R Faculty Workshop

Workshop overview

Day 1: Intro to R

Intro to R

Some notes about R and its ecosystem in general

Basics of writing R code

Syntax, data types, assignment, functions

Working with data

Importing data, cleaning, graphs, analyses

Day 2: Teaching with R

General thoughts on teaching with R

Pedagogy, grading, generative AI

Posit Cloud

A web-based version of RStudio

Quarto

An alternative document format with advantages for teaching

Example course materials

Examples of how I use R and these additional tools in Statistics and Labs

1 Introduction to R and RStudio

1.1 Why R?

R is a coding language specialized for statistical computing and data analysis. It is free and open-source (though there is a cloud-based version which can be paid for and can have advantages especially in the classroom).

Some of R's capabilities:

- Import and create data files in various formats
- Clean and organize data
- Analyze and visualize the data
- Communicate the results in various formats (pdf research paper, website, presentation slides)
- Generate random data, execute functions repeatedly, useful for simulations, bootstrapping etc
- Other programattic tasks, e.g. web-scraping, using APIs

1.2 The general workflow

1.2.1 Manipulating data

It might seem daunting to learn R if you have no experience with coding, but the basic idea is that you have some data, like you are familiar with from a regular Excel or Google Sheets spreadsheet, and you perform operations on your data using functions a lot like you would in Excel/Sheets. For example, you might compute an average in Sheets by typing `=AVERAGE(A1:A10)`. In R you might type `mean(my_data$column_a)`. The specifics of the function names are different, but the basic idea is the same.

1.2.2 Separation of data and code

A major difference between working with data in Excel vs. R is the separation of data from code. Rather than writing functions to manipulate or analyze data directly in your spreadsheet, code is written in a separate code file, which references **but does not modify** the source data file (unless you tell it to).

Excel Spreadsheet

A
1
2
3
4
5
=AVERAGE(A2:A6)

R Data

A
1
2
3
4
5

R Code

```
mean(data$A)
```

```
[1] 3
```

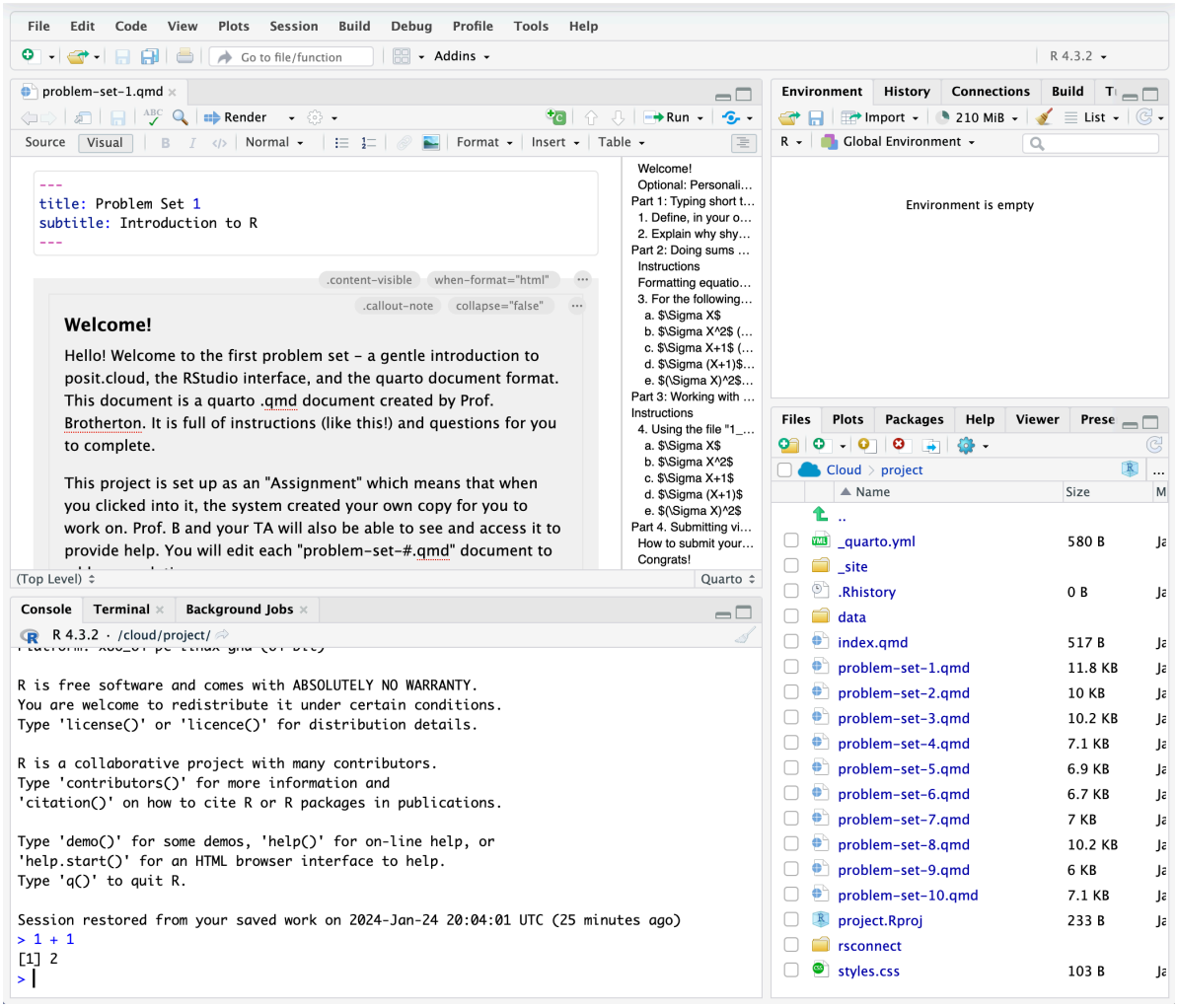
```
sum(data$A)
```

```
[1] 15
```

```r
sd(data$A)
```

```
[1] 1.581139
```

1.2.3 RStudio Interface



RStudio is the interface we'll use to write and run R code and see its output. The basic interface has 4 panels, each with a few tabs:

- Top-left: Code editor / data viewer
  - Open, edit, and save code documents
  - Execute code within files

  - View data
  - You can have multiple 'tabs' open at once,

- Bottom-left: R console
  - You can type code directly and run it by pressing enter.
  - You won't be saving your code as a document like when you type in in the editor, so this is useful for testing something simple out

- Top-right: Environment
  - As you execute code you may be creating objects like sets of numbers of data.frames. Those objects will appear here.
  - You can click the name of some objects, like data.frames, and it will open a view of the data as a tab in the editor pane

- Bottom-right: Files/folders, Plots, Viewer, help window
  - You can navigate the file tree

1.3 Additional packages

The R language has many functions built in. Generally speaking, you can find a way to do pretty much anything you would like to do using just 'base' R.

However there are many common tasks that are a bit tedious or unintuitive to do using base R. One of R's strengths is how extensible it is: anyone can write their own functions, turn the code into an R package, and make that package available to other R users.

1.3.1 Tidyverse

Actually, the tidyverse package is a container for multiple individual packages. The whole family of tidyverse packages are written with a consistent syntax and logic.

1.3.2 Specialized analyses

E.g....

- Structural equation modeling (`lavaan`)
- Meta-analysis (`metafor`)
- Linear mixed effects models (`lme4`, `simr`)
- Bootstrapping (`boot`)
- Bayesian analyses (`brms`, `rstanarm`)

- Network analyses (`igraph`, `ggraph`, `tidygraph`, `qgraph`, `bootnet`)
- Language analysis (`tidytext`, `quanteda`)
- Audio analysis (`tuneR`, `seewave`)
- Machine learning (`tidymodels`)

1.3.3 Additional capabilities

E.g. maps (`sf`, `leaflet`)

1.3.4 Installing packages

```
install.packages("tidyverse")

install.packages("lme4")
```

Packages only need to be installed on your system once.

1.3.5 Using packages

If you are just using one function from a package as a one-off, you can use the double-colon `::` operator in the form `package::function()`, i.e.

```
# package::function syntax

dplyr::filter(...)

tidyr::pivot_longer(...)

lme4::lmer(...)
```

If you will be using a package's functions repeatedly, it can be preferable to activate the entire package using the `library()` function.

```
# activate installed packages first with library()

library(tidyverse)
library(lme4)

# then use functions

lmer(...)
```

Note that a package only needs to be installed once on your system (or in a new posit.cloud project), but if you are using the `library()` method to activate the package, it must be done every time you have a new 'session' in R.

1.4 Help!

There is a help documentation page for every function. You can access it by typing a question mark and then the name of the function in the console and hitting Enter/Return:

```
?mean

?t.test
```

Doing so brings up the function documentation in the Help pane in the bottom-right of the RStudio interface.

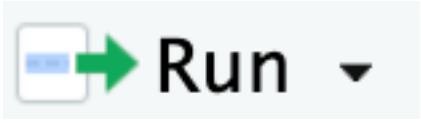
Alternatively, you can click into the Help pane directly and type a function or topic into the search bar near the top of the pane.

2 Basics of writing R code

> **Resources**
>
> Download this file to accompany this section:
>
> - my_first_r_file.R
>
> After you save it, double click it and it should open in the Editor pane in RStudio.

2.1 Writing and running code

Writing some code in an .R document does not cause it to be executed automatically. You need to run the code yourself. You can run a single line of code at a time, or a whole section, or an entire script.

- One line at a time:
  - Run button at top-right of editor pane 

  - Command (or Ctrl) Return (advances cursor to next line)
  - Option (or Alt) + Return (does not advance cursor)
- Whole script
  - Source (runs code, doesn't show output)
  - Source with Echo (shows output)

2.1.1 Let's run some code

To start getting used to writing and running code, let's use R as a calculator to do some sums.

```
1 + 1
```

```
[1] 2
```

```
(-3)^2
```

```
[1] 9
```

```
# here's a comment. comments do not get executed even if they contain valid code

# 2 + 2

# write a sum of your own.
# run the code and make sure you get the answer you're expecting
```

3 Data structures

3.1 Vectors

In R, a *vector* is a collection of values of a single type of data. You can make one by using the `c()` function to collect things together.

```
# numeric
c(1, 2, 3, 4, 5)
```

```
[1] 1 2 3 4 5
```

```
# colon can be used to produce a vector of integers
1:5
```

```
[1] 1 2 3 4 5
```

```
5:1
```

```
[1] 5 4 3 2 1
```

```
1 # is just a numeric vector of length 1
```

```
[1] 1
```

3.1.1 Other common data types

```
# character
c("hello", "world")
```

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

```
# logical
c(TRUE, FALSE)
```

```
[1]  TRUE FALSE
```

3.2 Assignment

R has a fancy assignment operator: `<-`.[1]

You assign things to a name by typing something like:

```
numbers <- c(1, 2, 3, 4, 5)
```

Almost anything can be assigned to a name. In the example here the vector `c(1, 2, 3, 4, 5)` was assigned to the name `numbers`. But in other situations you might assign an entire dataset, a statistical model object, a function, or something else. Whatever it is you're assigning, giving it a name allows to you perform subsequent operations more easily, and choosing appropriate names makes your code easier to understand.

3.2.1 Valid names

The `name` can be almost anything you like; it just can't start with a number or contain spaces or special characters other than `_` (underscore) and `. (period)`. It can have uppercase characters as well as lowercase, but note that when it comes time to use the name later you will need to type it exactly right, including capitalization. So you can make life a little easier for yourself by using a consistent naming convention, ideally avoiding capital letters altogether.

```
# valid name examples

data <- "works"

good_name <- "fine"

.ValidName <- "works, watch out for the capitals"
```

[1] Most other coding languages tend to use a boring `=` for assignment. Sure it's nice not having to type an extra character, but there's a keyboard shortcut to quickly add an `<-` in RStudio: Option/Alt + `-`. And philosophically, the `<-` arrow conveys the inherent directionality of the assignment operation. The object is assigned to the name; the object and its name are not equal and so the `=` arguably gives a misleading impression of the two things being one and the same. (Also, to let you in on a secret, `=` also works for assignment in R.)

```
long_name_for_a_variable <- "pixels are free, but time is limited"
```

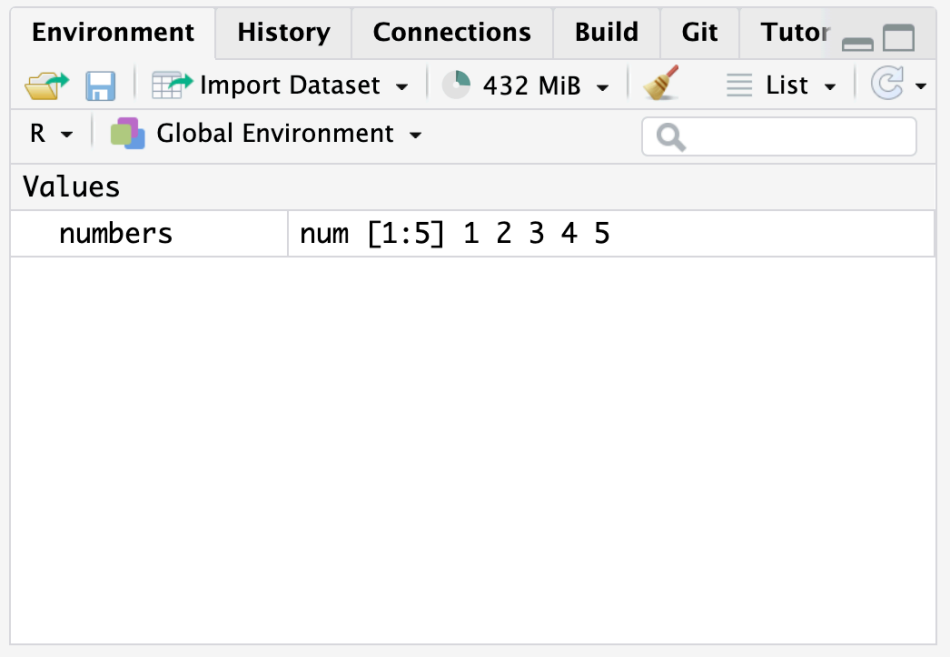
```
# invalid names

1badname <- "won't work"

worse name <- "can't have spaces"

# backticks allow for otherwise unacceptable names
`1bad name` <- "will work"
```

3.2.2 The Global Environment



When you run the code `numbers <- c(1, 2, 3, 4, 5)` or any other assignment operation, you generally won't see any output in the console. What you will see, however, is something new appear in your Global Environment, the pane in the top-right of the RStudio window. You have brought your named object into being.

Just as easily as you can bring an object into existence, so too can you remove it. Clicking the charming little sweeping brush near the top of the Environment pane will remove everything that currently exists from your Environment. It'll even ask if you're sure–a rare instance of compassion on R's part. It's a good habit not to be precious about the objects in your

Environment. Your code is the definite record and if the instructions to make something are in there it can always be recreated.

```
# removing objects from the Environment

rm(numbers) # remove a particular object by name

rm(list = ls()) # remove everything in the global environment

# or you can click the charming little sweeping brush
```

3.2.3 Overwriting

Note that R will allow you to reuse a name that you already assigned something to. It will simply replace the thing that the name refers to. It will not ask if you're sure you want to do that; it won't mention it at all.

```
number <- 1

number * 2
```

```
[1] 2
```

```
number <- 2

number * 2
```

```
[1] 4
```

It is entirely possible for you to run lines of code out of order and get potentially confusing results as a consequence. You might also run a line of code more than once, modifying an object in a way that you might not intend.

```
number <- 1

number <- number * 2 # what if you run this a few times?

number
```

```
[1] 2
```

3.3 Functions

Many of the things we eventually want to do involve functions. To use a function, type its name, followed by parentheses. Any inputs or other arguments you need to specify go inside the parentheses.

```
sum(c(1, 2, 3, 4, 5)) # sum() takes a numeric vector as input
```

```
[1] 15
```

```
# use sum() to get the total of a vector of numbers of your own
# run the code and make sure you get the answer you're expecting
```

3.3.1 Using functions with named objects

Most usefully, we can use a named object we have created as input to a function. So rather than having to type or copy/paste the original vector `c(1, 2, 3, 4, 5)`, we can give it a name and feed that name into a function that expects a numeric vector as input.

```
numbers <- c(1, 2, 3, 4, 5)

sum(numbers)
```

```
[1] 15
```

```
length(numbers)
```

```
[1] 5
```

```
mean(numbers)
```

```
[1] 3
```

```
sd(numbers)
```

```
[1] 1.581139
```

```
min(numbers)
```

```
[1] 1
```

```
max(numbers)
```

```
[1] 5
```

3.3.2 Function arguments

A function generally has one or more "arguments", to which you supply parameters. For example, the `mean()` function's first argument is the set of numbers you want to compute the mean of.

When there's more than one argument, they are separated by a comma. Arguments usually have names. You don't necessarily have to type the name of the argument, because of R's positional matching.

The `seq()` function, for example, produces a sequence of numbers according to three arguments, `from`, `to`, and `by`.

```
seq(from = 1, to = 10, by = 2)
```

```
[1] 1 3 5 7 9
```

If you don't type the names of the arguments, and just supply three values, R matches them by position, so this gives exactly the same output as the previous line of code because `from`, `to`, and `by` are the first three arguments respectively.

```
seq(1, 10, 2) # gives same result as above
```

```
[1] 1 3 5 7 9
```

Suppose we actually wanted a sequence of 6 values. We could use the `length.out` argument. Now we definitely have to type the name at least of the `by` and `length.out` arguments, because positional matching won't work.

```
seq(from = 1, by = 2, length.out = 6) # argument names required
```

```
[1]  1  3  5  7  9 11
```

So when do you type the argument names explicitly? Definitely when you need to, and maybe when you don't: remember someone (including your future self, might eventually want to read and understand your code.

3.3.3 Nesting functions

You can also nest functions inside one another. Make sure all the closing parentheses match up.

```
sqrt(mean(seq(1, 10, 2)))
```

```
[1] 2.236068
```

3.3.4 Getting help with functions

Remember, you can get help with a function (to see what arguments it accepts, for example) by typing a question mark followed by the function name (without parentheses) in your console.

```
?mean
```

Running the code will bring up the function's help documentation in RStudio's Help pane.

3.4 Doing stuff with named vectors

3.4.1 Indexing

You can access individual element of a vector by supplying an index within square brackets.

```
numbers <- c(3, 1, 4, 1, 5, 9)

numbers[1] # first element
```

```
[1] 3
```

```r
numbers[1:3] # multiple consecutive elements
```

```
[1] 3 1 4
```

```r
# can you pick out the 1st, 3rd, and 5th elements?
```

Note that R uses 1-indexing: the first element's index is 1. This differs from many other coding languages which are 0-indexed.

3.4.2 Doing math with vectors

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

numbers * 2
```

```
[1]  2  4  6  8 10
```

```r
6 - numbers
```

```
[1] 5 4 3 2 1
```

```r
numbers * c(1, 2)
```

```
Warning in numbers * c(1, 2): longer object length is not a multiple of shorter
object length
```

```
[1] 1 4 3 8 5
```

```r
numbers * numbers
```

```
[1]  1  4  9 16 25
```

3.4.3 Combining math and functions

```r
sd(numbers) / length(numbers) # standard error
```

```
[1] 0.3162278
```

```r
numbers - mean(numbers) # deviations
```

```
[1] -2 -1  0  1  2
```

```r
(numbers - mean(numbers))^2 # squared deviations
```

```
[1] 4 1 0 1 4
```

```r
# can you compute the sum of squared deviations?
```

3.4.4 Checking conditions

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

3 == numbers
```

```
[1] FALSE FALSE  TRUE FALSE FALSE  TRUE
```

```r
3 != numbers
```

```
[1]  TRUE  TRUE FALSE  TRUE  TRUE FALSE
```

```r
# check if a value is in a vector at least once
3 %in% numbers
```

```
[1] TRUE
```

```r
6 %in% numbers
```

```
[1] FALSE
```

```
# check if something is FALSE
!6 %in% numbers
```

```
[1] TRUE
```

```
3 >= numbers
```

```
[1]  TRUE  TRUE  TRUE FALSE FALSE  TRUE
```

3.4.5 Combining conditions with indexing

```
numbers[numbers > 3]
```

```
[1] 4 5
```

3.5 Other important things to know

3.5.1 Vector coercion

Every element in a vector must be of the same type (numeric, character, logical). If that is not the case, R will coerce the data into a single type.

```
numbers <- c(1, 2, 3, 4, 5)
numbers
```

```
[1] 1 2 3 4 5
```

```
numbers <- c(1, 2, "three", 4, 5)
numbers
```

```
[1] "1"     "2"     "three" "4"     "5"
```

```
numbers <- c(1, 2, "3", 4, 5)
numbers
```

```
[1] "1" "2" "3" "4" "5"
```

```r
mean(numbers)
```

```
Warning in mean.default(numbers): argument is not numeric or logical: returning
NA
```

```
[1] NA
```

3.5.2 Coercion confusion

Coercion can have some confusing consequences, if you are taken unawares by mixed data types.

```r
1 < "2"
```

```
[1] TRUE
```

```r
22 < "11"
```

```
[1] FALSE
```

```r
3 > "two"
```

```
[1] FALSE
```

```r
# why?
```

3.5.3 Coercion side effects

Coercion can have some happy consequences. For instance, logical values (`TRUE` and `FALSE`) can be coerced into the numbers 1 and 0. A function that requires numeric input, such as `sum()` or `mean()`, if given logical input, will coerce the vector to numeric.

```r
# doing math with logicals

bool <- c(TRUE, FALSE, FALSE, TRUE)
bool
```

```
[1]  TRUE FALSE FALSE  TRUE
```

```r
as.numeric(bool)
```

```
[1] 1 0 0 1
```

```r
sum(bool) # count of TRUEs
```

```
[1] 2
```

```r
mean(bool) # proportion of TRUEs
```

```
[1] 0.5
```

3.5.4 Factors

A factor is a special data type in R used to represent categorical data. Internally, it stores the data as integers, but each unique integer is associated with a text label (the *level*) for that category.

```r
data <- c("female", "male", "male", "female")

data_factor <- factor(data)

data_factor
```

```
[1] female male   male   female
Levels: female male
```

```r
as.numeric(data_factor)
```

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

Perhaps our raw data coded a variable like this as numeric to begin with, and we want to add the category labels ourselves.

```r
data <- c(1, 2, 2, 1) # gender coded numerically

factor(data, levels = c(1, 2), labels = c("female", "male"))
```

```
[1] female male   male   female
Levels: female male
```

3.5.4.1 Ordered factors

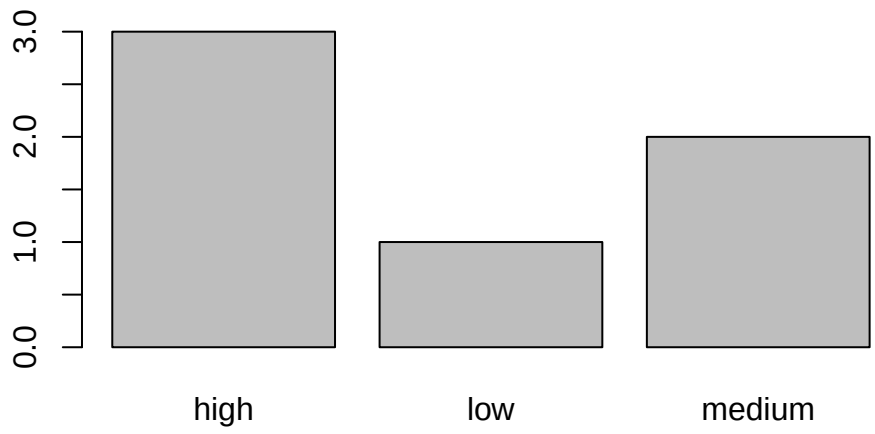
For ordinal data, where the order of categories matters, we can specify the levels in order and use the `ordered = TRUE` argument.

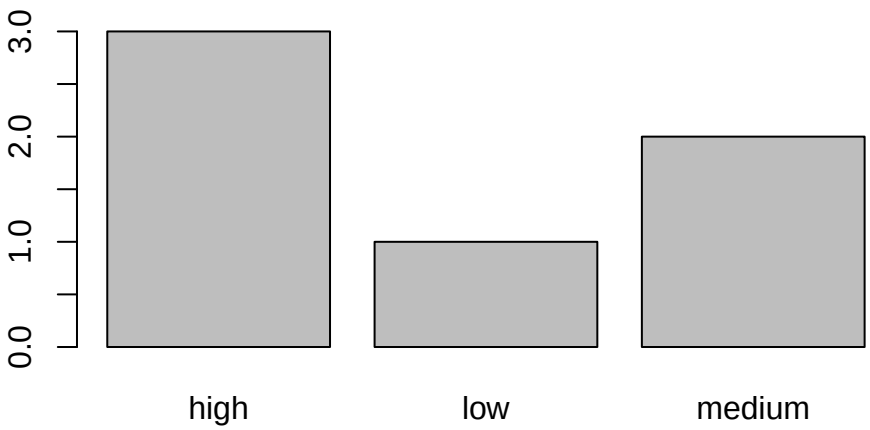
```
# with no order specified; levels are listed alphabetically

data <- c("medium", "low", "high", "medium", "high", "high")

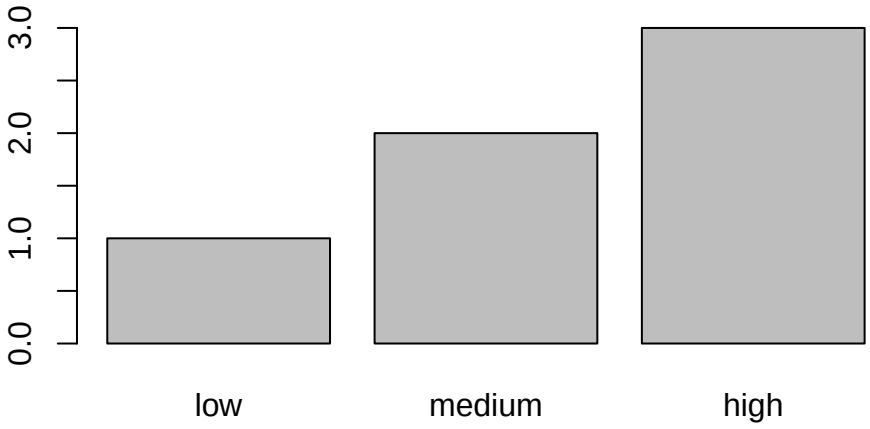
unordered <- factor(data)

plot(unordered)
```






```
ordered <- factor(data,
                  levels = c("low", "medium", "high"),
                  ordered = TRUE)
```



Note that this does not affect the raw data, but it means that if we plot a graph using this ordered factor later on, the values will appear in their correct, meaningful order, rather than just the default alphabetical order.

3.5.5 Missing Values

To anticipate a problem we often run into when working with real data, sometimes our data includes missing values. R has a special representation for missing values: `NA`.

```
numbers <- c(1, 2, NA, 4, 5)

mean(numbers)
```

```
[1] NA
```

```
# can you solve the problem by looking at the help page for the mean function?
```

3.6 Data.frames

So far we've been working with individual vectors. Sooner of later we're going to want to work with a collection of different sets of numbers: a spreadsheet. R's name for this kind of data structure is a *data.frame*. A data.frame is a collection of vectors; each column is a vector. Different columns can have different types (numeric, character, logical, date, etc), but each column will contain a single type of data. All columns must have the same length.

Most commonly we have a data file already (a .csv or maybe an Excel file or some other format) and we read it in to R. However, to get a sense of how these objects work, and how to work with them, we can make one from scratch.

```
df <- data.frame(a = c(1, 2, 3),
                 b = c("one", "two", "three"),
                 c = c(1, 2, "3"),
                 d = c(1, 2, NA),
                 e = c(TRUE, FALSE, FALSE),
                 f = factor(c("female", "female", "male")))

str(df)
```

```
'data.frame':   3 obs. of  6 variables:
 $ a: num  1 2 3
 $ b: chr  "one" "two" "three"
 $ c: chr  "1" "2" "3"
 $ d: num  1 2 NA
 $ e: logi  TRUE FALSE FALSE
 $ f: Factor w/ 2 levels "female","male": 1 1 2
```

:::
format:
html:
default
revealjs:
output-
file:
1_3_working-
with-
data_presentation.html

```
::: callout-note
## Start a new Project

In RStudio, click:

`File > New Project > New Directory > New Project`

Once you have created the Project, save these files into your Project folder:

-   ![](images/r-logo.png){height="0.8em" style="vertical-align: baseline;"} [triplett_analys
-   ![](images/table-icon.svg){height="0.8em" style="vertical-align: baseline;"} [triplett_da
```

````
Back in RStudio you should see those files appear in the Files pane in the bottom-right. Clic
:::

### The working directory

::: {.content-hidden when-format="revealjs"}
R can access your entire filesystem, so you can access and create files anywhere on your hard

`"/Users/robertbrotherton/Documents/r-workshop/triplett_data.csv"`

Another big drawback is that if you share your code for someone else to run on their own comp

Projects help to overcome these kinds of issues. When working in a project, the 'working dire
:::




::: {.cell}

```{.r .cell-code}
# check your current working directory
getwd()
```

::: {.cell-output .cell-output-stdout}

```
[1] "/Users/robertbrotherton/Documents/r-workshop"
```



:::

```{.r .cell-code}
# should be your project folder
```
:::




Whenever you use a function that requires you to specify a filename, the function will be loo
````

````
::: {.cell}

```{.r .cell-code}
read_csv(file = "triplett_data.csv")
```
:::
````

...and if the file is in your Project folder it will be found. And if someone else runs the

Getting packages ready

````
::: {.content-hidden when-format="revealjs"}
One of the strengths of R as a language for data analysis is its ecosystem of additional pac

The packages will need to be installed once, if they aren't already on your system. Once you
:::
````

````
::: {.cell}

```{.r .cell-code}
# install external packages if you don't already have them

# install.packages(c("tidyverse", "corrplot","effectsize", "lme4", "lmerTest))
```
:::
````

Then you can activate the packages with the \`library() function.

````
::: {.cell}

```{.r .cell-code}
library(tidyverse)
```

::: {.cell-output .cell-output-stderr}

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.0.4
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```


:::

```{.r .cell-code}
library(effectsize)
library(corrplot)
```

::: {.cell-output .cell-output-stderr}

```
corrplot 0.95 loaded
```


:::

```{.r .cell-code}
library(lme4)
```
````

```
::: {.cell-output .cell-output-stderr}

```
Loading required package: Matrix

Attaching package: 'Matrix'

The following objects are masked from 'package:tidyr':

    expand, pack, unpack
```


:::
:::

Importing data

::: {.content-hidden when-format="revealjs"}
R has a built-in function to read data from a .csv (comma-separated values) file like the one
:::

::: {.cell}

```{.r .cell-code}
triplett_data <- read_csv("triplett_data.csv")

# no output, but check your Global Environment
```
:::

::: {.cell}
::: {.cell-output .cell-output-stderr}

```
Rows: 40 Columns: 12
-- Column specification --------------------------------------------------------
```
```

```
Delimiter: ","
chr (4): subject, gender, group, classification
dbl (8): age, alone_0, competition_1, alone_1, competition_2, alone_2, compe...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

:::
:::

::: {.content-hidden when-format="revealjs"}
The raw comma-separated-values data get interpreted as a data.frame object that exists in R's

Note that as a data analysis project becomes more complicate it may be useful to keep data fi
:::

::: {.cell}

```{.r .cell-code}
# if your data was in a subdirectory...

triplett_data <- read_csv("data_raw/triplett_data.csv")
```
:::

Working with other data formats

R can import (and write) many other data types, should the need arise.

````
::: {.cell}

```{.r .cell-code}
# Stata
haven::read_stata("stata_file.dta")

# SPSS
haven::read_spss("spss_file.sav")

# SAS
haven::read_sas("sas_file.sas")

# Excel
readxl::read_excel("excel_file.xlsx", sheet = "sheet name")
```
:::




## Piping

::: {.content-hidden when-format="revealjs"}
You can string together different operations in a pipeline using the pipe operator: `|>`.[^1]
:::

[^1]: If you're looking at R code from elsewhere (e.g. looking up help online) you may see a




::: {.cell}

```{.r .cell-code}
my_data |>
  filter(a > 3)

# is equivalent to

filter(my_data, a > 3)
```
:::
````

```
::: {.content-hidden when-format="revealjs"}
The real power of this become apparent when you need to conduct a more elaborate sequence of
:::




::: {.cell}

```{.r .cell-code}
# building a more elaborate pipeline

my_data |>
  filter(a > 3) |>
  mutate(c = a + b) |>
  select(b, c)
```
:::

Data cleaning

`Select`ing columns

Sometimes your raw data file has more columns that you need. `dplyr`'s `select()` function le

::: {.cell}

```{.r .cell-code}
triplett_data |>
  select(subject, age, gender)

# you can also rename as you select
triplett_data |>
  select(participant = subject, age, gender)
```
```

```
:::

### `Filter`ing rows

`select()` allows you to pick which columns you want; `filter()` allows you to pick which rou

::: {.cell}

```{.r .cell-code}
triplett_data_subset <- triplett_data |>
  filter(group == "A")
```
:::

You can also specify multiple conditions as necessary. All conditions must evaluate to `TRUE

::: {.cell}

```{.r .cell-code}
triplett_data_subset <- triplett_data |>
  filter(group == "A", age >= 10)
```
:::

`Mutate` some data

::: {.content-hidden when-format="revealjs"}
```

````
The `dplyr` function `mutate()` creates new columns or modifies existing columns. The general
:::

#### Modify an existing column




::: {.cell}

```{.r .cell-code}
triplett_data <- triplett_data |>
  mutate(gender = factor(gender),
         group = factor(group))
```
:::




#### Create a new column




::: {.cell}

```{.r .cell-code}
triplett_data_recoded <- triplett_data |>
  mutate(alone_mean = rowMeans(across(contains("alone"))))

# produces NAs for some participants! oh no! why? can you fix it?
```
:::

::: {.cell}

```{.r .cell-code}
triplett_data_recoded <- triplett_data |>
  mutate(alone_mean = rowMeans(across(contains("alone")), na.rm = TRUE))

# can you add code to get the mean competition score, and a difference score?
````
````

```
triplett_data_recoded <- triplett_data |>
  mutate(alone_mean = rowMeans(across(contains("alone")), na.rm = TRUE),
         competition_mean = rowMeans(across(contains("competition")), na.rm = TRUE),
         diff = competition_mean - alone_mean)
```
:::

Mutate and `case_when()`

What if you need to create a new variable which differs depending on the value of an existing

::: {.cell}

```{.r .cell-code}
triplett_data_recoded <- triplett_data_recoded |>
  mutate(effect = case_when(
    diff < -sd(diff) ~ "improved",
    diff > sd(diff) ~ "impaired",
    TRUE ~ "no difference"
  ))
```
:::

The conditions are checked in order. The final step, `TRUE ~ "no difference"` is the default

Reshaping

Sometimes it is useful to reshape data from wide to long format. The Triplett data in its ini

::: {.cell}

```{.r .cell-code}
triplett_long <- triplett_data |>
  pivot_longer(contains(c("alone", "competition")),
               names_to = c("condition", "trial"),
               names_sep = "_",
               values_to = "performance")
```
:::

Saving cleaned data

Once you have a new version of your data, you may wish to save it as a new file for easy sha

One option is to save it as an R data file. Obviously this is specialized for R and cannot be

::: {.cell}

```{.r .cell-code}
saveRDS(triplett_long, "triplett_long.RDS")
```
:::

Another option is to save it as a .csv file.

::: {.cell}

```{.r .cell-code}
readr::write_csv(triplett_long, "triplett_long.csv")
```
:::

````
## Data exploration

### Descriptive statistics

A quick and easy way to get some summary statistics for a data.frame is to use the `summary()`




::: {.cell}

```{.r .cell-code}
triplett_data_recoded |>
  select(age, gender, group, alone_mean, competition_mean, diff) |>
  summary()
```

::: {.cell-output .cell-output-stdout}

```
      age        gender group    alone_mean    competition_mean
 Min.   : 8.00   f:26   A:20   Min.   :27.67   Min.   :27.20
 1st Qu.:10.75   m:14   B:20   1st Qu.:34.04   1st Qu.:33.05
 Median :11.00                 Median :39.22   Median :36.98
 Mean   :11.50                 Mean   :39.48   Mean   :37.41
 3rd Qu.:13.00                 3rd Qu.:44.62   3rd Qu.:41.43
 Max.   :17.00                 Max.   :57.13   Max.   :50.73
      diff
 Min.   :-6.4000
 1st Qu.:-4.1250
 Median :-1.7417
 Mean   :-2.0694
 3rd Qu.: 0.1625
 Max.   : 2.0000
```

:::
:::
````

Frequencies

::: {.cell}

```{.r .cell-code}
triplett_data_recoded |>
  count(gender, classification) |>
  mutate(prop = n / sum(n), .by = gender)
```

::: {.cell-output-display}

gender	classification	n	prop
f	little affected	6	0.2307692
f	stimulated adversely	5	0.1923077
f	stimulated positively	15	0.5769231
m	little affected	4	0.2857143
m	stimulated adversely	5	0.3571429
m	stimulated positively	5	0.3571429

:::
:::

Summarize

The `dplyr` function `summarize()` is a powerful way of producing summary statistics from a d

The difference between `summarize()` and `mutate()` is that `mutate()` modifies the full data

::: {.cell}

```{.r .cell-code}
triplett_data_recoded |>
  summarize(n = n(),
            mean_diff = mean(diff),
            sd_diff = sd(diff),
            range = max(diff) - min(diff))
```

::: {.cell-output-display}

n	mean_diff	sd_diff	range
40	-2.069417	2.475617	8.4

:::
:::

Summarize by group

`summarize()`'s superpower is it's special argument, `.by`. This lets us specify a grouping v

::: {.cell}

```{.r .cell-code}
triplett_data_recoded |>
  summarize(n = n(),
            mean_diff = mean(diff),
            sd_diff = sd(diff),
            range = max(diff) - min(diff),
            .by = gender)
```

::: {.cell-output-display}

gender	n	mean_diff	sd_diff	range

```
|f      | 26| -2.631795| 2.602639|  8.40|
|m      | 14| -1.025000| 1.884777|  6.05|

:::

```{.r .cell-code}
triplett_data_recoded |>
  summarize(n = n(),
            mean_diff = mean(diff),
            sd_diff = sd(diff),
            range = max(diff) - min(diff),
            .by = classification)
```

::: {.cell-output-display}

classification	n	mean_diff	sd_diff	range
stimulated positively	20	-3.841667	1.825137	6.550000
stimulated adversely	10	0.530000	1.346344	3.850000
little affected	10	-1.124333	1.494226	4.793333

:::
:::

Lots of grouping variables

You can have any number of grouping variables; just collect them together with the `c()` fun

::: {.cell}

```{.r .cell-code}
triplett_long |>
  summarize(average = mean(performance),
            .by = c(classification, condition, trial, group))
```
```

::: {.cell-output-display}

classification	condition	trial	group	average
stimulated positively	alone	0	A	47.49
stimulated positively	alone	1	A	42.60
stimulated positively	alone	2	A	38.42
stimulated positively	alone	3	A	NA
stimulated positively	competition	1	A	41.88
stimulated positively	competition	2	A	39.28
stimulated positively	competition	3	A	36.30
stimulated positively	alone	0	B	48.20
stimulated positively	alone	1	B	45.68
stimulated positively	alone	2	B	42.78
stimulated positively	alone	3	B	39.82
stimulated positively	competition	1	B	41.20
stimulated positively	competition	2	B	39.04
stimulated positively	competition	3	B	NA
stimulated adversely	alone	0	A	40.84
stimulated adversely	alone	1	A	39.72
stimulated adversely	alone	2	A	39.00
stimulated adversely	alone	3	A	NA
stimulated adversely	competition	1	A	43.32
stimulated adversely	competition	2	A	41.48
stimulated adversely	competition	3	A	37.40
stimulated adversely	alone	0	B	39.08
stimulated adversely	alone	1	B	36.48
stimulated adversely	alone	2	B	34.76
stimulated adversely	alone	3	B	34.56
stimulated adversely	competition	1	B	36.36
stimulated adversely	competition	2	B	36.44
stimulated adversely	competition	3	B	NA
little affected	alone	0	A	31.80
little affected	alone	1	A	30.80
little affected	alone	2	A	31.52
little affected	alone	3	A	NA
little affected	competition	1	A	30.24
little affected	competition	2	A	31.56
little affected	competition	3	A	31.32
little affected	alone	0	B	39.44
little affected	alone	1	B	33.64
little affected	alone	2	B	32.16
little affected	alone	3	B	32.56

```
|little affected       |competition |1     |B     |   32.84|
|little affected       |competition |2     |B     |   32.96|
|little affected       |competition |3     |B     |      NA|

:::
:::
```

Data Visualization

Using built in `plot`s

````
::: {.cell}

```{.r .cell-code}
hist(x = triplett_data_recoded$diff)
```

::: {.cell-output-display}
![](1_3_working-with-data_files/figure-pdf/unnamed-chunk-26-1.pdf){fig-pos='H'}
:::

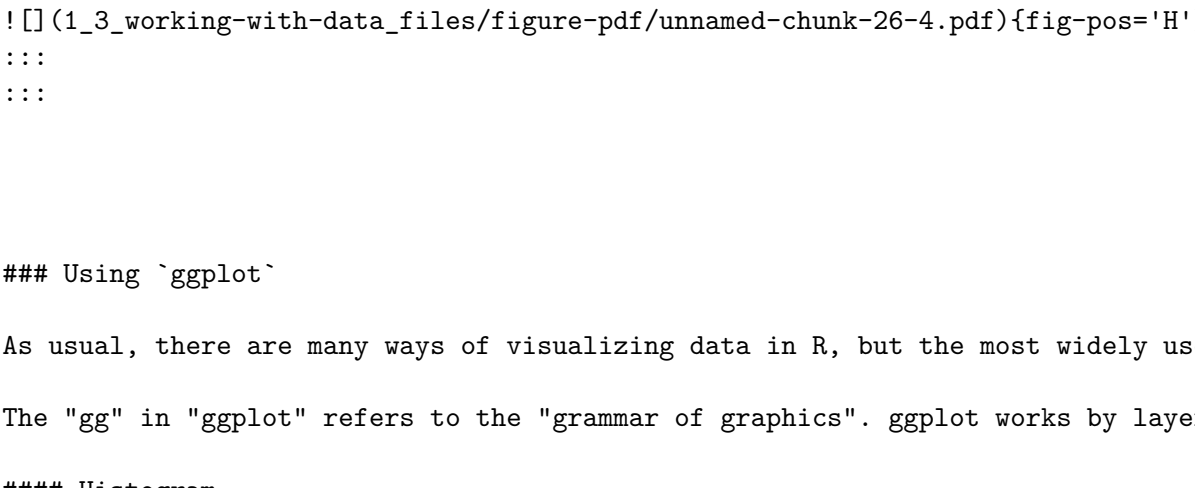
```{.r .cell-code}
plot(x = triplett_data_recoded$age,
     y = triplett_data_recoded$alone_0)
```

::: {.cell-output-display}
![](1_3_working-with-data_files/figure-pdf/unnamed-chunk-26-2.pdf){fig-pos='H'}
:::

```{.r .cell-code}
plot(diff ~ gender, data = triplett_data_recoded)
```

::: {.cell-output-display}
![](1_3_working-with-data_files/figure-pdf/unnamed-chunk-26-3.pdf){fig-pos='H'}
:::
````

```{.r .cell-code}
boxplot(diff ~ classification, data = triplett_data_recoded)
```

::: {.cell-output-display}

:::
:::

Using `ggplot`

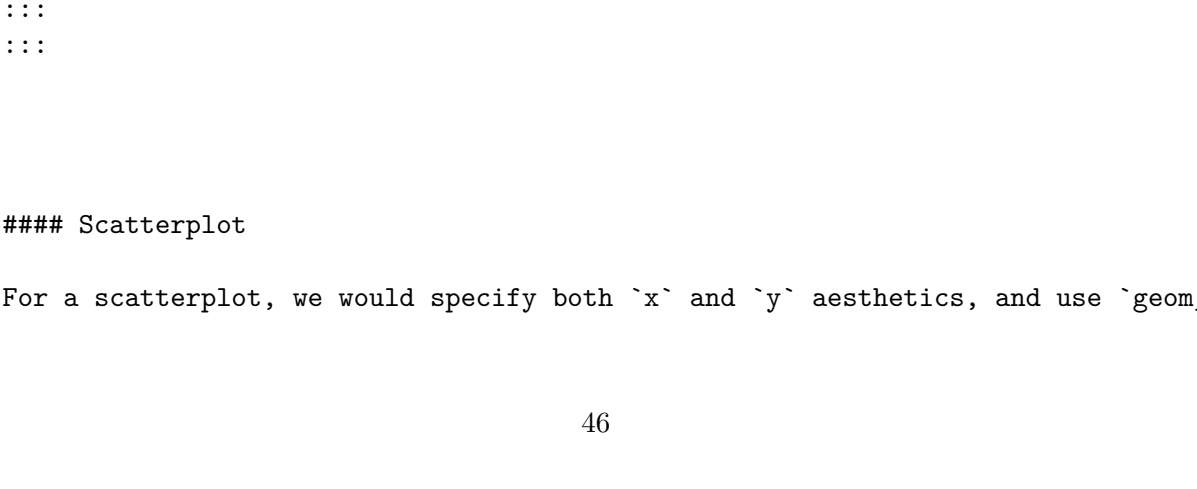
As usual, there are many ways of visualizing data in R, but the most widely used and flexible

The "gg" in "ggplot" refers to the "grammar of graphics". ggplot works by layering, using the

Histogram

::: {.cell}

```{.r .cell-code}
triplett_data_recoded |>
  ggplot(aes(x = diff)) +
  geom_histogram(bins = 10)
```

::: {.cell-output-display}

:::
:::

Scatterplot

For a scatterplot, we would specify both `x` and `y` aesthetics, and use `geom_point()` for t

````
::: {.cell}

```{.r .cell-code}
triplett_data_recoded |>
  ggplot(aes(x = age, y = alone_0)) +
  geom_point(position = "jitter")
```

::: {.cell-output-display}
![](1_3_working-with-data_files/figure-pdf/unnamed-chunk-28-1.pdf){fig-pos='H'}
:::
:::




#### Scatterplot with grouping variable

::: {.content-hidden when-format="revealjs"}
There are other aesthetics beyond `x` and `y`. We can also map data to a `color` or `fill` ae
:::




::: {.cell}

```{.r .cell-code}
triplett_data_recoded |>
  ggplot(aes(x = age, y = alone_0, color = gender)) +
  geom_point() +
  geom_smooth(method = "lm")
```

::: {.cell-output .cell-output-stderr}

```
`geom_smooth()` using formula = 'y ~ x'
```
````

````
:::

::: {.cell-output-display}
![](1_3_working-with-data_files/figure-pdf/unnamed-chunk-29-1.pdf){fig-pos='H'}
:::
:::




### Plotting descriptives




::: {.cell}

```{.r .cell-code}
triplett_data |>
  ggplot(aes(x = factor(age), y = alone_0, color = gender)) +
  geom_boxplot()
```

::: {.cell-output-display}
![](1_3_working-with-data_files/figure-pdf/unnamed-chunk-30-1.pdf){fig-pos='H'}
:::
:::




For more complex visuals, it can be useful to combine `summarize()` for computing summary sta




::: {.cell}

```{.r .cell-code}
triplett_long |>
  summarize(performance = mean(performance, na.rm = TRUE),
````
````

```
            .by = c(trial, condition)) |>
  ggplot(aes(x = trial, y = performance, fill = condition)) +
  geom_col(position = "dodge")
```

::: {.cell-output-display}

:::
:::

Data Analysis

Correlation

::: {.cell}

```{.r .cell-code}
cor.test(triplett_data$age, triplett_data$alone_0)
```

::: {.cell-output .cell-output-stdout}

```
    Pearson's product-moment correlation

data:  triplett_data$age and triplett_data$alone_0
t = -3.2537, df = 38, p-value = 0.002394
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.6794952 -0.1817032
sample estimates:
       cor
-0.4667911
```

````
:::
:::



### Lots of correlations



::: {.cell}

```{.r .cell-code}
triplett_data |>
  select(contains("alone"), contains("competition")) |>
  cor(use = "pairwise") |>
  corrplot::corrplot(method = 'shade')
```

::: {.cell-output-display}
![](1_3_working-with-data_files/figure-pdf/unnamed-chunk-33-1.pdf){fig-pos='H'}
:::
:::



### $\chi^2$ test



::: {.cell}

```{.r .cell-code}
# are the genders differently distributed among Triplett's classification categories

chisq.test(x = triplett_data_recoded$classification,
           y = triplett_data_recoded$gender)
```

::: {.cell-output .cell-output-stderr}
````

```
Warning in chisq.test(x = triplett_data_recoded$classification, y =
triplett_data_recoded$gender): Chi-squared approximation may be incorrect
```

:::

::: {.cell-output .cell-output-stdout}

```

    Pearson's Chi-squared test

data:  triplett_data_recoded$classification and triplett_data_recoded$gender
X-squared = 1.978, df = 2, p-value = 0.3719
```

:::
:::

t-test

Independent-samples

R has a `t.test()` function built in. When the DV is in one column and the IV grouping varial

::: {.cell}

```{.r .cell-code}
# is there a difference between the genders?

t.test(alone_0 ~ gender, data = triplett_data)
```

::: {.cell-output .cell-output-stdout}

```
    Welch Two Sample t-test

data:  alone_0 by gender
t = 2.9599, df = 31.645, p-value = 0.005788
alternative hypothesis: true difference in means between group f and group m is not equal to
95 percent confidence interval:
  2.315627 12.551406
sample estimates:
mean in group f mean in group m
       45.41923        37.98571
```

:::

```{.r .cell-code}
effectsize::cohens_d(alone_0 ~ gender, data = triplett_data)
```

::: {.cell-output-display}

Cohens_d	CI	CI_low	CI_high
0.9234365	0.95	0.2360446	1.599793

:::
:::

Related-samples

For a related-samples t-test we would generally need wide-format data, where each row of data

::: {.cell}

```{.r .cell-code}
t.test(triplett_data_recoded$alone_mean,
       triplett_data_recoded$competition_mean,
       paired = TRUE)
```

::: {.cell-output .cell-output-stdout}

```
    Paired t-test

data:  triplett_data_recoded$alone_mean and triplett_data_recoded$competition_mean
t = 5.2868, df = 39, p-value = 5.048e-06
alternative hypothesis: true mean difference is not equal to 0
95 percent confidence interval:
 1.277676 2.861157
sample estimates:
mean difference
       2.069417
```

:::

```{.r .cell-code}
effectsize::cohens_d(triplett_data_recoded$alone_mean,
       triplett_data_recoded$competition_mean,
       paired = TRUE)
```

::: {.cell-output .cell-output-stderr}

```
For paired samples, 'repeated_measures_d()' provides more options.
```

:::

::: {.cell-output-display}

```
|  Cohens_d|   CI|    CI_low|  CI_high|
|---------:|----:|---------:|--------:|
| 0.8359196| 0.95| 0.4710662| 1.192793|
```

:::
:::

ANOVA

Independent-samples

The `aov()` function computes an ANOVA model. It accepts a `formula` in the form `DV ~ IV`, j

::: {.cell}

```{.r .cell-code}
aov(diff ~ classification, data = triplett_data_recoded)
```

::: {.cell-output .cell-output-stdout}

```
Call:
   aov(formula = diff ~ classification, data = triplett_data_recoded)

Terms:
                classification Residuals
Sum of Squares       139.31890  99.69957
Deg. of Freedom              2        37

Residual standard error: 1.641518
Estimated effects may be unbalanced
```

:::
:::

Sometimes, like with `aov()`, the function that computes a model doesn't tell us everything w

::: {.cell}

```{.r .cell-code}
anova <- aov(diff ~ classification, data = triplett_data_recoded)

summary(anova)
```

::: {.cell-output .cell-output-stdout}

```
               Df Sum Sq Mean Sq F value   Pr(>F)
classification  2  139.3   69.66   25.85 9.44e-08 ***
Residuals      37   99.7    2.69
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

:::
:::

Related-samples

The `aov()` and `summary()` approach also works for within-participants designs. We just neec

::: {.cell}

```{.r .cell-code}
aov(performance ~ condition + Error(subject/condition), data = triplett_long) |>
  summary()
```

::: {.cell-output .cell-output-stdout}

```
Error: subject
          Df Sum Sq Mean Sq F value Pr(>F)
condition  1     22   21.85   0.092  0.763
Residuals 38   8993  236.65

Error: subject:condition
          Df Sum Sq Mean Sq F value   Pr(>F)
condition  1  242.2   242.2   27.84 5.22e-06 ***
Residuals 39  339.3     8.7
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Error: Within
           Df Sum Sq Mean Sq F value Pr(>F)
Residuals 161   2346   14.57
```

:::
:::

Regression

::: {.cell}

```{.r .cell-code}
regression_model <- lm(diff ~ age * gender, data = triplett_data_recoded)
```

````
summary(regression_model)
```

::: {.cell-output .cell-output-stdout}

```

Call:
lm(formula = diff ~ age * gender, data = triplett_data_recoded)

Residuals:
    Min      1Q  Median      3Q     Max
-3.7626 -1.8964 -0.0238  1.7587  5.0268

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -5.4370     2.8466  -1.910   0.0641 .
age           0.2456     0.2457   0.999   0.3243
genderm       3.2319     6.1234   0.528   0.6009
age:genderm  -0.1442     0.5236  -0.275   0.7846
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.412 on 36 degrees of freedom
Multiple R-squared:  0.1238,    Adjusted R-squared:  0.05075
F-statistic: 1.695 on 3 and 36 DF,  p-value: 0.1854
```



:::

```{.r .cell-code}
broom::tidy(regression_model)
```

::: {.cell-output-display}

|term        |   estimate| std.error|  statistic|   p.value|
|:-----------|----------:|---------:|----------:|---------:|
|(Intercept) | -5.4369621| 2.8465542| -1.9100153| 0.0641236|
|age         |  0.2455702| 0.2457286|  0.9993555| 0.3242927|
|genderm     |  3.2318877| 6.1234287|  0.5277905| 0.6008830|
|age:genderm | -0.1442141| 0.5235967| -0.2754298| 0.7845609|
````
````

:::

```{.r .cell-code}
broom::glance(regression_model)
```

::: {.cell-output-display}

r.squared	adj.r.squared	sigma	statistic	p.value	df	logLik	AIC	BI
0.1237728	0.0507539	2.411975	1.695078	0.1853631	3	-89.86817	189.7363	198.180

:::
:::

Mixed-effects

::: {.cell}

```{.r .cell-code}
lmerTest::lmer(performance ~ condition * age + (1 | subject),
           data = triplett_long)
```

::: {.cell-output .cell-output-stdout}

```
Linear mixed model fit by REML ['lmerModLmerTest']
Formula: performance ~ condition * age + (1 | subject)
   Data: triplett_long
REML criterion at convergence: 1406.319
Random effects:
 Groups   Name        Std.Dev.
 subject  (Intercept) 5.109
 Residual             3.667
Number of obs: 241, groups:  subject, 40
```

```
Fixed Effects:
             (Intercept)      conditioncompetition                       age
                 61.8905                   -4.6967                   -1.9463
conditioncompetition:age
                  0.2281
```
```

:::

```{.r .cell-code}
mixed_model <- lmerTest::lmer(performance ~ condition * age + group + (1 | subject),
                          data = triplett_long)

summary(mixed_model)
```

::: {.cell-output .cell-output-stdout}

```
Linear mixed model fit by REML. t-tests use Satterthwaite's method [
lmerModLmerTest]
Formula: performance ~ condition * age + group + (1 | subject)
   Data: triplett_long

REML criterion at convergence: 1403

Scaled residuals:
    Min      1Q  Median      3Q     Max
-2.1757 -0.6269 -0.1108  0.4602  3.4458

Random effects:
 Groups   Name        Variance Std.Dev.
 subject  (Intercept) 26.52    5.150
 Residual             13.44    3.667
Number of obs: 241, groups:  subject, 40

Fixed effects:
                        Estimate Std. Error       df t value Pr(>|t|)
(Intercept)              61.8434     5.7664  41.4686  10.725 1.57e-13 ***
conditioncompetition     -4.6886     3.1964 199.8308  -1.467  0.14399
age                      -1.9923     0.5002  41.3136  -3.983  0.00027 ***
groupB                    1.1393     1.7148  37.1406   0.664  0.51056
```
```

```
conditioncompetition:age   0.2287     0.2753 199.8510   0.831  0.40713
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Correlation of Fixed Effects:
            (Intr) cndtnc age    groupB
cndtncmpttn -0.236
age         -0.977  0.230
groupB      -0.013  0.005 -0.138
cndtncmptt:  0.232 -0.988 -0.233  0.002
```

:::
:::

::: {.cell}

```{.r .cell-code}
triplett_long |>
  mutate(trial = as.integer(trial)) |>
  ggplot(aes(x = trial, y = performance, group = subject, color = condition)) +
  geom_line(alpha = 0.3) +
  stat_summary(aes(group = condition), fun = mean, geom = "line", size = 1.2, color = "black
  facet_wrap(~condition) +
  labs(title = "Performance over Trials by Condition",
       x = "Trial", y = "Reeling Speed (or whatever units)",
       caption = "Gray lines = individual subjects; bold line = group mean") +
  theme_minimal()
```

::: {.cell-output .cell-output-stderr}

```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
```

:::

::: {.cell-output .cell-output-stderr}

```
Warning: Removed 39 rows containing non-finite outside the scale range
(`stat_summary()`).
```

:::

::: {.cell-output .cell-output-stderr}

```
Warning: Removed 19 rows containing missing values or values outside the scale range
(`geom_line()`).
```

:::

::: {.cell-output-display}

:::
:::

`<!-- quarto-file-metadata: eyJyZXNvdXJjZURpciI6Ii4ifQ== -->`{=html}

```{=html}
<!-- quarto-file-metadata: eyJyZXNvdXJjZURpciI6Ii4iLCJib29rSXRlbVR5cGUiOiJjaGFwdGVyIiwiYm9va
```

General thoughts on teaching with R

```````{.quarto-title-block template='/Applications/quarto/share/projects/book/pandoc/title-
---
format:
  html: default
  revealjs:
    output-file: 2_1_general-approach_presentation.html

---
```````

In this session we'll start working in Posit Cloud. I've created a 'Space' for this workshop. You can join using this link: https://posit.cloud/spaces/647833/join?access_code=d044OrLrWY83pbYiQXdsnuYbxO1_vDN_rN1Q7pYx
I'll link you to the actual R materials in due course.

3.7 Start with something cool

posit.cloud/spaces/647833/content/10308914

3.8 Justify using R

- Skill Development: Proficiency in R is increasingly sought after in both academic and industry settings, enhancing students' career prospects. See : https://r4stats.com/articles/popularity/
- Reproducibility: R promotes transparent and reproducible research practices, aligning with open science principles.
- Cost-Effectiveness: R is free and open-source, making it accessible to all students regardless of financial resources.
- Conceptual Understanding: Coding statistical procedures reinforces comprehension of underlying concepts. For example, writing functions to compute z-scores or simulate sampling distributions deepens understanding.

Requiring students to learn both statistical concepts (or research methods or whatever) *and* coding is a lot to ask in a single semester. Setting expectations and stating goals can help students see learning R as worthwhile from the outset.

3.9 Grading

Learning R can be challenging, especially for students new to programming. In R, there is often more than one way to do things. Some approaches may be more effective and straightforward than others, but there is value in the process of discovery. Making mistakes and learning how to resolve them is a crucial part of learning to code. In setting coding assignments, I want to emphasize that the value is in the effort, even if it does not immediately produce the ideal outcome.

To foster a supportive learning environment:

- Effort-Based Grading: Assign grades based on the completeness and sincerity of attempts rather than correctness. This encourages experimentation and reduces anxiety.
- Feedback-Oriented: Provide constructive feedback to guide improvement, focusing on the learning process over the final product.

My grading scheme:

- 0 points: No submission
- 1 point: Incomplete attempt
- 2 point: Complete, valid attempt

That's it. Every student gets full credit for a valid attempt. Even if every answer is wrong, they get full credit for the effort. It is entirely possible for every student to get full credit for the coding assignments across the semester, even if they never get a single correct answer–as long as they try. Of course, generally students will learn from their mistakes and improve along the way.

The drawback to this is that it does not differentiate the students who put in great effort, going above and beyond expectations, from those who simply meet the minimum requirements of the assignments. I feel that is a worthwhile trade-off, especially since such efforts can be rewarded as part of a participation grade, for example.

3.10 Most common problems

Students often encounter specific hurdles when learning R:

- Annoying technical problems: Forgetting to install or load necessary packages leads to errors.
- Syntax Errors: Missing commas, parentheses, or quotation marks are common. Encourage the use of RStudio's syntax highlighting and error messages to identify issues.

- Understanding Data Structures: Differentiating between vectors, data frames, and lists can be confusing. Use analogies and visual aids to clarify these concepts.
- Annoying technical problems: Forgetting to install or load necessary packages leads to errors.
- Function Arguments: Misunderstanding default arguments or the order of parameters can cause unexpected results. Demonstrate how to find and read function documentation effectively.

To address these challenges:

- Incremental Learning: Introduce concepts gradually, building upon previous knowledge.
- Active Practice: Incorporate hands-on exercises that allow students to apply new skills immediately.

3.11 Additional resources

One of the nice things about R is that many academic users have developed great teaching materials and made them available for free. A few examples:

- Danielle Navarro (2019) Learning Statistics with R
- Mine Çetinkaya-Rundel and Johanna Hardin (2022) Modern Statistical Methods for Psychology
- Russell A. Poldrack (2018) Statistical Thinking for the 21st Century

3.12 Generative AI

In its latest version RStudio has Github Copilot, an LLM interface, built-in. It suggests code that might come after what you have typed so far. You can even just type a comment about what you would like to achieve, and Copilot will suggest code. Here, I typed the comment and the function name, and Copilot completed the function body.

```
# a function to compute the sum of squared deviations
sum_squares <- function(x) {
  n <- length(x)
  mean_x <- mean(x)
  sum_sq_dev <- sum((x - mean_x)^2)
  return(sum_sq_dev)
}
```

```r
sum_squares(c(1, 2, 3, 4, 5))
```

```
[1] 10
```

4 Posit Cloud

4.1 What's what

We've encountered a few different names at this point. To clarify the ecosystem:

- **R** is a coding language for statistics and data analysis
- **RStudio** is a software interface for writing and running R code
- **Posit** is the name of the company that develops RStudio (formerly called RStudio, Inc.)
- **posit.cloud** is the browser-based version of RStudio, hosted and managed by Posit

As we've seen, you can install R and RStudio on your own computer for free and do things that way, but using posit.cloud simplifies things immensely when it comes to using R in the classroom. It substantially lowers the barrier to entry; frees us to focus on statistical fluency rather than tool fluency.

4.2 Cloud vs local: main differences

Feature	Local RStudio	Posit Cloud
Setup required	Yes — install R + RStudio	None — runs in browser
Platform consistency	Varies by OS (Mac/Win/etc)	Identical for all students
File storage	On user's machine	In the cloud — per project
Package installation	Manual per machine	Bundled with assignments
Instructor access to work	Only if student sends files	Instructor can directly view projects
Collaboration	Tricky / manual	Easy via shared projects

4.2.1 Downsides / Limitations

- Must be online: offline work not possible
- Projects are somewhat slow to launch, even on fast internet (but once it's up and running the experience is virtually identical to local installation)

- Limited compute resources: not great for giant datasets, heavy ML or large simulations
- Free tier may throttle or limit use if you go over project/hour quotas

4.2.2 Cost

No cost to students. Potentially cost to instructor/department.

Tier:	Free	Instructor (pay per compute hour)	Instructor (pay per student)
Cost	$0	$15/month up to 300 compute hours, $0.10 per hour over 300	$15 per instructor + $5 per student per month
Compute hours included	25 (once you reach the cap, you can no longer open or create projects during your current month)	300 (pay for overage)	Unlimited
Shared Spaces	1 shared space with up to 5 members and 10 content items in the space	Unlimited	Unlimited
Projects	25	Unlimited	Unlimited

4.3 Projects and assignments

4.3.1 Projects (cloud vs local)

Projects in the cloud function much like local R Projects: they provide a somewhat self-contained environment for collecting together R files, data files, etc relating to a particular project. However, Projects in the cloud are even more encapsulated than local Projects.

Locally, all your projects will be able to make use of any external packages installed on your system. In the cloud, each project essentially runs on its own system; packages must be installed in every new Project. A cloud-based Project also has its own filesystem; data files stored in one project in the cloud cannot be accessed in another Project; the file must be manually uploaded to each Project.

Any cloud Project in Your Workspace can be shared with All Posit Cloud users. When you share the link, a user can click into it. They will initially be working with a temporary copy of the project. They can click a button to save their own permanent copy. Note that sharing

a Project this way does not allow other users to modify your copy of the Project in any way; they will always be working on their own copy.

4.3.2 Assignments

Any posit.cloud Project in a Space you have created (but not in Your Workspace) can be made an **Assignment**. This has some advantages over just sharing Projects in the way described above. Primarily, as soon as a user opens the Assignment, the system creates a copy for them in the Space. The user does not have to click manually to save their own version.

4.3.2.1 Instructor workflow:

- Create a Space for your course (potentially a new Space each semester)
- Create a Project with all required files and packages
- Convert it to an Assignment
- Invite students to join the Space
- Share the Assignment link with students (or they can see it listed in the Space)
- Each student gets their own private copy when they click into it

4.3.2.2 You (the instructor) can:

- View their copies
- Open their Projects directly
- Run/debug their code
- Leave feedback in-line

4.3.2.3 Benefits:

- No setup required by students
- No broken paths, version mismatches, or install errors
- Packages installed in your source version will be already installed in students' copies
- See all student work in one place
- Effortless sharing of starter code, datasets, templates

5 Quarto

5.1 What is Quarto?

"**An open-source scientific and technical publishing system**"

Works seamlessly within RStudio

Next generation version of R Markdown.

Provides an integrated workflow: Combines narrative text with code, output and visualizations in a single document, or cohesive collection of documents.

5.1.1 Output formats

- Single documents
  - PDF article
  - Word document
  - HTML article
  - HTML presentation
- Collection of documents
  - Book (html and/or PDF)
  - Website (html pages, navigation)

5.1.2 Benefits of using Quarto over just R

- Reproducibility: Ensures that analyses and results can be consistently replicated, fostering transparency in research and teaching.
- Engaging Teaching Materials: Create interactive documents that combine theoretical explanations with practical code examples.
- Efficient Assignment Management: Design assignments where students can execute and modify code within the same document, streamlining grading and feedback.
- Professional Reporting: Generate publication-ready reports and presentations directly from your analyses.

5.2 Working with Quarto documents

5.3 Getting started

5.3.1 A single document

File > New File > Quarto Document

YAML section

```
---
title: "Untitled"
---
```

Text.

Code chunks.

```
# hello!
```

5.3.2 Typing text

Source vs Visual

5.3.3 Code chunks

5.3.4 Rendering

Use the "Render" button in RStudio to execute all code and generate the final output. Each render starts from a clean environment, ensuring that all results are reproducible; all code gets executed from scratch.

5.3.5 A Quarto Project

5.3.5.1 Project Features

The `_quarto.yml` file.

Contains rendering instructions for the entire project, and things that will apply to each file.

Listing 5.1 `_quarto.yml`

```yaml
project:
  type: website

format:
  html:
    toc: true
```

5.4 Advanced Features

5.4.1 Citations and References

Manage bibliographies using .bib files.

Insert citations using @citation_key.

Automatically generate reference sections.

5.4.2 Cross-referencing

Label figures and tables for easy cross-referencing within the text.

An image can be give an ID when inserted, e.g.

```
![The R Logo](images/r-logo.png){#fig-logo}
```

Figure 5.1: The R Logo

Typing "`As shown in @fig-logo, R has a perfectly pleasant logo.`" becomes:

As shown in Figure 5.1, R has a perfectly pleasant logo.

5.5 Additional Resources

- Tutorial: Hello, Quarto
- Quarto OfficialDocumentation
- Carpentries: Reproducible Publications with Quarto
- Quarto Manuscripts
- Quarto Gallery (Examples)

6 Course Materials

6.1 Statistics

> **PSYC BC1101 Materials**
>
> My Statistics materials are online here:
> https://robbrotherton.github.io/bc1101/

Structure of recitation:

- I talk through a topic hands-on for around 10 or 15 minutes, noting the new functions and ideas students will encounter, how they map on to content we covered in lectures, and flagging up potential sticking points in advance.
- Then I let students continue working. I encourage them to work independently, to collaborate, and/or to seek help from me or their TA as and when the need arises.
- The students are working in a single Quarto document for each problem set. The doc contains Instructional blocks and questions with space for students to answer using text and/or code. Instructions contain hints that show the general structure of the solution; students adapt those hints to get the final solution.
- Feedback: Following submission of their attempt, students receive full, detailed feedback noting both what they did well and what they could have done differently.

6.2 Labs

> **Social lab materials**
>
> My Social lab manual is online here:
> https://robbrotherton.github.io/bc2137/

For my labs, I have put my materials together as a Quarto website. Every lab session has its own page (including the syllabus on the site's home page).

In conjunction with that, I have a posit.cloud Project containing the required data files, and report templates.