

# Week 1 Workshop: Introduction to R Markdown

Charlotte Hadley

# 1. Getting to know R

## 2. Getting to know RStudio

## 3. Getting to know RMarkdown

## 4. Understanding the assessment template

# Getting to know R (and RStudio)

# The R Language

R is a scripting language and a very powerful tool for data analysis and presentation.

Primarily, this is thanks to the huge R user base (who are typically called users)

# What is R used for?

us R has developed a vast range of free and open source libraries/packages covering a vast range of different knowledge/computational domains:

- Statistical Analysis
- Machine Learning
- Image Analysis
- Network Analysis
- ... and many more

# R and CRAN

Unlike any other programming language there is a unified system for installing packages easily on Windows, macOS and Linux.

This is also where you download R from - the Comprehensive R Archive Network ([CRAN](#)).

# ... Windows users and RTools

There's always a slight complication to these things.

Both Linux and macOS come with a number of developer-focused tools pre-installed that aren't in Windows, it's really easy to install them.

Navigate to <https://cran.r-project.org/bin/windows/Rtools/> and download the most recent of the tools.

# R, CRAN and Packages

There are 18000+ packages on CRAN.

Deciding which packages to use (and trust) can be a daunting part of becoming confident and fluent in R.



# The R Console

R is the name of the programming language and console within which many users write and evaluate their code.

To use R on your local machine you must [download](#) and install the R Console, it's available on Windows, macOS and Linux.

Like most consoles this application provides only the following functionality:

- Write code and save script files
- Evaluate code and save script files

# The R Console

While it is possible to use the R console for all of your R needs, it's not the best experience you can have.

I highly recommend you use RStudio instead.



RStudio provides the most rich and fullest experience of the R language and what you can do with R.

RStudio is a free, open-source IDE (integrated development environment)...

## What's an IDE?

IDEs are applications designed and used by software developers to assist in the development, debugging and testing of code.

If you're writing R scripts, you're an R developer. You'll find benefits to using an IDE to develop your code even if it's only a few lines long.

In your CV you might want to add a section about IDEs, particularly if you learn more than one.

# RStudio and Data Science

RStudio is not just an IDE. It's awesome.

In this course, we'll make use of the following features:

- Relative file paths and transportable code thanks to RStudio projects
- Built-in documentation reader
- Built-in tools for debugging and running Shiny apps
- Built-in viewer for interactive data viz built with `{htmlwidgets}`
- Support for add-ins to improve user experience, including `{rmarkdown}`



# TASK: Get familiar with RStudio

Let's make sure you can do the following:

- Identify the console, environment and files tab
- Understand the difference between script files and console
- Create new script files

# Base R and R Packages

When R is installed on your computer the machinery necessary to run R code is added to your computer and a number of "base" packages<sup>1</sup> including; statistics and graphics

These packages will not get you far in life, unless you're prepared to write a lot of code from scratch. But you can guarantee that any code samples you see online referring to "base R" will work without having to install additional libraries.

[1] See [stackoverflow.com/a/9705725/1659890](https://stackoverflow.com/a/9705725/1659890) for further details.

# The 3 steps to using R packages

In general, if you want to make use of an R package these are the steps you must go through:

1. Install package on your machine<sup>1</sup>
2. Load package in your script file
3. Use the functions/data loaded from your package

[1] You only need to do this once per machine.

# 1. Installing R packages

We use the following function to install a package:

```
install.packages("tidyverse")
```

It's highly recommended you only write this code in your console and never in your script file.

- Packages only need to be installed once.
- `install.packages()` will try to get the latest CRAN version of a package
- The latest CRAN version of your package may affect your code's output





## ASIDE: Packages/Libraries

For all intents and purposes, library and package can be used completely interchangeably in R.

However, there is a general rule of thumb:

- When the thing is not being used on your local machine it is usually called a package:
  - We install and remove packages.
  - We develop packages.
  - We search for packages online.
  - We look up a function from a package.
- When the thing is being used on your local machine it is called a library:
  - We load a library.
  - The directory containing a package on your local machine is called the package library.



# Task: Install {ggplot2}

SLIDE 1 OF 2

1. Run `install.packages("ggplot2")`



# Task: Install {ggplot2}

## SLIDE 2 OF 2

1. Run `install.packages("ggplot2")`
2. In a R script file, expose the functions in the package by typing

```
1 ggplot2::
```

- ◆ `aes`
- ◆ `aes_`
- ◆ `aes_all`
- ◆ `aes_auto`
- ◆ `aes_q`
- ◆ `aes_string`
- ◆ `alpha`

- `{ggplot2}`
- `{ggplot2}`
- `{ggplot2}`
- `{ggplot2}`
- `{ggplot2}`
- `{ggplot2}`
- `{ggplot2}`

`alpha(colour, alpha = NA)`

These objects are imported from other packages. Follow the links below to see their documentation.

`grid`

`unit, arrow`

`scales`

`alpha`

Press F1 for additional help

## 2. Loading packages

Packages are designed to be loaded as follows:

```
library("ggplot2")
```

This achieves the following:

- `ggplot` functions are now available in RStudio's auto-completion
- any datasets from the package are available for us to use

[1] Loaded/attached... it's fairly complicated.

[2] For instance, a dataset about Star Wars is made available when the `dplyr` package is loaded.

## 3. Using packages

You're now free to use the package as the developer intended.

However, there are several big sanity warnings when it comes to using packages:

- The package might be written poorly (unoptimised, or with incorrect assumptions)
- The package may well become unsupported in the future if the developer does not maintain it
- Not all packages are designed to play nicely with one another, manipulating your code to play nice with `{ rmarkdown }` is harder than simply writing in base R



# Common Errors: Packages

SLIDE 1 OF 3

1. Forgetting to install a package

```
> library("foobar")  
Error in library("foobar") : there is no package called 'foobar'
```

If you see this error it's because you've either misspelled the package, or forgotten to install it.

You can fix this by installing the package in your console and then trying to load the package again.



# Common Errors: Packages

## SLIDE 2 OF 3

1. Forgetting to install a package
2. Forgetting to load a package

```
> fluidPage()
```

```
Error in fluidPage() : could not find function "fluidPage"
```

If you see this error it's because R can't find the function you've written.

It could be missing or you may have forgotten to load the library.

Fix this by running `library("package")`



# Common Errors: Packages

## SLIDE 3 OF 3

1. Forgetting to install a package
2. Forgetting to load a package

---

**Best Practice:** Always load packages at the top of your script files.

---

To aid you in reading these materials, we follow two conventions:

- `{ ggplot2 }` means the package `ggplot2`.
- `ggplot` means a function called `ggplot`<sup>1</sup>.



# Brackets in the R language

# Brackets matter

It's crucial you can quickly distinguish between the different brackets in the R language, so that when you are:

- Reading code e. we know what's what
- Writing code e. we know what to write
- Fixing code e. we can find missing brackets

Let's test your knowledge of what brackets mean in the R language.



# Test: Round brackets ()

What do we know about the following code because there are ( )?

```
praise : praise()
```

`praise` is a function, round brackets are used to encapsulate (or wrap) arguments passed to a function.

As in the example above, some functions have default arguments so can be called simply with

```
praise()
```

Multiple arguments are delimited by commas, for example:

```
rep("hello world")
```

```
## [1] "hello world" "hello world"
```



# Test: Square Brackets

What do we know about the following code because there are [ ]?

```
i s l a n d i [ m d r ]
```

i s l a n d i is a list object, the [ ] are brackets to extract an element from the object.

In this instance, we are extracting an element by its name "Timor".

Often items are extracted according to their "relative position in the object", which is called indexing

```
i s l a n d i [ 3 ]
```

```
# #      A s i a  
# #    1 6 9 8 8
```



# Test: Curly brackets or braces {}

What does the following code achieve?

```
var_1 <- 2
var_2 <- 1
var_3 <- { var_1 * var_2 + var_2 } / var_2
var_1 ^ { var_2 + var_3 }
```

Braces are used in the same way as parentheses when we write mathematics by hand, they allow multiple things to be done together.

# Braces in loops and conditions

Braces are used to denote the body of loops and conditions :

```
for (variable) {  
  ## This is the body of the loop  
}
```

```
if (condition) {  
  ## This is the body of TRUE condition  
} else {  
  ## This is the body of the FALSE condition  
}
```



# Test: foo\$bar

What do we know about `iris` because of the `$` in the following code?

```
iris$Species
```

`iris` is an object with named elements (specifically a `data.frame`), therefore we can provide our own completion of names as we type.

We could just have well indexed this object as follows:

```
iris["Species"]
```

# Indexing, subsetting and filtering data

Most R users work with rectangular data stored in .csv/.xlsx files and need to extract rows that meet certain criteria.

This is an extremely common task that we'll perform many times during this course.

It's called filtering, subsetting or indexing, depending on the method that we choose.

```
iris[which(iris$Species != "setosa"), ] # indexing
subset(iris, Species != "setosa") # subsetting
```

We'll strongly prefer filtering with the tidyverse for reasons we'll explain later

```
library("tidyverse")
filter(iris, Species != "setosa") # filtering
```



# RStudio Projects

# Good habits from the start

It's important when you start programming to keep good habits in mind, they're sometimes quite difficult to impose once you've started.

One of the best R habits to keep is to separate your work into distinct RStudio Projects.

This has the following benefits:

- Consistently separates work packages - making it easier to context switch
- Makes code more portable - as all file paths are relative
- Simplifies using git alongside R

# setwd() is bad practice

Many users of R begin their work with:

```
setwd("Users/charliejhady/GoogleDrive/courses/interactive-sta
```

This is a bad way to start.

- You've immediately tied your code to only working on your operating system AND your machine.

---

This advice is slightly out of context as we're going to start building simple { ggplot2 } before we import data so we'll come back to this point later.



# Task: RStudio Projects Round Trip

## SLIDE 1 OF 4

We're going to be creating many projects during this course, they should all be saved within the `g g p l o t 2 - e x a m p l e` folder ~~if you downloaded earlier~~

1. Add a sub folder to this location called "01-workshop"
2. Save each script file into this folder
3. Close all open script files in RStudio



# Task: RStudio Projects Round Trip

## SLIDE 2 OF 4

RStudio has a built-in menu for working with projects in the top-right hand corner of the screen.

1. Click on the menu
2. Select "New Project"

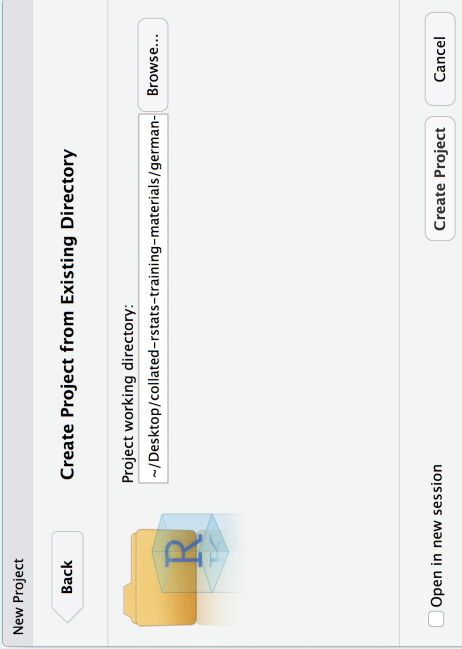
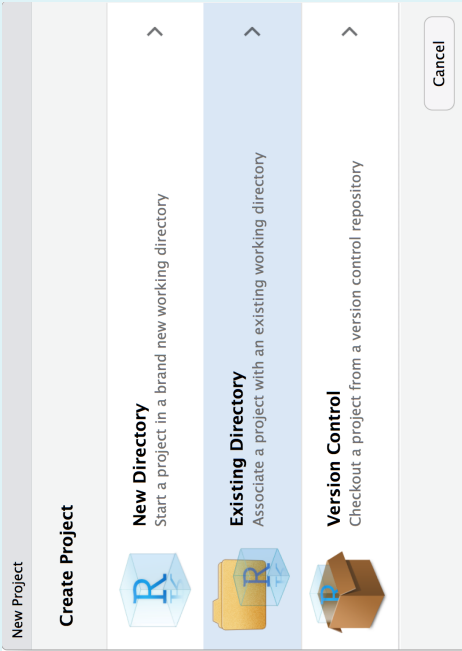




# Task: RStudio Projects Round Trip

## SLIDE 3 OF 4

1. Choose Existing Directory"
2. Navigate to the 01 - workshop directory
3. Create the project





# Task: RStudio Projects Round Trip

## SLIDE 4 OF 4

RStudio will then reload and the files tab will show the contents of your folder, plus a new file called `01 - workshop.Rproj`

This `01 - workshop.Rproj` is the project file. Let's prove it by completing the round trip.

1. Close RStudio
2. Navigate to the `01 - workshop` folder and open the `01 - workshop.Rproj`
3. Voila. RStudio has opened the project.

# R Markdown



# What does R Markdown allow us to do?

R Markdown is a technology that allows us to write reports side by side and more in R.

This is really powerful because it allows us to closely connect our data, code, analysis and the text we write about these things.

We can also programmatically generate many reports.

# What does R Markdown allow us to do?

R Markdown is a technology that allows us to write reports side-by-side and more in R.

R Markdown is also a very powerful tool for doing Exploratory Data Analysis (EDA).

EDA is the process we go through when we first get a dataset (or a question) and we want to understand it.

It's much easier to annotate your thought process in an R Markdown document than an R script file... we call these tools literate programming environments.

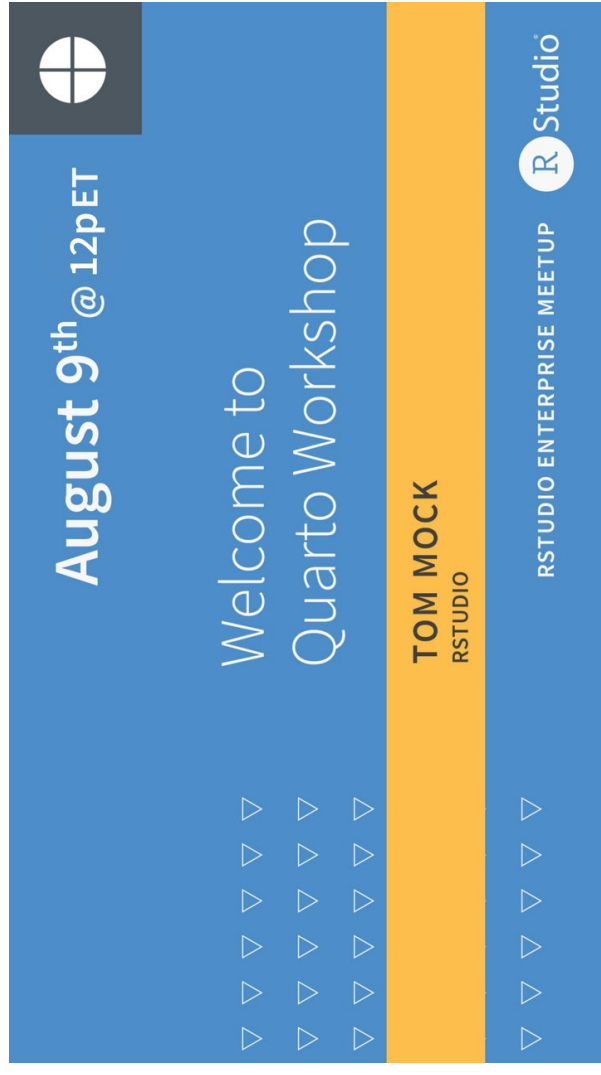
Jupyter notebooks are also useful for EDA, however on their own do not provide good solutions for generating [high-quality] reports and slide decks.

# We need to talk about Quarto

You might read things about something called Quarto.

Quarto is the next generation of R Markdown that works for; R, Python, Julia and Observable JS.

- Your assessment uses R Markdown (I can't change that!)
- It is very simple to change an R Markdown document to work with Quarto.
- Later in the course we can discuss if you want me to teach you Quarto.



# ... lets also talk about RStudio/Posit

The RStudio application we're using is built by a company that was called RStudio but was rebranded in July 2022 to Posit

The RStudio application will always be called RStudio, but other tools will change name

The company rebranded because they're expanding the scope of their toolings to support more languages than just R.

- Quarto is the biggest demonstration of this
- Shiny for Python is something we'll talk about at the end of the course.



# (Palate cleanser)

# What can we make with R Markdown?

# Output File Types

There are three very different types of "document" we can output with the `rmarkdown` package:

- PDF (for print and trans portability)
- HTML (for the web and mobile)
- .docx (for when you're held hostage)

Let's address each of these in turn.

# RMarkdown PDF

RMarkdown relies on LaTeX to generate PDF, which means you need to install more stuff:

- Windows users see <https://miktex.org/download>
- Everyone else see <https://www.latex-project.org/get/>

It's possible to build PDF presentations and reports with RMarkdown.

Reports (pdf document)

Presentations (beamer presentation)

Practical format for short articles.

Practical for presentations without interactive content.



# Bookdown for long-form PDF

R Markdown was not built for the heavy-duty work of creating books - it has some issues surrounding enumeration and citations that are frustrating.

We highly recommend that you use **bookdown** if writing long-form documents rather than short articles.

- **bookdown** was used to write the [R for Data Science](#) book.
- **bookdown** creates both a web (HTML) and print (PDF) version of your documents.



# R Markdown HTML

The r markdown package allows us to create two types of HTML document:

- `html_document`: A single web page
- `slidy_presenter`: A world different formats for creating a HTML presentation; a document split into slides.

# blogdown for websites

The `blogdown` package for creating a website (including a blog engine) directly from R Markdown.

Our website is developed with `blogdown`

- Our website lives on GitHub: <https://github.com/visbieldata/visibledata.github.io>
- GitHub uses Hugo to generate our website (this service is called [Github Pages](#)).
- We use [Netlify.com](#) for continuous integration and beta-testing of our site on top of this.



# xaringan for beautiful slides

The built-in R Markdown presentation formats (ioslides and slidey) have many limitations.

The xaringan package allows beautiful and well-featured presentations to be written in R Markdown.

- These lectures are written exclusively in xaringan



# Why xaringan?

One of my favourite features is what it prevents me from doing.

There's a natural limit on how much content you can fit into a slide.

This is initially how much stuff you should fit on a slide.

# R Markdown basics



# Task: Create a new R Markdown document

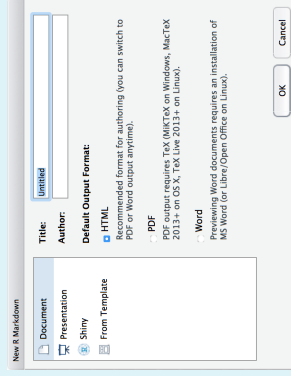
Let's create a new project called `r markdown - basics`

1. File -> New -> R Markdown

2. Click "Ok"

If you don't see this dialog, run

`install.packages("rmarkdown")`



preamble (or header) which tells RStudio what to output (and how), written in YAML

special code chunk that sets "global" code chunk options

code chunk

The screenshot shows an R Markdown document with the following content:

```

1 ---
2 title: "Untitled"
3 output: html_document
4 ---
5
6 {r setup, include=FALSE}
7 knitr::opts_chunk$set(echo = TRUE)
8
9
10 ## R Markdown
11
12 This is an R Markdown document. Markdown
13 syntax for authoring HTML, PDF, and MS
14 Word documents can be found at http://rmarkdown.rstudio.com.
15
16 {r cars}
17 summary(cars)
18
19
20 ## Including Plots
21
22 You can also embed plots, for example:
23
24 {r pressure, echo=FALSE}
25 plot(pressure)
26
27
28 Note that the `echo = FALSE` parameter was added to the code chunk to
29 prevent printing of the R code that generated the plot.

```

Annotations in the image include:

- A blue circle around the gear icon in the top right corner of the document, with a callout: "clicking the cog provides an interface for changing code chunk options (the global options in **this** case)".
- A blue oval around the "## R Markdown" header, with a callout: "This is an R Markdown document. Markdown syntax for authoring HTML, PDF, and MS Word documents can be found at <http://rmarkdown.rstudio.com>.".
- A blue oval around the "## Including Plots" header, with a callout: "You can also embed plots, for example:".
- A blue oval around the `{r pressure, echo=FALSE}` code chunk, with a callout: "Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.".
- A blue oval around the `{r cars}` code chunk, with a callout: "code chunk".
- A blue oval around the `{r setup, include=FALSE}` code chunk, with a callout: "special code chunk that sets 'global' code chunk options".
- A blue oval around the `{r cars}` code chunk, with a callout: "code chunk".
- A blue oval around the `{r pressure, echo=FALSE}` code chunk, with a callout: "code chunk".
- A blue oval around the "## Including Plots" header, with a callout: "naming code chunks allows you to easily navigate through your document and to find errors later".
- A blue oval around the "## R Markdown" header, with a callout: "## represent subheadings".

The "Default Chunk Options" dialog box is open, showing the "Output" tab with the following options:

- Show code and output (selected)
- Show warnings (checked)
- Show messages (checked)
- Chunk options (help icon)

The "Output" tab is selected, and the "Apply" button is visible.



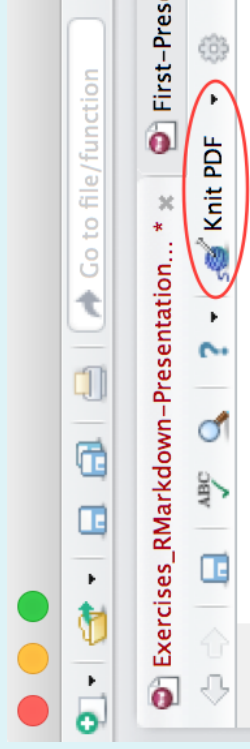


# Task: Generating output from .Rmd

We're currently looking at the source .Rmd file.

To generate the output document, we need to knit the document together:

1. Always save your .Rmd file before generating output.
2. Click the knit button



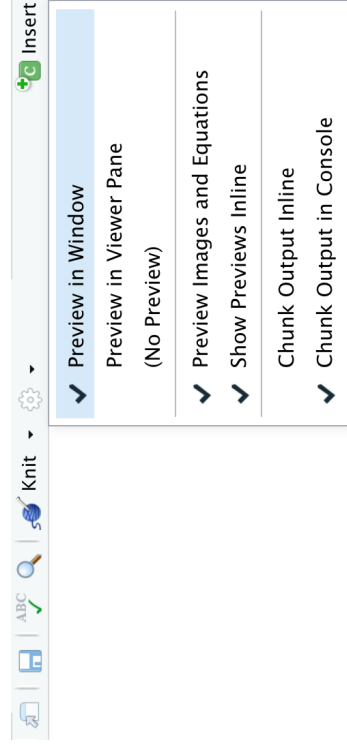
# Preview in... (I)

By default, RMarkdown documents will be previewed in a new window within the RStudio application.

It's important to understand what's happened here:

1. The `{knitr}` package has created a new `html` file in your project directory
2. RStudio has then opened the `html` file in either the `Viewer` tab or a new window.

You can change this behaviour by clicking on the cog next to the knit button:

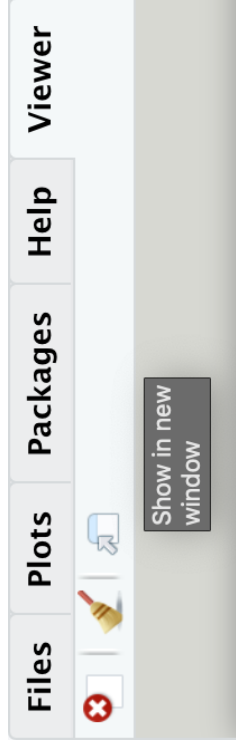


# Preview in... (II)

RStudio can preview the .html file because it's a browser built on Chromium.

However, it's a rubbish web browser. There are many types of content that won't display correctly in RStudio's viewer.

It's highly recommended that you view the output in your default browser<sup>1</sup>



[1] Microsoft recommend that Windows users switch from Internet Explorer (IE) to Microsoft Edge. IE will not correctly display your R Markdown .html output.

# RStudio's Visual Editor

I'm usually very positive about visual editors, but the RStudio visual editor for R Markdown.

However it does not work for ALL output types.

It DOES work for the assessments format you need to use

