

Erste Version: April 2012 (Christian Kleiber, Linda Walter, Jean-Michel Benkert)

Dieses Update: Mai 2024 (Christian Kleiber, Yannick Kohler)

## Datenmanagement in R: Eine Einführung

Diese Einführung soll Studierenden den Einstieg in praktisches Arbeiten mit R erleichtern. In der Regel ist der erste Schritt bei ökonomischer oder statistischer Arbeit die Datenbeschaffung. Da die Datenbeschaffung stark von der jeweiligen Untersuchung abhängt, wollen wir uns hier nicht weiter damit beschäftigen. Wir befassen uns direkt mit dem nächsten Schritt: Daten in R einlesen.

Um sinnvoll mit Daten in R arbeiten zu können, müssen die Daten in der Regel zunächst entsprechend aufbereitet werden. In dieser Einführung werden wir also sehen, wie Daten in R eingelesen, bearbeitet und schliesslich gesichert werden können. Wir verwenden Daten für eine Wachstumsregression aus einer Arbeit von Durlauf und Johnson (1995) [1]. Diese Daten sind im Datenarchiv des *Journal of Applied Econometrics* unter <https://journaldata.zbw.eu/dataset/multiple-regimes-and-crosscountry-growth-behaviour> verfügbar.

R-Code wird im vorliegenden Dokument **in dieser Schrift** geschrieben. Wenn Sie R installieren und die Voreinstellung verwenden, steht im Prompt am Anfang der Zeile ein `>`, resp. ein `+`, wenn die Eingabe von Code über mehrere Zeilen erfolgt. In diesem Dokument haben wir den Prompt verändert. Am Anfang der Zeile steht nun `R>`, was die Eingabe von Code in R verdeutlicht. Beim Nachrechnen müssen weder `R>` noch `+` am Anfang der Zeile in R eingegeben werden. Im Text wird mit zwei Klammern `()` verdeutlicht, wann es sich um Funktionen handelt (Beispiel: `head()`). In die Klammern `()` werden bei der Anwendung der Funktion die Argumente geschrieben, die der Funktion übergeben werden (Beispiel: `head(DataDJ)`).

### Inhaltsverzeichnis

|          |                               |          |
|----------|-------------------------------|----------|
| <b>1</b> | <b>Vorbereitungen</b>         | <b>2</b> |
| <b>2</b> | <b>Daten einlesen</b>         | <b>2</b> |
| 2.1      | Text-Dateien                  | 2        |
| 2.2      | csv-Dateien                   | 2        |
| 2.3      | Zwischenablage                | 3        |
| 2.4      | rda-Dateien und RData-Dateien | 3        |
| <b>3</b> | <b>Daten aufbereiten</b>      | <b>3</b> |
| 3.1      | Fehlende Werte                | 5        |
| 3.2      | Formatierung                  | 5        |
| <b>4</b> | <b>Daten sichern</b>          | <b>8</b> |
| 4.1      | rda-Dateien und RData-Dateien | 8        |
| 4.2      | csv-Dateien                   | 8        |
| 4.3      | txt-Dateien                   | 8        |
| <b>5</b> | <b>Schlussbemerkung</b>       | <b>8</b> |

## 1 Vorbereitungen

In einem ersten Schritt sollte das aktuelle Arbeitsverzeichnis wie gewünscht definiert werden. In R kann das aktuelle Arbeitsverzeichnis mit der Funktion `getwd()` abgefragt werden:

```
R> getwd()
```

```
[1] "C:/Documents and Settings/Default User/My Documents"
```

Bei uns liegt das Arbeitsverzeichnis im *My Documents*-Ordner. Das Arbeitsverzeichnis kann unter Windows via *Datei* → *Verzeichnis wechseln* oder allgemein mit der Funktion `setwd()` geändert werden:

```
R> setwd("Z:/MeinOrdner")
```

Gerade bei grösseren Projekten ist es sinnvoll, für das Projekt einen eigenen, neuen Ordner anzulegen.

## 2 Daten einlesen

Es gibt verschiedene Möglichkeiten, um Daten in R einzulesen. Das *R Development Core Team* liefert diesbezüglich unter <https://cran.r-project.org/doc/manuals/R-data.pdf> selbst eine Anleitung, in welcher alle wesentlichen Möglichkeiten zum Import von Daten erläutert werden. Wir werden hier nur eine Auswahl davon betrachten. Im Internet finden Sie viele weitere Anleitungen ... in recht unterschiedlicher Qualität.

Beachten Sie, dass das Datenarchiv des *Journal of Applied Econometrics* unsere Daten in der Datei `data.dj` zur Verfügung stellt. Wir haben `data.dj` für diese Anleitung in `DataDJ.txt` umbenannt, ansonsten ist die Datei nicht verändert. Wir erklären hier das Arbeiten mit genau dieser Textdatei, stellen die Daten aber auch in anderen Formaten zur Verfügung (`csv`, `rda`, `RData`, `xls`), so dass Sie mit diesen anderen Formaten wie unten beschrieben experimentieren können.

### 2.1 Text-Dateien

Liegen die Daten als Excel-Datei (siehe z.B. `DataDJ.xls`) vor, sollte man die Daten, die man einlesen möchte, zunächst als `csv`- oder Textdatei speichern. Textdateien können in Excel via *Datei* → *Speichern unter* gespeichert werden. Als *Dateityp* muss *Text (Tabstop-getrennt)(\*.txt)* angegeben werden. Aus der `txt`-Datei muss alles Überflüssige entfernt werden, bevor man sie in R einliest.

Das Einlesen der Daten in R erfolgt dann mit der Funktion `read.table()`.

Das Argument `header = TRUE` gibt an, dass die erste Zeile der einzulesenden Datei die Namen der Variablen enthält. Die Daten sind jetzt durch ein Leerzeichen voneinander getrennt, daher `sep = ""`:

```
R> DataDJ <- read.table("Z:/MeinOrdner/DataDJ.txt", sep = "", header = TRUE)
```

Falls das Arbeitsverzeichnis dem Ordner `Z:/MeinOrdner` entspricht, muss der Pfad wiederum nicht angegeben werden:

```
R> DataDJ <- read.table("DataDJ.txt", sep = "", header = TRUE)
```

### 2.2 csv-Dateien

Wie bereits erwähnt, sollte man Excel-Dateien als `csv`- oder Textdatei speichern, um sie in R einzulesen. `csv`-Dateien können in Excel via *Datei* → *Speichern unter* gespeichert werden. Als *Dateityp* muss *CSV (Trennzeichen-getrennt)(\*.csv)* angegeben werden. Auch aus der `csv`-Datei muss alles Überflüssige entfernt werden, bevor man sie in R einliest.

Das Einlesen der Daten in R erfolgt dann mit der Funktion `read.csv()`. Das Argument `sep = ";"` gibt an, dass die Daten mit einem Semikolon voneinander getrennt sind. `header = TRUE` gibt wieder an, dass die erste Zeile der csv-Datei die Namen der Variablen enthält:

```
R> DataDJ <- read.csv("Z:/MeinOrdner/DataDJ.csv", sep = ";", header = TRUE)
```

Falls das Arbeitsverzeichnis dem Ordner `Z:/MeinOrdner` entspricht, muss der Pfad wiederum nicht angegeben werden:

```
R> DataDJ <- read.csv("DataDJ.csv", sep = ";", header = TRUE)
```

## 2.3 Zwischenablage

Diese Variante kann insbesondere bei kleinen Datensätzen von Vorteil sein. Die gewünschten Daten werden z.B. in Excel mittels `Ctrl+C` in der Zwischenablage gespeichert und werden dann wie folgt in R geladen:

```
R> DataDJ <- read.table("clipboard", header = TRUE)
```

## 2.4 rda-Dateien und RData-Dateien

`rda` und `RData`-Dateien sind R-eigene Dateiformate, wobei `rda` das ältere der beiden Formate ist. Beide Formate sind allerdings im Wesentlichen identisch und können genau gleich verwendet werden. Liegen die Daten bereits als `rda`- oder `RData`-Datei vor, kann die Datei direkt mit der Funktion `load()` geladen werden:

```
R> load("Z:/MeinOrdner/DataDJ.rda")
```

Wenn sich die Datei im aktuellen Arbeitsverzeichnis `Z:/MeinOrdner` befindet, muss der Pfad wieder nicht angegeben werden:

```
R> load("DataDJ.rda")
```

In Betriebssystemen wie MS Windows oder macOS geht das auch über die Menüleiste im GUI: *Datei* → *Lade Workspace* anklicken. Bei *Dateityp* muss *R images - old extension (\*.rda)* ausgewählt werden. Dieser letzte Schritt müsste für eine `RData`-Datei natürlich nicht gemacht werden. Dann noch die Datei auswählen und *öffnen* klicken.

Mit dem Aufruf von `ls()` kann überprüft werden, was in der `rda`-Datei bzw. momentan im Arbeitsspeicher enthalten ist:

```
R> ls()
```

```
[1] "DataDJ"
```

## 3 Daten aufbereiten

Wir werden im Folgenden die als Textdatei eingelesenen Rohdaten so aufbereiten, dass sie im gleichen Format vorhanden sind wie im entsprechenden Datensatz `GrowthDJ` im R-Paket **AER**.

In einem ersten Schritt wollen wir uns nun einen Eindruck von den eingelesenen Daten verschaffen. Dazu geeignet ist die Funktion `str()`, welche auf kompakte Weise die interne Struktur eines R-Objekts wiedergibt.

```
R> str(DataDJ)
```

```
## 'data.frame':   121 obs. of  11 variables:
## $ NUMBER: int  1 2 3 4 5 6 7 8 9 10 ...
```

```
## $ NONOIL: num 1 1 1 1 1 1 1 1 1 1 ...
## $ INTER : num 1 0 0 1 0 0 1 0 0 0 ...
## $ OECD : num 0 0 0 0 0 0 0 0 0 0 ...
## $ GDP60 : num 2485 1588 1116 959 529 ...
## $ GDP85 : num 4371 1171 1071 3671 857 ...
## $ GDPGRO: num 4.8 0.8 2.2 8.6 2.9 1.2 5.7 1.5 -0.9 6.2 ...
## $ POPGRO: num 2.6 2.1 2.4 3.2 0.9 1.7 2.1 1.7 1.9 2.4 ...
## $ IONY : num 24.1 5.8 10.8 28.3 12.7 5.1 12.8 10.5 6.9 28.8 ...
## $ SCHOOL: num 4.5 1.8 1.8 2.9 0.4 0.4 3.4 1.4 0.4 3.8 ...
## $ LIT60 : num 10 5 5 -999 2 14 19 7 6 16 ...
```

Es handelt sich bei `DataDJ` um ein Objekt der Klasse `data.frame`. Wir können uns `DataDJ` als eine Tabelle der Grösse  $121 \times 11$  vorstellen, wie die Funktion `dim()` schnell bestätigt:

```
R> dim(DataDJ)
```

```
## [1] 121 11
```

Das Besondere an einem `data.frame` ist, dass in unterschiedlichen Spalten die Elemente auch von unterschiedliche Datentypen (`numeric`, `logical`, `character`) sein dürfen, was bspw. bei einer `matrix` nicht möglich ist. Offenbar sind alle Variablen in `DataDJ` vom Typ `numeric` (`integer` ist ein Spezialfall, eine Unterklasse, von `numeric`). Zu jeder der 11 Variablen werden zudem die ersten Beobachtungen angezeigt.

Um die ersten Beobachtungen eines Datensatzes anzuzeigen, kann man auch die Funktion `head()` verwenden:

```
R> head(DataDJ)
```

```
##   NUMBER NONOIL INTER OECD GDP60 GDP85 GDPGRO POPGRO IONY SCHOOL LIT60
## 1     1      1     1    0  2485  4371     4.8   2.6 24.1   4.5   10
## 2     2      1     0    0  1588  1171     0.8   2.1  5.8   1.8    5
## 3     3      1     0    0  1116  1071     2.2   2.4 10.8   1.8    5
## 4     4      1     1    0   959  3671     8.6   3.2 28.3   2.9  -999
## 5     5      1     0    0   529   857     2.9   0.9 12.7   0.4    2
## 6     6      1     0    0   755   663     1.2   1.7  5.1   0.4   14
```

Die Funktion `head()` zeigt per Voreinstellung die ersten 6 Zeilen resp. Elemente an. Wenn man beispielsweise nur die ersten vier Zeilen sehen möchte, kann man diese über den Aufruf `head(DataDJ, n = 4)` erhalten.

Alternativ können wir uns mit der Funktion `summary()` deskriptive Statistiken ansehen:

```
R> summary(DataDJ)
```

```
##      NUMBER      NONOIL      INTER      OECD      GDP60
## Min.   : 1   Min.   :0.00   Min.   :0.00   Min.   :0.000   Min.   : -999
## 1st Qu.: 31  1st Qu.:1.00   1st Qu.:0.00   1st Qu.:0.000   1st Qu.:  907
## Median : 61  Median :1.00   Median :1.00   Median :0.000   Median : 1842
## Mean   : 61  Mean   :0.81   Mean   :0.62   Mean   :0.182   Mean   : 3488
## 3rd Qu.: 91  3rd Qu.:1.00   3rd Qu.:1.00   3rd Qu.:0.000   3rd Qu.: 3766
## Max.   :121  Max.   :1.00   Max.   :1.00   Max.   :1.000   Max.   :77881
##      GDP85      GDPGRO      POPGRO      IONY
## Min.   : -999   Min.   : -999.0   Min.   : -999.0   Min.   :  4.1
## 1st Qu.:  974   1st Qu.:  2.6    1st Qu.:  1.1    1st Qu.:12.0
## Median : 2544   Median :  3.8    Median :  2.3    Median :17.7
## Mean   : 4965   Mean   : -29.1   Mean   : -113.6   Mean   :18.2
```

```
## 3rd Qu.: 6868 3rd Qu.: 5.2 3rd Qu.: 2.8 3rd Qu.:24.1
## Max. :25635 Max. : 9.2 Max. : 6.8 Max. :36.9
## SCHOOL LIT60
## Min. :-999.0 Min. :-999
## 1st Qu.: 2.3 1st Qu.: 7
## Median : 4.8 Median : 29
## Mean : -19.4 Mean :-108
## 3rd Qu.: 8.1 3rd Qu.: 75
## Max. : 12.1 Max. : 100
```

### 3.1 Fehlende Werte

Empirische Datensätze enthalten häufig fehlende Werte, d.h. der Wert einer oder mehrerer Variablen fehlt für manche Beobachtungen. Deshalb wurde unser Datensatz auch als Beispiel ausgewählt, er enthält fehlende Werte. U.a. weist die Variable `DataDJ$gdp60` ein Minimum von  $-999$  auf. Die Erklärung findet sich in der Dokumentation des Datensatzes, in der Datei `readme.dj.txt` im Datenarchiv des *Journal of Applied Econometrics*. Dort steht: “‘A value of -999 indicates that the observation is missing’”. Wir lesen deshalb die Daten nochmals ein und ersetzen beim Einlesen die betroffenen Werte gleich mit `NA` (für *not available*):

```
R> DataDJ <- read.table("DataDJ.txt", sep = "", header = TRUE,
+                       na.strings = c("-999.0", "-999.00"))
```

Beachten sie, dass auch ein Vektor von Strings eingegeben werden kann, wie hier erforderlich: die Datei `DataDJ.txt` enthält fehlende Werte sowohl in der Form  $-999.0$  als auch als  $-999.00$ .

Wenn man nicht-vorhandene Datenpunkte mittels dieser Variante entsprechend markieren möchte, ist allerdings Vorsicht angebracht. Es wäre bspw. möglich, dass der Wert  $-999.0$  nicht immer nicht-vorhandene Datenpunkte angibt, sondern für gewisse Beobachtungen tatsächlich ein korrekter Wert ist. Die Software könnte dies natürlich nicht ahnen und würde in diesem Fall oben alle Werte durch `NA` ersetzen. Darauf müssen Sie bei der Datenaufbereitung selbst achten.

Alternativ hätten wir den Datensatz wie bisher einlesen und die betroffenen Werte nachträglich durch `NA` ersetzen können:

```
R> DataDJ$GDP60[DataDJ$GDP60 == -999] <- NA
```

Dies hätten wir für jede betroffene Variable so gemacht.

### 3.2 Formatierung

Bevor wir die Daten nun aufbereiten, schauen wir uns an, in welchem Format wir sie am Ende haben möchten. Dazu installieren wir, falls noch nicht vorhanden, zunächst das R-Paket **AER**,

```
R> install.packages("AER")
```

laden es,

```
R> library("AER")
```

laden die Daten

```
R> data("GrowthDJ", package = "AER")
```

und sehen uns die Struktur von `GrowthDJ` an:

```
R> str(GrowthDJ)
```

```
## 'data.frame': 121 obs. of 10 variables:
## $ oil : Factor w/ 2 levels "yes","no": 2 2 2 2 2 2 2 2 2 2 ...
## $ inter : Factor w/ 2 levels "no","yes": 2 1 1 2 1 1 2 1 1 1 ...
## $ oecd : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
## $ gdp60 : num 2485 1588 1116 959 529 ...
## $ gdp85 : num 4371 1171 1071 3671 857 ...
## $ gdpgrowth : num 4.8 0.8 2.2 8.6 2.9 1.2 5.7 1.5 -0.9 6.2 ...
## $ popgrowth : num 2.6 2.1 2.4 3.2 0.9 1.7 2.1 1.7 1.9 2.4 ...
## $ invest : num 24.1 5.8 10.8 28.3 12.7 5.1 12.8 10.5 6.9 28.8 ...
## $ school : num 4.5 1.8 1.8 2.9 0.4 0.4 3.4 1.4 0.4 3.8 ...
## $ literacy60: num 10 5 5 NA 2 14 19 7 6 16 ...
```

Als erstes fällt auf, dass es eine Variable weniger gibt: die Variable NUMBER fehlt. Diese Variable läuft von 1 bis 121, nummeriert also die Daten durch. Als ersten Schritt werden wir nun diese Variable, also Spalte, aus dem Datensatz DataDJ entfernen. Da NUMBER in der ersten Spalte liegt, können wir dies mit folgendem Aufruf erreichen:

```
R> DataDJ <- DataDJ[, -1]
```

Mit [i, j] geben wir an, welche Zeilen i und welche Spalten j wir auswählen möchten. [, -1] bedeutet, dass wir alle Zeilen auswählen (deshalb steht nichts vor dem Komma) und die erste Spalte weglassen möchten (dies drücken wir mit dem Minus-Zeichen aus).

Jedoch kann es gerade bei grösseren Datensätzen sehr aufwändig und mühsam sein, den Index der zu entfernenden Spalte herauszufinden. Die Funktion match() übernimmt dies für uns. An dieser Stelle erinnern wir an die Hilfsfunktion in R, aufrufbar mit ?match, welche uns in diesem Fall die Beschreibung der Funktion liefert.

```
R> DataDJ <- DataDJ[, -match(c("NUMBER"), names(DataDJ))]
```

In diesem einfachen Beispiel, in welchem nur eine Spalte und dazu noch die erste entfernt werden muss, wirkt dies umständlich, kann aber bei grösseren Datensätzen viel Arbeit ersparen.

Bevor wir uns anspruchsvolleren Datenmanipulationen zuwenden, passen wir noch kurz die Namen an. Die bisherigen Namen können mit der Funktion names() abgefragt werden:

```
R> names(DataDJ)
```

```
## [1] "NONOIL" "INTER" "OECD" "GDP60" "GDP85" "GDPGRO" "POPGRO" "IONY"
## [9] "SCHOOL" "LIT60"
```

```
R> names(GrowthDJ)
```

```
## [1] "oil" "inter" "oecd" "gdp60" "gdp85"
## [6] "gdpgrowth" "popgrowth" "invest" "school" "literacy60"
```

Wir überschreiben die Namen in DataDJ wie folgt:

```
R> colnames(DataDJ) <- c("oil", "inter", "oecd", "gdp60", "gdp85", "gdpgrowth",
+ "popgrowth", "invest", "school", "literacy60")
```

Im Unterschied zu DataDJ enthält GrowthDJ nicht nur numerische Datentypen, sondern auch sogenannte Faktoren. In Faktoren kann man qualitative (kategoriale) Informationen speichern, wie bspw. das Geschlecht. Ein Faktor kann auch mehr als zwei Ausprägungen umfassen.

Tatsächlich ist DataDJ\$oil zur Zeit kein Faktor, sondern nur ein dummy-codierter numerischer Vektor. Mit der Funktion factor() lässt sich eine Variable wie folgt in einen Faktor umwandeln:

```
R> DataDJ$oil <- factor(DataDJ$oil, levels = c(0,1), labels = c("yes", "no"))
R> DataDJ$inter <- factor(DataDJ$inter, levels = c(0,1), labels = c("no", "yes"))
R> DataDJ$oecd <- factor(DataDJ$oecd, levels = c(0,1), labels = c("no", "yes"))
```

Für DataDJ\$oil haben wir 0 der Kategorie “yes” und 1 der Kategorie “no” zugeordnet. Der Grund dafür ist, dass im eingelesenen DataDJ Datensatz die Variable NONOIL hiess und laut Dokumentation also diejenigen Länder markierte, welche *keine* Ölvorkommen besitzen. Im GrowthDJ Datensatz sind allerdings Länder markiert, welche Ölvorkommen besitzen, daher die Vertauschung. Man beachte noch, dass R aus internen Gründen nicht 0 und 1, sondern 1 und 2 als numerische Repräsentation der Ausprägungen eines Faktors ausgibt.

Ein Überblick über die aufbereiteten Daten erhält man wieder mit `summary()` und `str()`. Wir sehen, dass die Datensätze jetzt übereinstimmen:

```
R> summary(DataDJ)
```

```
##  oil      inter      oecd      gdp60      gdp85      gdpgrowth
## yes:23   no :46   no :99   Min.   : 383   Min.   : 412   Min.   : -0.90
## no :98   yes:75   yes:22   1st Qu.: 973   1st Qu.: 1209  1st Qu.: 2.80
##
##          Median : 1962   Median : 3484   Median : 3.90
##          Mean    : 3682   Mean    : 5683   Mean    : 4.09
##          3rd Qu.: 4274   3rd Qu.: 7719   3rd Qu.: 5.30
##          Max.    : 77881   Max.    : 25635   Max.    : 9.20
##          NA's    : 5      NA's    : 13     NA's    : 4
##  popgrowth      invest      school      literacy60
## Min.   :0.30   Min.   : 4.1   Min.   : 0.40   Min.   : 1.0
## 1st Qu.:1.70   1st Qu.:12.0  1st Qu.: 2.40   1st Qu.: 15.0
## Median :2.40   Median :17.7  Median : 4.95   Median : 39.0
## Mean   :2.28   Mean   :18.2  Mean   : 5.53   Mean   : 48.2
## 3rd Qu.:2.90   3rd Qu.:24.1  3rd Qu.: 8.18   3rd Qu.: 83.5
## Max.   :6.80   Max.   :36.9  Max.   :12.10   Max.   :100.0
## NA's   :14      NA's   : 3     NA's   :18
```

```
R> summary(GrowthDJ)
```

```
##  oil      inter      oecd      gdp60      gdp85      gdpgrowth
## yes:23   no :46   no :99   Min.   : 383   Min.   : 412   Min.   : -0.90
## no :98   yes:75   yes:22   1st Qu.: 973   1st Qu.: 1209  1st Qu.: 2.80
##
##          Median : 1962   Median : 3484   Median : 3.90
##          Mean    : 3682   Mean    : 5683   Mean    : 4.09
##          3rd Qu.: 4274   3rd Qu.: 7719   3rd Qu.: 5.30
##          Max.    : 77881   Max.    : 25635   Max.    : 9.20
##          NA's    : 5      NA's    : 13     NA's    : 4
##  popgrowth      invest      school      literacy60
## Min.   :0.30   Min.   : 4.1   Min.   : 0.40   Min.   : 1.0
## 1st Qu.:1.70   1st Qu.:12.0  1st Qu.: 2.40   1st Qu.: 15.0
## Median :2.40   Median :17.7  Median : 4.95   Median : 39.0
## Mean   :2.28   Mean   :18.2  Mean   : 5.53   Mean   : 48.2
## 3rd Qu.:2.90   3rd Qu.:24.1  3rd Qu.: 8.18   3rd Qu.: 83.5
## Max.   :6.80   Max.   :36.9  Max.   :12.10   Max.   :100.0
## NA's   :14      NA's   : 3     NA's   :18
```

## 4 Daten sichern

Wenn man die Daten aufbereitet hat, möchte man sie oft auch sichern, um sie auch in Zukunft in dieser Form verwenden zu können. Natürlich lassen sich Daten in R in diverse Dateiformate exportieren; wir werden die nützlichsten kurz erläutern.

### 4.1 rda-Dateien und RData-Dateien

Falls die Daten zu einem späteren Zeitpunkt wieder in R verwendet werden sollen, ist es naheliegend die Daten im R-eigenen Binärformat (Endungen `.rda` oder `.RData`) zu speichern, da so beim späteren Laden mögliche Komplikationen vermieden werden. Wir verwenden hierbei die Funktion `save()`:

```
R> save(DataDJ, file = "DataDJ2.rda")
```

`file = "DataDJ2.rda"` bestimmt den Dateinamen der `rda`-Datei, in welcher wir das Objekt `DataDJ` speichern. Die Datei wurde jetzt im aktuellen Arbeitsverzeichnis gespeichert. Durch die Angabe eines Pfades könnte eine Datei auch ausserhalb des aktuellen Arbeitsverzeichnisses gespeichert werden. Wir können überprüfen, ob dies wirklich geklappt hat, indem wir `DataDJ` mit der Funktion `rm()` aus dem aktuellen Arbeitsspeicher entfernen und dann wieder mit `load()` laden oder indem wir mit der Funktion `dir()` abfragen, was sich im aktuellen Arbeitsverzeichnis befindet.

### 4.2 csv-Dateien

Natürlich können Daten auch in das `csv`-Format exportiert werden. Dafür verwenden wir die Funktion `write.csv()` oder `write.csv2()`. Für die Unterschiede zwischen den beiden Funktionen verweisen wir auf die Dokumentation (`?write.csv`).

```
R> write.csv(DataDJ, file = "DataDJ2.csv")
```

### 4.3 txt-Dateien

Eine nützliche Variante ist der Export der Daten als Text-Datei, da Text-Dateien in jedes Programm eingelesen werden können. In R ist dies mit der Funktion `write.table()` möglich:

```
R> write.table(DataDJ, file = "DataDJ2.txt", col.names = TRUE,  
+             sep = " ", dec = ".", na = "NA")
```

Wir bestimmen hier, ob die Namen der Variablen gelistet werden (`col.names`), wie die einzelnen Variablen voneinander getrennt werden (`sep`), welcher String den Dezimalpunkt darstellt (`dec`) und wie nicht vorhandene Datenpunkte benannt werden (`na`).

## 5 Schlussbemerkung

Dieses Dokument soll als Leitfaden und erste, kleine Einführung ins Datenmanagement in R dienen. Tatsächlich kann man Daten in R auf zahlreiche Arten einlesen, bearbeiten und speichern. Am besten lernt man die Möglichkeiten von R kennen, wenn man sich selbst an einen Datensatz wagt und versucht die Daten ein wenig zu manipulieren und in einem nächsten Schritt explorativ zu analysieren oder graphisch darzustellen.

Ein Blick in die Dokumentation einer Funktion kann in der Regel die meisten Probleme schon beseitigen (wie schon erwähnt aufrufbar mit `?` und dem Funktionsnamen, z.B. `?write.table`).

## Literatur

- [1] Steven N. Durlauf und Paul A. Johnson, "Multiple Regimes and Cross-Country Growth Behavior", *Journal of Applied Econometrics*, Vol. 10, No. 4, 1995, 365–384.