CS&SS 321 - Data Science and Statistics for Social Sciences

Module I - Getting started with R/RStudio

Ramses Llobet

Welcome!

(But Seriously) Welcome!

- Welcome to the first quiz section of CS&SS / SOC / STAT 321!
- I am Ramses Llobet (rllobet@uw.edu), I am a Ph.D. candidate in Political Science.
- My research interest are in **political economy** and **applied statistics**.
- Please **DO NOT** hesitate to stop me if you don't hear or understand me properly.
- **DO NOT** hesitate to ask questions. No question is silly. :)

Now it's your turn

- **Name** and major/year (or intended major)?
- **Why** are you take this course?
- **What** is your experience with R (zero shame)?

`R` setup

- How to install `R` and `R-studio`.
  - R-4.3.2 for Windows
  - R-4.3.2 for macOS
- R-studio can be downloaded from posit's repository.
- I recommend this tutorial from Casey Bates for an overview of R and RStudio.
- **Live coding**: how to install packages and start tutorials (`setting_up.R`).
- For Mac users, installation of the **qss** package may sometimes fail if **pandoc** or **curl** is not installed or updated on your Mac. To resolve this, you can:
  1. Install the package manager Homebrew package.
  2. Then use the macOS terminal to install **pandoc** or **curl** using the commands `brew install pandoc` or `brew install curl`.

Useful free online `R` resources

- Introductory:
  - Grolemund (2014) *Hands-On Programming with R.*
- Intermediary:
  - Wickham et al. (2023) *R for Data Science.* 2nd Edition.
- `R Markdown`
  - Xie et al. (2022) *R Markdown: The Definitive Guide*
- Others
  - Stack Overflow.
  - ChatGPT

Project management and working directory

- A good practice is to keep your projects and files organized and tidy.
  - **Avoid** accumulating data and R files in your **downloads folder**.
- I recommend creating an **R project** file in your course folder materials. R projects have several advantages:
  - Centralized and efficient *workflow*.
  - Sets the **current** (*root*) working directory.
  - See more in Martin Chan's beginner's guide.

What are working directories?

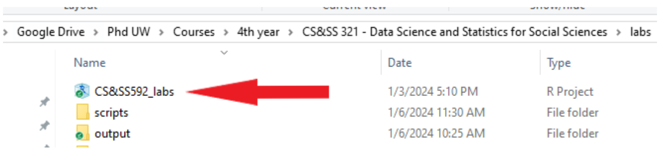
- A **directory** is a **folder** in a file system that stores files and other sub-directories.
- A **path** is a string that specifies the **location** of a directory in a file system.
- For example:
  - `D:\Google Drive`
  - `D:\Google Drive\Phd UW\Courses\Third Year\CSSS 594 - Text as Data`
- When you **run** a command or script, R looks for files and sub-directories based on **relative paths** to your current working directory.

Absolute and relative paths

- **Absolute Path**: Specifies the full path from the **root** directory to the file or directory.
  - For example: `D:\Google Drive\Phd UW\Courses\Third Year\CSSS 321\scripts\setting_up.R` is an **absolute path**.
- **Relative Path**: Specifies the path relative to the current working directory.
  - For example, if the **working directory** is `D:\Google Drive\Phd UW\Courses\Third Year\CSSS 321`, then
    - `scripts\setting_up.R` is a **relative path**.

Project management: working directory

- **.Rproj** (R Project File) in your project folder establishes the working directory as its absolute path upon opening R.

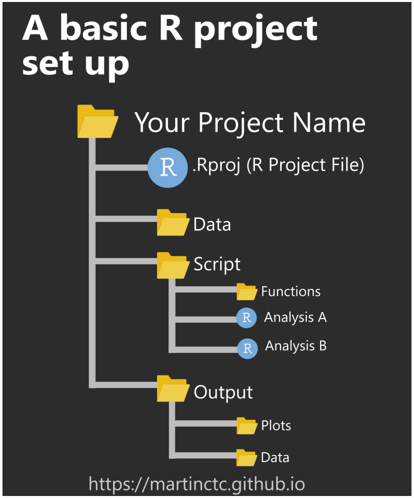


Google Drive > Phd UW > Courses > 4th year > CS&SS 321 - Data Science and Statistics for Social Sciences > labs

Name	Date	Type
CS&SS592_labs	1/3/2024 5:10 PM	R Project
scripts	1/6/2024 11:30 AM	File folder
output	1/6/2024 10:25 AM	File folder

- Employing **.Rproj** and **relative paths** in R streamlines project management and collaboration by overseeing files, inputs, and outputs.
  - **Live demostration** of how to create and manage an R Project File.

Project management: workflow



Working directories: obsolete practices

- Workflow with *.Rproj* is relatively **new**.
- Until recently, users had to **manually set** working directories using **functions** or specialized **packages**. See example:

```
setwd() # function to set directory

setwd("D:/Google Drive/Phd UW/Courses
      /Third Year/CSSS 594 - Text as Data
      /presentation") # remember to put quotes
```

What are functions?

- They are a **set of instructions** that performs a specific task in R.
- Functions often take one or more **arguments**, which are inputs that are used to customize the behavior of the function.
- The `mean()` function takes one **required** argument, which is the vector of numbers to calculate the mean of.

```r
# create a vector consisting of midterm scores.
grades_M <- c(76, 82, 94, 45, 75)

# calculate the mean using the mean() function
mean(grades_M)
```

```
## [1] 74.4
```

What are functions?

- the `mean()` function also has additional optional arguments, which can be used to further customize the behavior of the function.

```
# create a vector consisting of final scores.
grades_F <- c(82, 90, 89, NA, 64)

# calculate the mean using the mean() function
mean(grades_F)
```

```
## [1] NA
```

```
# use the argument `na.rm` to evaluate the removal of NAs
mean(grades_F, na.rm= TRUE)
```

```
## [1] 81.25
```

- **Remember**: use `?` or `help()` to see the documentation of a function.

Reading the error messages

Reference: David Robinson

```
x <- 10 + foo
```

`Error: object 'foo' not found`: You tried to access a variable that doesn't exist.

You might have:

- **misspelled** the variable name
- incorrectly **capitalized** the variable name (R is case sensitive!)
- **forgotten** to run the line that defines the variable in the first place, or run into an error on that line.

Reading the error messages

```r
x <- foo(...)
```

`Error: could not find function "foo"`: You tried to use a function that doesn't exist. You might have:

- **misspelled** the function name
- incorrectly **capitalized** the function name
- forgotten to **load the library** that provides this function.

Reading the error messages

```r
x <- c(1:10))
```

`Error: unexpected ')' in ...`: There is an extra end parenthesis in your line

```r
x <- 10; y <- 20
mean(x y)
```

`Error: unexpected symbol in ...`: The most common cause of this is forgetting a punctuation mark such as a comma: for example, `foo(bar1 bar2)` instead of `foo(bar1, bar2)`.

`Error: unexpected numeric constant` is similar: it just means the value after the missing punctuation is a number (for example, `x 2` instead of `x = 2`).

Reading the error messages

```
paste("welcome to CSSS, 321)
```

- You might see a + sign in the interpreter after you hit return. This means the previous statement is unfinished:
- it might have an **open parenthesis** that never closes,
  - an open `"` or `'` that is unmatched, or
  - it could end with an operator like + or − that expects the line to continue afterwards.
- Find the problem in your previous lines (count parentheses, and check your quotes) and fix it.

Seek for help: reproducible code

- If you feel stuck with an error, seek help but remember to provide **reproducible code** in an R-script file:
  1. Load necessary **packages** at the beginning.
  2. Include all code up to the error, or at least the **necessary** to reproduce it.
  3. **Comment** your code for clarity.
  4. If applicable, send the necessary **data** to reproduce the error.

R-Markdown

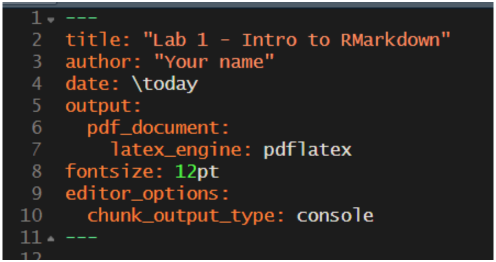
- Save the following Cheat Sheet for `RMarkdown`.
- If any of you is looking for an general introduction for `RMarkdown`, I suggest you to check Chapter 27 from Wickham and Grolemund (2017) - **R for Data Science**.
- If you want a more comprehensive guide, then check Xie et al. (2021) - **R Markdown: The Definitive Guide**.
- Another, more applied, resource is Xie et al. (2022) - **R Markdown Cookbook**.

R-Markdown

- `RMarkdown` is a document format that allows you to integrate `R` **code** and **output** into a single document.
- Besides `R` code and output, it can also include **text**, **images**, and other **multimedia elements**, allowing for rich and informative documents.
- *Pandoc* is a free and open-source **document converter** that can convert documents from one markup language to another.
    - In the context of `Rmarkdown`, pandoc is the underlying document converter (sfotware) that converts the R-markdown file into a final output format, such as **HTML**, **PDF**, or **Word**.

R-Markdown

- The output format of the final document can be customized using options in the **YAML header** or external templates.

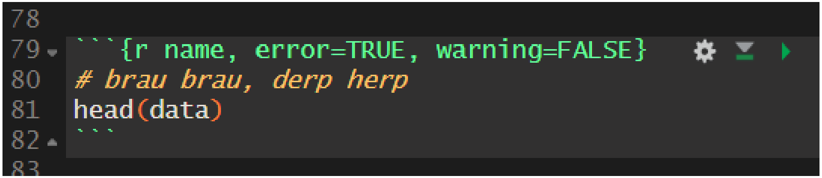


```
---
title: "Lab 1 - Intro to RMarkdown"
author: "Your name"
date: \today
output:
  pdf_document:
    latex_engine: pdflatex
fontsize: 12pt
editor_options:
  chunk_output_type: console
---
```

- The YAML header in RMarkdown is a block of configuration settings at the beginning of the document enclosed by three hyphens (---).
- It is used to specify document metadata and other settings such as the document title, author, output format, and more.

R-Markdown

- **Code chunks** are sections of R code that can be executed and embedded within an RMarkdown document.



~~~
```{r name, error=TRUE, warning=FALSE}
# brau brau, derp herp
head(data)
```
~~~

- Code chunks can be inserted using the syntax {r} and closed with "'.
  - Short cut in Windows: `Ctrl + Alt + I`
  - Short cut in macOS: `Cmd + Option + I`
- Code chunks can be customized with various **chunk options**.
- **Note**: set the function `knitr::opts_chunk$set()` with any general setting without repeating it in every code chunk.

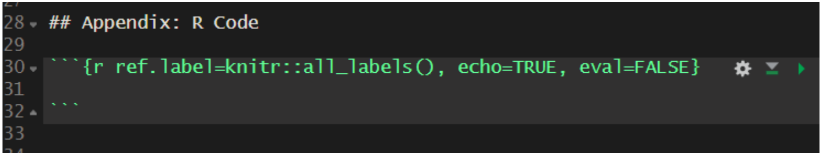
R-Markdown

- Frequently used chunk options

Option	Description
include	If `FALSE`, knitr will run the chunk but **not** include the chunk in the final document
echo	If `FALSE`, knitr will **not** display the code in the code chunk above it's results in the final document.
error	If `FALSE`, knitr will **not** display any error messages generated by the code.
message	If `FALSE`, knitr will **not** display any messages generated by the code.
warning	If `FALSE`, knitr will **not** display any warning messages generated by the code.

Recommendation for Homework

Option	HW setting
include	TRUE
echo	TRUE
error	FALSE
message	FALSE
warning	FALSE



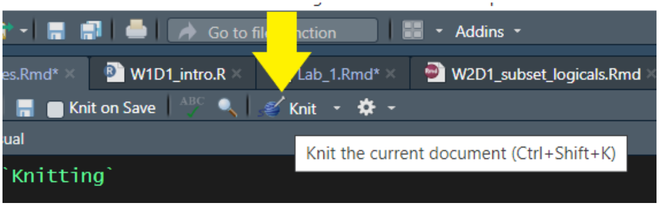
~~~
## Appendix: R Code

```{r ref.label=knitr::all_labels(), echo=TRUE, eval=FALSE}

```

~~~

R-Markdown

- In `RMarkdown`, **rendering** a document means converting the source `RMarkdown` file into its final output format (using pandoc).
- To render a document, we need to `Knit`, knitting is the process of taking the RMarkdown file and converting it into a single, cohesive document that can be rendered into different formats (HTML, PDF, etc).
  - To produce **PDF file**, you need `TeX` files.
- Easy way: Install the `tinytex` package: `install.packages("tinytex")`. Then run `tinytex::install_tinytex()`.

Knitting

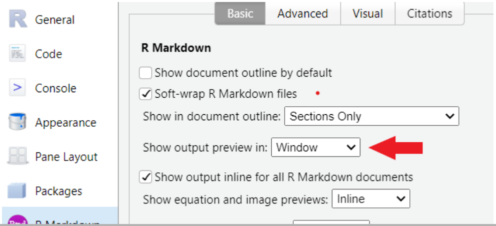
- To knit:



Knit the current document (Ctrl+Shift+K)

`Knitting`

- Auxiliary window for output preview:



R Markdown

- [ ] Show document outline by default
- [✓] Soft-wrap R Markdown files

Show in document outline: Sections Only

Show output preview in: Window

- [✓] Show output inline for all R Markdown documents

Show equation and image previews: Inline

Review of basics

- The subsequent slides show some R basics that we have already covered during the live demonstrations but feel free to review them on your own.

Running R code and operators

```r
# Arithmetic Operators
1 + 1
```

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

```r
2 * 8
```

```
## [1] 16
```

```r
9 / 3
```

```
## [1] 3
```

```r
2^3
```

```
## [1] 8
```

Running R code and operators

```
# Relational Operators
10 > 8 # is 10 bigger than 8?
```

```
## [1] TRUE
```

```
7 <= 6 # is 7 less or equal to 6?
```

```
## [1] FALSE
```

```
(2 * 5) == 10 # is 2*5 equal to 10?
```

```
## [1] TRUE
```

```
1 != 2 # is 1 unequal to 2?
```

```
## [1] TRUE
```

Objects in R: vectors and assignment

```r
# Concatenate vectors into a new vector
c(1, 2, 3)
```

```
## [1] 1 2 3
```

```r
# Assign them to a new object for manipulation
x <- c(1, 2, 3)
print(x) # or simply, x
```

```
## [1] 1 2 3
```

```r
# Operators on vector
x + 1
```

```
## [1] 2 3 4
```

```r
# Logical test on vector
x == 1
```

Objects in R: vectors and functions

```r
# Use an object as input to a function
x <- c(1, 2, 3)

# Functions take input(s) and produce output(s)
class(x)
```

```
## [1] "numeric"
```

```r
length(x)
```

```
## [1] 3
```

```r
mean(x)
```

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

Objects in R: introductory tips

- Unless you assign (<-) some operations or transformations to an object, those values will not be registered

```r
x <- c(1, 2, 3)
print(x + 1)
```

```
## [1] 2 3 4
```

```r
print(x)
```

```
## [1] 1 2 3
```

```r
x <- x + 1
print(x)
```

```
## [1] 2 3 4
```

Objects in R: introductory tips

- New assignment will overwrite the original values if you assign some values to an existing object. It is a **major** source of errors. One advise is to keep distinct object names

```r
x <- c(1, 2, 3)
length(x)
```

```
## [1] 3
```

```r
x <- c(1, 2, 3, 4, 5)
length(x)
```

```
## [1] 5
```

Objects in R: atomic vectors

- Most common types of atomic vectors: **numeric (integer, double)**, **logical**, **character**

```r
x <- c(1, 2, 3)
class(x)
```

```
## [1] "numeric"
```

```r
y <- c(TRUE, FALSE, FALSE)
class(y)
```

```
## [1] "logical"
```

```r
names <- c("Peter", "Paul", "Mary")
class(names)
```

```
## [1] "character"
```

Objects in R: atomic vectors

- You can also coerce one type of vector into another:

```r
x <- c(1, 2, 3)
x <- as.character(x)

print(x)
```

```
## [1] "1" "2" "3"
```

```r
class(x)
```

```
## [1] "character"
```

Objects in R: reading data

- You can import any data file and assign it into an object

```r
x <- c(1, 2, 3)
x <- as.character(x)

print(x)
```

```
## [1] "1" "2" "3"
```

```r
class(x)
```

```
## [1] "character"
```