ECON 172 - Section 2 : Introduction to R

Dennis Egger and Maddie Duhon

9/3/2019

1 Welcome to R

R is a programming language and environment for statistical computing and graphics which is free to use and runs on Windows, Unix and MacOS. To do analysis in R, one can either write one's own commands or take advantage of the huge number of pre-existing packages written by R users and freely available.

1.1 Getting Started

1.1.1 Using R on your local machine (optional)

R can be downloaded from https://cran.rstudio.com/. We also recommend downloading RStudio, a user interface which makes R easier to use. It can be downloaded from https://www.rstudio.com/products/rstudio/download/.

If you run R on your local computer, you will need to install any packages you want to use (just one time), then load them into R's library for the current session. After that, it's easy to access any of the commands in the package (which are also generally extensively documented in a sort of user guide for the package).

1.1.2 Using R on your UC Berkeley's Datahub (recommended)

RStudio also runs on the UC Berkeley Datahub, so you don't need to install it locally on your computer. To start RStudio on Datahub, simply go to: https://r.datahub.berkeley.edu/

In our sections (and for the problem sets), we are going to show you this way of using RStudio. However, anything that we show you should also work on your local machine.

1.2 R Code

When we use R, we want to write and save our commands in a script. This will let us reproduce everything we do without having to retype our commands every time we work on a project. To do this, you can write and save a file as a .R file, which typically contains only code and code comments. You can find a separate .R file containing the code used to conduct the analysis below in the folder with the material for this section.

1.3 RMarkdown

What's even nicer about R is a file format called RMarkdown, such as the one you are currently working in. This file type allows you to combine text, mathematical equations (in a format called LaTex), code as well as dynamic output from this code, and automatically compile / output everything into a pdf document. (In RMarkdown language, this process is called "knitting" the document.)

1.4 Setting up the Environment

First, we need to set up the R environment, including installing all the required packages, loading all the required packages. This is done using R-code. The way to let RMarkdown know that the following output is a 'code chunk' is by enclosing the code chunk in triple quotations, as below. The code chunk indicator is followed by curly brackets, and r (indicating this code chunk is in the R language), a name for the code chunk (here "setup"), and some additional parameters. The parameter "results='hide' " and "message=FALSE" lets RMarkdown know that we will not want to include the output of or any messages generated by this code chunk in the pdf document (but the code will still run).

In your code, you can add comments, using # (beginning of the line) and ## (end of the line). These comments are not run as code, but will help your later self (and others reading your code) to understand what you were doing.

```
knitr::opts_chunk$set(echo = TRUE) ## sets global option to include code chunks in-line
#install.packages("haven") ## only run once to install
library(haven) #this library allows you to load datasets in Stata format
#install.packages("tidyverse") ## only run once to install
library(tidyverse)
#install.packages("summarytools") ## only run once to install
library(summarytools)
#install.packages("stargazer") ## only run once to install
library(stargazer) ##This package is great for making tables in .html, .tex and many other formats.
#install.packages("broom") ## only run once to install
library(broom)
```

The code chunk specifies that, by default, code chunks are part of the pdf output. Next, it installs a few packages. You only need to run this installation once. Afterwards, the package will remain installed on your datahub. (Remove the # before the install.packages commands to install each package for the first time, then "comment out" these lines using #, as above.) Next, the "library" command lets R know that you will want to use these packages in your code.

1.5 Loading Data

The first step in using R for statistical analysis is to store data in what R calls a "data frame," a matrix whose columns have different modes (like numeric, words, etc.) and each row is a unit of observation (i.e: a person or a country, etc.)

R can load data from many types of files, including .csv and .txt. R has its own type of data file, with a .rds or .Rdata extension, and can also load data from Stata, SPSS, and SAS (other statistics computing programs) with the appropriate commands. The .rds format allows for a single object to be saved at a time, like a single dataset. The .RData format allows for multiple objects to be saved at once. Either one is a great way to save your data in R format.

For this section, we will be using data from Robinson, Acemoglu and Johnson (2001). For your convenience, the following link imports this data directly into your Datahub repository: https://r.datahub.berkeley.edu/hub/user-redirect/git-pull?repo=https%3A%2F%2Fgithub.com%2Fdennistegger%2FECON172_Fall2019_SectionMaterial&urlpath=rstudio%2F The folder should now appear as "ECON172_Fall2019_SectionMaterial" on your workspace (bottom right).

When we load a dataframe in R, we give it a name for reference, such as "mydata" or something more specific, like "colonials". We also need to tell R where to find the data. We can do that by specifying a full file path to its exact location of the data. For the purposes of this section, we'll use the dataset "colonials" which is an example dataset that contains data on property rights index, gdp per capita and other variables for countries that were European colonies in the past.

```
## Load in data in csv form, from the folder just loaded into your directory.
## Important: Paths are always relative to the RMarkdown file location
##            Alternatively, specify your working directory using
##            knitr::opts_chunk$set(root.dir = "xxx")
colonials <- read.csv("colonials.csv")
```

Next we can visualize the database we just imported, and take a look at what variables are included. To do so, it's easiest to click directly on the dataset in your Environment pane on the top right in RStudio.

```
## Let's take a look on the dataset.
colonials ## or print first 10 observations
## Let's take a look the variables in this dataset.
```

```
names(colonials) ## just print
colonials_variables <- names(colonials) ## store list, assigned "colonials_variables"
colonials_variables ## take a look at this list
```

2 Analysis in R

2.1 Summary Statistics

Once a dataframe is loaded in R, it's very easy to run basic summary statistics. Unlike many other statistics programs, R provides fairly minimal output, and it's possible to store the results of a command without ever displaying them. For example, we may be interested in summary statistics for the GDP per capita of countries in the sample, which we can get by typing summary(colonials$gdppc). This command instructs R to look in the dataframe listed before the $ symbol (in this case, colonials), and to summarize the variable that follows the $ symbol (in this case, gdppc). Note, the "results = 'markup'" bit tells RMarkdown to display the output of the code exactly as displayed in the R console.

```
## Let's look at some summary statistics.
## Here is the min, 25%-ile, median, mean, 75%-ile, and max for the variable gdppc
summary(colonials$gdppc) ## just print
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     450    1480    2835    5445    6968   27330
```

```
colonials_sumstats <- summary(colonials$gdppc) ## store summary, "colonials_variables"
colonials_sumstats ## take a look at this summary
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     450    1480    2835    5445    6968   27330
```

Or, we may be interested specifically in the **mean** value of GDP per capita for countries in the sample specifically, which we can get by typing mean(colonials$gdppc). As before, this command instructs R to look in the colonials dataframe and calculate the mean for the variable gdppc. We can store value resulting from this calculation, by using the <- symbol to store the results in a new object called meangdp, which can be by typing meangdp into the console, or used in another command or context later on.

```
## Let's look specifically at mean GDP
mean(colonials$gdppc) ## just print
```

```
## [1] 5445.458
```

```
meangdp <- mean(colonials$gdppc) ## save as a variable, called "meangdp"
meangdp ## take a look at the value stored in of "meangdp"
```

```
## [1] 5445.458
```

Similar commands exist for standard deviation (**sd**), variance (**var**), minimum (**min**), maximum (**max**), median (**med**), range (**range**) and quantile (**quantile**).

2.2 Regression

Linear regression is also very straightforward to apply in R. The lm command, which comes preloaded, can be used for univariate or multivariate regression:

$$Y_i = \alpha + \beta X_{1,i} + \gamma X_{2,i} + ... + e_i$$

(Note: The above mathematical expression was written in a typesetting environment called LaTex, and can easily be included in RMarkdown. For examples, see the Script.) Examples of how to run a regression of log GDP per capita on protection against expropriation and latitude are below:

```
## Univariate regression of log gdp per capita on property rights index.
lm(logGDP ~ protection, data=colonials) # display regression results
```

```
##
## Call:
## lm(formula = logGDP ~ protection, data = colonials)
##
## Coefficients:
## (Intercept)   protection
##      4.6604       0.5221
```

```
## Multivariate, adding absolute latitude.
lm(logGDP ~ protection + lat_abst, data=colonials) # display regression results
```

```
##
## Call:
## lm(formula = logGDP ~ protection + lat_abst, data = colonials)
##
## Coefficients:
## (Intercept)   protection     lat_abst
##      4.7281       0.4679       1.5769
```

```
## Storing results
reg1 <- lm(logGDP ~ protection, data=colonials) # display regression results
reg2 <- lm(logGDP ~ protection + lat_abst, data=colonials) # display regression results
```

In the last example, we store the results in the object reg1. If you try this in R, you'll notice the results don't display on the screen when you store the results, but you can access them by naming the object.

```
## Now access regression results by:
summary(reg1) # access complete regression results
```

```
##
## Call:
## lm(formula = logGDP ~ protection, data = colonials)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.8715 -0.4644  0.1683  0.4610  1.1413
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.66038    0.40851  11.408  < 2e-16 ***
## protection   0.52211    0.06119   8.533 4.72e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7132 on 62 degrees of freedom
## Multiple R-squared:  0.5401, Adjusted R-squared:  0.5327
## F-statistic: 72.82 on 1 and 62 DF,  p-value: 4.724e-12
```

2.3 Tables

There are many packages in R that let one create very nice tables. We recommend stargazer. Take the last regression we ran, stored as reg1 and say we want to export that to a table to be used in word or latex.

To include the final table in the knitted pdf created by RMarkdown, you can use the following template.

(Note, we want to use results=‘asis’ and header=FALSE, in order for RMarkdown to compile the LaTex table into a pretty format when knitting the pdf):

```r
stargazer(reg1, reg2,
          out="Table 1",type="latex",header=FALSE,  table.placement = "!h",
          title="Property Rights and Development",align=TRUE,
          report = "vc*st", omit.stat=c("LL","ser","f","rsq","adj.rsq"),no.space=TRUE)
```

Table 1: Property Rights and Development

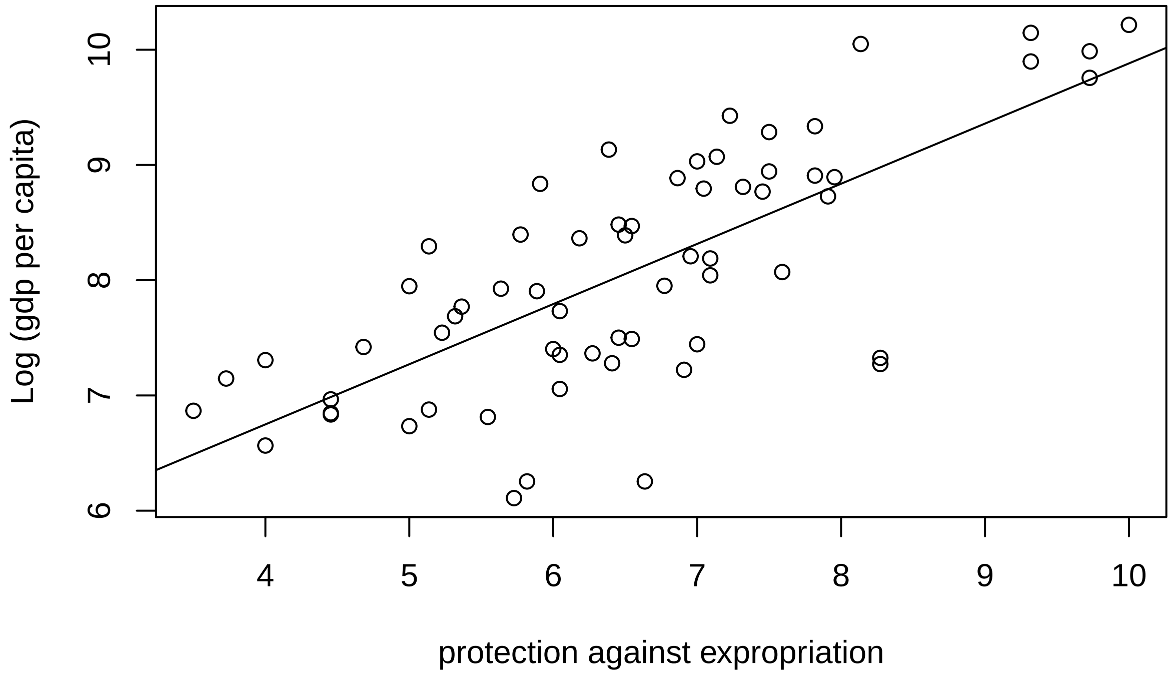
	Dependent variable:	
	logGDP	
	(1)	(2)
protection	0.522^{***}	0.468^{***}
	(0.061)	(0.064)
	$t = 8.533$	$t = 7.292$
lat_abst		1.577^{**}
		(0.710)
		$t = 2.220$
Constant	4.660^{***}	4.728^{***}
	(0.409)	(0.397)
	$t = 11.408$	$t = 11.900$
Observations	64	64
Note:	*p<0.1; **p<0.05; ***p<0.01	

The above command combines regression output saved in reg1 and reg2 into one table. Stargazer lets you specify title, alignment of coefficients inside columns and rows, which statistics to report (here, the variable names “v”, coefficients “c”, significance stars "*“, standard errors”s" and t-statistics “t”). Moreover, you can tell it to omit statistics such as the log-likelihood “LL”, the R2 “rsq”, etc. You can also set the table placement in the final pdf (here, table.placement=“H” means that the table will be placed directly after the code chunk). Stargazer has extensive documentation (try ??stargazer in R) and can also be used to make great summary statistics tables - try it out!

2.4 Plotting

R also makes it easy to do basic plots. For example, one might want to create a scatterplot from two variables, and graph the regression line - or line of best fit - on the plot. This is easy to do with the plot command, which as a default plots the first variable listed on the horizontal axis, and the second on the vertical axis. Below are two ways of plotting protection against expropriation index and log of GDP per capita, one very simple and one with titles and labels. The second version also includes the abline command to add a regression line to the graph.

```r
## A scatterplot with titles
reg2 <- lm(logGDP ~ protection , data=colonials)
plot(colonials$protection,colonials$logGDP,
     xlab="protection against expropriation", ylab="Log (gdp per capita)")
abline(lm(logGDP ~ protection , data=colonials)) # line of best fit
```



```
## Remember that abline adds to the plot the regression line (linear fit line)
```

3 Resources for further study

- UCLA has a number of excellent resources to help you learn more about how to use R, available at: http://www.ats.ucla.edu/stat/r/.
- You can also find interactive lessons at Try R http://tryr.codeschool.com/ for practice writing code.
- We also recommend the text on the syllabus, Hanck, Christoph, Martin Arnold, Alexander Gerber and Martin Schmelzer. (2018). Introduction to Econometrics with R, https://www.econometrics-with-r.org/.
- You may also find https://www.r-bloggers.com/ to be a useful resource for specific questions.
- Finally, we include two useful resources that summarize useful R commands and common RMarkdown features.

Base R
Cheat Sheet

Getting Help

Accessing the help files

?mean
Get help of a particular function.

help.search('weighted mean')
Search the help files for a word or phrase.

help(package = 'dplyr')
Find help for a package.

More about an object

str(iris)
Get a summary of an object's structure.

class(iris)
Find the class an object belongs to.

Using Libraries

install.packages('dplyr')
Download and install a package from CRAN.

library(dplyr)
Load the package into the session, making all its functions available to use.

dplyr::select
Use a particular function from a package.

data(iris)
Load a built-in dataset into the environment.

Working Directory

getwd()
Find the current working directory (where inputs are found and outputs are sent).

setwd('C://file/path')
Change the current working directory.

Use projects in RStudio to set the working directory to the folder you are working in.

Vectors

Creating Vectors

`c(2, 4, 6)`	2 4 6	Join elements into a vector
`2:6`	2 3 4 5 6	An integer sequence
`seq(2, 3, by=0.5)`	2.0 2.5 3.0	A complex sequence
`rep(1:2, times=3)`	1 2 1 2 1 2	Repeat a vector
`rep(1:2, each=3)`	1 1 1 2 2 2	Repeat elements of a vector

Vector Functions

sort(x)
Return x sorted.

table(x)
See counts of values.

rev(x)
Return x reversed.

unique(x)
See unique values.

Selecting Vector Elements

By Position

`x[4]`	The fourth element.
`x[-4]`	All but the fourth.
`x[2:4]`	Elements two to four.
`x[-(2:4)]`	All elements except two to four.
`x[c(1, 5)]`	Elements one and five.

By Value

`x[x == 10]`	Elements which are equal to 10.
`x[x < 0]`	All elements less than zero.
`x[x %in% c(1, 2, 5)]`	Elements in the set 1, 2, 5.

Named Vectors

`x['apple']`	Element with name 'apple'.

Programming

For Loop

```
for (variable in sequence){

   Do something

}
```

Example

```
for (i in 1:4){

   j <- i + 10

   print(j)

}
```

While Loop

```
while (condition){

   Do something

}
```

Example

```
while (i < 5){

   print(i)

   i <- i + 1

}
```

If Statements

```
if (condition){
   Do something
} else {
   Do something different
}
```

Example

```
if (i > 3){
   print('Yes')
} else {
   print('No')
}
```

Functions

```
function_name <- function(var){

   Do something

   return(new_variable)
}
```

Example

```
square <- function(x){

   squared <- x*x

   return(squared)
}
```

Reading and Writing Data

Input	Ouput	Description
`df <- read.table('file.txt')`	`write.table(df, 'file.txt')`	Read and write a delimited text file.
`df <- read.csv('file.csv')`	`write.csv(df, 'file.csv')`	Read and write a comma separated value file. This is a special case of read.table/write.table.
`load('file.RData')`	`save(df, file = 'file.Rdata')`	Read and write an R data file, a file type special for R.

Conditions								
	`a == b`	Are equal	`a > b`	Greater than	`a >= b`	Greater than or equal to	`is.na(a)`	Is missing
	`a != b`	Not equal	`a < b`	Less than	`a <= b`	Less than or equal to	`is.null(a)`	Is null

RStudio® is a trademark of RStudio, Inc. • CC BY Mhairi McNeill • mhairihmcneill@gmail.com

Learn more at **web page or vignette** • package version • Updated: 3/15

Types

Converting between common data types in R. Can always go from a higher value in the table to a lower value.

`as.logical`	`TRUE, FALSE, TRUE`	Boolean values (TRUE or FALSE).
`as.numeric`	`1, 0, 1`	Integers or floating point numbers.
`as.character`	`'1', '0', '1'`	Character strings. Generally preferred to factors.

`as.factor`	`'1', '0', '1',`
	`levels: '1', '0'`
		Character strings with preset levels. Needed for some statistical models.

Maths Functions

`log(x)`	Natural log.	`sum(x)`	Sum.
`exp(x)`	Exponential.	`mean(x)`	Mean.
`max(x)`	Largest element.	`median(x)`	Median.
`min(x)`	Smallest element.	`quantile(x)`	Percentage quantiles.
`round(x, n)`	Round to n decimal places.	`rank(x)`	Rank of elements.
`signif(x, n)`	Round to n significant figures.	`var(x)`	The variance.
`cor(x, y)`	Correlation.	`sd(x)`	The standard deviation.

Variable Assignment

```
> a <- 'apple'
> a
[1] 'apple'
```

The Environment

`ls()`	List all variables in the environment.
`rm(x)`	Remove x from the environment.
`rm(list = ls())`	Remove all variables from the environment.

You can use the environment panel in RStudio to browse variables in your environment.

Matrixes

`m <- matrix(x, nrow = 3, ncol = 3)`
Create a matrix from x.

`m[2, ]` - Select a row

`m[ , 1]` - Select a column

`m[2, 3]` - Select an element

`t(m)`
Transpose

`m %*% n`
Matrix Multiplication

`solve(m, n)`
Find x in: m * x = n

Lists

`l <- list(x = 1:5, y = c('a', 'b'))`
A list is collection of elements which can be of different types.

`l[[2]]`	`l[1]`	`l$x`	`l['y']`
Second element of l.	New list with only the first element.	Element named x.	New list with only element named y.

Also see the **dplyr** library.

Data Frames

`df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))`
A special case of a list where all elements are the same length.

x	y
1	a
2	b
3	c

List subsetting

`df$x` `df[[2]]`

Understanding a data frame

`View(df)` See the full data frame.

`head(df)` See the first 6 rows.

Matrix subsetting

`df[ , 2]`

`df[2, ]`

`df[2, 2]`

`nrow(df)`
Number of rows.

`ncol(df)`
Number of columns.

`dim(df)`
Number of columns and rows.

`cbind` - Bind columns.

`rbind` - Bind rows.

Strings

Also see the **stringr** library.

`paste(x, y, sep = ' ')`	Join multiple vectors together.
`paste(x, collapse = ' ')`	Join elements of a vector together.
`grep(pattern, x)`	Find regular expression matches in x.
`gsub(pattern, replace, x)`	Replace matches in x with a string.
`toupper(x)`	Convert to uppercase.
`tolower(x)`	Convert to lowercase.
`nchar(x)`	Number of characters in a string.

Factors

`factor(x)`
Turn a vector into a factor. Can set the levels of the factor and the order.

`cut(x, breaks = 4)`
Turn a numeric vector into a factor but 'cutting' into sections.

Statistics

`lm(x ~ y, data=df)`
Linear model.

`glm(x ~ y, data=df)`
Generalised linear model.

`summary`
Get more detailed information out a model.

`t.test(x, y)`
Preform a t-test for difference between means.

`pairwise.t.test`
Preform a t-test for paired data.

`prop.test`
Test for a difference between proportions.

`aov`
Analysis of variance.

Distributions

	Random Variates	Density Function	Cumulative Distribution	Quantile
Normal	`rnorm`	`dnorm`	`pnorm`	`qnorm`
Poison	`rpois`	`dpois`	`ppois`	`qpois`
Binomial	`rbinom`	`dbinom`	`pbinom`	`qbinom`
Uniform	`runif`	`dunif`	`punif`	`qunif`

Plotting

Also see the **ggplot2** library.

`plot(x)`
Values of x in order.

`plot(x, y)`
Values of x against y.

`hist(x)`
Histogram of x.

Dates

See the **lubridate** library.

R Markdown Cheat Sheet

learn more at rmarkdown.rstudio.com

RStudio

.Rmd files

An R Markdown (.Rmd) file is a record of your research. It contains the code that a scientist needs to reproduce your work along with the narration that a reader needs to understand your work.

Reproducible Research

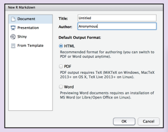
At the click of a button, or the type of a command, you can rerun the code in an R Markdown file to reproduce your work and export the results as a finished report.

Dynamic Documents

You can choose to export the finished report as a html, pdf, MS Word, ODT, RTF, or markdown document; or as a html or pdf based slide show.

Workflow

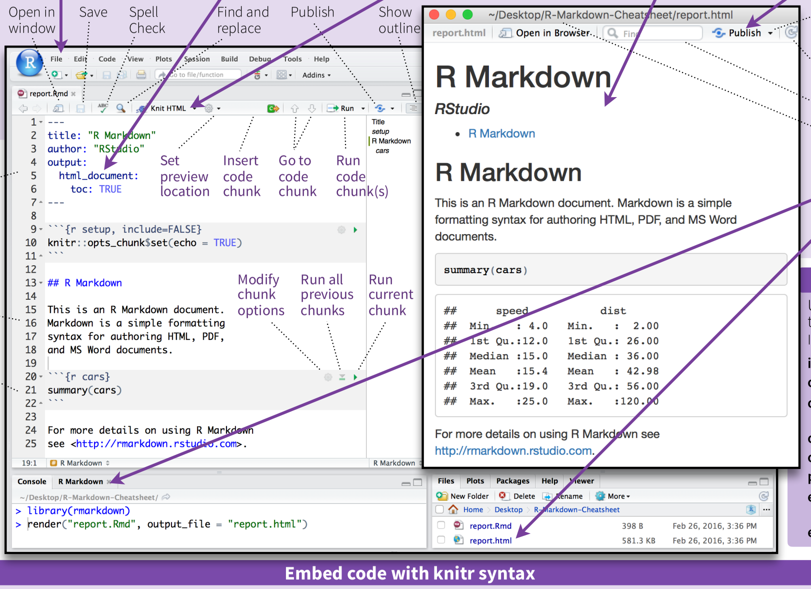
1 Open a new .Rmd file at File ▸ New File ▸ R Markdown. Use the wizard that opens to pre-populate the file with a template



2 Write document by editing template

3 Knit document to create report Use knit button or **render()** to knit

4 Preview Output in IDE window



~~~
---
title: "R Markdown"
author: "RStudio"
output:
  html_document:
    toc: TRUE
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
~~~

R Markdown

This is an R Markdown document.
Markdown is a simple formatting
syntax for authoring HTML, PDF,
and MS Word documents.

```{r cars}
summary(cars)
```

For more details on using R Markdown
see <http://rmarkdown.rstudio.com>.
```

19:1 R Markdown

~/Desktop/R-Markdown-Cheatsheet/

```
> library(rmarkdown)
> render("report.Rmd", output_file = "report.html")
```

~/Desktop/R-Markdown-Cheatsheet/report.html

R Markdown

RStudio

- R Markdown

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents.

```
summary(cars)
```

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00  
```

For more details on using R Markdown see http://rmarkdown.rstudio.com.

report.Rmd	398 B	Feb 26, 2016, 3:36 PM
report.html	581.3 KB	Feb 26, 2016, 3:36 PM

5 Publish (optional) to web or server

- Synch publish button to accounts at
  - rpubs.com,
  - shinyapps.io
  - RStudio Connect
- Reload document
- Find in document
- File path to output document

6 Examine build log in R Markdown console

7 Use output file that is saved alongside .Rmd

.Rmd structure

YAML Header
Optional section of render (e.g. pandoc) options written as key:value pairs (YAML).

- At start of file
- Between lines of - - -

Text
Narration formatted with markdown, mixed with:

Code chunks
Chunks of embedded code. Each chunk:

- Begins with ```{r}
- ends with ```

R Markdown will run the code and append the results to the doc.

It will use the location of the .Rmd file as the **working directory**

render()

Use rmarkdown::**render()** to render/knit at cmd line. Important args:

input - file to render

output_format

output_options - List of render options (as in YAML)

output_file

output_dir

params - list of params to use

envir - environment to evaluate code chunks in

encoding - of input file

Interactive Documents

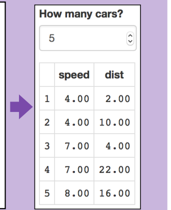
Turn your report into an interactive Shiny document in 4 steps

1. Add **runtime: shiny** to the YAML header.
2. Call Shiny **input** functions to embed input objects.
3. Call Shiny **render** functions to embed reactive output.
4. Render with rmarkdown::**run** or click **Run Document** in RStudio IDE

```
---
output: html_document
runtime: shiny
---

```{r, echo = FALSE}
numericInput("n",
  "How many cars?", 5)

renderTable({
  head(cars, input$n)
})
```
```


How many cars?

5

	speed	dist
1	4.00	2.00
2	4.00	10.00
3	7.00	4.00
4	7.00	22.00
5	8.00	16.00

Embed a complete app into your document with shiny::shinyAppDir()

* *Your report will rendered as a Shiny app, which means you must choose an html output format, like* ***html_document****, and serve it with an active R Session.*

Embed code with knitr syntax

Inline code

Insert with \`r <code>\`. Results appear as text without code.

```
Built with
`r getRversion()`
```

→ Built with 3.2.3

Code chunks

One or more lines surrounded with ```{r} and ```. Place chunk options within curly braces, after r. Insert with

```
```{r echo=TRUE}
getRversion()
```
```

→

```
getRversion()
## [1] '3.2.3'
```

Global options

Set with knitr::**opts_chunk$set()**, e.g.

```
```{r include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```
```

Important chunk options

cache - cache results for future knits (default = FALSE)

cache.path - directory to save cached results in (default = "cache/")

child - file(s) to knit and then include (default = NULL)

collapse - collapse all output into single block (default = FALSE)

comment - prefix for each line of results (default = '##')

dependson - chunk dependencies for caching (default = NULL)

echo - Display code in output document (default = TRUE)

engine - code language used in chunk (default = 'R')

error - Display error messages in doc (TRUE) or stop render when errors occur (FALSE) (default = FALSE)

eval - Run code in chunk (default = TRUE)

fig.align - 'left', 'right', or 'center' (default = 'default')

fig.cap - figure caption as character string (default = NULL)

fig.height, fig.width - Dimensions of plots in inches

highlight - highlight source code (default = TRUE)

include - Include chunk in doc after running (default = TRUE)

message - display code messages in document (default = TRUE)

results (default = 'markup')
'asis' - passthrough results
'hide' - do not display results
'hold' - put all results below all code

tidy - tidy code for display (default = FALSE)

warning - display code warnings in document (default = TRUE)

Options not listed above: R.options, aniopts, autodep, background, cache.comments, cache.lazy, cache.rebuild, cache.vars, dev, dev.args, dpi, engine.opts, engine.path, fig.asp, fig.env, fig.ext, fig.keep, fig.lp, fig.path, fig.pos, fig.process, fig.retina, fig.scap, fig.show, fig.showtext, fig.subcap, interval, out.extra, out.height, out.width, prompt, purl, ref.label, render, size, split, tidy.opts

Parameters

Parameterize your documents to reuse with different inputs (e.g., data sets, values, etc.)

1 Add parameters
Create and set parameters in the header as sub-values of **params**

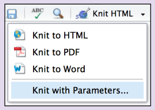
```
---
params:
  n: 100
  d: !r Sys.Date()
---
```

Indent 2 spaces

2 Call parameters
Call parameter values in code as **params$<name>**

```
Today's date
is `r params$d`
```

3 Set parameters
Set values wth **Knit with parameters** or the params argument of render():



render("doc.Rmd",
  params = list(n = 1, d = as.Date("2015-01-01"))
```

## Pandoc's Markdown

Write with syntax on the left to create effect on right (after render)

```
Plain text
End a line with two spaces
to start a new paragraph.
*italics* and **bold**
`verbatim code`
sub/superscript^2^~2~
~~strikethrough~~
escaped: \* \_ \\
endash: --, emdash: ---
equation: $A = \pi*r^{2}$
equation block:

$$E = mc^{2}$$

> block quote

# Header1  {#anchor}

## Header 2  {#css_id}

### Header 3  {.css_class}

#### Header 4

##### Header 5

###### Header 6

<!--Text comment-->

\textbf{Tex ignored in HTML}
<em>HTML ignored in pdfs</em>

<http://www.rstudio.com>
[link](www.rstudio.com)
Jump to [Header 1](#anchor)
image:

![Caption](smallorb.png)

* unordered list
    + sub-item 1
    + sub-item 2
        - sub-sub-item 1

* item 2

    Continued (indent 4 spaces)

1. ordered list
2. item 2
       i) sub-item 1
          A.  sub-sub-item 1

(@)  A list whose numbering

continues after

(@)  an interruption

Term 1

:   Definition 1

| Right | Left | Default | Center |
|------:|:-----|---------|:------:|
|   12  |  12  |    12   |    12  |
|  123  |  123 |   123   |   123  |
|    1  |    1 |     1   |     1  |

- slide bullet 1
- slide bullet 2

(>- to have bullets appear on click)

horizontal rule/slide break:

***

A footnote [^1]

[^1]: Here is the footnote.
```

Plain text

End a line with two spaces to start a new paragraph.

italics and **bold**

`verbatim code`

sub/superscript^2$_2$

~~strikethrough~~

escaped: * _ \

endash: –, emdash: —

equation: $A = \pi * r^2$

equation block:

$$E = mc^2$$

> block quote

Header1

Header 2

Header 3

Header 4

Header 5

Header 6

HTML ignored in pdfs

http://www.rstudio.com

link

Jump to Header 1

image:

Caption

- unordered list
    - sub-item 1
    - sub-item 2
        - sub-sub-item 1
- item 2

    Continued (indent 4 spaces)

1. ordered list
2. item 2
    i. sub-item 1
        A. sub-sub-item 1

1. A list whose numbering

continues after

2. an interruption

Term 1

Definition 1

Right	Left	Default	Center
12	12	12	12
123	123	123	123
1	1	1	1

- slide bullet 1
- slide bullet 2

(>- to have bullets appear on click)

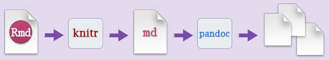
horizontal rule/slide break:

A footnote [1]

[1] Here is the footnote.↩

Set render options with YAML

When you render, R Markdown
1. runs the R code, embeds results and text into .md file with knitr
2. then converts the .md file into the finished format with pandoc



Set a document's default output format in the YAML header:

```
---
output: html_document
---
# Body
```

output value	creates
html_document	html
pdf_document	pdf (requires Tex)
word_document	Microsoft Word (.docx)
odt_document	OpenDocument Text
rtf_document	Rich Text Format
md_document	Markdown
github_document	Github compatible markdown
ioslides_presentation	ioslides HTML slides
slidy_presentation	slidy HTML slides
beamer_presentation	Beamer pdf slides (requires Tex)

Customize output with sub-options (listed at right):

```
---
output:
  html_document:
    code_folding: hide
    toc_float: TRUE
---
# Body
```

Indent 2 spaces

Indent 4 spaces

html tabsets

Use .tabset css class to place sub-headers into tabs

```
# Tabset {.tabset .tabset-fade .tabset-pills}
## Tab 1
text 1
## Tab 2
text 2
### End tabset
```

Tabset

Tab 1 | Tab 2

text 1

End tabset

sub-option	description	html	pdf	word	odt	rtf	md	github	ioslides	slidy	beamer
citation_package	The LaTeX package to process citations, `natbib`, `biblatex` or `none`		X				X				X
code_folding	Let readers to toggle the display of R code, `"none"`, `"hide"`, or `"show"`	X									
colortheme	Beamer color theme to use										X
css	CSS file to use to style document	X							X	X	
dev	Graphics device to use for figure output (e.g. `"png"`)	X	X				X	X	X	X	X
duration	Add a countdown timer (in minutes) to footer of slides									X	
fig_caption	Should figures be rendered with captions?	X	X	X	X				X	X	X
fig_height, fig_width	Default figure height and width (in inches) for document	X	X	X	X	X	X	X	X	X	X
highlight	Syntax highlighting: `"tango"`, `"pygments"`, `"kate"`, `"zenburn"`, `"textmate"`	X	X	X						X	X
includes	File of content to place in document (`in_header`, `before_body`, `after_body`)	X	X		X		X	X	X	X	X
incremental	Should bullets appear one at a time (on presenter mouse clicks)?								X	X	X
keep_md	Save a copy of .md file that contains knitr output	X		X	X	X			X	X	
keep_tex	Save a copy of .tex file that contains knitr output		X								X
latex_engine	Engine to render latex, `"pdflatex"`, `"xelatex"`, or `"lualatex"`		X								X
lib_dir	Directory of dependency files to use (Bootstrap, MathJax, etc.)	X							X	X	
mathjax	Set to `local` or a URL to use a local/URL version of MathJax to render	X							X	X	
md_extensions	Markdown extensions to add to default definition or R Markdown	X	X	X	X	X	X	X	X	X	X
number_sections	Add section numbering to headers	X	X								
pandoc_args	Additional arguments to pass to Pandoc	X	X	X	X	X	X	X	X	X	X
preserve_yaml	Preserve YAML front matter in final document?						X				
reference_docx	docx file whose styles should be copied when producing docx output			X							
self_contained	Embed dependencies into the doc	X							X	X	
slide_level	The lowest heading level that defines individual slides										X
smaller	Use the smaller font size in the presentation?								X		
smart	Convert straight quotes to curly, dashes to em-dashes, … to ellipses, etc.	X							X	X	
template	Pandoc template to use when rendering file	X	X		X					X	X
theme	Bootswatch or Beamer theme to use for page	X									X
toc	Add a table of contents at start of document	X	X	X		X	X	X			X
toc_depth	The lowest level of headings to add to table of contents	X	X	X		X	X	X			
toc_float	Float the table of contents to the left of the main content	X									

Options not listed: extra_dependencies, fig_crop, fig_retina, font_adjustment, font_theme, footer, logo, html_preview, reference_odt, transition, variant, widescreen

Create a Reusable template

1 **Create a new package** with a inst/rmarkdown/templates directory

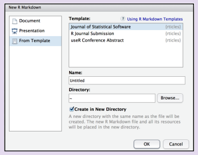
2 In the directory, **Place a folder** that contains:
- **template.yaml** (see below)
- **skeleton.Rmd** (contents of the template)
- any supporting files

3 **Install the package**

4 **Access template** in wizard at File ▸ New File ▸ R Markdown

template.yaml

```
---
name: My Template
---
```



Template:
- Journal of Statistical Software {rticles}
- R Journal Submission {rticles}
- useR Conference Abstract {rticles}

Name: Untitled

Directory: ~

☑ Create in New Directory

A new directory with the same name as the file will be created. The new R Markdown file and all its resources will be placed in the new directory.

Table suggestions

Several functions format R data into tables

Table with kable

eruptions	waiting
3.600	79
1.800	54
3.333	74
2.283	62

	eruptions	waiting
1	3.60	79.00
2	1.80	54.00
3	3.33	74.00
4	2.28	62.00

Table with xtable

Table with stargazer

	eruptions	waiting
1	3.600	79
2	1.800	54
3	3.333	74
4	2.283	62

data <- faithful[1:4,]

~~~
```{r results = 'asis'}
knitr::kable(data, caption = "Table with kable")
```
~~~

~~~
```{r results = "asis"}
print(xtable::xtable(data, caption = "Table with xtable"),
  type = "html", html.table.attributes = "border=0"))
```
~~~

~~~
```{r results = "asis"}
stargazer::stargazer(data, type = "html",
  title = "Table with stargazer")
```
~~~

Learn more in the **stargazer, xtable,** and **knitr** packages.

Citations and Bibliographies

Create citations with .bib, .bibtex, .copac, .enl, .json, .medline, .mods, .ris, .wos, and .xml files

1 **Set bibliography file** and CSL 1.0 Style file (optional) in the YAML header

```
---
bibliography: refs.bib
csl: style.csl
---
```

2 **Use citation keys in text**

```
Smith cited [@smith04].
Smith cited without author [-@smith04].
@smith04 cited in line.
```

3 **Render.** Bibliography will be added to end of document

Smith cited (Joe Smith 2004).
Smith cited without author (2004).
Joe Smith (2004) cited in line.

RStudio® is a trademark of RStudio, Inc. • CC BY RStudio • info@rstudio.com • 844-448-1212 • rstudio.com   More cheat sheets at http://www.rstudio.com/resources/cheatsheets/   Learn more at rmarkdown.rstudio.com • RStudio IDE 0.99.879 • Updated: 02/16