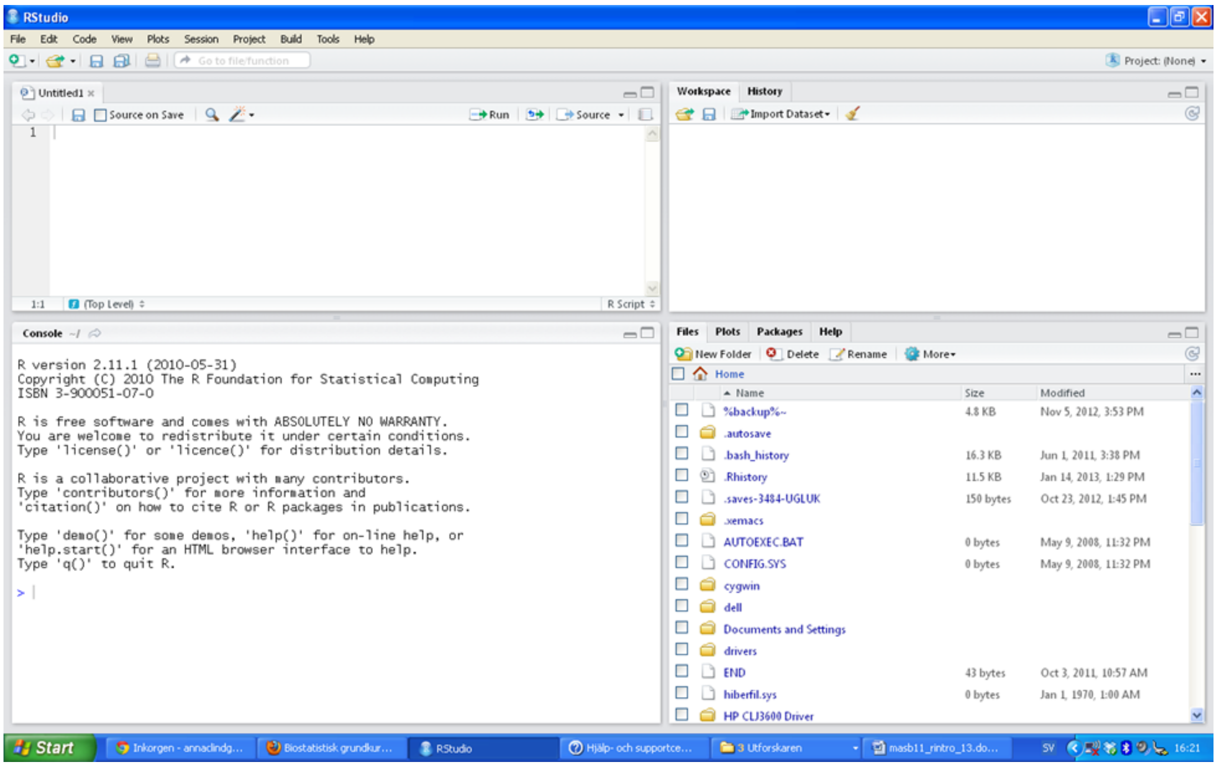
Computer exercise 0: Introduction to R

R Studio, working directory and syntax file

In this short exercise we will become familiar with the basic functions in R. The programme can be downloaded for free from `www.r-project.org`. There is also a very useful user interface, RStudio, which can be downloaded from `www.rstudio.com`.

Install R first and then install RStudio. You should then start RStudio, not R. After a few seconds the screen will look something like the figure below.



```
R version 2.11.1 (2010-05-31)
Copyright (C) 2010 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Name	Size	Modified
%backup%~	4.8 KB	Nov 5, 2012, 3:53 PM
.autosave		
.bash_history	16.3 KB	Jun 1, 2011, 3:38 PM
.Rhistory	11.5 KB	Jan 14, 2013, 1:29 PM
.saves-3484-UGLUK	150 bytes	Oct 23, 2012, 1:45 PM
.xemacs		
AUTOEXEC.BAT	0 bytes	May 9, 2008, 11:32 PM
CONFIG.SYS	0 bytes	May 9, 2008, 11:32 PM
cygwin		
dell		
Documents and Settings		
drivers		
END	43 bytes	Oct 3, 2011, 10:57 AM
hiberfil.sys	0 bytes	Jan 1, 1970, 1:00 AM
HP CLJ3600 Driver		

At the bottom left we have the **console** or **command** window, where you can perform commands directly by writing after the prompt > and pressing `Enter`. This is also where the output of the commands will appear unless they result in new variables, in which case the variables show up in the environment window instead.

At the top right we find the **environment/history** window. Under the environment tab you can see all the objects that R has in its memory. Under the history tab you have a list of all the commands you have given.

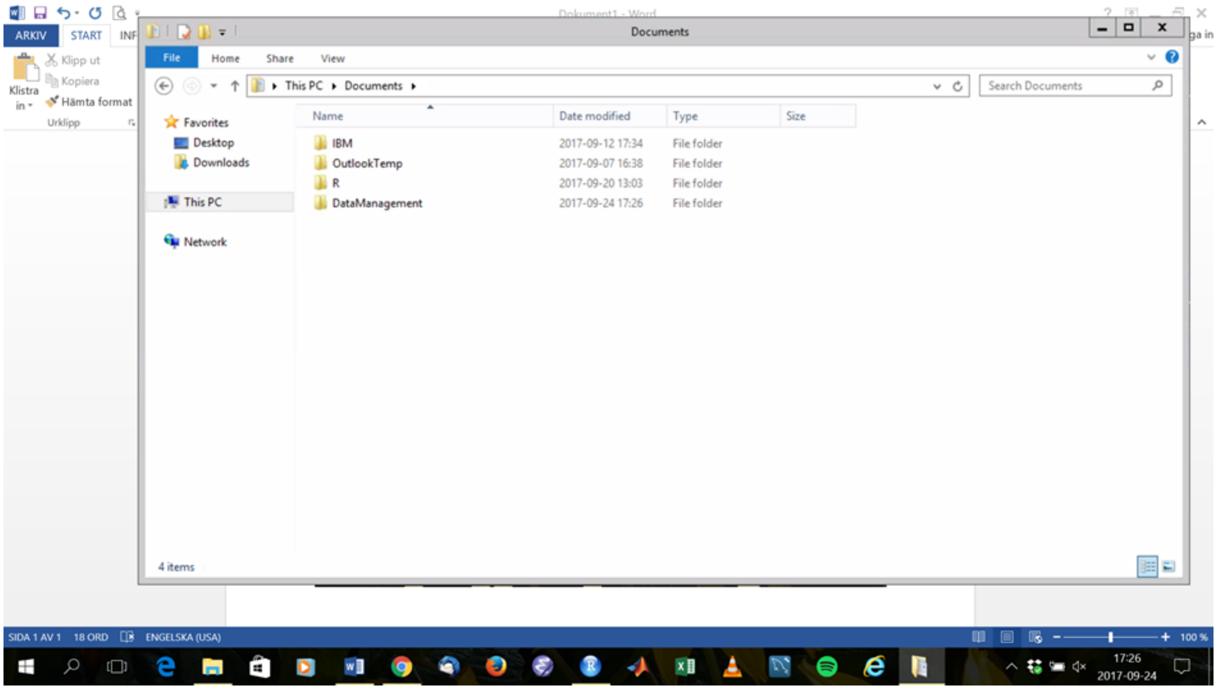
At the bottom right is the **files/plots/packages/help/viewer** window. Here you can browse among your files and see the plots you have made. You can also manage your R packages, i.e., extensions to R. Here you will also find the help function.

At the top left is the **editor** window or **script** window. This is where you edit your programs or scripts. It may be hidden under a large console. You can start a new script by choosing File – New File – R Script and it will show up. The simplest form of script is just a sequence of R commands.

You should save these so that you can reproduce your calculations later. You run the commands here by pressing `Ctrl-R` on each successive line. We will use this extensively on the projects.

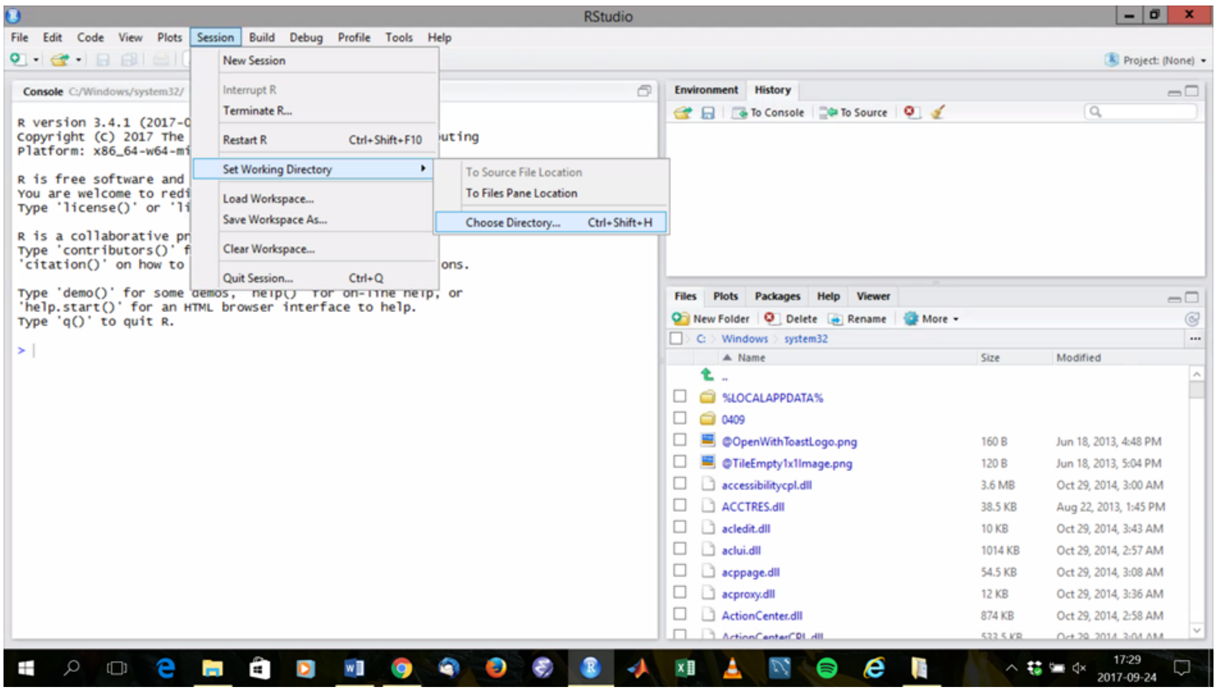
In contrast to spreadsheet based programmes like Excel, we will not see the datasheet unless we ask for it. The focus here is on the commands and the results. Data is, as always, entered so that each row represents one individual and every column represents one variable, e.g. age, HbA1c, etc. If you have a small dataset you can enter the data by hand but it is easier to import larger datasets from some other programme, e.g. Excel or a database program. There are many similarities between R and Matlab, but R is much better at handling data sets, including missing data and string variables. The functions for statistical tests and different types of regression are also much more user friendly in R.

When doing more complicated calculations you should always use a script file (with the extension .R). This works in the same way as an .m-file in Matlab. You should first tell R where to look for your files, and where to save the R-file. Create a folder somewhere on your computer where you will save your R-files and other related material. In this example the folder is called DataManagement:



Name	Date modified	Type	Size
IBM	2017-09-12 17:34	File folder	
OutlookTemp	2017-09-07 16:38	File folder	
R	2017-09-20 13:03	File folder	
DataManagement	2017-09-24 17:26	File folder	

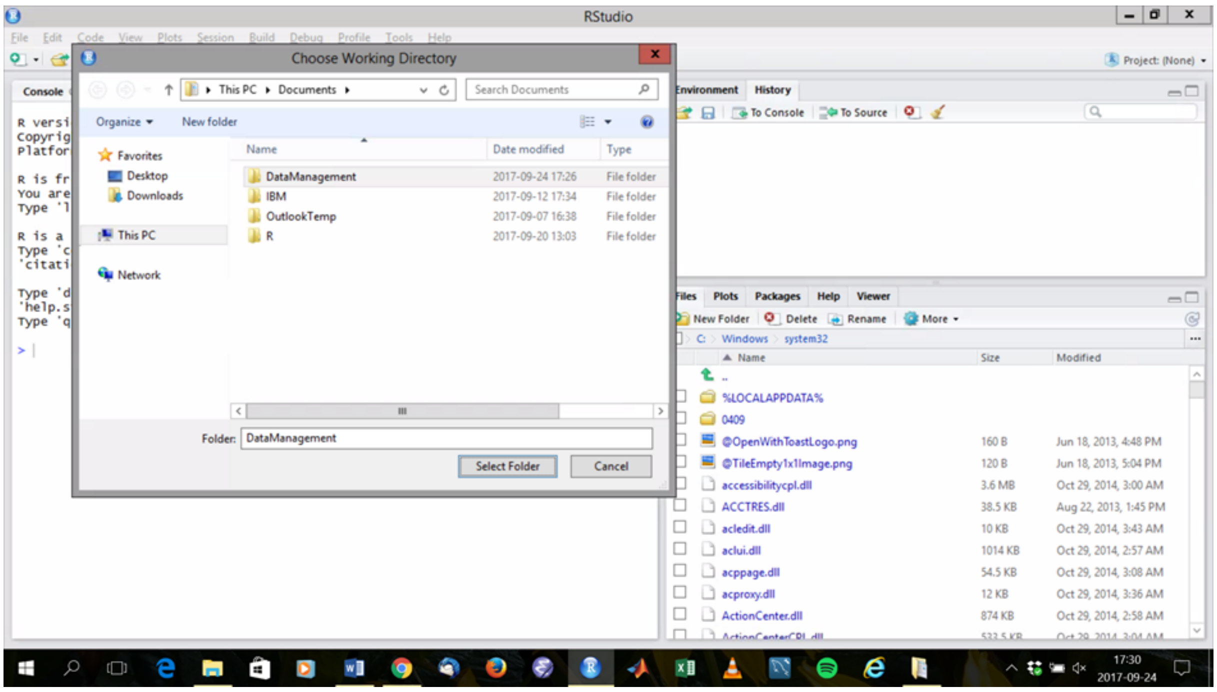
In R Studio, set the working directory:



```
R version 3.4.1 (2017-0
Copyright (C) 2017 The
Platform: x86_64-w64-mi                          uting

R is free software and
You are welcome to redi
Type 'license()' or 'li

R is a collaborative pr
Type 'contributors()' f
'citation()' on how to                           ons.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```

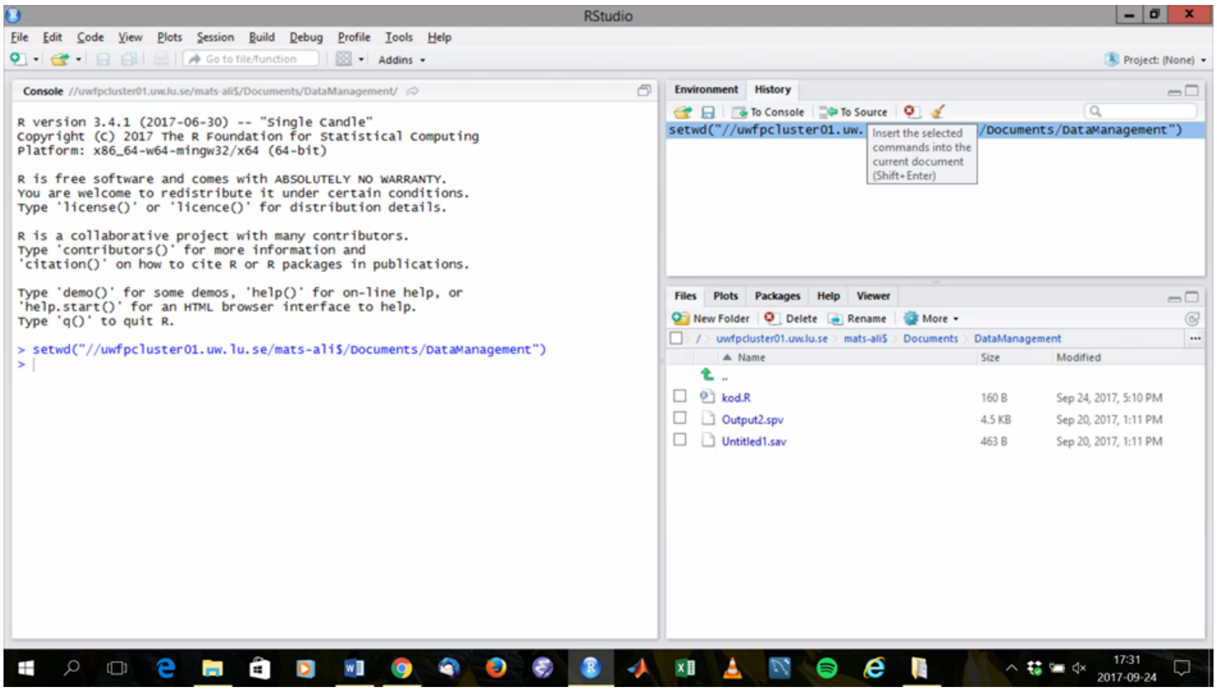
Name	Size	Modified
..		
%LOCALAPPDATA%		
0409		
@OpenWithToastLogo.png	160 B	Jun 18, 2013, 4:48 PM
@TileEmpty1x1Image.png	120 B	Jun 18, 2013, 5:04 PM
accessibilitycpl.dll	3.6 MB	Oct 29, 2014, 3:00 AM
ACCTRES.dll	38.5 KB	Aug 22, 2013, 1:45 PM
acledit.dll	10 KB	Oct 29, 2014, 3:43 AM
aclui.dll	1014 KB	Oct 29, 2014, 2:57 AM
acppage.dll	54.5 KB	Oct 29, 2014, 3:08 AM
acproxy.dll	12 KB	Oct 29, 2014, 3:36 AM
ActionCenter.dll	874 KB	Oct 29, 2014, 2:58 AM

Choose your new folder:



Name	Date modified	Type
DataManagement	2017-09-24 17:26	File folder
IBM	2017-09-12 17:34	File folder
OutlookTemp	2017-09-07 16:38	File folder
R	2017-09-20 13:03	File folder

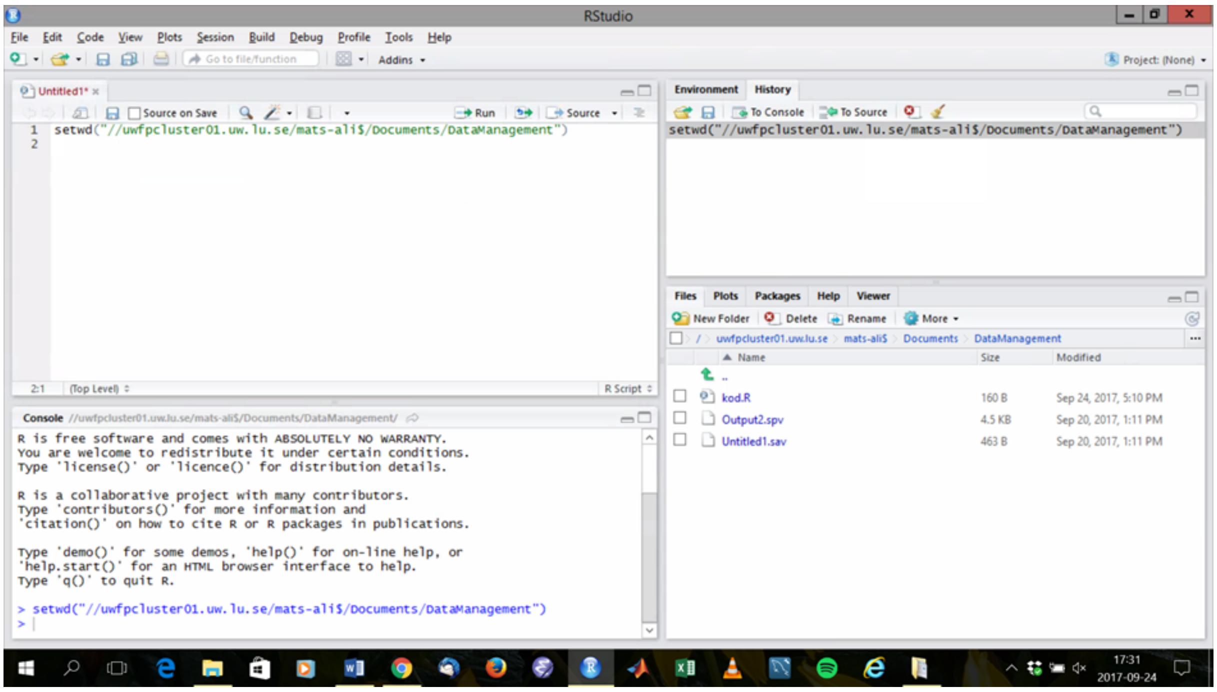
Save the syntax for setting the working directory in a syntax file using the History tab and to Source:


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Environment History
To Console To Source
Insert the selected commands into the current document (Shift+Enter)
Files Plots Packages Help Viewer
New Folder Delete Rename More
uwfpcluster01.uw.lu.se mats-ali$ Documents DataManagement

```
R version 3.4.1 (2017-06-30) -- "Single Candle"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> setwd("//uwfpcluster01.uw.lu.se/mats-ali$/Documents/DataManagement")
>
```

Name	Size	Modified
kod.R	160 B	Sep 24, 2017, 5:10 PM
Output2.spv	4.5 KB	Sep 20, 2017, 1:11 PM
Untitled1.sav	463 B	Sep 20, 2017, 1:11 PM

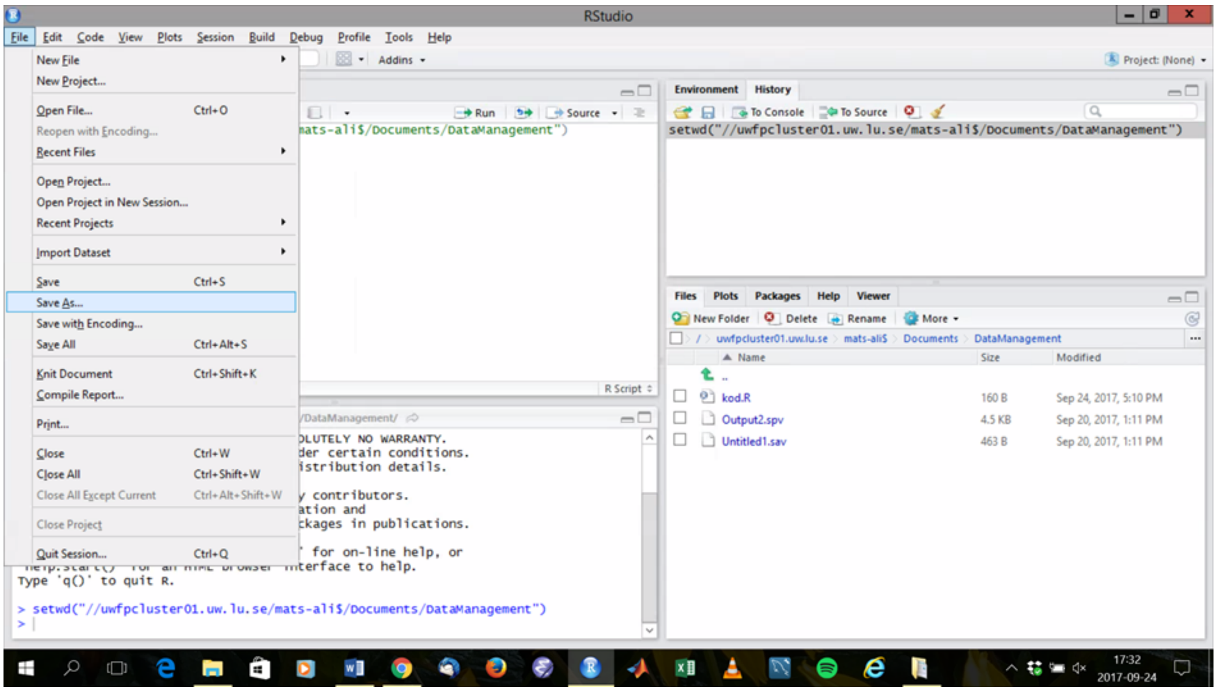
This will create a syntax file and copy the command:


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Untitled1*
Source on Save
Run
Source
Environment History
To Console To Source
Files Plots Packages Help Viewer
New Folder Delete Rename More
uwfpcluster01.uw.lu.se mats-ali$ Documents DataManagement

```
setwd("//uwfpcluster01.uw.lu.se/mats-ali$/Documents/DataManagement")
```

```
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> setwd("//uwfpcluster01.uw.lu.se/mats-ali$/Documents/DataManagement")
>
```

Save the syntax file in your folder:

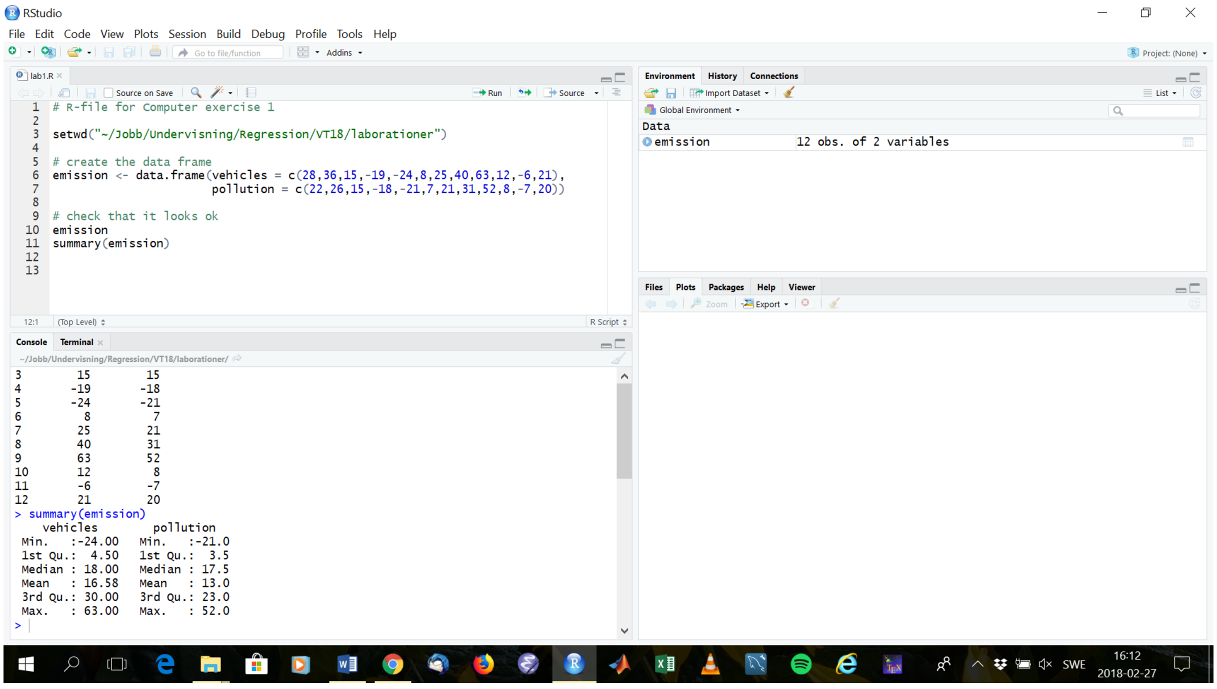


```
setwd("//uwfpcluster01.uw.lu.se/mats-ali$/Documents/DataManagement")
```

```
OLUTELY NO WARRANTY.
der certain conditions.
istribution details.

y contributors.
ation and
ckages in publications.

' for on-line help, or
nterface to help.
Type 'q()' to quit R.

> setwd("//uwfpcluster01.uw.lu.se/mats-ali$/Documents/DataManagement")
> 
```

Continue writing your syntax in this file, saving often! Add comments so that you can remember what it is your code is supposed to do, later.

Run the commands from the syntax file by pressing `Ctrl-R` on each line.



```
# R-file for Computer exercise 1

setwd("~/Jobb/Undervisning/Regression/VT18/laborationer")

# create the data frame
emission <- data.frame(vehicles = c(28,36,15,-19,-24,8,25,40,63,12,-6,21),
                       pollution = c(22,26,15,-18,-21,7,21,31,52,8,-7,20))

# check that it looks ok
emission
summary(emission)
```

```
~/Jobb/Undervisning/Regression/VT18/laborationer/
3        15        15
4       -19       -18
5       -24       -21
6         8         7
7        25        21
8        40        31
9        63        52
10       12         8
11       -6        -7
12       21        20
> summary(emission)
    vehicles         pollution   
 Min.   :-24.00   Min.   :-21.0  
 1st Qu.:  4.50   1st Qu.:  3.5  
 Median : 18.00   Median : 17.5  
 Mean   : 16.58   Mean   : 13.0  
 3rd Qu.: 30.00   3rd Qu.: 23.0  
 Max.   : 63.00   Max.   : 52.0  
> 
```

When you are done, close R Studio (`q()` means Quit). If you are using our computers, you should copy the folder and its contents to a USB stick, or similar, so that you can access them later.

The next time, you can start R Studio by opening your saved R-file.

Basic computing

This is a quick introduction to R. You will note that it has many similarities with Matlab, as well as some confusing differences.

R is made for computing things. If you want to find the result of $2 + 4$ you simply write

```
2+4
```

and R will answer

```
> 2+4
[1] 6
```

The notation "`[1] 6`" means that the first value (in this case the only value) of the answer is 6. If you want to do a multiplication you write

```
2*4
```

All common mathematical functions are available. In order to calculate 42, $\sqrt{4}$, $\ln(4)$ and e^4 you write

```
4^2
sqrt(4)
log(4)
exp(4)
```

Note that R uses decimal period, never decimal comma.

If you want to save the result of a calculation you have to give the result a name. This is done using the notation `<-` (less-than immediately followed by a dash). If we want to calculate $2 + 4$ and save the result under the name `myresult` you give the command

```
myresult <- 2+4
```

A variable named `myresult` should now be listed in the Environment window. You can also see that it contains the value 6. You can ask R to print the answer by writing

```
myresult
```

We can use this variable in other functions. For example, we can write

```
sqrt(myresult)
```

to get the square root of 6. The expression `sqrt()` is a function. All functions in R end in brackets, even if they have no argument, e.g. `getwd()`.

Variables

You can collect several values into one variable, a vector, using the function `c()` (`c` for combine or collect):

```
x <- c(3, 5, 7, 11, 13)
```

You can then perform the same calculation as before but on all the values at the same time:

```
x + 3
sqrt(x)
```

You can also combine several variables into one longer variable:

```
y <- c(17, 19, 23, 29, 31)
z <- c(x, y)
z
```

Sometimes you will want to create structured data, e.g. series or repeated sequences. There are two commands for this: `seq()` and `rep()`. In addition you can use the colon sign :. Try out the following commands and try to understand what they do:

```
seq(1, 100, 9)
seq(to=100, from=1, by=9)
seq(f=1, t=100, length.out=10)
1:3
3:1
rep(c(1,2,3), times=3)
rep(1:3, each=4)
rep(1:3, t=3, e=4)
rep(1:3, length.out=20)
```

If you need help on a particular function you can use the help function by writing `help(seq)` or `?seq`. You can also use the "Help" window in R Studio. The colon sign is not a function but an operator so you have to write `help(":")` using quotes.
Sometimes you only want some of the values in a variable. We can choose values using []:

```
myvalues <- 21:30
myvalues
myvalues[1]
myvalues[c(1, 3, 5)]
myvalues[1:3]
```

You can also choose to exclude values:

```
myvalues[-1]
myvalues[-c(1, 3, 5)]
myvalues[-(2:4)]
```

Some functions

There is a large number of functions in R. Here are some examples of basic statistical functions. The first one creates 100 random numbers from a standard normal distribution. Run help on the others to find out what they do.

```
x <- rnorm(100)
x
mean(x)
var(x)
sd(x)
median(x)
boxplot(x)
boxplot(x, horizontal=TRUE)
hist(x)
```

Objects

All variables in R are objects. You can see the objects you have created in the Environment window. You can also list them using the command `ls()`. If you want to remove an object you use the command `remove()` or, shorter, `rm()`:

```
rubbish <- c(1, 19, 23.4)
ls()
remove(rubbish)
ls()
```

If you want to remove all objects you can combine the two commands into

```
remove(list=ls())
```

Be careful! R will NOT warn you that you are removing anything. It assumes you know what you are doing. Now we have a nice empty environment. Save your script file and close R Studio. You can answer No when asked to save the workspace. Since you saved your script file you can run the commands and recreate then again next time you run R.

Create a vector of zeroes

We can use `numeric` to initialize a vector of a given size. For example

```
> numeric(3)
[1] 0 0 0
```

and afterwards you can fill it with some values. For example here we assign values to vector `x`:

```
> x <- numeric(3)
> x[1] <- 2
> x
[1] 2  0  0
```

If you wish to type all values by hand (you should do this only for very few values), then use `c()` as explained in the Variables section, for example

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

Create a matrix initialised to NAs

`NA` stands for "Not available", and is an important "value" in statistics, where missing data are not uncommon. Here we use it to initialize a matrix of a given size.
For example

```
> matrix(,nrow=2,ncol=3)  # matrix with 2 rows and 3 columns
     [,1] [,2] [,3]
[1,]   NA   NA   NA
[2,]   NA   NA   NA
```

The above creates a matrix of NAs with number of rows specified in `nrows` and number of columns specified in `ncol`. As usual, type `?matrix` for more options.
Say that we call the matrix `mymat`, via `mymat <- matrix(,nrow=2,ncol=3)`, we can then fill it with values, for example

```
> mymat[1,1]<-3  # put a 3 in the [1,1] slot
> mymat
     [,1] [,2] [,3]
[1,]    3   NA   NA
[2,]   NA   NA   NA
```

More typically, you will want to fill-up a matrix using a `for` loop (further below).

Matrix multiplication

While the `*` performs elementwise multiplication, if you want to perform matrix multiplication you use `%*%`.

```
> A <- matrix(c(2,3,-2,1,2,2),3,2)
> B <- matrix(c(2,-2,1,2,3,1),2,3)
> A*B  # this result in an error
Error in A * B : non-conformable arrays
> A%*%B  # this is ok
     [,1] [,2] [,3]
[1,]    2    4    7
[2,]    2    7   11
[3,]   -8    2   -4
```

for loops

A `for` loop in R is represented with

```
for(i in 1:trials){
# write your code here
}
```

The above is just an example, the scope of the `i` counter can be very general. For example, let's fill the first row of the `mymat` matrix we created further above:

```
for(i in 1:3){
   mymat[1,i] <- runif(1)  # generates a uniform random number in (0,1)
}
```

and we obtain

```
> mymat
          [,1]      [,2]     [,3]
[1,] 0.4981466 0.3348102 0.296695
[2,]        NA        NA       NA
```

Notice, if you run the code you will almost certainly obtain different values than the ones I reported. That's because I invoked a "(pseudo) random number" generator to fill-up the matrix first row. See the next section.

Pseudo-random numbers

We have already encountered `runif()` which generates independent (pseudo-)random draws from the uniform distribution on the real interval $(0, 1)$. You can generate more independent draws, say three draws using `runif(3)`. Or generate three draws in the interval (a,b) using `runif(3,a,b)` for some $a < b$. Example

```
> runif(3,2,5)
[1] 2.331082 4.521521 2.953891
```

generates three draws sampled uniformly on the interval $(2, 5)$.

Another popular distribution is the standard Gaussian $N(0, 1)$ (having mean $\mu = 0$ and standard deviation $\sigma = 1$). You can sample n independent draws from $N(0, 1)$ using `rnorm(n)` for some positive integer n.

```
> rnorm(3)
[1]  0.7818592 -0.7767766 -0.6159899
```

Of course it is possible to set arbitrary values for the mean and the standard deviation:

```
> rnorm(3,2,0.1)
[1] 2.004658 1.886961 2.057672
```

the above has generated three draws from $N(\mu, \sigma)$, with $\mu = 2$ and $\sigma = 0.1$. **Important**: a common mistake is to pass the variance σ^2 as third argument to $rnorm(n, \mu, \sigma)$. No! You should pass the standard deviation σ.

Controlling pseudo-random numbers generation

As mentioned in the section devoted to "for loops", every time you execute a code involving pseudo-random numbers generation, you get different values. However, it is sometimes desirable to be able to control that the *stream* of generated random numbers is repeatable, for reproducibility of certain results, or when we wish to compare two different codes and we want them to use the same random numbers, for code debugging purposes etc. Then you can use `set.seed()` and put it at the beginning of the script you want to run. `set.seed` takes an integer value as argument:

```
> set.seed(123)
> runif(3)
[1] 0.2875775 0.7883051 0.4089769
> runif(3)
[1] 0.8830174 0.9404673 0.0455565
> set.seed(123)
> runif(3)
[1] 0.2875775 0.7883051 0.4089769
> runif(3)
[1] 0.8830174 0.9404673 0.0455565
```

Notice I used `set.seed(123)` then generated some uniform random numbers. Then I called again `set.seed(123)` and as you can see I am re-obtaining the same stream of random numbers as in the previous attempts. You can use any integer within `set.seed`, it does not have to be `123`. This number has no specific meaning.

More topics

It does not really make much sense to write a detailed guide to R here. There are endless excellent guides on the web. See for example `http://r.sund.ku.dk/index.html` or `https://www.statmethods.net/`. Just google around for specific topics.