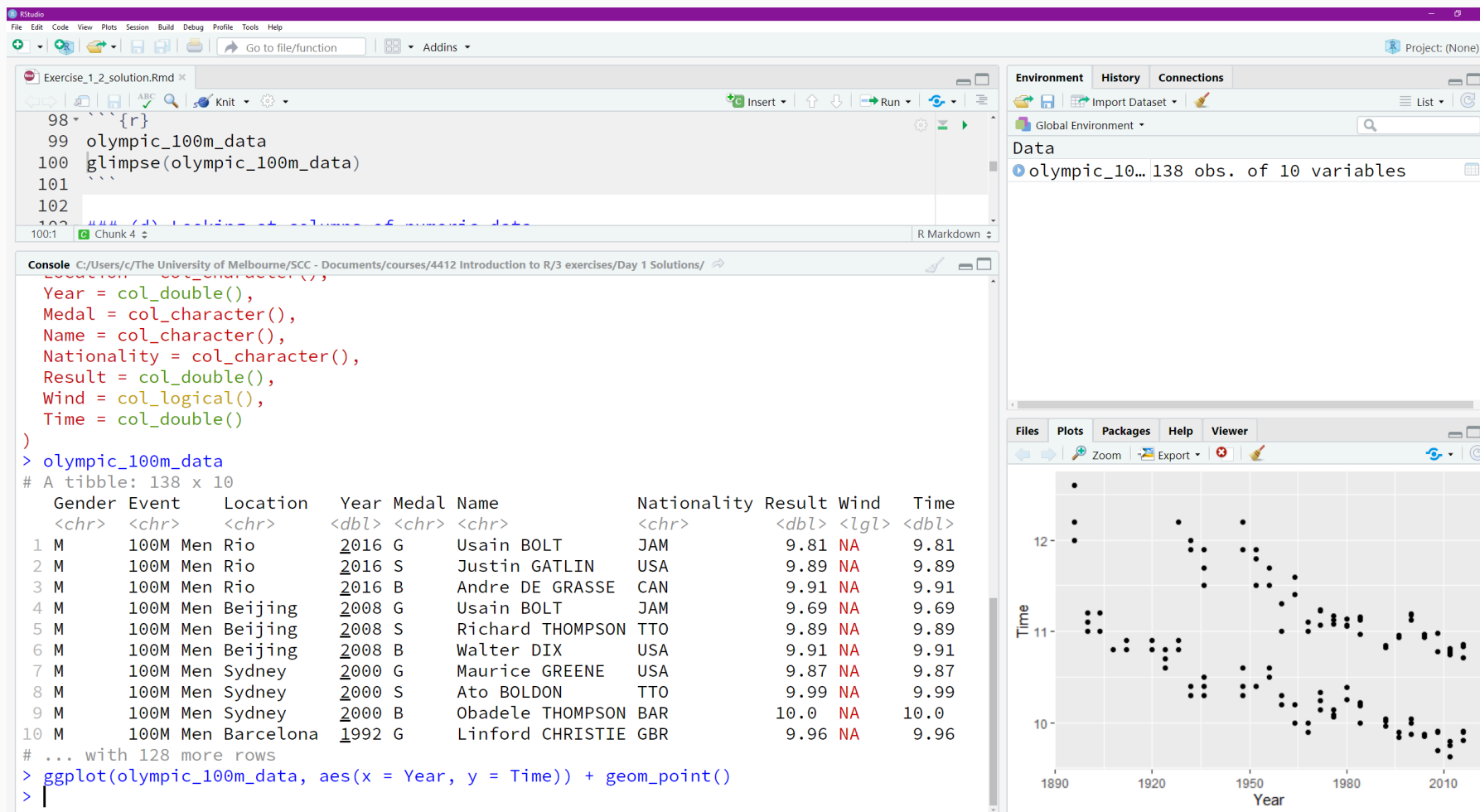


Importing data into R

Melbourne Statistical Consulting Platform
University of Melbourne
April 2024

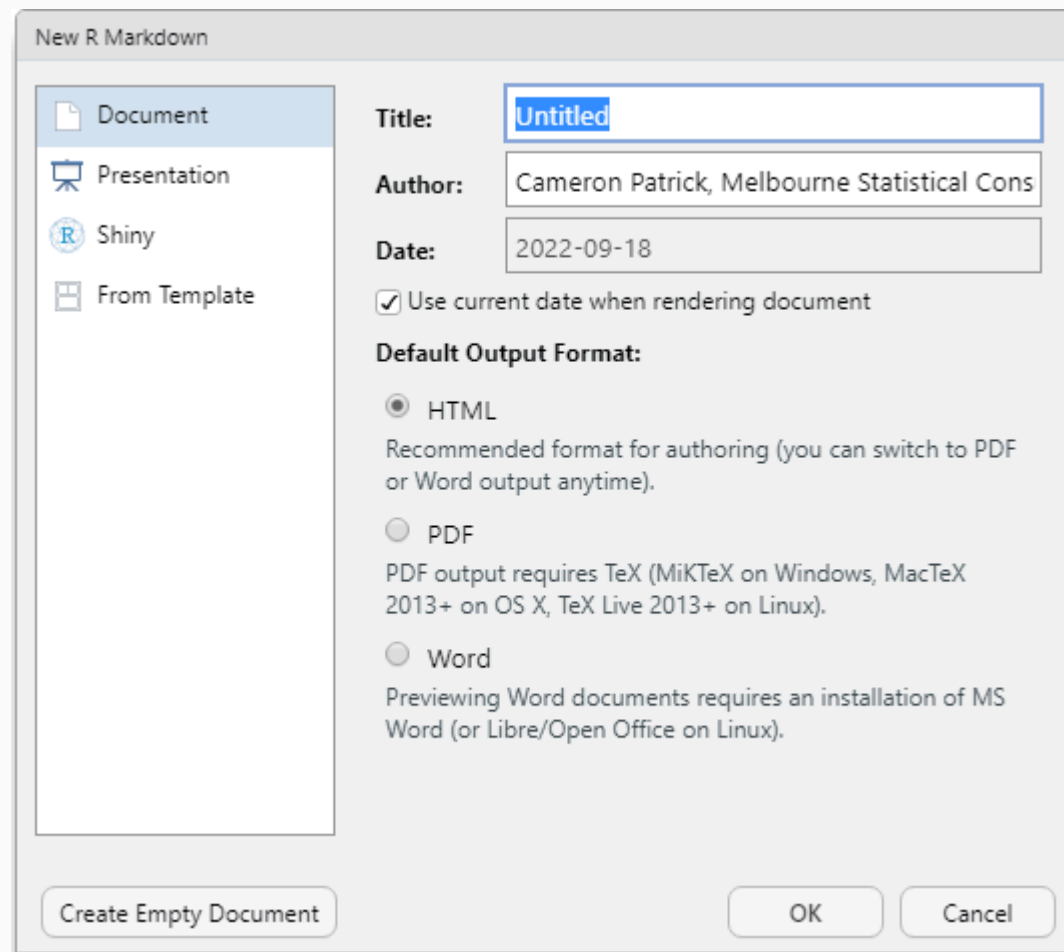
Entering code at the R console

The bottom-left pane in RStudio is the R console. It's good for experimenting. Code and output here isn't kept anywhere after you close RStudio.



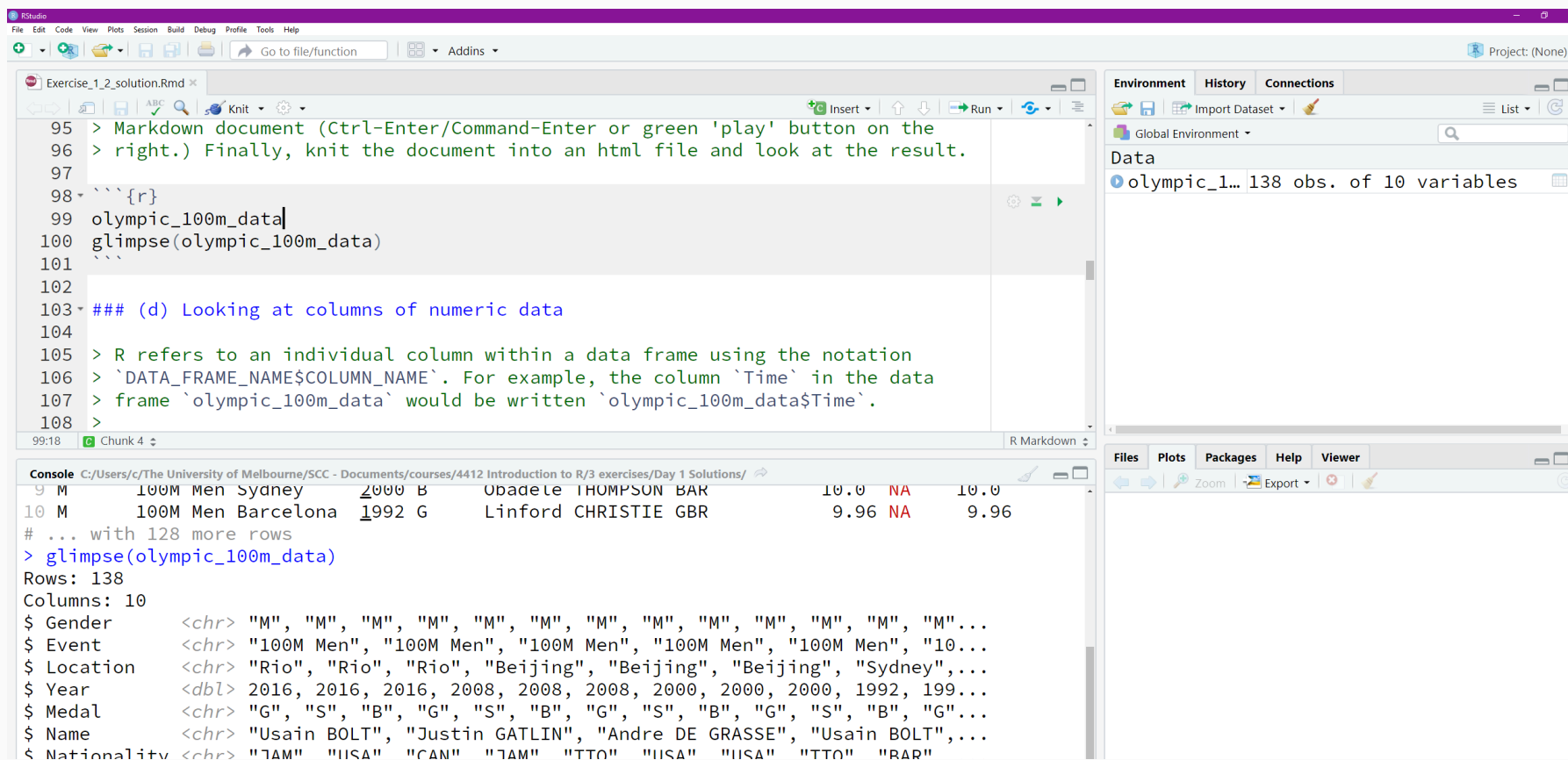
Creating a new R Markdown document

You won't need to do this during the course since you'll be working with the R Markdown documents we provide in the Exercises folder. For your own projects: **File > New File > R Markdown...**



R Markdown code chunks

Code chunks in your `.Rmd` file are for code you want to keep. **Make sure there's a blank line above and below the code chunk, or RStudio may get confused.** Click the green "play" button to the right of the chunk to run it. Think of the output as a preview to help you test your code rather than a permanent record: the "real" output is your knitted HTML file. If you've followed our setup instructions, output will go in the console. (RStudio default places it directly below the code in the editor.)



The screenshot shows the RStudio interface with a file named 'Exercise_1_2_solution.Rmd'. The editor contains the following R code and Markdown text:

```
95 > Markdown document (Ctrl-Enter/Command-Enter or green 'play' button on the
96 > right.) Finally, knit the document into an html file and look at the result.
97
98 ```{r}
99 olympic_100m_data
100 glimpse(olympic_100m_data)
101 ```
102
103 ### (d) Looking at columns of numeric data
104
105 > R refers to an individual column within a data frame using the notation
106 > `DATA_FRAME_NAME$COLUMN_NAME`. For example, the column `Time` in the data
107 > frame `olympic_100m_data` would be written `olympic_100m_data$Time`.
108 >
```

The console output shows the result of the `glimpse()` function:

```
Console C:/Users/c/The University of Melbourne/SCC - Documents/courses/4412 Introduction to R/3 exercises/Day 1 Solutions/
9 M      100M Men Sydney      2000 B      Obadete THOMPSON BAR      10.0 NA      10.0
10 M      100M Men Barcelona  1992 G      Linford CHRISTIE GBR      9.96 NA      9.96
# ... with 128 more rows
> glimpse(olympic_100m_data)
Rows: 138
Columns: 10
$ Gender      <chr> "M", "M", "M", "M", "M", "M", "M", "M", "M", "M", "M", "M", "M"...
$ Event       <chr> "100M Men", "100M Men", "100M Men", "100M Men", "100M Men", "10...
$ Location    <chr> "Rio", "Rio", "Rio", "Beijing", "Beijing", "Beijing", "Sydney",...
$ Year       <dbl> 2016, 2016, 2016, 2008, 2008, 2008, 2000, 2000, 2000, 1992, 199...
$ Medal      <chr> "G", "S", "B", "G", "S", "B", "G", "S", "B", "G", "S", "B", "G"...
$ Name       <chr> "Usain BOLT", "Justin GATLIN", "Andre DE GRASSE", "Usain BOLT",...
$ Nationality <chr> "JAM" "USA" "CAN" "JAM" "TTO" "USA" "USA" "TTO" "BAR"
```

R Markdown code chunk options

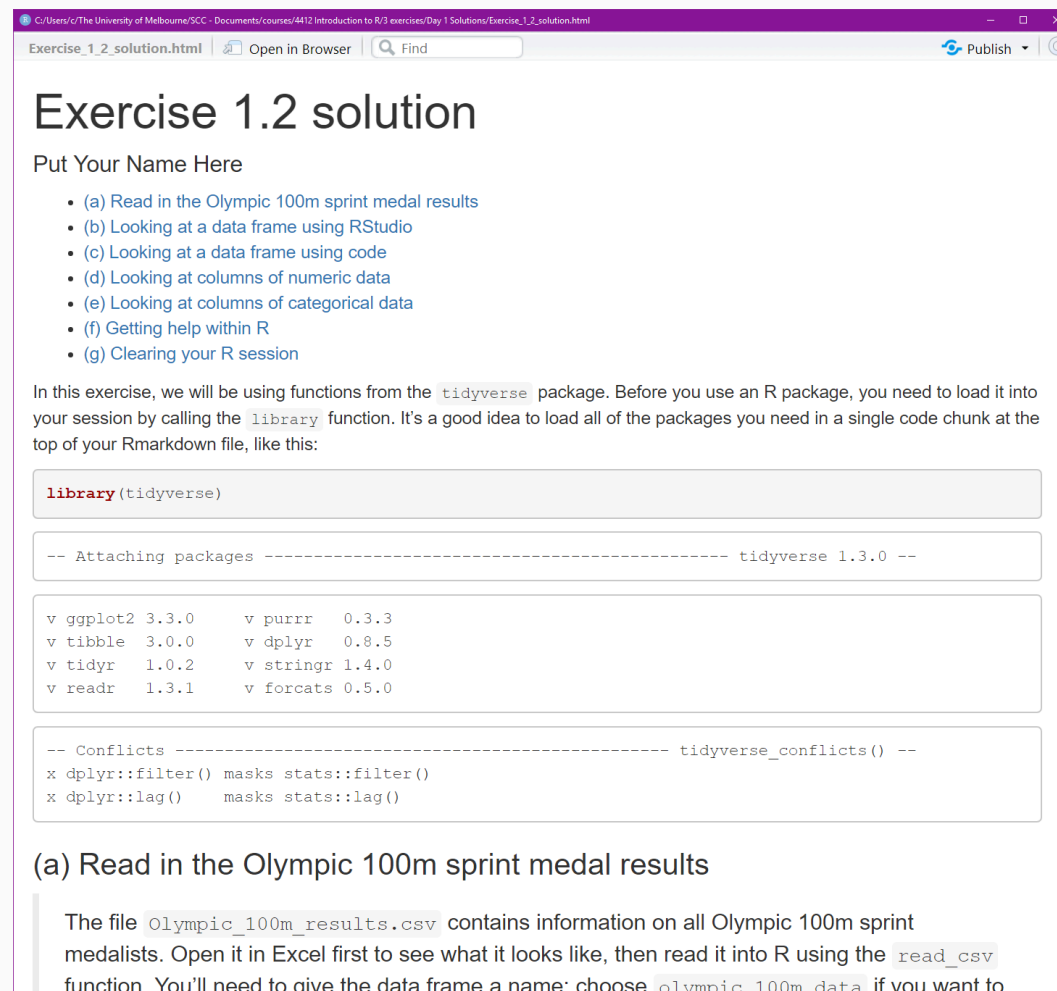
We can include options within the curly braces of a code chunk. The main option you will see utilised in this workshop is the chunk name or label.

```
31 ▾ ### (a) Loading the Tidyverse
32
33 Throughout the course, we will be using functions from the `tidyverse` package.
34 Before you use an R package, you need to load it into your session by calling
35 the `library` function. It's a good idea to load all of the packages you need
36 in a single code chunk at the top of your R Markdown file, like this:
37
38 ▾ ```{r packages}
39 library(tidyverse)
40 ▴ ```
41
42 You can insert new code chunks either by typing in the triple backticks and
43 curly braces by hand, using the **Insert > R** button in RStudio, or pressing
44 Control-Alt-I. (The keyboard shortcut won't work on University owned machines
45 due to software installed by central IT.)
46
47 You can run a code chunk by pressing the green "play" button on the right of
48 the chunk, or pressing Ctrl-Shift-Enter when the cursor is in the chunk.
49 There are more options for running code in the **Run** button above
50 and to the right of the code editor.
51
52 Code chunks can have a name (like packages above) or they can be unnamed
53 (like the one below). Code chunks can also have options - for example, the
54 setup chunk at the top of this file has an option include = FALSE which
55 prevents it from showing up in the knitted document.
56
```

Chunk names are not necessary but have some advantages. Chunk names must be unique and cannot have spaces.

Knit your document to produce an HTML file

This is the final product of your analysis, produced in a reproducible way: starting with fresh R session with no data or packages loaded, running all of your code in order. Knit early, knit often.



The screenshot shows a web browser window with the address bar displaying 'Exercise_1_2_solution.html'. The page content includes a title 'Exercise 1.2 solution', a placeholder 'Put Your Name Here', a list of seven tasks (a) through (g), a paragraph explaining the use of the tidyverse package, and three code blocks. The first code block contains `library(tidyverse)`. The second code block contains a comment '-- Attaching packages ---- tidyverse 1.3.0 --' followed by a table of installed packages and their versions. The third code block contains a comment '-- Conflicts ---- tidyverse_conflicts() --' followed by a list of conflicts between dplyr and stats packages. The page also features a 'Publish' button in the top right corner.

Exercise 1.2 solution

Put Your Name Here

- (a) Read in the Olympic 100m sprint medal results
- (b) Looking at a data frame using RStudio
- (c) Looking at a data frame using code
- (d) Looking at columns of numeric data
- (e) Looking at columns of categorical data
- (f) Getting help within R
- (g) Clearing your R session

In this exercise, we will be using functions from the `tidyverse` package. Before you use an R package, you need to load it into your session by calling the `library` function. It's a good idea to load all of the packages you need in a single code chunk at the top of your Rmarkdown file, like this:

```
library(tidyverse)
```

```
-- Attaching packages ---- tidyverse 1.3.0 --
```

v ggplot2 3.3.0	v purrr 0.3.3
v tibble 3.0.0	v dplyr 0.8.5
v tidyr 1.0.2	v stringr 1.4.0
v readr 1.3.1	v forcats 0.5.0

```
-- Conflicts ---- tidyverse_conflicts() --
```

```
x dplyr::filter() masks stats::filter()
x dplyr::lag() masks stats::lag()
```

(a) Read in the Olympic 100m sprint medal results

The file `Olympic_100m_results.csv` contains information on all Olympic 100m sprint medalists. Open it in Excel first to see what it looks like, then read it into R using the `read_csv` function. You'll need to give the data frame a name: choose `olympic_100m_data` if you want to

Loading the tidyverse package

We're going to be using functions from the `tidyverse` package throughout this course, so all of our R Markdown files will begin by loading it:

```
library(tidyverse)
```

```
install.packages("light")
```



```
library("light")
```



Images sourced from <https://www.wikihow.com/Change-a-Light-Bulb>

R packages and sessions

R packages need to be installed (once) and loaded (every R session).

Best practice: load all needed R packages in a single code chunk, near the top of your `.Rmd` file.

Don't put the code to install packages in your R Markdown files or other R scripts.

Use **Session > Restart R** to give yourself a clean slate (especially if R is acting weird).

Importing data from a CSV file

```
pig_behaviour_data <- read_csv("pig_behaviour_by_time.csv")
```

The `read_csv()` function loads data stored in a CSV file into an R data frame. File names go inside quotation marks `"` (or `'`).

The left-arrow `<-` stores the result of an R statement in an object (variable) with a name you choose.

- Always give your object meaningful names.
- Don't use the same name to store two different things in the same `.Rmd` file.

Technical note: that's `read_csv` with an underscore, which is part of Tidyverse; not `read.csv` with a dot, which is part of base R. `read.csv` will often produce slightly different output, sometimes fails to read CSV files saved by Excel, and is very slow for big files - so we always recommend `read_csv`.

How does R find my data?

Every R session has a **working directory**.

When knitting a document, this will always be the folder your `.Rmd` file is in.


In RStudio generally: could be anywhere, depending on your settings.

If in doubt: **Session > Set Working Directory > To Source File Location**.

Set it in the R terminal: `setwd('...')`

Check it in the R terminal: `getwd()`

Hard-coded paths make colleagues sad



The first line of your R script was

```
setwd("C:\\Users\\smith\\path\\that\\only\\I\\have")
```

Now nothing works for me...

Examining a data frame using code

```
glimpse(pig_behaviour_data)
```

Rows: 280

Columns: 16

```
$ Pen          <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 1...
$ Housing      <chr> "FC", "FC", "FC", "FC", "FC", "FC", ...
$ Treatment    <chr> "HC", "HC", "HC", "HC", "HC", "HC", ...
$ HousTreat    <chr> "FC, HC", "FC, HC", "FC, HC", "FC, H...
$ Sex          <chr> "M", "M", "M", "M", "M", "M", "M", "...
$ Total_pigs   <dbl> 10, 10, 10, 10, 10, 10, 10, 10, 10, ...
$ Time         <chr> "15 mins", "15 mins", "15 mins", "15...
$ Time_brief   <chr> "15m", "15m", "15m", "15m", "15m", "...
$ Time_hours   <dbl> 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, ...
$ Upright      <dbl> 10, 10, 8, 10, 10, 10, 10, 10, 10, 1...
$ Aggression   <dbl> 0, 0, 1, 2, 0, 2, 0, 0, 0, 2, 0, 0, ...
$ Chewing_pig  <dbl> 2, 0, 3, 0, 0, 0, 2, 1, 0, 3, 5, 0, ...
$ Playing      <dbl> 0, 1, 0, 0, 3, 4, 2, 1, 0, 0, 1, 0, ...
$ Vocalising   <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
$ Exploring_pen <dbl> 8, 10, 5, 8, 7, 6, 6, 8, 7, 6, 7, 9,...
$ Nosing_pen   <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, ...
```

Some more R language

An R expression or statement can be:

- A mathematical calculation, e.g. `24 * 60 * 60`

- Saving a result in an object,

e.g. `seconds_per_day <- 24 * 60 * 60`

Almost anything in R can be saved in an object, including data frames, statistical models, and plots.

- Calling a function, e.g. `log10(seconds_per_day)`
- A few special constructs used for programming
- Any of the above combined (lego bricks!)

You can refer to a particular column of a data frame as

`DATA_FRAME_NAME$COLUMN_NAME`,

e.g. `pig_behaviour_data$Time`

Some R functions are mathematical or statistical. Others are more computer-y.

```
mean(Weight, na.rm = TRUE)
```

```
mean(x = Weight, na.rm = TRUE)
```

```
mean(Weight, , TRUE)
```

The things inside the brackets are called *parameters* or *arguments*. They are given meaning either by their position (e.g. first argument to `mean` is a vector to calculate the mean of) or their name (e.g. `na.rm` indicates missing values should be ignored).

All of the above calls to `mean()` do the same. Which do you prefer?

Getting help

Inside R

Press F1 when the cursor is on a function name in a code block. (Some laptops, e.g. MacBooks, require Fn+F1.)

```
?functionname
```

```
help("functionname")
```

```
help(package = "packagename")
```

```
help.search("keyword")
```

Tip: use these in your R console (used for playing and experimenting), not your R Markdown file (used for code and results you want to keep). The last command above searches all documentation text.

Online resources

Stack Overflow <https://stackoverflow.com/>
for programming and technical questions

Cross Validated <https://stats.stackexchange.com/>
for statistical questions

Google search, paying special note to results from the above.

RStudio Cheatsheets (**Help > Cheatsheets > ...** or online)

ChatGPT is good for programming questions, including "how do I" - but can be confidently wrong for conceptual statistical questions

Reading data from an Excel file

Use the `read_excel()` function in the `readxl` package.

Near the top of your R Markdown, make sure to load the package:

```
library(readxl)
```

e.g. To read the first sheet of an Excel file:

```
bird_data <- read_excel("bird_data.xlsx")
```

e.g. To read the third sheet of an Excel file:

```
bird_data <- read_excel("bird_data.xlsx", sheet = 3)
```

e.g. To read the sheet called "Magpies" of an Excel file:

```
magpie_data <- read_excel("bird_data.xlsx",  
                          sheet = "Magpies")
```


Extension: .rds files, R's native data format

The `.rds` file format has a couple of advantages:

- it saves a data frame (or any other data in R) exactly as it is stored inside R
- it's highly compact so takes up much less space and is fast to load or save

Load data using the `readRDS()` function:

```
magpie_data <- readRDS("magpies.rds")
```

You probably don't have any `.rds` files lying around, but you can save data out of R using the `saveRDS()` function:

```
saveRDS(magpie_data, "magpies.rds")
```

This is useful if you have a more complex data set.

Extension: reading SPSS and Stata data files

The `haven` package provides functions for reading data files in formats used by other popular statistical software, including SPSS and Stata.

Load data from a Stata `.dta` file using `read_dta()`:

```
library(haven)
pigeon_data <- read_dta("pigeons.dta")
```

Load data from an SPSS `.sav` file using `read_sav()`:

```
library(haven)
survey_data <- read_sav("survey.sav")
```

Including a data frame directly in your code

The `tribble()` function makes a data frame out of some data directly included in your R code.

It's most useful for small amounts of data.

```
melbourne_postcodes <- tribble(  
  ~postcode, ~suburb_name,  
  3000,      "Melbourne",  
  3002,      "East Melbourne",  
  3003,      "West Melbourne",  
  3051,      "North Melbourne",  
  3205,      "South Melbourne"  
)  
melbourne_postcodes
```

```
# A tibble: 5 × 2  
  postcode suburb_name  
    <dbl> <chr>  
1     3000 Melbourne  
2     3002 East Melbourne  
3     3003 West Melbourne  
4     3051 North Melbourne  
5     3205 South Melbourne
```

Making vectors

Sometimes it's useful to manually construct vectors (columns) in some way. Here are a few ways to do this:

```
c(1900, 1926, 1939, 1940, 1941, 1948, 1955,  
   1956, 1957, 1959, 1960, 1964, 2021)
```

```
[1] 1900 1926 1939 1940 1941 1948 1955  
[8] 1956 1957 1959 1960 1964 2021
```

```
c("Monday", "Wednesday", "Friday")
```

```
[1] "Monday"      "Wednesday" "Friday"
```

```
1:10
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
2001:2022
```

```
[1] 2001 2002 2003 2004 2005 2006 2007  
[8] 2008 2009 2010 2011 2012 2013 2014  
[15] 2015 2016 2017 2018 2019 2020 2021  
[22] 2022
```

```
seq(1, 10)
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
seq(1, 20, by = 3)
```

```
[1] 1 4 7 10 13 16 19
```

```
seq(100, 500, length.out = 5)
```

```
[1] 100 200 300 400 500
```

```
rep(42, times = 5)
```

```
[1] 42 42 42 42 42
```

```
rep(c(1, 2, 3), times = 5)
```

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

```
rep(c(1, 2, 3), each = 5)
```

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

Making data frames out of vectors

The `tibble()` function makes a data frame (tibble) out of a separate vector for each column.

```
hipster_postcodes <- tibble(  
  postcode = c(3055, 3056, 3057, 3070),  
  suburb_name = c("Brunswick West", "Brunswick", "Brunswick East", "Northcote")  
)  
hipster_postcodes
```

```
# A tibble: 4 × 2  
  postcode suburb_name  
    <dbl> <chr>  
1     3055 Brunswick West  
2     3056 Brunswick  
3     3057 Brunswick East  
4     3070 Northcote
```

Exercise 1.2.