R/RStudio – First Steps : : CHEAT SHEET v1.0

Basic Workflow Tips

How to treat datafiles

	A	B	C	D	E
1	Country	Year	CO2em.pp	Income.pp	HDI
2	Canada	2011	15.6	41600	0.907
3	Denmark	2011	7.28	44400	0.922
4	Nigeria	2011	0.588	5260	0.507
5	South Kor	2011	11.8	31200	0.889
6	Canada	2012	14.8	41800	0.909
7	Denmark	2012	6.49	44300	0.924
8	Nigeria	2012	0.589	5340	0.514
9	South Kor	2012	11.7	31800	0.891
10	Canada	2013	14.7	42300	0.912
11	Denmark	2013	6.83	44600	0.926
12	Nigeria	2013	0.571	5480	0.521

Leave your original data set unchanged. In this way you cannot make any changes that cannot be reversed!
Raw data are read-only!

Instead we write commands/code in R which instructs R to import the data and make any changes and calculations required. This makes your work reproducible!
These files are called script files.

```
## Code for Cheatsheet

#Load packages ----
library(tidyverse)

# Data Upload ----
dat_c <- read_csv("Data/ExData.csv")

# Summarise Data ----
summary(dat_c) # plots basic summary stats
```

NEVER WRITE OVER YOUR ORIGINAL DATAFILE

Save your work and make it reproducible

```
CSCode.R*
Source on Save
## Code for Cheatsheet

setwd("YOUR_DIR_NAME")

# load libraries ----
library(tidyverse,ggplot2)

# Data Upload ----
dat_c <- read_csv("Data/ExData.csv")

names(dat_c)
str(dat_c)
summary(dat_c)
```

The way to work in R is by using scripts. These are files in which you collect all the actions you perform on the data (data cleaning, summary stats, graphing, etc.).

This is a record of your work which you can pass on to others but also makes it easy to re-run the analysis, e.g. after fixing a mistake or after obtaining additional data.

Commenting

```
# load libraries ----
library(tidyverse,ggplot2)

# Data Upload ----
dat_c <- read_csv("Data/ExData.csv")

names(dat_c)    # List of variables
summary(dat_c) # Summary Stats
```

Everything after a "#" is a comment and only used to help you and others understand what the code intends to do.

By creating a comment line ending in "----" you create a "sort of chapter" in your code. Collapse a chapter by clicking on the little triangle on the left of that line.

Test your code as frequently as possible

When you write code make sure that you check every line of code straight after writing it. As you develop your code you will encounter many mistakes and if you wrote many lines of code without testing, then it may be difficult to figure out which line is faulty.

Closing RStudio

As you close RStudio you will be asked whether you want to save the Workspace. On most occasions there is no need.

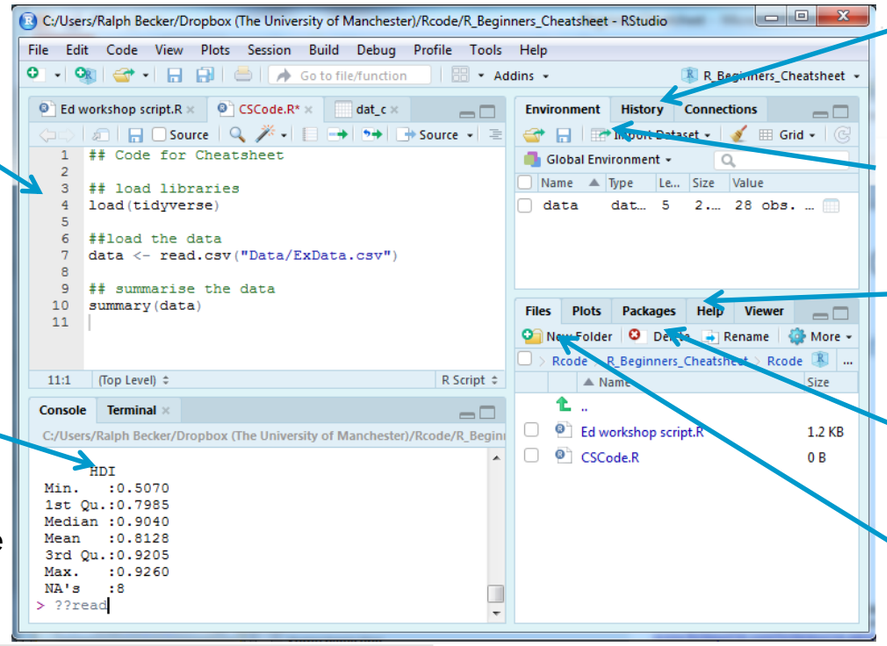
The RStudio Layout

CODE EDITOR

In this window you will see your code/script files.
This is where you record what you want to do.

CONSOLE

Here you will see any output produced by your code.
You can also enter individual commands to either test them before you include them into your script or because you are not planning to include these in your script.



HISTORY

In this tab you can find a history of the commands you used.

ENVIRONMENT

Here you can see all objects/variables that have been loaded or created by your code.

Help

Shows help files if you type "?read.csv" (or any other function name) into the console.

PACKAGES

In this tab you manage (install, update) your packages.

FILES

The files and folders in your working directory appear here.

File Management

Folders

Make sure you know where your files are. It is advisable to create a folder into which you save all files that you use for a particular project. If your project has many files you should consider creating sensible subfolders such as "code", "data", "images", "documentation". This will make you file management much easier.

Working Directory

You should let R know what directory you are working off.

```
# set working directory
setwd("YOUR DIRECTORY")
```

We call this the working directory and you set it with the "setwd" command. Put the full file path inbetween the quotation marks.

RStudio Project files

You should consider using RStudio project files. This will facilitate file management and make references to data and other files more straightforward. In particular you will not have to set the working directory to your current folder as, when you open a project, Rstudio will automatically choses the project folder as its working directory. Hence there is no need to change the working directory if you are working on different computers.
This is particularly useful if you want to share your work with others. But it also greatly facilitates the management folders. Help on how to use them is available from RStudio or Softwarecarpentry.

Useful RStudio Shortcuts

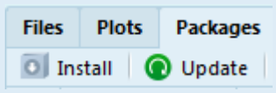
What	Icon	Windows	Mac
Save script		CTRL+S	COMMAND+S
Run entire script	Source	CTRL+ SHIFT+S	COMMAND+ SHIFT+S
Run current line/selection	Run	CTRL+ ENTER	COMMAND+ ENTER
Clear Console		CTRL+L	CPMMAND+L
Undo		CTRL+Z	COMMAND+Z
Re-indent		CTRL+I	COMMAND+I
Clear Workspace			
Piping operator. %>%		CTRL+ SHIFT+M	COMMAND+ SHIFT+M

R/RStudio – First Steps : : CHEAT SHEET v1.0

Packages

R is an open source software which has a lot of functionality build-in. However, a lot of very useful additional functions are provided by extra packages. TO be able to use these you need to:

1. Find the package you need (Google or check out CRAN Task view for starters).
2. The package needs to be installed on the machine you are using. Run `install.packages("PACKAGENAME")` (quotation marks need to be included) or use the interactive function in the packages tab:



Click install and search for package name

3. In the script in which you use a function which is provided by a package you need to load that package by running the following line in your script:
`library(PACKAGENAME)`

Some useful packages

Package	What it does
tidyverse	This is the swiss army-knife for the R-analyst; helps with most data tasks
readxl	This will support the import of Excel spreadsheets
ggplot2	Creates amazing plots
rmarkdown	Allows you to create Rmarkdown documents which produce excellent documents that mix code, output and text.
forecast	Provides a lot of time-series functionality
car	A package with plenty of functions for regression analysis

BOOLEAN VARIABLES

```
> a <- 3
> (a < 4)
[1] TRUE
> (a == 5)
[1] FALSE
> (a == 5)|(a < 4)
[1] TRUE
```

Variables which take TRUE or FALSE as values. You can assign the result to a new variable. These variables have many uses. Combine conditions with "&" (and) or "|" (or).

Basic data management

Analyse the CO2 intensity of per capita income for some countries over a range of years.

Upload packages (assuming that these are installed)

```
library(tidyverse,ggplot2)
library(car)
```

Load data (assumed to be in a "Data" folder)

```
dat_c <- read_csv("../Data/ExData.csv")
```

Check content of datafile

```
names(dat_c)   # lists all variable names
summary(dat_c) # plots basic summary stats
```

Datafile contains these variables

```
"Country" "Year" "CO2emission.pp" "Income.pp" "HDI"
```

Check variable types by clicking on `dat_c` in Environment tab

Drop incomplete observations (you need to think whether this is what you want to do!) and save in `dat_comp`.

```
dat_comp <- dat_c %>% drop_na()
```

Calculate new intensity variable (`mutate`), group data by country (`group_by`) and summarise (`summarise`) the grouped observations with their mean value (`mean`). Ensure that results are ordered from largest to smallest (`arrange`).

```
dat_comp %>%
  mutate(co2int = CO2emission.pp/Income.pp) %>%
  group_by(Country) %>%
  summarise(mean.CO2int=mean(co2int,na.rm = TRUE)) %>%
  arrange(desc(mean.CO2int))
```

	Country	mean.CO2int
	<chr>	<dbl>
1	South Korea	0.000365
2	Canada	0.000357
3	Denmark	0.000149
4	Nigeria	0.000106

We see that income is generated with significantly more CO2 emissions in South Korea and Canada when compared to Denmark and Nigeria

Here we used the piping technology (`%>%`) facilitated by the `tidyverse` package.

Error Management

When writing code you will encounter problems and error messages. This happened to everyone and is a normal part of code writing.

When you encounter an issue do this first:

1. Re-read what you typed, is it 'exactly' what you wanted? (R is case sensitive! Don't use spaces in variable names!)
2. Re-run code one line at a time and identify line with error.
3. Read error message and try to find anything that you can understand. Don't worry about the parts you don't understand – there will be lots!

Common error messages

- 'No such file or directory' - R cannot find the file, Check the file really exists in the specified folder. This could be because it is looking in the wrong place or you have mis-typed. Check `getwd()` to see what R's working directory is. Use `setwd("yourdir")` to change working directory.
- 'Error: object 'name' not found' - R cannot find the object `name`. This could be because you've mistyped, because you need quotes around the name or because you are referring to a variable that does not yet exist.
- 'Could not find function "name"' - R cannot find the function. This could be because you've mistyped or because you haven't used `library()` to load the package containing that function.

Searching for help

Possibly the most important programming skill (and everyone does it!). Either google an error code or a question you have (e.g. "R delete variable from dataframe"). You may have to look at a number of the first links to find useful info. Many links will be from "stackoverflow.com". Posts on here can be very useful. You may need to copy some code and try to adjust it to your problem at hand.

Also don't forget to use the R help function. E.g. type `?lm` into the Console to get help on using the regression function.

Start solving your problem step-by-step, meaning that you try to create the smallest possible problem first before you make your problem more complicated.

REGRESSION

`lm` is the function to run a regression, `summary` prints the results and `lht` can be used to test simple or multiple hypotheses

```
reg1 <- lm(Income.pp~CO2emission.pp,data = dat_c)
summary(reg1)
lht(reg1, "CO2emission.pp = 1000")
```