Using R for Data Processing

Glenn Williams

University of Sunderland

2021-10-25 (updated: 2021-10-31)

The What and Why of R

What is R?

- **Statistical programming language** used for processing, summarising, analysing, and graphing data.
- **Free and open source**, so anyone can see how it works and add to the development of the program.
- Packages and changes are **vetted by reviewers**, so we can be sure most things we do in R are appropriate.
- Often used with **RStudio** which makes working with **notebooks**, **projects**, and **version control** easier. More on these features later.

Why Should I Care?

- Free software means **anyone can use it at no cost**. This makes science more inclusive, and allows for easier confirmation of analyses.
- Programming languages force you to **document all decisions** made with the data. This makes your research more **transparent**.

You supporting Open Science

- With **Open Source** Software we can see how the program works and improve/extend it where needed. **Finding and fixing problems is easier and quicker**.
- Free, Open Source software is crucial for **Open Science**.

Sharing is Caring

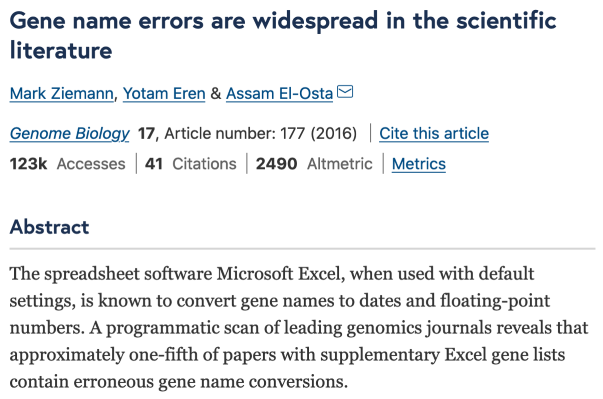
- Is your reseach **replicable**? If I follow your methods with **new participants**, will I get **similar results**?
- Is your research **reproducible**? If I follow your methods with **your data**, will I get **the same results**?
- Checking these things is made possible if you share your materials, data, and steps for analysis.
- Sites like the Open Science Foundation (OSF) and GitHub allow us to host our research products online. They also track changes to your work.
- Opening up your research can be scary, but **it'll probably make you more careful**, allow people to build on your work, and allow science to be more reliable and able to self-correct.

Why Should I Care?

- Veldkamp et al. (2014): 63% of articles contained **at least one *p*-value that was incorrect**. In 20.5% of cases, these errors led to **erroneous decisions** about the statistical significance of an effect.
- How does this happen?
    - Not documenting your methods fully makes **detecting mistakes much more difficult**.
    - Transcribing results from visual interfaces like SPSS means you introduce **human error**.
- How can we fix this problem? **Notebooks**!
    - You never write the results, just the methods for making them.
    - Every step of your analysis is documented.
    - **If your data changes, your results are instantly updated**.

Still Not Convinced?

- Easy to learn is not Easy to use.
  - How do you **filter observations** in Excel? In R, use `filter()`
  - How do you do a **t-test** in SPSS? In R, use `t.test()`
  - Learning to code might take longer, but implementing it is often easier down the line.
- **You won't document everything you do in Excel**. That's a problem if you notice a mistake.
- **R scales up**. Imagine processing 1,000,000 rows of data in Excel. R can do that as easily as 10 rows.



Gene name errors are widespread in the scientific literature

Mark Ziemann, Yotam Eren & Assam El-Osta

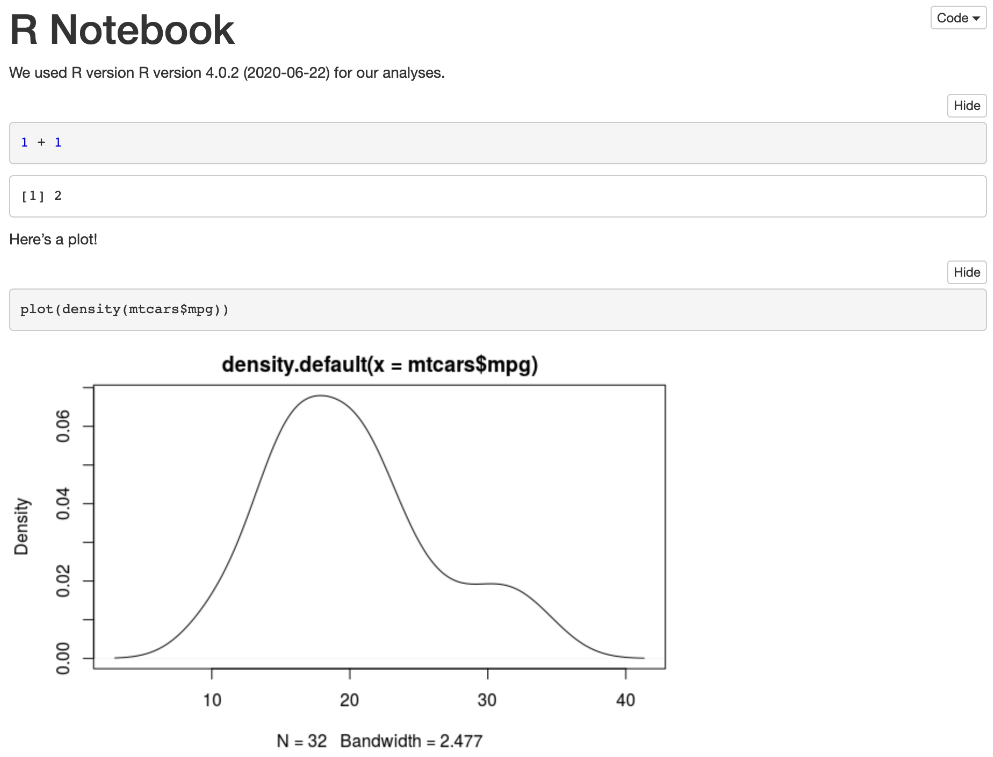
Genome Biology **17**, Article number: 177 (2016) | Cite this article

123k Accesses | **41** Citations | **2490** Altmetric | Metrics

Abstract

The spreadsheet software Microsoft Excel, when used with default settings, is known to convert gene names to dates and floating-point numbers. A programmatic scan of leading genomics journals reveals that approximately one-fifth of papers with supplementary Excel gene lists contain erroneous gene name conversions.

Example Notebook



R Notebook

We used R version R version 4.0.2 (2020-06-22) for our analyses.

```
1 + 1
```

```
[1] 2
```

Here's a plot!

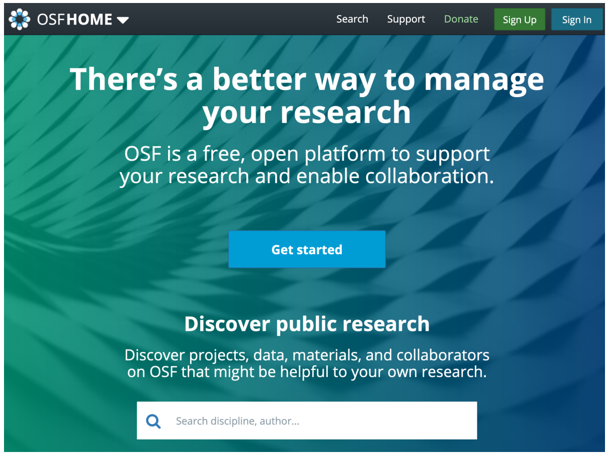
```
plot(density(mtcars$mpg))
```

Software versions are easy to report, and we can **see both the code to produce results as well as the results**.

How Do I Share My Work?

- Using R without sharing will help to solve many issues with reproducibility, but making it accessible to others is even better.

- The OSF is the most user-friendly method of sharing all your research.
    - Servers paid for 50 years, with servers in Europe.
    - **Drag and drop** to upload files.
    - Automatically **tracks updates** to files.
    - Can link **pre-registrations, licenses, and pre-prints** to your project.



There's a better way to manage your research

OSF is a free, open platform to support your research and enable collaboration.

Discover public research

Discover projects, data, materials, and collaborators on OSF that might be helpful to your own research.

The OSF

- GitHub is another option, which has more advanced **version control**, but can be trickier to use.

What We'll Do

After these lessons, you will have:

- used rstudio.cloud to learn the basics of how R works.
- created your first notebook.
- processed some raw data and cleaned it up.
- made summaries of our data.
- made graphs of data.

It can be difficult at first, but it **will make things easier** when it comes to the assessment. Don't worry:

- You will struggle to remember the code, but **that's fine**.
- It's standard in most sciences, so if you have a question/issue, it's already been asked and answered somewhere. **Google is your friend**.
- You'll learn a valuable, **transferable skill**.

Getting Started in R

How Do I Install R?

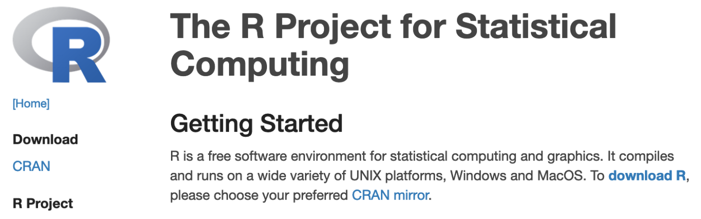
Local Installation

- To get started, at the very least you'll have to download R from CRAN.

- Choose a mirror from which to download R. Any will do.

- Select the distribution for your operating system. On Windows, click **install R for the first time**. On Mac/Linux click the most recent install version.

- You'll see a new page; For Windows, click on **"Download R [version number] for Windows."**.



[Home]

Download

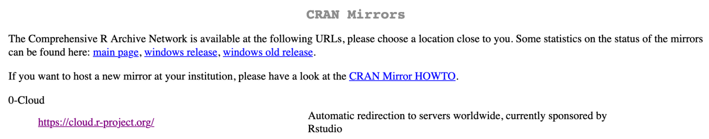
CRAN

R Project

The R Project for Statistical Computing

Getting Started

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To **download R**, please choose your preferred CRAN mirror.



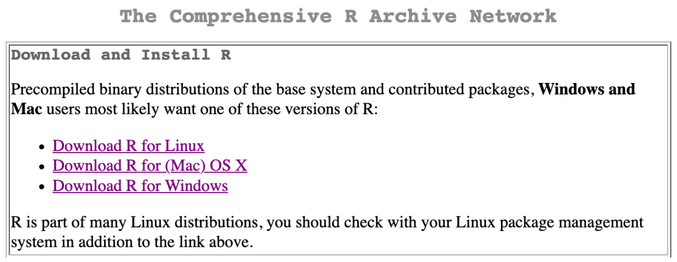
CRAN Mirrors

The Comprehensive R Archive Network is available at the following URLs, please choose a location close to you. Some statistics on the status of the mirrors can be found here: main page, windows release, windows old release.

If you want to host a new mirror at your institution, please have a look at the CRAN Mirror HOWTO.

0-Cloud

https://cloud.r-project.org/	Automatic redirection to servers worldwide, currently sponsored by Rstudio



The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

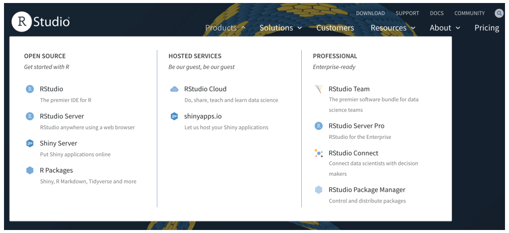
R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

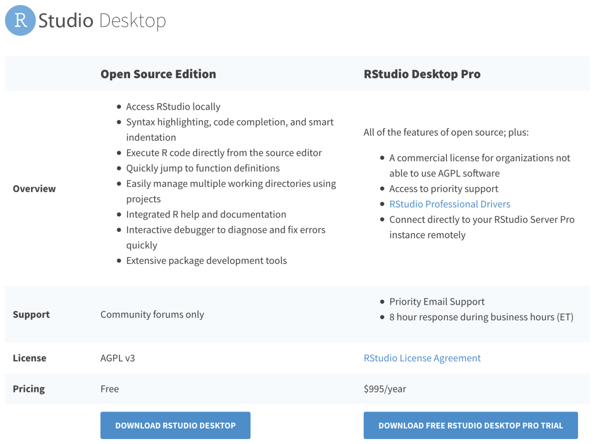
What is RStudio? How Do I Install It?

Local Installation

Integrated Development Environment for R: makes working with R and extensions to it easier.

- Download from the RStudio website.
- Select products and choose **RStudio**. Scroll down until you see **Download RStudio Desktop**.


RStudio
DOWNLOAD
SUPPORT
DOCS
COMMUNITY
Products
Solutions
Customers
Resources
About
Pricing
OPEN SOURCE
Get started with R
RStudio
The premier IDE for R
RStudio Server
RStudio anywhere using a web browser
Shiny Server
Put Shiny applications online
R Packages
Shiny, R Markdown, Tidyverse and more
HOSTED SERVICES
Be our guest, be our guest
RStudio Cloud
Do, share, teach and learn data science
shinyapps.io
Let us host your Shiny applications
PROFESSIONAL
Enterprise-ready
RStudio Team
The premier software bundle for data science teams
RStudio Server Pro
RStudio for the Enterprise
RStudio Connect
Connect data scientists with decision makers
RStudio Package Manager
Control and distribute packages


RStudio Desktop

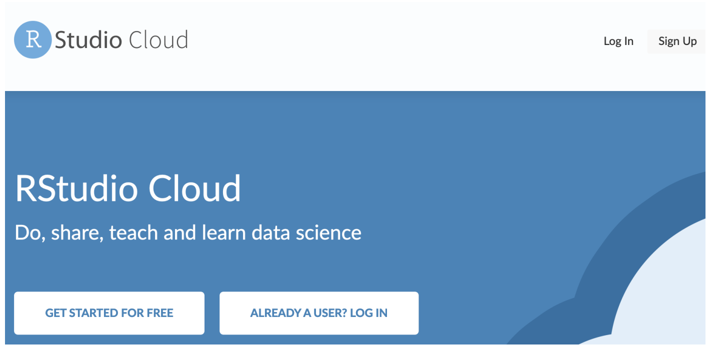
	Open Source Edition	RStudio Desktop Pro
Overview	• Access RStudio locally • Syntax highlighting, code completion, and smart indentation • Execute R code directly from the source editor • Quickly jump to function definitions • Easily manage multiple working directories using projects • Integrated R help and documentation • Interactive debugger to diagnose and fix errors quickly • Extensive package development tools	All of the features of open source; plus: • A commercial license for organizations not able to use AGPL software • Access to priority support • RStudio Professional Drivers • Connect directly to your RStudio Server Pro instance remotely
Support	Community forums only	• Priority Email Support • 8 hour response during business hours (ET)
License	AGPL v3	RStudio License Agreement
Pricing	Free	$995/year
	DOWNLOAD RSTUDIO DESKTOP	DOWNLOAD FREE RSTUDIO DESKTOP PRO TRIAL

- Click the **Download** button in the free tier and select the correct installer for your operating system.

Getting Started in the Cloud

Alternatively, we can use RStudio Cloud to do everything online.

- This may be the **easiest** route, especially if your system is locked down (e.g. on University controlled computers).
- This may be a little more limited than using R on your machine, but **most things you need will be available without effort on your behalf**.



rstudio.cloud signup page

- **Works with Chromebooks** etc. that do not allow local installs of R.
- Please note that rstudio.cloud **has usage limits**. You can only work for 15 hours per month in it at the free tier. If you can, get a local install.

Sign up for an account by clicking sign up on the homepage.

Starting Your First Project

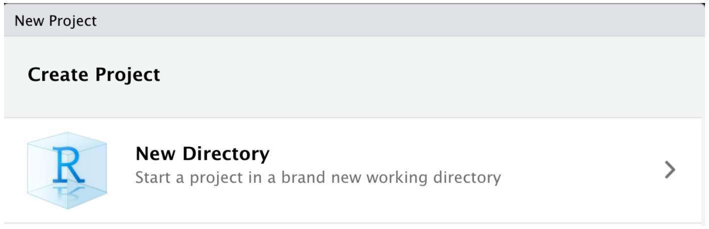
Assuming you're using a local installation of R/RStudio:

- Make a folder somewhere on your computer.
- Open RStudio.
- **Click File** $\rightarrow$ **New Project** $\rightarrow$ **New Directory** $\rightarrow$ **New Project**.
- Give this a name and save it somewhere you can access on your computer (e.g. Desktop)

Assuming you're using rstudio.cloud:

- Make a new Space and give it a name, then **click New Project**.

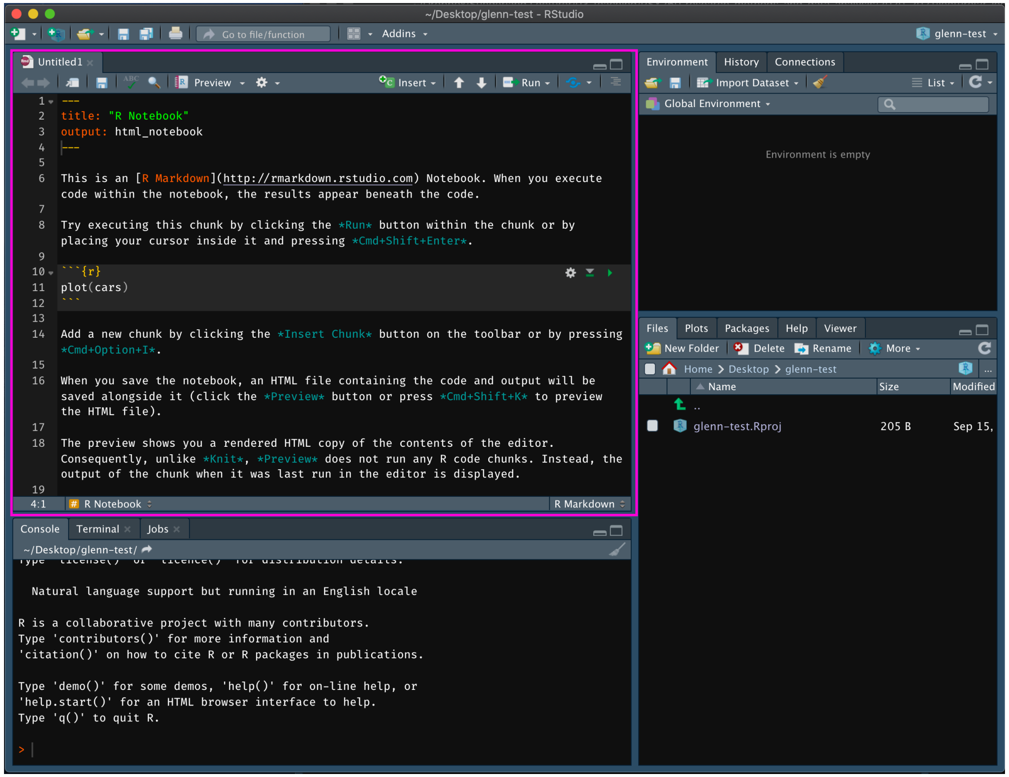
You now have a **folder and an rstudio project** for your work. You can put data and code in here, and all outputs will be saved in this place.



creating a new project

Now What?

Click File → New File → R Notebook. This will open an R notebook in the Editor.



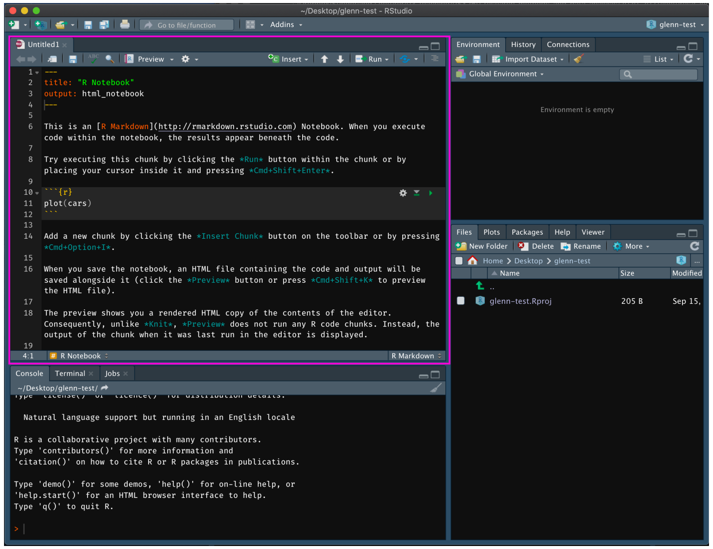
~~~
---
title: "R Notebook"
output: html_notebook
---

This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook. When you execute
code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by
placing your cursor inside it and pressing *Cmd+Shift+Enter*.

```{r}
plot(cars)
```

Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing
*Cmd+Option+I*.

When you save the notebook, an HTML file containing the code and output will be
saved alongside it (click the *Preview* button or press *Cmd+Shift+K* to preview
the HTML file).

The preview shows you a rendered HTML copy of the contents of the editor.
Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the
output of the chunk when it was last run in the editor is displayed.
~~~

```
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

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
..		
glenn-test.Rproj	205 B	Sep 15,

RStudio Interface

Understanding the Editor



~~~
---
title: "R Notebook"
output: html_notebook
---

This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook. When you execute
code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by
placing your cursor inside it and pressing *Cmd+Shift+Enter*.

```{r}
plot(cars)
```

Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing
*Cmd+Option+I*.

When you save the notebook, an HTML file containing the code and output will be
saved alongside it (click the *Preview* button or press *Cmd+Shift+K* to preview
the HTML file).

The preview shows you a rendered HTML copy of the contents of the editor.
Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the
output of the chunk when it was last run in the editor is displayed.
~~~

```
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

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.

> 
```

The RStudio Interface

Editor (top-left): Where you **write your code**. Anything you write here can be saved to the file.

Console (bottom-left): Where we run your code. Once code is entered here you cannot edit it. **Don't work in the console!!!**

Environment (top-right): Lists any **variables avaiable in your global environment**. More on this later.

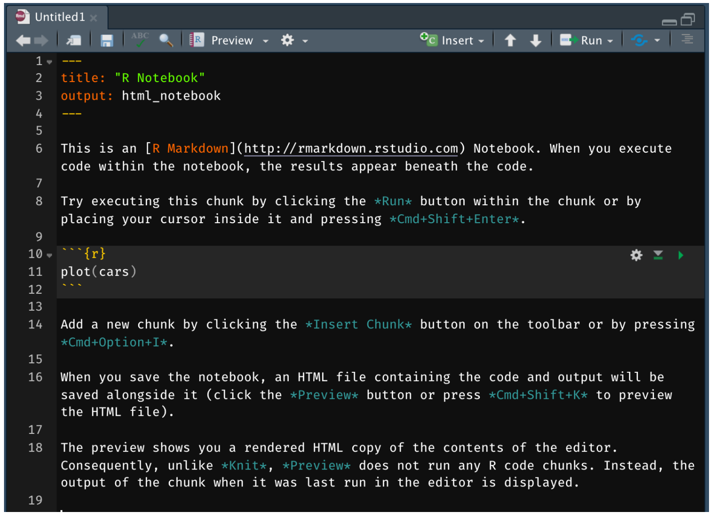
Files/Plots/Help/Viewer (bottom-right): See any plots in your **working dirctory (i.e. in your project)**, view any recently created plots or help on how to use R functions.

Working with Notebooks

After making a notebook (by clicking **File** → **New File** → **R Notebook**). Give your notebook a title by changing the text following `title:` at the top.

Notebooks are made up of:

- **Markdown text**: which allows you to **write in plain English** (or other languages). You **add decoration** to text (e.g. italics, bold, links) using markdown commands. They have some examples in the text for you.
- **R code chunks**: These only accept R code. Press the play button to run the code.



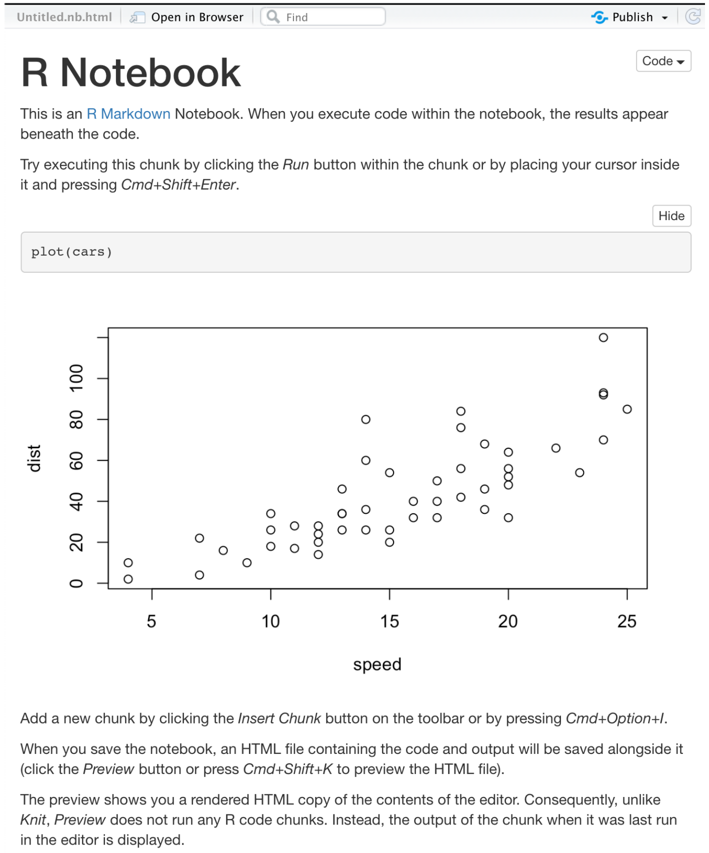
~~~
---
title: "R Notebook"
output: html_notebook
---

This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook. When you execute
code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by
placing your cursor inside it and pressing *Cmd+Shift+Enter*.

```{r}
plot(cars)
```

Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing
*Cmd+Option+I*.

When you save the notebook, an HTML file containing the code and output will be
saved alongside it (click the *Preview* button or press *Cmd+Shift+K* to preview
the HTML file).

The preview shows you a rendered HTML copy of the contents of the editor.
Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the
output of the chunk when it was last run in the editor is displayed.
~~~

An R Notebook

When you save your notebook, you can **preview it**. It will create an HTML file that contains your markdown text and r code with output made pretty.

Preview Your Notebook

Make sure you press play on your code chunks. This will run your code and show you the output. When done, save your work (Command/Ctrl + S) and **click Preview**. You should see this:



R Notebook

This is an R Markdown Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Cmd+Shift+Enter*.

```
plot(cars)
```

Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Cmd+Option+I*.

When you save the notebook, an HTML file containing the code and output will be saved alongside it (click the *Preview* button or press *Cmd+Shift+K* to preview the HTML file).

The preview shows you a rendered HTML copy of the contents of the editor. Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the output of the chunk when it was last run in the editor is displayed.

Avoiding Repetition

In **R code chunks**, we can code things up by hand, or make **functions** that allow us to repeat a sequence of commands easily. For example, let's say we want to add 1 to several numbers.

We could do it by hand:

```
1 + 1
```

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

```
2 + 1
```

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

```
3 + 1
```

```
## [1] 4
```

Or we could write a function, and apply this to our numbers:

```
add_one <- function(x) {x + 1}

add_one(1)
```

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

```
add_one(2)
```

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

```
add_one(3)
```

```
## [1] 4
```

Making Life Easy with Pre-built Functions

Some developers make functions available to others as a **package**.

- The two packages (or package of packages) we'll use are `tidyverse` and `here`.
    - **Tidyverse**: has several functions for making working with data and creating plots easier.
    - **Here**: makes your R-scripts read and write files relative to where the project is. This means you can write scripts that work on any PC (not just your own).

This will be made clear once we start using these packages.

You only need to **install a package once per computer** (or cloud project):

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

But, you **must load the packages (or libraries) every time you start R**:

```
library("tidyverse")
library("here")
```

Using R for Data Processing

Overview

- **Data Types**: How is data stored in R, and how does this affect your choices?
- **Objects and Functions**: How to save data to a variable you can work with, and how to work with it.
- **Packages**: Adding functionality to R with additional packages.

This will give you some basic experience in R which we will build upon in future sessions for doing serious data processing and graphing.

Data Types

Basic Types

R can read your data in and parse it in different ways. There are a few basic data types available to R:

- **Strings** = **Text** data, e.g. "Dog", "Cat", "Dogs and Cats".
- **Integers** = **Whole numbers**, e.g. 1, 2, 3.
- **Doubles/Floats** = **Decimal** numbers, e.g. 0.1, 0.22, 0.14132.
- **Factors** = Ordered or unordered **factors with levels**. These take numbers and associated labels, e.g. 1 = "Dog", 2 = "Cat".
- **Booleans** = These are TRUE or FALSE statements, and only take those values.

These can be stored on their own in **objects** called vectors.

Data Types

Using R to Print Data

You can use R for basic things like printing out your data or using it as a calculator.

```r
"cat"
```

```
## [1] "cat"
```

```r
1 + 1
```

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

Data Types

Using R to Print Data

R will wait for you to finish an expression before executing the operation, e.g. if your line ends with a + or you haven't finished a quote for a string.

```r
1 + 1 + 1 + 1 + 2 +
  1 + 2
```

```
## [1] 9
```

```r
"this is a really long string on
two separate lines."
```

```
## [1] "this is a really long string on\ntwo separate lines."
```

This isn't very useful on its own though...

Data Types

Storing Data

In R, we can assign values to an object using the assignment operator, `<-`.

When we do this, we can perform operations on the object and access it later to retrieve its values:

```r
# make an object called my_object containing the string,
# "hello, class!"
my_object <- "hello, class!"

# access the value of my_object
my_object
```

```
## [1] "hello, class!"
```

Note: We can include comments to our code using the symbol #, which is ignored by R. Comments are good for explaining the aim of your code.

Data Types

Storing Data Together

What if we have several strings or numbers we'd like to store together? We can combine them by **concatenation** with `c()`.

This is a **function** for combining data. Like all R functions, it has a name, and begins and ends with `()`.

```
my_strings <- c("A", "B")
my_numbers <- c(1, 2, 3)

# print them out
my_strings
```

```
## [1] "A" "B"
```

```
my_numbers
```

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

Be careful when **mixing data types**. If you combine a string and a number, R will make everything a string. Why? You can write out a "1", or "2" as text, but you can't make "hello, class" into a number.

```
c(my_strings, my_numbers)
```

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

Data Types

Storing More Data Together

We can store data together in different ways.

The most common way in R is to use a `data.frame`. This lets you to make a table of data with columns of different data types.

```
my_frame <- data.frame(
  instructor = c("Glenn", "Becci", "Shannon"),
  has_cat = c(TRUE, TRUE, FALSE),
  n_cats = c(2, 1, 0)
)

my_frame
```

```
##   instructor has_cat n_cats
## 1      Glenn    TRUE      2
## 2      Becci    TRUE      1
## 3    Shannon   FALSE      0
```

Data Types

Storing More Data Together

We can also store data in matrices if they are numbers:

```
matrix(
  c(1:6),
  nrow = 2,
  ncol = 3
)
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

Or in lists where we can have nested levels of detail:

```
list(
  instructor = c("Glenn", "Becci
  is_mint = c(TRUE, TRUE, TRUE)
)
```

```
## $instructor
## [1] "Glenn"   "Becci"   "Shannon"
##
## $is_mint
## [1] TRUE TRUE TRUE
```

We won't focus on these in the following sessions.

Operations

You've seen we can do things with data. Here's some basic operations we can perform:

R has the simple arithmetic operations, e.g.:

- add: `x + y`,
- subtract: `x - y`,
- multiply: `x * y`
- divide: `x/y`,

R also has some logical operations built in. For example:

- less than or equal to: `x <= y`
- exactly equal to: `x == y`
- not equal to: `x != y`
- x OR y: `x | y`
- x AND y: `x & y`

We can do additional operations with functions.

Functions

R has a number of inbuilt functions, which can be expanded by loading new packages.

Some basic arithmetic can be seen below:

```r
sum(c(1, 2, 3, 4))
```

```
## [1] 10
```

```r
mean(c(1, 2, 3, 4))
```

```
## [1] 2.5
```

This is just the same:

```r
a <- c(1, 2, 3, 4)
mean(a)
```

```
## [1] 2.5
```

Accessing Data from Vectors

When we have data stored in an R object, we can **pull out specific values**.

With vectors, we do this by their index (position) in the vector, starting with 1. These values are accessed with square brackets after the vector name.

```
my_numbers <- c(1, 2, 5, 10, 20)
my_numbers[3]
```

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

- We can also change these values if we want.

```
my_numbers[3] <- 100
my_numbers
```

```
## [1]   1   2 100  10  20
```

Accessing Data from Data Frames

When we have data stored in an R object, we can **pull out specific values**.

With **data frames**, we can do this by their **column name, position**, or some mix of the two. Values are indexed by square brackets after the data.frame name, separated like: **[row number, column number]**.

```r
df <- data.frame(
  a = c(1, 2, 3),
  b = c("x", "y", "z")
)

df # show the data
```

```
##   a b
## 1 1 x
## 2 2 y
## 3 3 z
```

```r
df[2, 1] # row 2, column 1
```

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

```r
df[2, "a"] # row 2, column "a"
```

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

```r
df$a # access column "a"
```

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

If you want all rows from column b, use `df[, "b"]`. If you want a specific row for all columns, use e.g. `df[1, ]`.

Using Packages

We've seen how to install and load packages. Once loaded (e.g. `library(tidyverse)`), you can use functions from it:

```r
library(tidyverse)

df <- data.frame(
  a = c(1, 2, 3, 10, 20),
  b = c("a", "b", "c", "d", "e")
)

filter(df, a > 2)
```

```
##    a b
## 1  3 c
## 2 10 d
## 3 20 e
```

Here, **filter subsets your data frame** only to observations where column a has values greater than 2.

Conclusion

In the last 3 sessions you've learned a lot, including:

- **What R is**, and why it's important and helpful in your research.
- How to **install R**, navigate the RStudio interface, and set up a **project**.
- How to **create a notebook** to document your work in R.
- How to **perform basic operations on data in R**, including storing data in objects, accessing that data, and creating tables of data.

What Next?

Next week, you'll learn about how to read and write data in R, how to inspect your data, and how to perform operations on the data you have.