The "live" code of Lecture 8

The first script

We first used the new "dressed-up" data in `dataForAnalysis.RData` to repeat the analysis from Lecture 6. However, we went to great length to always keep nice variable names and column and row names, wheverever needed, such that we will get nice labels and titles in figures, tables, and titles.

The first script we worked on is the following.

```
##############################################################################
#     Redoing analysis from Lecture 6, but with dressed-up data        #
##############################################################################


rm(list = ls())

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


load("dataForAnalysis.RData")

# SET PARAMETERS HERE
######################
# These for lines just so you can get
# what the potential parameter values can be
unique(dataAussen$D0)
unique(dataAussen$D1)
unique(dataAussen$D2)
unique(dataXrates$D1)

startYear = 2000
curr = unique(dataXrates$D1)[1] # A trick to avoid typing variable names
tradeDir = "Ausfuhr"
goodsType = unique(dataAussen$D1)[1]
measure = "Wert in Millionen Franken"
saveGraph = "no"

# NEW!
dataAsChangeRates = "yes"
# This parameter determines whether data should
# be analysed in levels or percentage change rates


########################################


# The function to convert data to percentage
# change rates, like in Lecture 6
toGrowth = function(x){
  out = (tail(x, -1)/head(x, -1) -1 )*100
  out = c(NA, out)
  return(out)
```

```
}

# Select the subset of rows and columns from
# dataAussen and dataXrates that we need,
# assign new name to resulting object.

aussen =
  subset(dataAussen,

         D0 == tradeDir  &
         D1 == goodsType &
         D2 == measure   &
         timeID >= startYear ,

         select = c("timeID", "D0","D1","D2", "Value"))

xrates =
  subset(dataXrates,
         D1 == curr  &
         timeID >= startYear,
         select = c("timeID", "D1","Value"))


# Bring data into wide format
# in case you are missing reshape2, run the next line
#install.packages("reshape2")
library(reshape2)
xrates = dcast(xrates, timeID ~ D1, value.var = "Value")

aussen = dcast(aussen, timeID ~ D0 + D1 + D2, value.var = "Value")

# !!!!! QUITE SOME NEW STUFF FROM HERE ON !!!!!
# make variable name for variable in "aussen" prettier,
# such that we can use it for labels in graphs
# In particular, we do not want the underscores
# and not "in Millionen Franken".

# In case you are missing stringr
# install.packages("stringr")
library(stringr)
  # library to work on character ("string") variables

x = str_locate_all(names(aussen)[2], "_") # find ALL positions of underscores
class(x) # x is a list
x = max(x[[1]])  # the LAST position of an underscore, use [[]]
                 # to extract elements from lists

# cut off the pieces after the last underscore
# ("Wert in Millionen Franken")
newName = substr(names(aussen)[2], start = 1, stop = x-1)

# and replace the remaining underscore(s)
newName = str_replace_all(newName, "_", " - ")
```

```
# Assign the new name to second column in "aussen".

names(aussen)[2] = newName

# NOTE: All this code works for ANY type of export/import variable
# we may want to analyse. It is GENERIC. That's why it's a little
# tedious to write, but it ALWAYS works.
# Compare this to a "manual" specific adjustment without the stringr
# commands. This would only work for ONE SPECIFIC case, not ALL!!!


# Now merge the two data sets
D = merge(xrates, aussen, by = "timeID")
  # D is my shortcut for Data we analyse

# rename "timeID" to something that looks pretty
# in a graph
names(D)[1] = "Zeit"



# convert to rate of change, if required
# by parameter dataAsChangeRates

# For this to work for any of our potential variables
# we need a generic names for the thing in the third column of D.
# In the regression analysis, this will be our y-variable.
yvar = names(D)[3]

# convert to percentage changes
if (dataAsChangeRates == "yes") {
  D[[curr]] = toGrowth(D[[curr]])
  D[[yvar]] = toGrowth(D[[yvar]])
}

# # This is only for inspections of the data
# # for potentially later use
# hist(D[[curr]] )
# max(D[[curr]], na.rm = T )
# which.max(D[[curr]])


# One more piece about the variable names
# to make them useful for labels in graphs.
# If we change the data to percentage changes
# we want to be able to get this in the graphs
# from the labels, otherwise, we may forget
# whether we plotted levels or change rates

addToLab = ""  # initialize an empty character variable

# make it nonempty in case that we convert data to
# percentage changes
```

```
if (dataAsChangeRates == "yes"){
  addToLab = "(Veraenderung in \u0025)"
}
addToLab

# the "\u0025" code is an example of UTF-8 encoding
# See http://www.utf8-chartable.de/unicode-utf8-table.pl?utf8=oct&unicodeinhtml=dec&htmlent=1

# Preparing orderly file names
# for option of saving graphs to file
#######################################

#!!!NEW

# Create a folder inside your working directory,
# call it plots

# We want to have file names that indicate all parameters, such
# as "CHFproEuro_Ausfuhr_Total_WertNom_Veraend"
# If you then want to select a few graphs for
# your document (e.g. thesis), you can quickly identify
# the one you want.
# This code is run ONLY IF saveGraph is "yes"
if (saveGraph=="yes"){
  x = "" # This is becoming the piece that replaces
         # "Wert in Millionen Franken", in case that
         # measure takes on this value.
         # Otherwise, x stays empty (i.e. "")
  if (measure == "Wert in Millionen Franken"){
    x = "WertNom"
    if (dataAsChangeRates == "yes"){
      x = paste0(x, "_Veraend")
      # if data are converted to percentage changes
      # add this to x
    }
  }
  # Now put all the pieces together, separated by
  # underscores, which are practical for the purpose
  # of file names
  fname = paste(curr, tradeDir, goodsType, x, sep = "_")
  # There is still a forward slash. This can be a problem for
  # file names, so replace it with something else
  fname = str_replace(fname, "/", "pro")

  # This is the command that actually initiating the process of
  # writing to a png file
  # note that addition of "plots/", meaning that the file
  # should go into the plot folder. Also not the ".png".
  png(filename = paste0("plots/", fname, ".png"), width = 800, height = 800)
}
#!!! END OF NEW
```

```r
# ANALYSIS
############


# Make a plot

# !!!NEW!!!
par(mar=c(5, 6, 4, 2) + 0.1)
# par() is used to set graphical parameters.
# In this case, our aim is to set the parameters
# of the margain of the graph. We want to have space for 3 lines
# of text on the left side, instead of only one!
# We want to indicate the full characterization of our
# y variable along the y axis.

# mar consists of a numerical vector of the form
# c(bottom, left, top, right) which gives the number of lines
# of margin to be specified on the four sides of the plot.
# The default is c(5, 4, 4, 2) + 0.1.


plot(D[[curr]], D[[yvar]],

        pch = 16, # data points as dots; google "r plot pch"

        cex = .7, # the size of the points

        xlab = paste(curr, addToLab, sep = " "),

        ylab = paste(tradeDir, goodsType, addToLab, sep = "\n"),
          # Note, here we use "\n" as a separater between the
          # arguments of paste. This means a line break!

        col = c("turquoise"), # color of dots

        main = paste0(tradeDir, " (", goodsType, ") \nund ",
                      curr, "-Wechselkurs")
                          # The title of the graph.
                          # Note the line break!

     )  # The closing parenthesis of plot

grid()  # add grid lines, so it is easier to judge the slope of
        # the regression line we will add next.

# Add a regression line
abline(  lm(D[[yvar]]  ~   D[[curr]]
         ), col="red3", lwd = 3)

#!!!NEW
# If we wrote the plot to a png file, we
# have to "close that channel" to return to
# the normal output mode.
```

```
# This is done with the next line of code.
if (saveGraph=="yes") dev.off()
#!!! END NEW


# Now e also run a regression
# explicitly, so we can get the estimated
# parameters
reg = lm(D[[yvar]]~D[[curr]])
    # Save the regression as an object called "reg"
s = summary(reg)
    # This gives us the estimated parameters

class(s) # This is a new type of object

s$coefficients
class(s$coefficients) # This is what we actually want
                      # It's just a matrixm you can
# give it row and column names

rownames(s$coefficients)
colnames(s$coefficients)

# For a nice table in a document, these
# names look ugly. So let's change them

rownames(s$coefficients) = c("Konstante", curr)
colnames(s$coefficients)[1:3] = c("Koeffizient", "Standardfehler",
                                  "t-Wert")

# Some text output for later use, when we get
# all the results automatically
# printed into a pdf document

cat("Die zugehoerige Regressionstabelle sieht wie folgt aus.\n\n")

print(s$coefficients)

# This one looks a bit weird. You are going to see later...
cat("\n\nDas $R^2$ betraegt ", round(s$r.squared, digits = 4), ".", sep = "")

# Since s$coefficients is just a matrix
# we can easily extract the pvalue that tells us
# whether the estimated relationship is statistically significant
pval = s$coefficients[2,4]

# The below looks also weird. Can you guess what it is
# supposed to do?
if (pval<00.05){
  cat("\\textcolor{red}{Die Beziehung ist statistisch signifikant!}")
} else {
  cat("\\textcolor{blue}{Wir haben hier keine statistisch
      signifikante Beziehung.}")
```

```
  }

# And this is the weirdest of all...
# It's kind of a table translation program...
# For later use

# RUN THE NEXT LINE ONLY ONCE ON YOUR DEVICE
#install.packages("xtable")
library(xtable)
options(xtable.comment = FALSE)
tab = xtable(s)
print(tab, type="latex")
```

The second script

In the second script, we wrap all of the previous script into a function. Or, I should say, we wrap almost all of the previous script into a function. We no longer need the parameters at the beginning of the previous script. These become the arguments of the function. See the very end of the below script.

```
################################################
#      Get a function do the analysis!       #
################################################


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")


load("dataForAnalysis.RData")

# Below COMMENTED OUT compared TO Lec7_forClass_Analysis1.R

    # SET PARAMETERS HERE
    ######################
    # unique(dataAussen$D0)
    # unique(dataAussen$D1)
    # unique(dataAussen$D2)
    # unique(dataXrates$D1)
    #
    # startYear = 2000
    # curr = unique(dataXrates$D1)[1]
    # tradeDir = "Ausfuhr"
    # goodsType = "Total"
    # measure = "Wert in Millionen Franken"
    #
    # dataAsChangeRates = "yes"

# BEGIN ADDED TO Lec7_forClass_Analysis1.R !!!!
getAnalysis = function(currency, tradeDirection, typeOfGoods, measure,
                startYear = 2000, dataAsChangeRates = "yes",
```

```
                saveGraph = "no"){

    # To make function variables nice and understandable,
    # but keep variables in function definition code
    # a bit shorter

    curr=currency; tradeDir= tradeDirection;
    goodsType = typeOfGoods

# END ADDED TO Lec7_forClass_Analysis1.R !!!!


    toGrowth = function(x){
      out = (tail(x, -1)/head(x, -1) -1 )*100
      out = c(NA, out)
      return(out)
    }

    # Select the subset of rows and columns from
    # dataAussen and dataXrates that we need,
    # assign new name to resulting object.

    aussen =
      subset(dataAussen,

             D0 == tradeDir  &
               D1 == goodsType &
               D2 == measure   &
               timeID >= startYear ,

             select = c("timeID", "D0","D1","D2", "Value"))

    xrates =
      subset(dataXrates,
             D1 == curr  &
               timeID >= startYear,
             select = c("timeID", "D1","Value"))


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

    aussen = dcast(aussen, timeID ~ D0 + D1 + D2, value.var = "Value")

    # !!!!! QUITE SOME NEW STUFF FROM HERE ON !!!!!
    # make variable name for variable in "aussen" prettier,
    # such that we can use it for labels in graphs
    # In particular, we do not want the underscores
    # and not "in Millionen Franken".

    library(stringr)
    # library to work on character ("string") variables
```

```
x = str_locate_all(names(aussen)[2], "_") # find ALL positions of underscores
class(x) # x is a list
x = max(x[[1]])  # the LAST position of an underscore, use [[]]
# to extract elements from lists

# cut off the pieces after the last underscore
# ("Wert in Millionen Franken")
newName = substr(names(aussen)[2], start = 1, stop = x-1)

# and replace the remaining underscore(s)
newName = str_replace_all(newName, "_", " - ")

# Assign the new name to second column in "aussen".

names(aussen)[2] = newName

# NOTE: All this code works for ANY type of export/import variable
# we may want to analyse. It is GENERIC. That's why it's a little
# tedious to write, but it ALWAYS works.
# Compare this to a "manual" specific adjustment without the stringr
# commands. This would only work for ONE SPECIFIC case, not ALL!!!


# Now merge the two data sets
D = merge(xrates, aussen, by = "timeID")
# D is my shortcut for Data we analyse

# rename "timeID" to something that looks pretty
# in a graph
names(D)[1] = "Zeit"



# convert to rate of change, if required
# by parameter dataAsChangeRates

# For this to work for any of our potential variables
# we need a generic names for the thing in the third column of D.
# In the regression analysis, this will be our y-variable.
yvar = names(D)[3]

# convert to percentage changes
if (dataAsChangeRates == "yes") {
  D[[curr]] = toGrowth(D[[curr]])
  D[[yvar]] = toGrowth(D[[yvar]])
}

# # This is only for inspections of the data
# # for potentially later use
# hist(D[[curr]] )
# max(D[[curr]], na.rm = T )
# which.max(D[[curr]])
```

```
# One more piece about the variable names
# to make them useful for labels in graphs.
# If we change the data to percentage changes
# we want to be able to get this in the graphs
# from the labels, otherwise, we may forget
# whether we plotted levels or change rates

addToLab = ""  # initialize an empty character variable

# make it nonempty in case that we convert data to
# percentage changes
if (dataAsChangeRates == "yes"){
  addToLab = "(Veraenderung in \u0025)"
}
addToLab

# the "\u0025" code is an example of UTF-8 encoding
# See http://www.utf8-chartable.de/unicode-utf8-table.pl?utf8=oct&unicodeinhtml=dec&htmlent=1




# Preparing orderly file names
# for option of saving graphs to file
########################################

# Create a folder inside your working directory,
# call it plots

# We want to have file names that indicate all parameters, such
# as "CHFproEuro_Ausfuhr_Total_WertNom_Veraend"
# If you then want to select a few graphs for
# your document (e.g. thesis), you can quickly identify
# the one you want.
# This code is run ONLY IF saveGraph is "yes"
if (saveGraph=="yes"){
  x = "" # This is becoming the piece that replaces
  # "Wert in Millionen Franken", in case that
  # measure takes on this value.
  # Otherwise, x stays empty (i.e. "")
  if (measure == "Wert in Millionen Franken"){
    x = "WertNom"
    if (dataAsChangeRates == "yes"){
      x = paste0(x, "_Veraend")
      # if data are converted to percentage changes
      # add this to x
    }
  }
  # Now put all the pieces together, separated by
  # underscores, which are practical for the purpose
  # of file names
  fname = paste(curr, tradeDir, goodsType, x, sep = "_")
  # There is still a forward slash. This can be a problem for
```

```
  # file names, so replace it with something else
  fname = str_replace(fname, "/", "pro")

  # This is the command that actually initiating the process of
  # writing to a png file
  # note that addition of "plots/", meaning that the file
  # should go into the plot folder. Also not the ".png".
  png(filename = paste0("plots/", fname, ".png"), width = 800, height = 800)
}




# ANALYSIS
###########


# Make a plot

# !!!NEW!!!
par(mar=c(5, 6, 4, 2) + 0.1)
# par() is used to set graphical parameters.
# In this case, our aim is to set the parameters
# of the margain of the graph. We want to have space for 3 lines
# of text on the left side, instead of only one!
# We want to indicate the full characterization of our
# y variable along the y axis.

# mar consists of a numerical vector of the form
# c(bottom, left, top, right) which gives the number of lines
# of margin to be specified on the four sides of the plot.
# The default is c(5, 4, 4, 2) + 0.1.


plot(D[[curr]], D[[yvar]],

     pch = 16, # data points as dots; google "r plot pch"

     cex = .7, # the size of the points

     xlab = paste(curr, addToLab, sep = " "),

     ylab = paste(tradeDir, goodsType, addToLab, sep = "\n"),
     # Note, here we use "\n" as a separater between the
     # arguments of paste. This means a line break!

     col = c("turquoise"), # color of dots

     main = paste0(tradeDir, " (", goodsType, ") \nund ",
                   curr, "-Wechselkurs")
     # The title of the graph.
     # Note the line break!
```

```
)  # The closing parenthesis of plot

grid()  # add grid lines, so it is easier to judge the slope of
# the regression line we will add next.

# Add a regression line
abline(  lm(D[[yvar]]  ~   D[[curr]]
), col="red3", lwd = 3)

# If we wrote the plot to a png file, we
# have to "close that channel" to return to
# the normal output mode.
# This is done with the next line of code.
if (saveGraph=="yes") dev.off()

# Now e also run a regression
# explicitly, so we can get the estimated
# parameters
reg = lm(D[[yvar]]~D[[curr]])
# Save the regression as an object called "reg"
s = summary(reg)
# This gives us the estimated parameters

class(s) # This is a new type of object

s$coefficients
class(s$coefficients) # This is what we actually want
# It's just a matrixm you can
# give it row and column names

rownames(s$coefficients)
colnames(s$coefficients)

# For a nice table in a document, these
# names look ugly. So let's change them

rownames(s$coefficients) = c("Konstante", curr)
colnames(s$coefficients)[1:3] = c("Koeffizient", "Standardfehler",
                                  "t-Wert")

# Some text output for later use, when we get
# all the results automatically
# printed into a pdf document
cat("Regression table\n\n")

cat("Die zugehoerige Regressionstabelle sieht wie folgt aus.\n\n")

print(s$coefficients)

# This one looks a bit weird. You are going to see later...
cat("\n\nDas $R^2$ betraegt ", round(s$r.squared, digits = 4), ".", sep = "")

# Since s$coefficients is just a matrix
```

```
    # we can easily extract the pvalue that tells us
    # whether the estimated relationship is statistically significant
    pval = s$coefficients[2,4]

    # The below looks also weird. Can you guess what it is
    # supposed to do?
    if (pval<00.05){
      cat("\\textcolor{red}{Die Beziehung ist statistisch signifikant!}")
    } else {
      cat("\\textcolor{blue}{Wir haben hier keine statistisch
          signifikante Beziehung.}")
}

    # And this is the weirdest of all...
    # It's kind of a table translation program...
    # For later use
    library(xtable)
    options(xtable.comment = FALSE)
    tab = xtable(s)
    print(tab, type="latex")



} # closing curly bracket of function definition
  # NEW with respect to Lec7_forClass_Analysis1.R !!!!

# NEW with respect to Lec7_forClass_Analysis1.R:
# Check it out whether function works!
getAnalysis(currency = unique(dataXrates$D1)[2],
            tradeDirection = "Ausfuhr",
            typeOfGoods = "Total",
            measure = "Wert in Millionen Franken"
            )
```

Third: Not a script, but a function definition

Third, we considered a `.R` file that just contains the definition of the function `get.Analysis()`.

```
# This file just contains a copy-cat version of the
# function definition in the previous script!
# Well, a very few things are either removed or rearranged.
# But it's basically a carbon copy!

# Note the name of this .R file!


getAnalysis = function(currency, tradeDirection, typeOfGoods, measure,
                       startYear = 2000, dataAsChangeRates = "yes",
                       saveGraph = "no"){

  # To make function variables nice and understandable,
  # but keep variables in function definition code
```

```r
# a bit shorter

curr=currency; tradeDir= tradeDirection;
goodsType = typeOfGoods

# END ADDED TO Lec7_forClass_Analysis1.R !!!!


toGrowth = function(x){
  out = (tail(x, -1)/head(x, -1) -1 )*100
  out = c(NA, out)
  return(out)
}

# Select the subset of rows and columns from
# dataAussen and dataXrates that we need,
# assign new name to resulting object.

aussen =
  subset(dataAussen,

         D0 == tradeDir  &
           D1 == goodsType &
           D2 == measure   &
           timeID >= startYear ,

         select = c("timeID", "D0","D1","D2", "Value"))

xrates =
  subset(dataXrates,
         D1 == curr  &
           timeID >= startYear,
         select = c("timeID", "D1","Value"))


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

aussen = dcast(aussen, timeID ~ D0 + D1 + D2, value.var = "Value")

# !!!!! QUITE SOME NEW STUFF FROM HERE ON !!!!!
# make variable name for variable in "aussen" prettier,
# such that we can use it for labels in graphs
# In particular, we do not want the underscores
# and not "in Millionen Franken".

library(stringr)
# library to work on character ("string") variables

x = str_locate_all(names(aussen)[2], "_") # find ALL positions of underscores
class(x) # x is a list
x = max(x[[1]])  # the LAST position of an underscore, use [[]]
```

```
# to extract elements from lists

# cut off the pieces after the last underscore
# ("Wert in Millionen Franken")
newName = substr(names(aussen)[2], start = 1, stop = x-1)

# and replace the remaining underscore(s)
newName = str_replace_all(newName, "_", " - ")

# Assign the new name to second column in "aussen".

names(aussen)[2] = newName

# NOTE: All this code works for ANY type of export/import variable
# we may want to analyse. It is GENERIC. That's why it's a little
# tedious to write, but it ALWAYS works.
# Compare this to a "manual" specific adjustment without the stringr
# commands. This would only work for ONE SPECIFIC case, not ALL!!!


# Now merge the two data sets
D = merge(xrates, aussen, by = "timeID")
# D is my shortcut for Data we analyse

# rename "timeID" to something that looks pretty
# in a graph
names(D)[1] = "Zeit"



# convert to rate of change, if required
# by parameter dataAsChangeRates

# For this to work for any of our potential variables
# we need a generic names for the thing in the third column of D.
# In the regression analysis, this will be our y-variable.
yvar = names(D)[3]

# convert to percentage changes
if (dataAsChangeRates == "yes") {
  D[[curr]] = toGrowth(D[[curr]])
  D[[yvar]] = toGrowth(D[[yvar]])
}

# # This is only for inspections of the data
# # for potentially later use
# hist(D[[curr]] )
# max(D[[curr]], na.rm = T )
# which.max(D[[curr]])


# One more piece about the variable names
# to make them useful for labels in graphs.
```

```
# If we change the data to percentage changes
# we want to be able to get this in the graphs
# from the labels, otherwise, we may forget
# whether we plotted levels or change rates

addToLab = ""  # initialize an empty character variable

# make it nonempty in case that we convert data to
# percentage changes
if (dataAsChangeRates == "yes"){
  addToLab = "(Veraenderung in \u0025)"
}
addToLab

# the "\u0025" code is an example of UTF-8 encoding
# See http://www.utf8-chartable.de/unicode-utf8-table.pl?utf8=oct&unicodeinhtml=dec&htmlent=1




# Preparing orderly file names
# for option of saving graphs to file
########################################

# Create a folder inside your working directory,
# call it plots

# We want to have file names that indicate all parameters, such
# as "CHFproEuro_Ausfuhr_Total_WertNom_Veraend"
# If you then want to select a few graphs for
# your document (e.g. thesis), you can quickly identify
# the one you want.
# This code is run ONLY IF saveGraph is "yes"
if (saveGraph=="yes"){
  x = "" # This is becoming the piece that replaces
  # "Wert in Millionen Franken", in case that
  # measure takes on this value.
  # Otherwise, x stays empty (i.e. "")
  if (measure == "Wert in Millionen Franken"){
    x = "WertNom"
    if (dataAsChangeRates == "yes"){
      x = paste0(x, "_Veraend")
      # if data are converted to percentage changes
      # add this to x
    }
  }
  # Now put all the pieces together, separated by
  # underscores, which are practical for the purpose
  # of file names
  fname = paste(curr, tradeDir, goodsType, x, sep = "_")
  # There is still a forward slash. This can be a problem for
  # file names, so replace it with something else
  fname = str_replace(fname, "/", "pro")
```

```r
  # This is the command that actually initiating the process of
  # writing to a png file
  # note that addition of "plots/", meaning that the file
  # should go into the plot folder. Also not the ".png".
  png(filename = paste0("plots/", fname, ".png"), width = 800, height = 800)
}




# ANALYSIS
############


# Make a plot

# !!!NEW!!!
par(mar=c(5, 6, 4, 2) + 0.1)
# par() is used to set graphical parameters.
# In this case, our aim is to set the parameters
# of the margain of the graph. We want to have space for 3 lines
# of text on the left side, instead of only one!
# We want to indicate the full characterization of our
# y variable along the y axis.

# mar consists of a numerical vector of the form
# c(bottom, left, top, right) which gives the number of lines
# of margin to be specified on the four sides of the plot.
# The default is c(5, 4, 4, 2) + 0.1.


plot(D[[curr]], D[[yvar]],

     pch = 16, # data points as dots; google "r plot pch"

     cex = .7, # the size of the points

     xlab = paste(curr, addToLab, sep = " "),

     ylab = paste(tradeDir, goodsType, addToLab, sep = "\n"),
     # Note, here we use "\n" as a separater between the
     # arguments of paste. This means a line break!

     col = c("turquoise"), # color of dots

     main = paste0(tradeDir, " (", goodsType, ") \nund ",
                   curr, "-Wechselkurs")
     # The title of the graph.
     # Note the line break!

)  # The closing parenthesis of plot
```

```r
grid()  # add grid lines, so it is easier to judge the slope of
# the regression line we will add next.

# Add a regression line
abline(  lm(D[[yvar]]  ~   D[[curr]]
), col="red3", lwd = 3)

# If we wrote the plot to a png file, we
# have to "close that channel" to return to
# the normal output mode.
# This is done with the next line of code.
if (saveGraph=="yes") dev.off()

# Now e also run a regression
# explicitly, so we can get the estimated
# parameters
reg = lm(D[[yvar]]~D[[curr]])
# Save the regression as an object called "reg"
s = summary(reg)
# This gives us the estimated parameters

class(s) # This is a new type of object

s$coefficients
class(s$coefficients) # This is what we actually want
# It's just a matrixm you can
# give it row and column names

rownames(s$coefficients)
colnames(s$coefficients)

# For a nice table in a document, these
# names look ugly. So let's change them

rownames(s$coefficients) = c("Konstante", curr)
colnames(s$coefficients)[1:3] = c("Koeffizient", "Standardfehler",
                                  "t-Wert")


cat("Die zugehoerige Regressionstabelle sieht wie folgt aus.\n\n")

library(xtable)
options(xtable.comment = FALSE)
tab = xtable(s)
print(tab, type="latex")

# This one looks a bit weird. You are going to see later...
cat("\n\nDas $R^2$ betraegt ", round(s$r.squared, digits = 4), ".", sep = "")

# Since s$coefficients is just a matrix
# we can easily extract the pvalue that tells us
# whether the estimated relationship is statistically significant
pval = s$coefficients[2,4]
```

```
  # The below looks also weird. Can you guess what it is
  # supposed to do?
  if (pval<00.05){
    cat("\\textcolor{red}{Die Beziehung ist statistisch signifikant!}")
  } else {
    cat("\\textcolor{blue}{Wir haben hier keine statistisch
        signifikante Beziehung.}")
}









  } # closing curly bracket of function definition
```

A fourth script

In our fourth script we call the function `getAnalysis()` to use it inside a loop and produce a large number of graphs.

```
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")

load("dataForAnalysis.RData")

# NEW!!!
# This is how you call a function that you define in a
# separate file!
# It's like you wrote your own "package" and call it here!!!
source("../R_Scripts/getAnalysis.R")

# Note on the path above:
# This is the example of a RELATIVE path

# My working directory is
# "D:/Dropbox/Mac&Surf/Programmierkurs Dropb/Data"
# This is where I have the data. However, my R scripts
# are in a different directory:
# "D:/Dropbox/Mac&Surf/Programmierkurs Dropb/R_Scripts"
# So, starting from the Data directory, I have to go
# one directory up, and then down to R_Scripts.
# This is what "../R_Scripts/getAnalysis.R" means.


getAnalysis(currency = unique(dataXrates$D1)[2],
            tradeDirection = "Ausfuhr",
            typeOfGoods = "Total",
            measure = "Wert in Millionen Franken",
            saveGraph = "yes"
```

```
)

# Now we want to start massproduction of graphs!
# Create a folder inside your working directory,
# call it plots

typeList = unique(dataAussen$D1[dataAussen$D0 == "Ausfuhr"])
# This is a list we want to loop over to get all the
# respective plots
# Note that not all values in dataAussen$D1 are available for
# "Ausfuhr". Some are only available for "Einfuhr". This
# explains the slightly more involved code above.

for (i in typeList){
  getAnalysis(currency = unique(dataXrates$D1)[1],
              tradeDirection = "Ausfuhr",
              typeOfGoods = i,
              measure = "Wert in Millionen Franken",
              saveGraph = "yes"    # TURN THIS ON!!!!
  )
}









# If you are not yet impressed, then lets expand capacity!


currList = unique(dataXrates$D1)


for (i in currList){
  for (j in typeList[2:length(typeList)]){

    currency = i
    tradeDirection = "Ausfuhr"
    typeOfGoods = j

    cat("\\subsection{Reaktion von ", tradeDirection,
        " (", typeOfGoods, ") auf ", currency,
        "-Wechselkurs}", sep = "")

    getAnalysis(currency,
                tradeDirection,
                typeOfGoods,
                measure = "Wert in Millionen Franken",
                saveGraph = "yes"
```

```
    )
  }
}



# The best way to inspect the output is to open a new word document
# and drag the files into that document from the file explorer/finder.


# This is all pretty cool. There is one drawback of this procedure,
# however. We cannot automate any text comments on our results,
# nor automate the printing and commenting of regression results.

# In order to automate documentation further, we need an additional
# tool: RMarkdown!
```