ein Editor verwendet werden. Längerer Code und kompliziertere Funktionen können mit dem Editor geschrieben werden und dann in eine Datei gespeichert werden. Mit `source("Dateiname")`

```
> plot(UKgas)
> lines(UKgas - 50, col = "red")
```

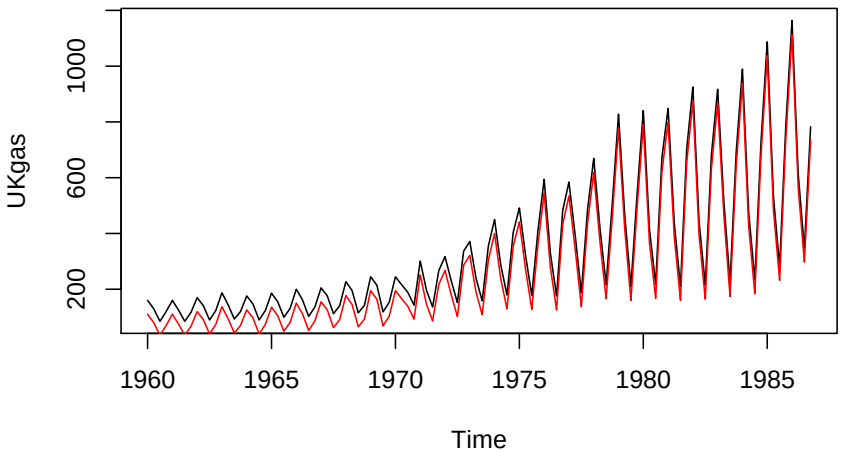


Figure 1: Zeitreihenplot der `UKgas` Daten.

kann man die Funktion dann wieder in den Workspace laden. Zudem bieten die meisten Syntax-Highlighting, Klammerhervorhebungen und andere nützliche Funktionen.

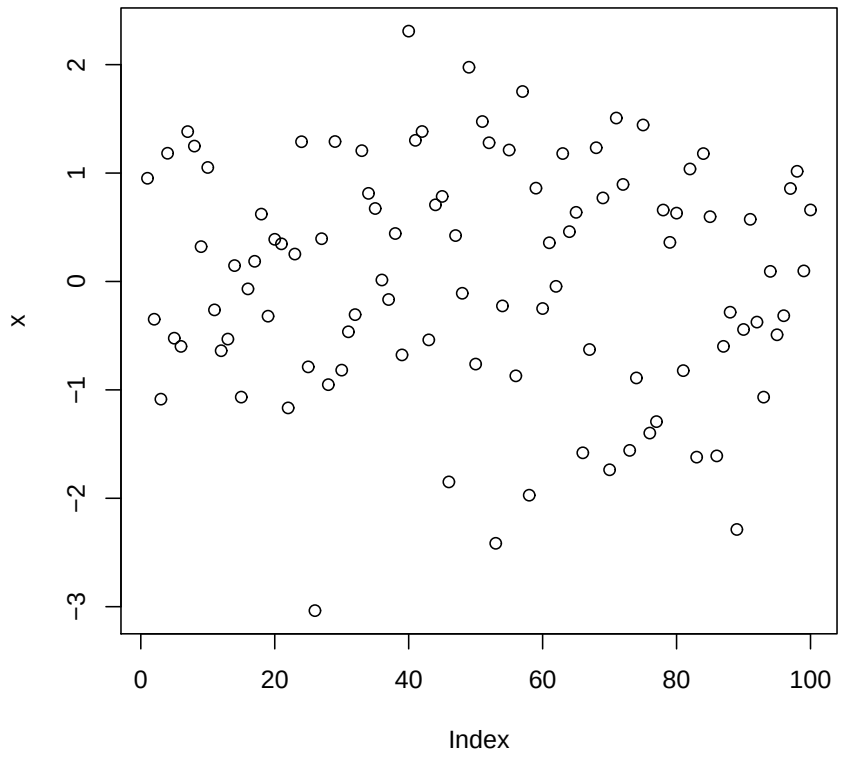
Für unsere LV ist z.B. Tinn-R (`sourceforge.net/projects/tinn-r`) unter Windows gut geeignet. Ein Editor für Mac ist z.B. SubEthaEdit (`www.codingmonkeys.de/subethaedit/`). Für Linux bietet sich Emacs mit dem ESS Plug-In (`ess.r-project.org/`) an (das verwende ich in der LV).

Für Windows ist der Editor "Tinn-R" so zu installieren und zu verwenden:

- Laden Sie sich R und Tinn-R in der aktuellsten Version herunter.
- Installieren Sie R und dann Tinn-R (der Einfachheit halber sollten Sie die voreingestellten Pfade beibehalten).
- Eigentlich sollte jetzt alles schon funktionieren. Starten Sie Tinn-R und klicken Sie auf den Menüpunkt R->RGui. Dann müsste sich ein zweites Fenster öffnen, das sich direkt an das Tinn-R Fenster anlegt. Wenn Sie jetzt File->New wählen können Sie R-Befehle eintippen und editieren und dieses File dann speichern, z.B.

  ```
  > x <- rnorm(100)
  > plot(x)
  ```

  Mit dem Befehl `Strg-L` schicken Sie eine Zeile nach R, mit `Strg-R` eine markierte Region. Sie sollten dann einen solchen Output sehen:



- Wenn das nicht klappt, dann öffnen Sie mit z.B. Notepad die Datei Rprofile.site und schauen ob folgende Zeilen vorhanden sind:

  Unter XP:

```
#===============================================================
# Tinn-R: necessary packages and functions
# Tinn-R: >= 2.0.0.1
#===============================================================
library(utils)

# check necessary packages
necessary = c('TinnR', 'svSocket')
installed = necessary %in% installed.packages()[, 'Package']
if (length(necessary[!installed]) >=1)
install.packages(necessary[!installed], dep=T)

# set options
options(IDE='C:/Tinn-R/bin/Tinn-R.exe')
options(use.DDE=T)

# load packages
library(TinnR)
library(svSocket)
```

```
# start DDE
trDDEInstall()

.trPaths = c(
'C:/Documents and Settings/IHRUSERNAME/Application Data/Tinn-R/tmp/',
'C:/Documents and Settings/IHRUSERNAME/Application Data/Tinn-R/tmp/search.txt',
'C:/Documents and Settings/IHRUSERNAME/Application Data/Tinn-R/tmp/objects.txt',
'C:/Documents and Settings/IHRUSERNAME/Application Data/Tinn-R/tmp/file.r',
'C:/Documents and Settings/IHRUSERNAME/Application Data/Tinn-R/tmp/selection.r',
'C:/Documents and Settings/IHRUSERNAME/Application Data/Tinn-R/tmp/block.r',
'C:/Documents and Settings/IHRUSERNAME/Application Data/Tinn-R/tmp/lines.r')
```

Unter Vista:

```
#===========================================================
# Tinn-R: necessary packages and functions
# Tinn-R: >= 2.0.0.1
#===========================================================
library(utils)

# check necessary packages
necessary = c('TinnR', 'svSocket')
installed = necessary %in% installed.packages()[, 'Package']
if (length(necessary[!installed]) >=1)
install.packages(necessary[!installed], dep=T)

# set options
options(use.DDE=T)
# uncoment the line below if you want Tinn-R starts
# always R starts
#options(IDE='C:/Tinn-R/bin/Tinn-R.exe')

# load packages
library(TinnR)
library(svSocket)

# start DDE
trDDEInstall()

.trPaths = c(
'C:/Users/IHRUSERNAME/AppData/Roaming/Tinn-R/tmp/',
'C:/Users/IHRUSERNAME/AppData/Roaming/Tinn-R/tmp/search.txt',
'C:/Users/IHRUSERNAME/AppData/Roaming/Tinn-R/tmp/objects.txt',
'C:/Users/IHRUSERNAME/AppData/Roaming/Tinn-R/tmp/file.r',
'C:/Users/IHRUSERNAME/AppData/Roaming/Tinn-R/tmp/selection.r',
'C:/Users/IHRUSERNAME/AppData/Roaming/Tinn-R/tmp/block.r',
'C:/Users/IHRUSERNAME/AppData/Roaming/Tinn-R/tmp/lines.r')
```

Falls nicht, ergänzen sie die fehlenden Einträge.