

R for the Excel User

Julie Lowndes & Allison Horst

2019-09-29

Contents

1	Welcome	7
1.1	Prerequisites	7
2	Overview	9
2.1	Summary (a few sentences)	9
2.2	Objectives (more detailed, bulletpoints?)	9
2.3	Resources	9
2.4	Overview	9
2.5	RStudio Orientation	12
2.6	R Scripts	15
2.7	Don't save the workspace	16
2.8	Deep thought: keep the raw data raw.	16
2.9	Interludes (deep thoughts/openscapes)	16
2.10	Our Turn Your Turn 1	16
2.11	Our Turn Your Turn 2	16
2.12	Efficiency Tips	16
3	readxl	17
3.1	Summary	17
3.2	Objectives	17
3.3	Resources	17
3.4	Lesson	18
3.5	Fun facts ideas:	26
3.6	Interludes (deep thoughts/openscapes)	26
3.7	Activity 1	26
3.8	Activity 2	26
3.9	Efficiency Tips	26
4	RMarkdown	27
4.1	Summary (a few sentences)	27
4.2	Objectives (more detailed, bulletpoints?)	27
4.3	Resources	27
4.4	Lessons teaching for each objective..... (objectives, examples) . .	27

4.5	Fun facts (quirky things) - making a note of these wherever possible for interest (little “Did you know?” sections)	27
4.6	Interludes (deep thoughts/openscapes)	27
4.7	Our Turn Your Turn 1	27
4.8	Our Turn Your Turn 2	27
4.9	Efficiency Tips	27
5	Dplyr and Pivot Tables	29
5.1	Summary (a few sentences)	29
5.2	Objectives (more detailed, bulletpoints?)	29
5.3	Resources	29
5.4	Lessons teaching for each objective..... (objectives, examples)	29
5.5	Fun facts (quirky things) - making a note of these wherever possible for interest (little “Did you know?” sections)	29
5.6	Interludes (deep thoughts/openscapes)	29
5.7	Our Turn Your Turn 1	29
5.8	Our Turn Your Turn 2	29
5.9	Efficiency Tips	29
6	Dplyr and vlookups	31
6.1	Summary (a few sentences)	31
6.2	Objectives (more detailed, bulletpoints?)	31
6.3	Resources	31
6.4	Lessons teaching for each objective..... (objectives, examples)	31
6.5	Fun facts (quirky things) - making a note of these wherever possible for interest (little “Did you know?” sections)	31
6.6	Interludes (deep thoughts/openscapes)	31
6.7	Our Turn Your Turn 1	31
6.8	Our Turn Your Turn 2	31
6.9	Efficiency Tips	31
7	Tidying	33
7.1	Better practices [needs a better name]	33
7.2	Summary (a few sentences)	33
7.3	Objectives (more detailed, bulletpoints?)	33
7.4	Resources	34
7.5	Lessons teaching for each objective..... (objectives, examples)	34
7.6	Fun facts (quirky things) - making a note of these wherever possible for interest (little “Did you know?” sections)	35
7.7	Interludes (deep thoughts/openscapes)	35
7.8	Our Turn Your Turn 2	35
7.9	Efficiency Tips	35
8	Formatting and Sharing	37
8.1	Summary (a few sentences)	37
8.2	Objectives (more detailed, bulletpoints?)	37

8.3	Resources	37
8.4	Lessons teaching for each objective..... (objectives, examples) . .	37
8.5	Fun facts (quirky things) - making a note of these wherever possible for interest (little “Did you know?” sections)	38
8.6	Interludes (deep thoughts/openscapes)	38
8.7	Our Turn Your Turn 1	38
8.8	Our Turn Your Turn 2	38
8.9	Efficiency Tips	38
9	Synthesis	39
9.1	Summary (a few sentences)	39
9.2	Objectives (more detailed, bulletpoints?)	39
9.3	Resources	39
9.4	Lessons teaching for each objective..... (objectives, examples) . .	39
9.5	Fun facts (quirky things) - making a note of these wherever possible for interest (little “Did you know?” sections)	39
9.6	Interludes (deep thoughts/openscapes)	39
9.7	Our Turn Your Turn 1	39
9.8	Our Turn Your Turn 2	39
9.9	Efficiency Tips	39

Chapter 1

Welcome

TODO: shorten, add about RStudio::conf

Excel is a widely used and powerful tool for working with data. As automation, reproducibility, collaboration, and frequent reporting become increasingly expected in data analysis, a good option for Excel users is to extend their workflows with R. Integrating R into data analysis with Excel can bridge the technical gap between collaborators using either software. R enables use of existing tools built for specific tasks and overcomes some limitations that arise when working with large datasets or repeated analyses. This course is for Excel users who want to add or integrate R and RStudio into their existing data analysis toolkit. Participants will get hands-on experience working with data across R, Excel, and Google Sheets, focusing on: data import and export, basic wrangling, visualization, and reporting with RMarkdown. Throughout, we will emphasize conventions and best practices for working reproducibly and collaboratively with data, including naming conventions, documentation, organization, all while “keeping the raw data raw”. Whether you are working in Excel and want to get started in R, already working in R and want tools for working more seamlessly with collaborators who use Excel, or whether you are new to data analysis entirely, this is the course for you!

1.0.1 About this book

1.1 Prerequisites

Before the training, please make sure you have done the following:

1. Download and install **up-to-date versions** of:
 - R: <https://cloud.r-project.org>
 - RStudio: <http://www.rstudio.com/download>

2. Install the Tidyverse
3. Get comfortable: if you're not in a physical workshop, be set up with two screens if possible. You will be following along in RStudio on your own computer while also watching a virtual training or following this tutorial on your own.

Chapter 2

Overview

2.1 Summary (a few sentences)

2.2 Objectives (more detailed, bulletpoints?)

2.3 Resources

R is not only a language, it is an active community of developers, users, and educators (often these traits are in each person). This workshop and book based on many excellent materials created by other members in the R community, who share their work freely to help others learn. Using community materials is how WE learned R, and each chapter of the book will have Resources listed for further reading into the topics we discuss. And, when there is no better way to explain something (ahem Jenny Bryan), we will quote or reference that work directly.

- What They Forgot to Teach You About R — Jenny Bryan & Jim Hester
- Stat545 — Jenny Bryan & Stat545 TAs
- Where do Things Live in R? REX Analytics

2.4 Overview

Welcome.

This workshop you will learn hands-on how to begin to interoperate between Excel and R.

A main theme throughout is to produce analyses people can understand and build from — including Future You. Not so brittle/sensitive to minor changes.

We will learn and reinforce X main things all at the same time: coding with best practices (R/RStudio), how this relates to operations in Excel, Z. This training will teach these all together to reinforce skills and best practices, and get you comfortable with a workflow that you can use in your own projects.

2.4.1 What to expect

This is going to be a fun workshop.

The plan is to expose you to X that you can have confidence using in your work. You'll be working hands-on and doing the same things on your own computer as we do live on up on the screen. We're going to go through a lot in these two days and it's less important that you remember it all. More importantly, you'll have experience with it and confidence that you can do it. The main thing to take away is that there *are* good ways to work between R and Excel; we will teach you to expect that so you can find what you need and use it! A theme throughout is that tools exist and are being developed by real, and extraordinarily nice, people to meet you where you are and help you do what you need to do. If you expect and appreciate that, you will be more efficient in doing your awesome science.

You are all welcome here, please be respectful of one another. You are encouraged to help each other.

Everyone in this workshop is coming from a different place with different experiences and expectations. But everyone will learn something new here, because there is so much innovation in the data science world. Instructors and helpers learn something new every time, from each other and from your questions. If you are already familiar with some of this material, focus on how we teach, and how you might teach it to others. Use these workshop materials not only as a reference in the future but also for talking points so you can communicate the importance of these tools to your communities. A big part of this training is not only for you to learn these skills, but for you to also teach others and increase the value and practice of open data science in science as a whole.

2.4.2 What you'll learn

TODO: dev

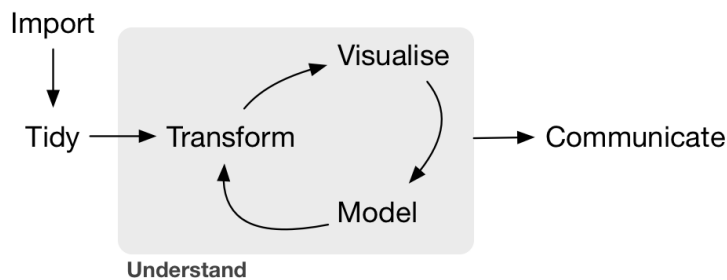
- Motivation is to bridge and/or get out of excel
- We're not going to replicate all of your fancy things in R,
- We use Excel to look at data that we're reading into R

- Spreadsheets are great; blend data entry with analyses and we're going to try to help you think about them a bit more distinctively.
- Most important collaborator is future you, and future us

An important theme for this workshop is being deliberate about your analyses and setting things up in a way that will make your analytical life better downstream in the current task, and better when Future You or Future Us revisit it in the future (i.e. avoiding: what happens next? What does this name mean?)

This graphic by Hadley Wickham and Garrett Golemund in their book R for Data Science is simple but incredibly powerful:

```
knitr::include_graphics("img/r4ds_data-science.png")
```



You may not have ever thought about analysis in such discrete steps: I certainly hadn't before seeing this. That is partly because in Excel, it can be easy to blend these steps together. We are going to keep them separate, and talk about why. The first step is Import: and implicit in this as a first step is that the data is stored elsewhere and is not manipulated directly, which **keeps the raw data raw**.

We will be focusing on:

- **Import:** `readr`, `readxl` to read raw data stored in CSV or Excel files directly into R
- **Tidy:** `tidyr` to (re)organize rows of data into unique values
- **Transform:** `dplyr` to “wrangle” data based on subsetting by rows or columns, sorting and joining
- **Visualize:** `ggplot2` static plots, using grammar of graphics principles
- **Communicate**
 - `writexl` to export intermediate and final data
 - GitHub File Upload and Issues for online publishing and collaboration

2.4.3 Emphasizing collaboration

TODO: rewrite/update (from OHI book):

Collaborating efficiently has historically been really hard to do. It's only been the last 20 years or so that we've moved beyond mailing things with the postal service. Being able to email and get feedback on files through track changes was a huge step forward, but it comes with a lot of bookkeeping and reproducibility issues (did I send that report based on `analysis_final_final.xls` or `analysis_final_usethisone.xls`?). But now, open tools make it much easier to collaborate.

Working with collaborators in mind is critical for reproducibility. And, your most important collaborator is Future You. This training will introduce best practices using open tools, so that collaboration will become second nature to you!

2.4.4 By the end of the course...

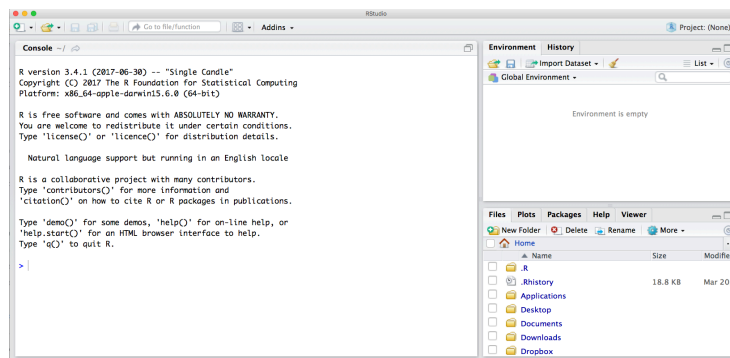
By the end of this course you'll produce this report that you can reproduce, which means... Introduce the problem we will solve. Eg: (just an idea maybe time-series is not a great idea) SMALL PROBLEM. (4 mins) Show data files, We will discuss our analysis plan (only enough to motivate!) Create a report, that looks great.

2.5 RStudio Orientation

Open RStudio for the first time.

Launch RStudio/R.

```
knitr::include_graphics("img/RStudio_IDE.png")
```



Notice the default panes:

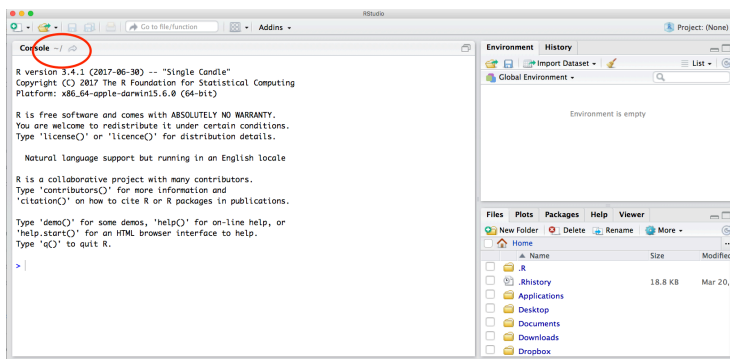
- Console (entire left)
- Environment/History (tabbed in upper right)
- Files/Plots/Packages/Help (tabbed in lower right)

FYI: you can change the default location of the panes, among many other things: Customizing RStudio.

An important first question: **where are we?**

If you've have opened RStudio for the first time, you'll be in your Home directory. This is noted by the `~/` at the top of the console. You can see too that the Files pane in the lower right shows what is in the Home directory where you are. You can navigate around within that Files pane and explore, but note that you won't change where you are: even as you click through you'll still be Home: `~/`.

```
knitr::include_graphics("img/RStudio_IDE_homedir.png")
```



2.5.1 RStudio Projects

Create a new Project called 'r-for-excel-users'. File > New Project... > New Directory > New Project. Give your Project a name browse to a place to keep it. And then click to Create Project!

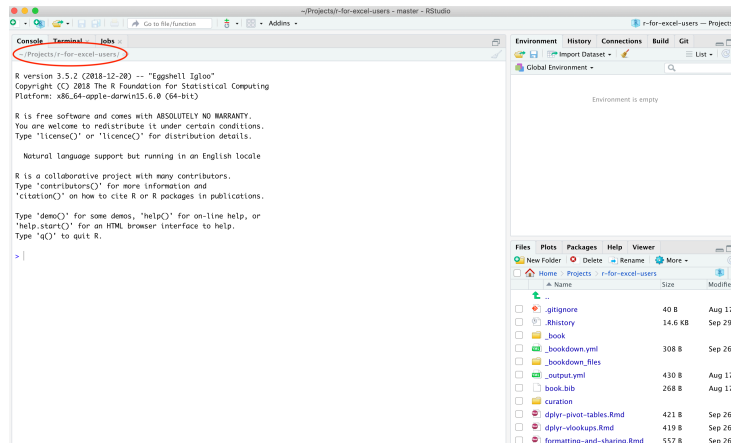
What is a Project? It is a way to organize all of the relevant things you need for an analysis in the same place. This means data, code, figures, notes, etc.

Why does this matter? Keeping everything you need for your analysis together makes it less brittle and more portable — across people, time, and computers.

Working directory = no file path/broken path issues. Notice that a folder now appears wherever you saved this project with the same name, and it contains a `.Rproj` file.

Now that we have our Project, here is an important question: where are we? Now we are in our Project. Everything we do will by default be saved here so we can be nice and organized.

```
knitr::include_graphics("img/RStudio_IDE_projdir.png")
```



2.5.2 R Console

Watch me work in the Console.

I can do math:

```
52*40
365/12
```

But like Excel, the power comes not from doing small operations by hand (like $8*22.3$), it's by being able to operate on swaths of numbers. Columns, Rows, etc. In Excel, they are stored in the spreadsheet. In R, they are stored in variables.

R stores data in variables, and that data can be a variety of formats, like numeric and text. Let's have a look at some data in R. R has several built-in data sets that we can look at and work with.

One of these datasets is called `mtcars`. If I write this in the Console, it will print the data in the console.

```
mtcars
```

I can also use RStudio's Viewer to see this in a more familiar-looking format:

```
View(mtcars)
```

This opens the fourth pane of the RStudio IDE. In the Viewer I can do things like filter or sort. This does not do anything to the actual data, it just changes how you are viewing the data.

There are also functions like in Excel; I can do `?sum` and look at the help pages.

Average mile per gallon. In excel there is a function called AVERAGE. Let's see

OK so working in the Console is great, but it gets messy. It's good for quick things. But looking back at what I've done and trying to build upon it would be a nightmare.

2.5.3 Deep thought: Error messages are your friends

Implicit contract with the computer / scripting language: Computer will do tedious computation for you. In return, you will be completely precise in your instructions. Typos matter. Case matters. Pay attention to how you type.

Remember that this is a language, not unsimilar to English! There are times you aren't understood – it's going to happen. There are different ways this can happen. Sometimes you'll get an error. This is like someone saying 'What?' or 'Pardon'? Error messages can also be more useful, like when they say 'I didn't understand what you said, I was expecting you to say blah'. That is a great type of error message. Error messages are your friend. Google them (copy-and-paste!) to figure out what they mean.

—>

And also know that there are errors that can creep in more subtly, when you are giving information that is understood, but not in the way you meant. Like if I am telling a story about suspenders that my British friend hears but silently interprets in a very different way (true story). This can leave me thinking I've gotten something across that the listener (or R) might silently interpreted very differently. And as I continue telling my story you get more and more confused... Clear communication is critical when you code: write clean, well documented code and check your work as you go to minimize these circumstances!

2.6 R Scripts

Instead of working in the Console that can get messy, we can be more organized. In a script. Let's all do this together.

File > New File > R Script.

This is a blank slate for us to write our code; but there are some good practices we can start off with. Let's add a useful header to the top of it. For example, at a minimum:

```
# -----  
# A descriptive title  
# Summary of what this script is for  
# Your name  
# Contact information  
# Date
```

```
# -----  
  
# Other things you might include: required packages or datasets, relevant links (e.g.
```

Since we're working in or Project, this script is now nicely saved in our Project.

Let's attach a package. Since you've already installed tidyverse

```
# Attach the tidyverse  
library(tidyverse)
```

Cool looking plot with diamonds.

You can run code line by line or sourcing the script

2.7 Don't save the workspace

2.8 Deep thought: keep the raw data raw.

Discussing using Excel for variables.

Horror Stories! Economist etc.

2.9 Interludes (deep thoughts/openscapes)

Comments! Organization (spacing, subsections, vertical structure, indentation, etc.)! Well-named variables! Also, well-named operations so analyses (select(data, columnname)) instead of data[1:6,5] and excel equivalent. (Ex with strings) Not so brittle/sensitive to minor changes.

2.10 Our Turn Your Turn 1

2.11 Our Turn Your Turn 2

2.12 Efficiency Tips

Chapter 3

readxl

3.1 Summary

Check this, may need to be a block quote: The **readxl** package makes it easy to import tabular data from Excel spreadsheets (.xls or .xlsx files) and includes several options for cleaning data during import. **readxl** has no external dependencies and functions on any operating system, making it an OS- and user-friendly package that simplifies getting your data from Excel into R.

3.2 Objectives

- Use `readr::read_csv()` to read in a comma separated value (CSV) file
- Use `readxl::read_excel()` to read in an Excel worksheet from a workbook
- Replace a specific string/value in a spreadsheet with `NA`
- Skip *n* rows when importing an Excel worksheet
- Use `readxl::read_excel()` to read in parts of a worksheet (by cell range)
- Specify column names when importing Excel data
- Read and combine data from multiple Excel worksheets into a single df using `purrr::map_df()`
- Write data using `readr::write_csv()` or `writexl::write_excel()`
- Workflows with **readxl**: considerations, limitations, reproducibility

3.3 Resources

- <https://readxl.tidyverse.org/>

- readxl Workflows article (from tidyverse.org)

3.4 Lesson

3.4.1 Lesson prep: get data files into your working directory

In Session 1, we introduced how and why R Projects are great for reproducibility, because our self-contained working directory will be the **first** place R looks for files.

You downloaded four files for this workshop:

- fish_counts_curated.csv
- invert_counts_curated.xlsx
- kelp_counts_curated.xlsx
- substrate_cover_curated.xlsx

Copy and paste those files into the ‘RforExcelUsers’ folder on your computer. Notice that now these files are in your working directory when you go back to that Project in RStudio (check the ‘Files’ tab). That means they’re going to be in the first place R will look when you ask it to find a file to read in.

3.4.2 Create a new .R script, attach the tidyverse, readxl and writexl packages

In your RforExcelUsers project in RStudio, open a new .R script and add a useful header to the top of it. For example, at a minimum:

```
# -----
# A descriptive title
# Summary of what this script is for
# Your name
# Contact information
# -----

# Other things you might include: required packages or datasets, relevant links (e.g.
```

In this lesson, we’ll read in a CSV file with the `readr::read_csv()` function, so we need to have the `readr` package attached. Since it’s part of the `tidyverse`, we’ll go ahead and attach the `tidyverse` package below our script header using `library(package_name)`. It’s a good idea to attach all necessary packages near the top of a script, so we’ll also attach the `readxl` packages here.

```
# Attach the tidyverse, readxl and writexl packages:
library(tidyverse)
library(readxl)
library(writexl)
```

Now, all of the packages and functions within the `tidyverse` and `readxl`, including `readr::read_csv()` and `readxl::read_excel()`, are available for use.

3.4.3 Use `readr::read_csv()` to read in data from a CSV file

There are many types of files containing data that you might want to work with in R. A common one is a comma separated value (CSV) file, which contains values with each column entry separated by a comma delimiter. CSVs can be opened, viewed, and worked with in Excel just like an `.xls` or `.xlsx` file - but let's learn how to get data directly from a CSV into R where we can work with it more reproducibly.

The CSV we'll read in here is called “`fish_counts_curated.csv`”, and contains observations for “the abundance and size of fish species as part of SBCLTER's kelp forest monitoring program to track long-term patterns in species abundance and diversity” from the Santa Barbara Channel Long Term Ecological Research program.

Source: Reed D. 2018. SBC LTER: Reef: Kelp Forest Community Dynamics: Fish abundance. Environmental Data Initiative. <https://doi.org/10.6073/pasta/dbd1d5f0b225d903371ce89b09ee7379>. Dataset accessed 9/26/2019.

Read in the “`fish_counts_curated.csv`” file `read_csv("file_name.csv")`, and store it in R as an object called `fish_counts`:

```
fish_counts <- read_csv("fish_counts_curated.csv")
```

Notice that the name of the stored object (here, `fish_counts`) will show up in our Environment tab in RStudio.

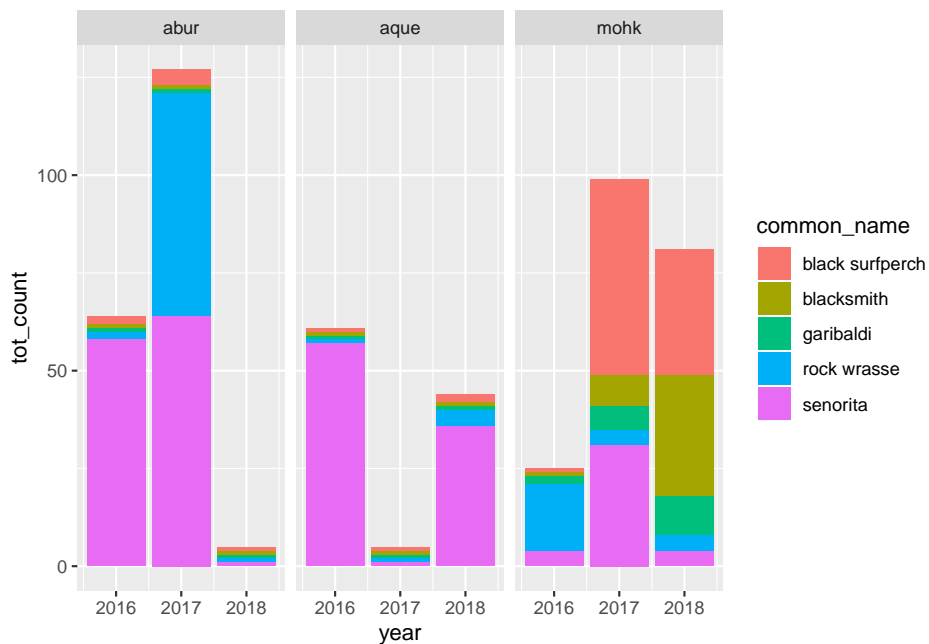
Click on the object in the Environment, and R will automatically run the `View()` function for you to pull up your data in a separate viewing tab. Now we can look at it in the spreadsheet format we're used to.

Here are a few other functions for quickly exploring imported data:

- `summary()`: summary of class, dimensions, NA values, etc.
- `names()`: variable names (column headers)
- `ls()`: list all objects in environment
- `head()`: Show the first x rows (default is 6 lines)
- `tail()`: Show the last x rows (default is 6 lines)

Now let's make a simple plot of some fish counts with `ggplot2`.

```
ggplot(fish_counts, aes(x = year, y = tot_count)) +
  geom_col(aes(fill = common_name)) +
  facet_wrap(~site)
```



Now that we have our fish counts data ready to work with in R, let's get the substrate cover and kelp data (both .xlsx files). In the following sections, we'll learn that we can use `readxl::read_excel()` to read in Excel files directly.

3.4.4 Use `readxl::read_excel()` to read in a single Excel worksheet

First, take a look at *substrate_cover_curated.xlsx* in Excel, which contains a single worksheet with substrate type and percent cover observations at different sampling locations in the Santa Barbara Channel.

A few things to notice:

- The file contains a single worksheet
- There are multiple rows containing text information up top
- Where observations were not recorded, there exists '-9999'

Let's go ahead and read in the data. If the file is in our working directory, we can read in a single worksheet .xlsx file using `readxl::read_excel("file_name.xlsx")`. *Note: `readxl::read_excel()` works for both .xlsx and .xls types.*

Like this:

```
substrate_cover <- read_excel("substrate_cover_curated.xlsx")
```

Tada? Not quite.

Click on the object name (*substrate_cover*) in the Environment to view the data in a new tab. A few things aren't ideal:

```
substrate_cover

## # A tibble: 23,942 x 9
##   `Substrate cover data` ...2 ...3 ...4 ...5 ...6 ...7 ...8 ...9
##   <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
## 1 Source: https://porta~ <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA>
## 2 Accessed: 9/28/2019 <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA>
## 3 <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA>
## 4 year month date site trans~ quad side subs~ perc~
## 5 -9999 -9999 -9999 carp 1 20 i b -9999
## 6 2000 9 -9999 carp 1 20 o b -9999
## 7 2000 9 9/8/00 carp 1 20 i b 100
## 8 2000 9 9/8/00 carp 1 20 o b 100
## 9 2000 9 9/8/00 carp 1 40 i b 100
## 10 2000 9 9/8/00 carp 1 40 o b 100
## # ... with 23,932 more rows
```

- The top row of text has automatically become the (messy) column headers
- There are multiple descriptive rows before we actually get to the data
- There are -9999s that we want R to understand NA instead

We can deal with those issues by adding arguments within `read_excel()`. Include argument `skip = n` to skip the first 'n' rows when importing data, and `na = "this"` to replace "this" with NA when importing:

```
substrate_cover <- read_excel("curation/substrate_cover_curated.xlsx", skip = 4, na = "-9999")
```

```
substrate_cover

## # A tibble: 23,938 x 9
##   year month date site transect quad side substrate_type
##   <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
## 1 <NA> <NA> <NA> carp 1 20 i b
## 2 2000 9 <NA> carp 1 20 o b
## 3 2000 9 9/8/~ carp 1 20 i b
## 4 2000 9 9/8/~ carp 1 20 o b
## 5 2000 9 9/8/~ carp 1 40 i b
## 6 2000 9 9/8/~ carp 1 40 o b
## 7 2000 9 9/8/~ carp 2 20 i b
## 8 2000 9 9/8/~ carp 2 20 o b
## 9 2000 9 9/8/~ carp 2 40 i b
## 10 2000 9 9/8/~ carp 2 40 o b
```

```
## # ... with 23,928 more rows, and 1 more variable: percent_cover <chr>
```

Check out *substrate_cover*, and see that the first row *after* the 4 skipped are the column names, and all -9999s have been updated to NA. Hooray!

3.4.5 Use `readxl::read_excel()` to read in only *part* of an Excel worksheet

We always advocate for leaving the raw data raw, and writing a complete script containing all steps of data wrangling & transformation. But in *some* situations (be careful), you may want to specify a range of cells to read in from an Excel worksheet.

You can specify a range of cells to read in using the `range =` argument in `read_excel()`. For example, if I want to read in the rectangle from D12:I15 in *substrate_cover_curated.xlsx* - only observations for Carpenteria Beach (Transect 2) in September 2000 - I can use:

```
carp_cover_2000 <- readxl::read_excel("substrate_cover_curated.xlsx", range = "D12:I15")
```

But yuck. Look at *carp_cover_2000* and you'll notice that the first row *of that range* is automatically made the column headers. To keep all rows within a range and **add your own column names**, add a `col_names =` argument:

```
carp_cover_2000 <- readxl::read_excel("substrate_cover_curated.xlsx", range = "D12:I15", col_names =
```

```
carp_cover_2000
```

```
## # A tibble: 4 x 6
##   site_name transect quad plot_side type coverage
##   <chr>      <chr>   <chr> <chr>   <chr> <chr>
## 1 carp      2       20    i      b     90
## 2 carp      2       20    o      b     80
## 3 carp      2       40    i      b     80
## 4 carp      2       40    o      b     85
```

So far we've read in a single CSV file using `readr::read_csv()`, and an Excel file containing a single worksheet with `readxl::read_excel()`. Now let's read in data from an Excel workbook with multiple worksheets.

3.4.6 Use `readxl::read_excel()` to read in selected worksheets from a workbook

Now, we'll read in the kelp fronds data from file *kelp_counts_curated.xlsx*. If you open the Excel workbook, you'll see that it contains multiple worksheets with giant kelp observations in the Santa Barbara Channel during July 2016, 2017, and 2018, with data collected at each *site* in a separate worksheet.

To read in a single Excel worksheet from a workbook we'll again use `readxl::read_excel("file_name.xlsx")`, but we'll need to let R know which worksheet to get.

Let's read in the kelp data just like we did above, as an object called *kelp_counts*.

```
kelp_counts <- readxl::read_excel("kelp_counts_curated.xlsx")
```

You might be thinking, "Hooray, I got all of my Excel workbook data!" But remember to always look at your data - you will see that actually only the first worksheet was read in. The default in `readxl::read_excel()` is to read in the **first worksheet** in a multi-sheet Excel workbook.

To check the worksheet names in an Excel workbook, use `readxl::excel_sheets()`:

```
readxl::excel_sheets("kelp_counts_curated.xlsx")
```

If we want to read in data from a worksheet other than the first one in an Excel workbook, we can specify the correct worksheet by name or position by adding the `sheet` argument.

Let's read in data from the worksheet named *golb* (Goleta Beach) in the *kelp_counts_curated.xlsx* workbook:

```
kelp_golb <- readxl::read_excel("kelp_counts_curated.xlsx", sheet = "golb")
```

Note that you can also specify a worksheet by position: since *golb* is the 6th worksheet in the workbook, we could also use the following:

```
kelp_golb <- readxl::read_excel("kelp_counts_curated.xlsx", sheet = 6)
```

```
kelp_golb
```

```
## # A tibble: 3 x 5
##   year month site  common_name tot_fronde
##   <chr> <chr> <chr> <chr>          <dbl>
## 1 2016   7    golb  giant kelp      2557
## 2 2017   7    golb  giant kelp      1575
## 3 2018   7    golb  giant kelp      1629
```

3.4.7 Read in and combine data from multiple worksheets into a data frame simultaneously with `purrr::map_df()`

So far, we've read in entire Excel worksheets and pieces of a worksheet. What if we have a workbook (like *kelp_counts_curated.xlsx*) that contains worksheets that contain observations for the same variables, in the same organization? Then we may want to read in data from *all* worksheets, and combine them into a single data frame.

We'll use `purrr::map_df()` to loop through all the worksheets in a workbook, reading them in & putting them together into a single df in the process.

The steps we'll go through in the code below are:

- Set a pathway so that R knows where to look for an Excel workbook
- Get the names of all worksheets in that workbook with `excel_sheets()`
- Set names of a vector with `set_names()`
- Read in all worksheets, and put them together into a single data frame with `purrr::map_df()`

QUESTION: Have they learned the pipe operator at this point?

Expect the question: Why do I need to use `read_excel()` instead of just giving it the file path (as below)?

```
kelp_path <- "kelp_counts_curated.xlsx"

kelp_all_sites <- kelp_path %>%
  excel_sheets() %>%
  set_names() %>%
  purrr::map_df(read_excel, kelp_path)
```

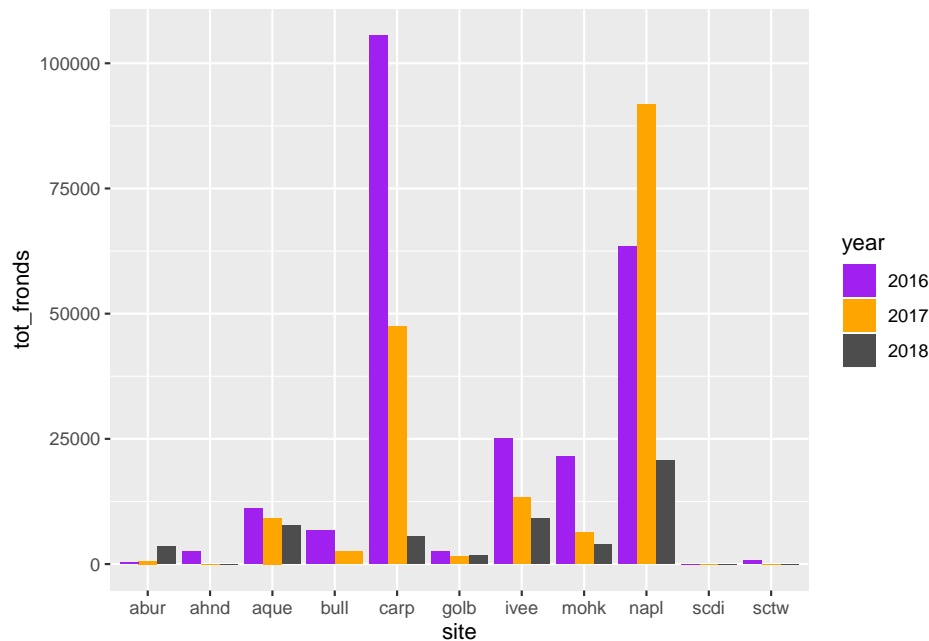
Check out `kelp_all_sites`, and notice that now the data from all 11 sites is now collected into a single data frame:

```
kelp_all_sites

## # A tibble: 32 x 5
##   year month site common_name tot_fronds
##   <chr> <chr> <chr> <chr>          <dbl>
## 1 2016 7     abur giant kelp          307
## 2 2017 7     abur giant kelp          604
## 3 2018 7     abur giant kelp        3532
## 4 2016 7     ahnd giant kelp        2572
## 5 2017 7     ahnd giant kelp          16
## 6 2018 7     ahnd giant kelp          16
## 7 2016 7     aque giant kelp       11152
## 8 2017 7     aque giant kelp       9194
## 9 2018 7     aque giant kelp       7754
## 10 2016 7     bull giant kelp       6706
## # ... with 22 more rows
```

Awesome! Let's make a graph with `ggplot2`:

```
ggplot(kelp_all_sites, aes(x = site, y = tot_fronds)) +
  geom_col(aes(fill = year), position = "dodge") +
  scale_fill_manual(values = c("purple", "orange", "gray30"))
```

3.4.8 Save data frames as .csv or .xlsx with `readr::write_csv()` or `writexl::write_xlsx()`

There are a number of reasons you might want to save (/export) data in a data frame as a .csv or Excel worksheet, including:

- To cache raw data within your project
- To store copies of intermediate data frames
- To convert your data back to a format that your coworkers/clients/colleagues will be able to use it more easily

Use `readr::write_csv(object, "file_name.csv")` to write a data frame to a CSV, or `writexl::write_xlsx(object, "file_name.xlsx")` to similarly export as a .xlsx (or .xls) worksheet.

In the previous step, we combined all of our kelp frond observations into a single data frame. Wouldn't it make sense to store a copy?

As a CSV:

```
readr::write_csv(kelp_all_sites, "kelp_all_sites.csv")
```

A cool thing about `readr::read_csv()` is that it just quietly *works* without wrecking anything else you do in a sequence, so it's great to add at the end of a piped sequence.

For example, if I want to read in the ‘ivee’ worksheet from `kelp_counts_curated.xlsx`, select only columns ‘year’ and ‘tot_fronds’, then write that new subset to a .csv file, I can pipe all the way through:

```
kelp_ivee <- readxl::read_excel("kelp_counts_curated.xlsx", sheet = "ivee") %>%  
  select(year, tot_fronds) %>%  
  write_csv("kelp_ivee.csv")
```

Now I’ve created `kelp_ivee.csv`, but the object `kelp_ivee` also exists for me to use in R.

If needed, I can also export a data frame as an Excel (.xlsx) worksheet:

```
writexl::write_xlsx(kelp_all_sites, "kelp_all_sites.xlsx")
```

3.5 Fun facts ideas:

- Did you know that Clippy shows up to help you in the documentation for `?writexl::write_xlsx()`?
- The name of the `purrr` package? Why map?

3.6 Interludes (deep thoughts/openscapes)

- Workflow/reproducibility/readxl workflows article
- Respecting the tools people are working with already (e.g. don’t make your Excel using co-workers hate you)

3.7 Activity 1

3.8 Activity 2

3.9 Efficiency Tips

- Add an assignment arrow in script/code chunk (`<-`): Alt + minus (-)
- Undo shortcut: Command + Z
- Redo shortcut: Command + Shift + Z

Chapter 4

RMarkdown

- 4.1 Summary (a few sentences)
- 4.2 Objectives (more detailed, bulletpoints?)
- 4.3 Resources
- 4.4 Lessons teaching for each objective..... (objectives, examples)
- 4.5 Fun facts (quirky things) - making a note of these wherever possible for interest (little “Did you know?” sections)
- 4.6 Interludes (deep thoughts/openscapes)
- 4.7 Our Turn Your Turn 1
- 4.8 Our Turn Your Turn 2
- 4.9 Efficiency Tips

Chapter 5

Dplyr and Pivot Tables

- 5.1 Summary (a few sentences)
- 5.2 Objectives (more detailed, bulletpoints?)
- 5.3 Resources
- 5.4 Lessons teaching for each objective..... (objectives, examples)
- 5.5 Fun facts (quirky things) - making a note of these wherever possible for interest (little “Did you know?” sections)
- 5.6 Interludes (deep thoughts/openscapes)
- 5.7 Our Turn Your Turn 1
- 5.8 Our Turn Your Turn 2
- 5.9 Efficiency Tips

Chapter 6

Dplyr and vlookups

- 6.1 Summary (a few sentences)
- 6.2 Objectives (more detailed, bulletpoints?)
- 6.3 Resources
- 6.4 Lessons teaching for each objective..... (objectives, examples)
- 6.5 Fun facts (quirky things) - making a note of these wherever possible for interest (little “Did you know?” sections)
- 6.6 Interludes (deep thoughts/openscapes)
- 6.7 Our Turn Your Turn 1
- 6.8 Our Turn Your Turn 2
- 6.9 Efficiency Tips

Chapter 7

Tidying

7.1 Better practices [needs a better name]

How to be a nimble useR Modern useRs are nimble internet useRs something clever about cleaning I am the worst at naming things

7.2 Summary (a few sentences)

R ecosystem evolves and improves due to contributed work by the community, and this is a good thing. Being a nimble useR means being able to navigate/keep tabs on this ecosystem and find what you need. It also means working reproducibly, so you can re-run and update things more easily. Here we will teach you how to expect things and help yourself. Pay attention to urls.

7.3 Objectives (more detailed, bulletpoints?)

- expect there is a better way, how and where to look (20 mins)
 - CRAN
 - Twitter #rstats
 - rOpenSci
 - RStudio
 - Example: how to Google.
- hands-on with janitor (30+ mins)
 - discovery and quality assurance
 - installing from GitHub
 - big payoff for little effort

- hands-on with another excel-useful example: skimr?
- reproducibility (20 mins)
 - it's important, scripted

7.4 Resources

- Wilson et al. 2014 “Good enough practices”

7.5 Lessons teaching for each objective..... (objectives, examples)

7.5.1 Expect there's a better way chat

- give time for them to google?

7.5.2 Janitor

janitor & other things that will make your life easier with limited effort Janitor: up till now the column names have been fine. Until now.

7.5.2.1 Our turn your turn

Walk through and example and leave our code up, and have you do it but clean another dataset. Work with a neighbor.

7.5.3 Example: How to Google

Pay attention to URLs, build github/rmarkdown savviness (ex: raw.githubusercontent.com)

- I read this blog: <https://blog.revolutionanalytics.com/2018/08/how-to-use-r-with-excel.html>
- I've never heard of click on **openxlsx**, what is it
- Takes me here <https://www.rdocumentation.org/packages/openxlsx/versions/4.1.0.1>, but I want more info. How recently was it worked on? Does it interface with tidyverse? Click on “news”
- Takes me here. <https://raw.githubusercontent.com/awalker89/openxlsx/master/NEWS> . Not useful. But from this URL, - I see the username so I can edit this url to be <https://github.com/awalker89/openxlsx/>

7.6. *FUN FACTS (QUIRKY THINGS) - MAKING A NOTE OF THESE WHEREVER POSSIBLE FOR INTEREST*

- 1st thing: most recent commit was a year ago. Can poke around more, are there issues open, are they taken care of? Etc. I will probably not pursue using this right now. But good to have learned about it.

7.6 Fun facts (quirky things) - making a note of these wherever possible for interest (little “Did you know?” sections)

7.7 Interludes (deep thoughts/openscapes)

7.8 Our Turn Your Turn 2

7.9 Efficiency Tips

- browser efficiency tips
 - Rmd/github anchors for urls
 - press command to open a new tab

Reproducibility is important (this might be new to some people) Example: run everything start to finish and then closing it all and trying to do again In excel Vs R If your computer shuts off are you nervous to close it? Recreate it “What they didn’t forget to teach you about R” WTDF. uncool

Chapter 8

Formatting and Sharing

8.1 Summary (a few sentences)

8.2 Objectives (more detailed, bulletpoints?)

8.3 Resources

8.4 Lessons teaching for each objective..... (objectives, examples)

1. Create a GitHub account: <https://github.com> *Note! Shorter names that kind of identify you are better, and use your work email!*

- 8.5 Fun facts (quirky things) - making a note of these wherever possible for interest (little “Did you know?” sections)
- 8.6 Interludes (deep thoughts/openscapes)
- 8.7 Our Turn Your Turn 1
- 8.8 Our Turn Your Turn 2
- 8.9 Efficiency Tips

Chapter 9

Synthesis

- 9.1 Summary (a few sentences)
- 9.2 Objectives (more detailed, bulletpoints?)
- 9.3 Resources
- 9.4 Lessons teaching for each objective..... (objectives, examples)
- 9.5 Fun facts (quirky things) - making a note of these wherever possible for interest (little “Did you know?” sections)
- 9.6 Interludes (deep thoughts/openscapes)
- 9.7 Our Turn Your Turn 1
- 9.8 Our Turn Your Turn 2
- 9.9 Efficiency Tips