The "Live" Code of Lecture 5

```
##############################################
# LECTURE 5: Handling data and data.frames  #
##############################################


# We will continute working with the data sets for the SNB
# That you downloaded for Lecture 4.

# This time, we are assuming that the data already look neat
# So no deletion of empty columns any more.

# Preparatory steps
####################

# Almost everytime you work with data, you should do the
# following steps...

rm(list = ls()) # Empty workspace to start with a "clean sheet"

# REPLACE THE WORKING DIRECTORY BELOW WITH THE ONE FOR YOUR DEVICE
setwd("D:/Dropbox/Mac&Surf/Programmierkurs Dropb/Data")


# Read the data
###############

# Last time, we read the data like this:
rawXrates = read.csv(
  file = "SNB Xrates downloaded clean noEmptyCol.csv", sep = ",")

# For some of you, sep should take the value ";".

# However, for some of you, the column names may land in the first row,
# which is not the idea. So add "header = TRUE":
rawXrates = read.csv(file =
   "SNB Xrates downloaded clean noEmptyCol.csv",
       sep = ",", header = TRUE)



# Referring to columns in data.frames
#####################################

# There are (at least) two ways to refer to a column:

#  rawXrates[["DO"]]
#  rawXrates$DO

# For instance, you can use this for
# getting all the values in a column
```

```
length(unique(rawXrates$D0))
```

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

```
table(rawXrates$D0)
```

```
##
##    M0    M1
## 32058 32058
```

```
# or the type

class(rawXrates[["D0"]])
```

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

```
# Note the subtle difference between
class(rawXrates[["D0"]])
```

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

```
# and
class(rawXrates["D0"])
```

```
## [1] "data.frame"
```

```
# Sometimes, this matters, sometimes not...
# For deleting empty columns, it does not.

# NOTE: Our data is in the so-called "long" format: all variables (in the statistical
# sense) are "stacked".

# The statistical variable names are a combination of D0 and D1.

# The SNB does not make it too easy to get the meaning of D0. But if you go
# to the Data Portal https://data.snb.ch/de/topics/ziredev#!/cube/devkum
# and download the data in Excel format (which, by the way, is useless for
# reading the data into R), you get the meaning of the exchange rates.


# Converting data from "long" to "wide" and back to "long"
################################################################

# See "R for Everyone", Section 12.3

# R comes with lots of "packages". For converting data from long to wide
# we need the package "reshape2"

# A package is intalled like this:
# install.packages("reshape2")
```

```r
# Do this only once on your machine

# Everytime you use a package, you have to "call" or "load" it
library(reshape2)
# or
require(reshape2)


wide = dcast(rawXrates, # the data frame
             Date ~ # the variable that is to
               # 'IDENTIFY' ROWS of new
               # variables! Note the '~'!
               D0 + D1 , # the column(s) containing
             # what you want to become
             # the new variable NAMES
             value.var = "Value")  # column that contains the
              # VALUES of the new variables
              # (often, you can ommit this)

# In short:
wide = dcast(rawXrates, Date ~ D0 + D1)
```

```
## Using Value as value column: use value.var to override.
```

```r
# Check out, what the following would do:
wide = dcast(rawXrates, Date ~ D0)
```

```
## Using Value as value column: use value.var to override.

## Aggregation function missing: defaulting to length
```

```r
wide = dcast(rawXrates, Date ~ D1)
```

```
## Using Value as value column: use value.var to override.
## Aggregation function missing: defaulting to length
```

```r
wide = dcast(rawXrates, Date ~ D0 + D1)
```

```
## Using Value as value column: use value.var to override.
```

```r
# Let's go back to "long". For the moment, this is more useful

long = melt(wide, id.vars = "Date",
            value.name = "Value")

# D0 and D1 are now merged. We could change this, but
# we have more important things to do...


# Recoding date as numerical, and a unique time identifier
```

```
############################################################

# Our data looks really enormously big
# Let's say we only care about data from 2000 on
# to start with...

# The next lines of code are preparations for
# data from 2000 on (or any other year)


#install.packages("stringr")
library(stringr)

rawXrates$year = as.numeric(
  substr(rawXrates$Date, start = 1, stop = 4)  )

rawXrates$month = as.numeric(
  substr(rawXrates$Date, start = 6, stop = 7)  )


# A unique identifier for time
rawXrates$timeID = rawXrates$year +
  (rawXrates$month-1)/12



# Eliminating rows and columns from data.frames
# (= selection of subsets of data)
#####################################################

# Let's get rid of all data before 2000

Xrates = rawXrates

Xrates = Xrates[Xrates$year>=2000, ]

# Now you can see why we needed the dates in numerical format!


# Next we get rid of other information we are not interested in...
unique(Xrates$D0)
```

```
## [1] M0 M1
## Levels: M0 M1
```

```
Xrates = Xrates[Xrates$D0 == "M0", ]

# Let's select only Euro exchange rates

# How is the EUR coded? For a factor
unique(Xrates$D1)
```

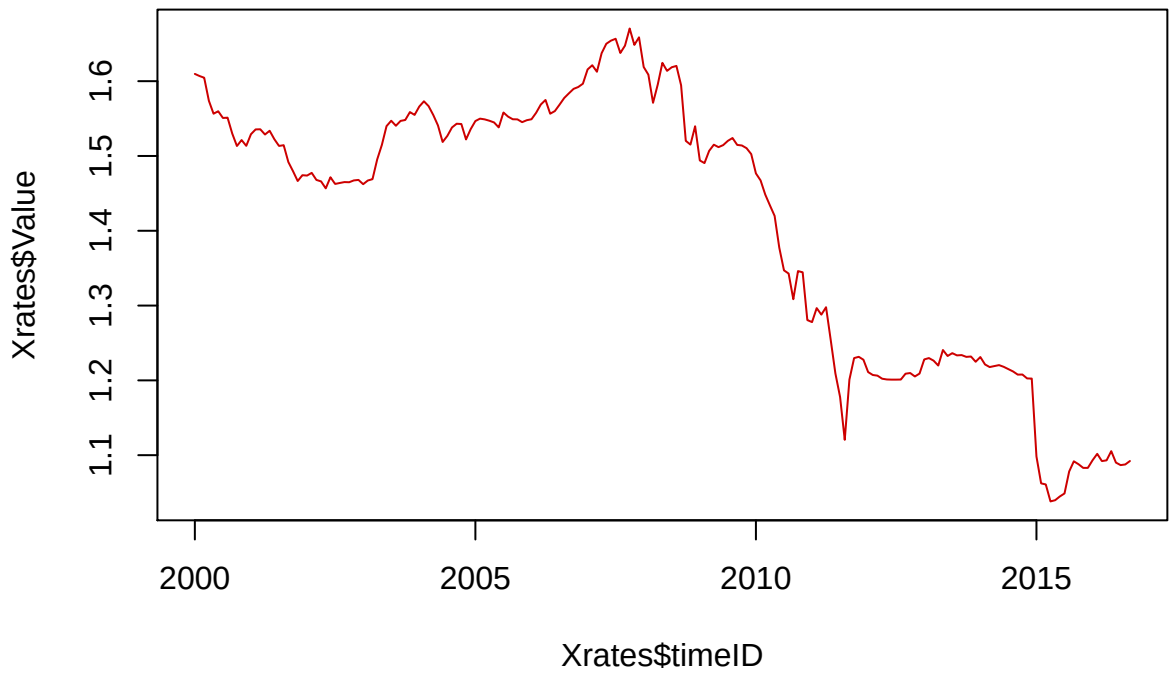
```
##  [1] EUR1   GBP1   DKK100 NOK100 CZK100 HUF100 PLN100 RUB100 SEK100 TRY100
```

```
## [11] USD1   CAD1   ARS100 BRL100 MXN100 ZAR1   JPY100 AUD1   CNY100 HKD100
## [21] KRW100 MYR100 NZD1   SGD100 THB100 XDR1
## 26 Levels: ARS100 AUD1 BRL100 CAD1 CNY100 CZK100 DKK100 EUR1 ... ZAR1
```

```
# does sometimes not work so well. In that case, use
levels(Xrates$D1)
```

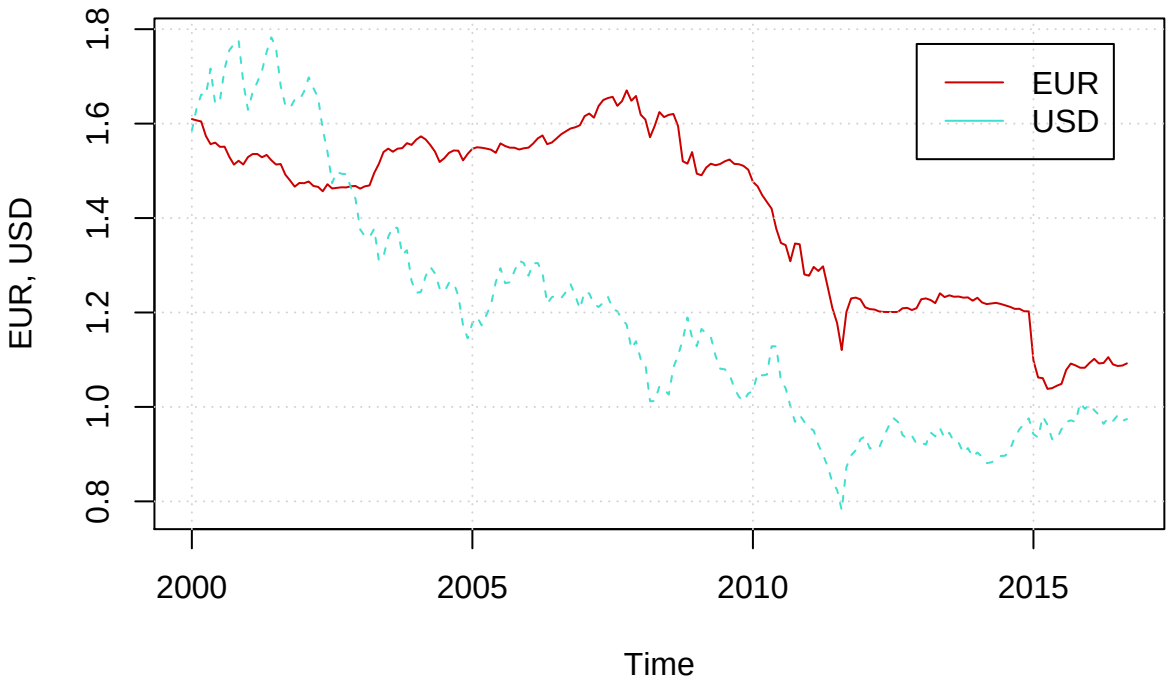
```
##  [1] "ARS100" "AUD1"   "BRL100" "CAD1"   "CNY100" "CZK100" "DKK100"
##  [8] "EUR1"   "GBP1"   "HKD100" "HUF100" "JPY100" "KRW100" "MXN100"
## [15] "MYR100" "NOK100" "NZD1"   "PLN100" "RUB100" "SEK100" "SGD100"
## [22] "THB100" "TRY100" "USD1"   "XDR1"   "ZAR1"
```

```
Xrates = Xrates[Xrates$D1 == "EUR1", ]


plot(Xrates$timeID, Xrates$Value,
     type = "l", col = "red3")
```




```
# You can also use the subset command

XratesAlt = subset(rawXrates,
      year >= 2000 & D0 == "M0"  & D1 == "EUR1",
       select = c(timeID, D1, Value))

# And if we again converted the dates, we could select data >= 2010
```

```r
# Remove objects we no longer need

rm(long, wide, XratesAlt)


# Analyze the correlation between the USD and EUR exchange rate
#####################################################################



data = subset(rawXrates,
        (D1 == "EUR1" | D1 == "USD1") &
          D0 == "M0" &
          timeID >= 2000 ,
        select = c("timeID", "D1", "Value"))

# Bring data into wide format
library(reshape2)
data = dcast(data, timeID ~ D1, value.var = "Value")


# Make a plot
matplot(data$timeID, cbind(data$EUR1, data$USD1),
        type = "l", xlab = "Time", ylab = "EUR, USD", col = c("red3", "turquoise"))
grid()
legend('topright', inset=.05, legend = c("EUR", "USD"),
       lty = 1, col = c("red3", "turquoise"))
```




```
# Correlation
cor(data$EUR1, data$USD1)
```

```
## [1] 0.6299145
```

```
# Run a regression
reg0 = lm(data$USD1 ~ data$EUR1)
summary(reg0)
```

```
##
## Call:
## lm(formula = data$USD1 ~ data$EUR1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.34302 -0.12688 -0.06599  0.08268  0.49984
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.08376    0.11153  -0.751    0.454
## data$EUR1    0.89761    0.07845  11.441   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2024 on 199 degrees of freedom
```

```
## Multiple R-squared:  0.3968, Adjusted R-squared:  0.3938
## F-statistic: 130.9 on 1 and 199 DF,  p-value: < 2.2e-16
```