R Faculty Workshop

Workshop overview

Day 1: Intro to R

Intro to R

Some notes about the general

Basics of writing R code

Syntax, functions, assignment, pipe, data types

Working with data

Importing data, cleaning, graphs, analyses

Day 2: Teaching with R

General approach

Pedagogy, grading, generative AI

Tools

Quarto

Examples

Lab manual, problem sets

1 Introduction to R and RStudio

1.1 Why R?

A coding language specialized for statistical computing and data analysis. Free and open-source. (Though there is a cloud-based version which can be paid).

- Import data from your computer, websites, databases, via webscraping
- Clean and organize data
- Analyze and visualize the data
- Communicate the results in various formats (pdf research paper, website, presentation slides)

1.2 The general workflow

1.2.1 Manipulating data

It might seem daunting if you've 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

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

```
[1] 3
```

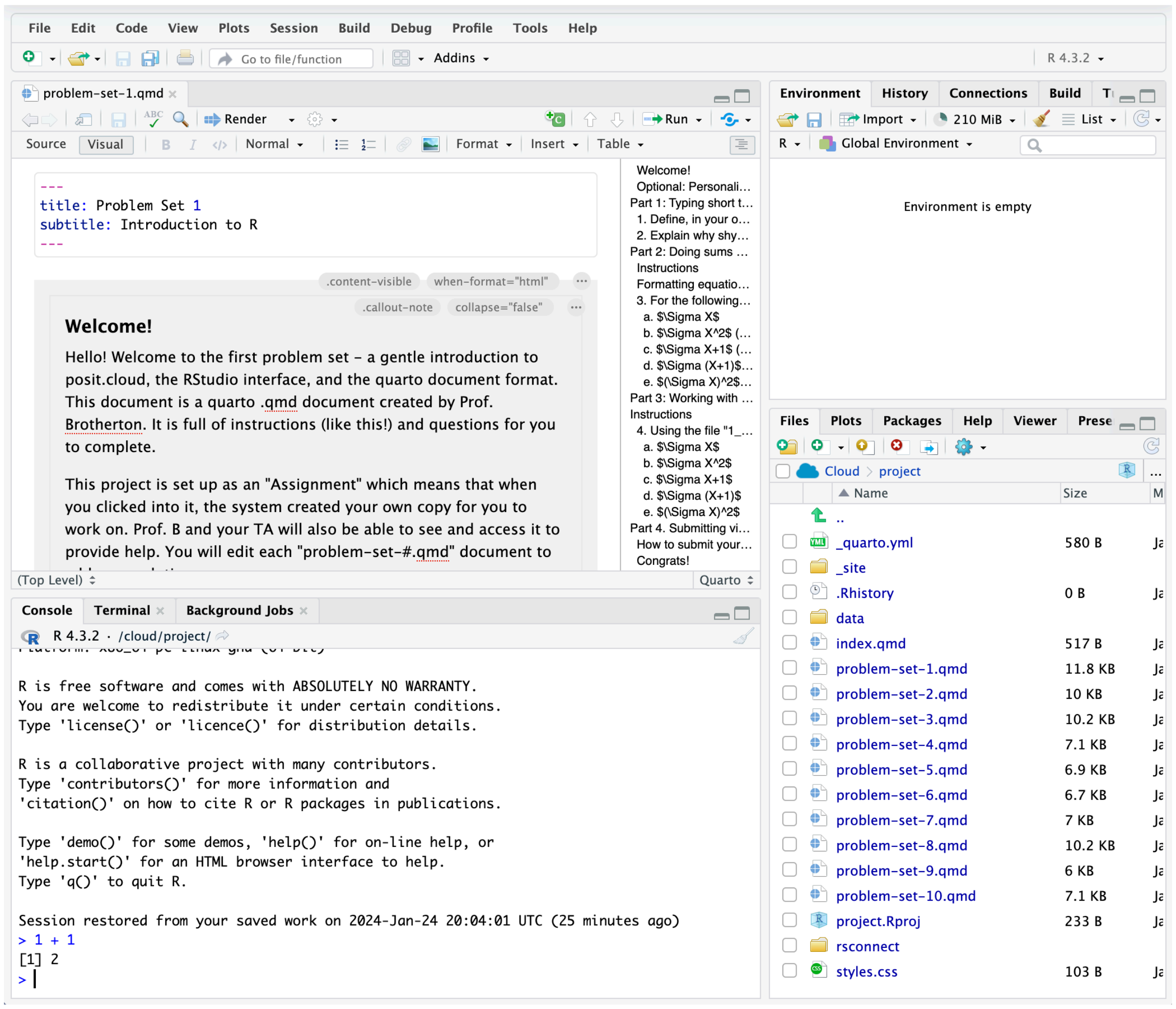
```r
sum(data$A)
```

```
[1] 15
```

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

```
[1] 1.581139
```

1.2.3 RStudio Interface


File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
R 4.3.2
problem-set-1.qmd
Render
Run
Source Visual
Normal
Format Insert Table
Environment History Connections Build
Import
210 MiB
List
R
Global Environment
Files Plots Packages Help Viewer Prese
Cloud > project
Console Terminal Background Jobs
(Top Level)
Quarto

```
---
title: Problem Set 1
subtitle: Introduction to R
---
```

.content-visible when-format="html"

.callout-note collapse="false"

Welcome!

Hello! Welcome to the first problem set – a gentle introduction to posit.cloud, the RStudio interface, and the quarto document format. This document is a quarto .qmd document created by Prof. Brotherton. It is full of instructions (like this!) and questions for you to complete.

This project is set up as an "Assignment" which means that when you clicked into it, the system created your own copy for you to work on. Prof. B and your TA will also be able to see and access it to provide help. You will edit each "problem-set-#.qmd" document to

```
Welcome!
  Optional: Personali…
Part 1: Typing short t…
  1. Define, in your o…
  2. Explain why shy…
Part 2: Doing sums …
  Instructions
  Formatting equatio…
  3. For the following…
    a. $\Sigma X$
    b. $\Sigma X^2$ (…
    c. $\Sigma X+1$ (…
    d. $\Sigma (X+1)$…
    e. $(\Sigma X)^2$…
Part 3: Working with …
Instructions
  4. Using the file "1_…
    a. $\Sigma X$
    b. $\Sigma X^2$
    c. $\Sigma X+1$
    d. $\Sigma (X+1)$
    e. $(\Sigma X)^2$
Part 4. Submitting vi…
  How to submit your…
  Congrats!
```

Environment is empty

```
R 4.3.2 · /cloud/project/

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

Session restored from your saved work on 2024-Jan-24 20:04:01 UTC (25 minutes ago)
> 1 + 1
[1] 2
>
```

Name	Size
..	
_quarto.yml	580 B
_site	
.Rhistory	0 B
data	
index.qmd	517 B
problem-set-1.qmd	11.8 KB
problem-set-2.qmd	10 KB
problem-set-3.qmd	10.2 KB
problem-set-4.qmd	7.1 KB
problem-set-5.qmd	6.9 KB
problem-set-6.qmd	6.7 KB
problem-set-7.qmd	7 KB
problem-set-8.qmd	10.2 KB
problem-set-9.qmd	6 KB
problem-set-10.qmd	7.1 KB
project.Rproj	233 B
rsconnect	
styles.css	103 B

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.

```r
# 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:

```r
?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

2.1 Make your first R code file

File > New File > R Script. Creates a new, temporary `Untitled.R` file. Give it a meaningful name when you save it.

2.2 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.

- 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)

```r
# comment

# 1 + 1

1 + 1
```

```
[1] 2
```

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

```
[1] 9
```

2.2.1 Using R like a calculator

```r
1 + 1
```

```
[1] 2
```

```r
2 * 2
```

```
[1] 4
```

```r
3^2
```

```
[1] 9
```

3 Data structures

3.1 Vectors

A vector is a collection of things.

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

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

```
1:5
```

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

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

```
[1] 1
```

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

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

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

```
[1]  TRUE FALSE
```

Doing math with vectors

```r
1:5 * 2
```

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

```r
1:5 * c(1, 2)
```

```
Warning in 1:5 * c(1, 2): longer object length is not a multiple of shorter
object length
```

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

```r
6 - 1:5
```

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

```r
1:5 * 1:5
```

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

3.1.1 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.

```r
c(1, 2, "three", 4, 5)
```

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

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

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

```r
mean(c(1, 2, "3", 4, 5))
```

```
Warning in mean.default(c(1, 2, "3", 4, 5)): argument is not numeric or
logical: returning NA
```

```
[1] NA
```

```r
22 < "1"
```

```
[1] FALSE
```

Coercion can have some happy consequences. For instance, logical values (`TRUE` and `FALSE`) will 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

c(TRUE, FALSE, FALSE, TRUE, 1)
```

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

```r
sum(c(TRUE, TRUE, FALSE, FALSE))
```

```
[1] 2
```

```r
mean(c(TRUE, TRUE, FALSE, FALSE))
```

```
[1] 0.5
```

3.1.2 Factors

```r
factor(c("male", "female", "female", "male"))
```

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

```r
factor(c("medium", "small", "large"), levels = c("small", "medium", "large"), ordered = TRUE
```

```
[1] medium small  large 
Levels: small < medium < large
```

3.1.3 Special values

```r
NA
```

```
[1] NA
```

```r
NULL
```

```
NULL
```

```r
NaN
```

```
[1] NaN
```

3.2 Functions

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

```r
sum(1:5)
```

```
[1] 15
```

```r
length(1:5)
```

```
[1] 5
```

```r
mean(1:5)
```

```
[1] 3
```

```r
sd(1:5)
```

```
[1] 1.581139
```

```r
var(1:5)
```

```
[1] 2.5
```

```r
min(1:5)
```

```
[1] 1
```

```r
max(1:5)
```

```
[1] 5
```

You can also nest functions inside one another.

```r
sqrt(mean(1:5))
```

```
[1] 1.732051
```

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; in the previous examples `original_numbers` and `doubled_numbers` were the parameters I supplied. You don't necessarily have to type the name of the argument, but it can be helpful. The `seq()` function, for example, produces a sequence of numbers according to three arguments, `from`, `to`, and `by`.

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

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

When you don't type the names of the arguments, R matches them by position, so this gives exactly the same output as the previous line of code:

```r
seq(1, 10, 2)
```

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

```r
seq(1, 2, 10)
```

```
[1] 1
```

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.

```r
?mean

# see if you can compute a 'trimmed' mean for this set of skewed numbers
c(1, 2, 3, 4, 100)

mean(c(1, 2, 3, 4, 100))

mean(c(1, 2, 3, 4, 100), trim = 0.2)
```

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

3.3 Assignment

R has a fancy assignment operator: `<-`.[1] You assign things to a name by typing something like:

```r
name <- thing
```

The `thing` there might be a set of numbers, an entire dataset, a statistical model object, or something else. Giving it a name allows to you perform subsequent operations more easily, and choosing appropriate names makes your code easier to understand.

```r
original_numbers <- 1:10
original_numbers
```

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

[1]If you're looking at R code from elsewhere (e.g. looking up help online) you may see a different pipe: `%>%`. The `|>` pipe, called the "native" pipe, was only included as a feature of base R relatively recently. Until then, the `%>%` pipe was provided by an external package (called `magrittr`. Get it?). In practice the pipes work similarly, so you can often just replace `%>%` with `|>` and it'll work fine, but it's worth being aware of.

```r
doubled_numbers <- original_numbers * 2
doubled_numbers
```

```
 [1]  2  4  6  8 10 12 14 16 18 20
```

Most usefully, we can use these nicely named objects as input to functions.

```r
mean(original_numbers)
```

```
[1] 5.5
```

```r
mean(doubled_numbers)
```

```
[1] 11
```

3.4 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`.

```r
1 + NA
```

```
[1] NA
```

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

mean(numbers)
```

```
[1] NA
```

```r
mean(numbers, na.rm = TRUE)
```

```
[1] 3
```

```r
mean(numbers[!is.na(numbers)])
```

```
[1] 3
```

3.5 Data.frames

R's versions of a spreadsheet in Excel or SPSS is called a data.frame. A data.frame is a collection of vectors; each column is a vector. As a result, 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.

```r
df <- data.frame(a = c(1, 2, 3),
                 b = c("one", "two", "three"),
                 c = c(1, 2, "3"))

str(df)
```

```
'data.frame':	3 obs. of  3 variables:
 $ a: num  1 2 3
 $ b: chr  "one" "two" "three"
 $ c: chr  "1" "2" "3"
```

```r
summary(df)
```

```
       a            b                   c            
 Min.   :1.0   Length:3           Length:3          
 1st Qu.:1.5   Class :character   Class :character  
 Median :2.0   Mode  :character   Mode  :character  
 Mean   :2.0                                        
 3rd Qu.:2.5                                        
 Max.   :3.0                                        
```

3.5.1 Selecting a data.frame column

```r
df$a
```

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

```r
mean(df$a)
```

```
[1] 2
```

```r
table(df$b)
```

```

  one three   two 
    1     1     1 
```

4 Working with data

4.1 Start a new Project

File > New Project > New Directory > New Project

4.2 The working directory

R can access your entire filesystem, so you can access and create files anywhere on your hard drive. It can be a bit tedious to type explicit complete file paths every time you need to read or write a file, however. E.g.:

`"/Users/robertbrotherton/Documents/r-workshop/my_first_r_file.R"`

Another big drawback is that if you share your code for someone else to run on their own computer, the filepaths will not work!

```r
getwd()
```

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

Now whenever you use a function that requires you to point to a file will be looking in that directory. So you can type

```r
read.csv(file = "my_first_r_file.R")
```

4.3 Writing and running code

4.4 Importing data

```r
# read.csv()
# foreign::read
foreign::read.dta("stats_file.dta")
haven::read
foreign::read.spss("spss_file.sav")

readxl::read_excel("excel_file.xlsx")
```

4.5 Piping

You can string together different operations in a pipeline using the pipe operator: `|>`.[1] The result of each line of code gets "piped" into the function on the next line as its first argument. For example, below I take some data (named `my_data`) and perform a series of operations, first changing its shape using `pivot_longer()`, then creating summary statistics for the mean and standard deviation separately by a grouping-variable, then I pipe the summary statistics into `ggplot()` to create a graph with a `geom_col()` layer for the geometry.

4.6 Data cleaning

```r
df |>
  filter() |>
  mutate(something = case_when(
    condition ~ new_value,
    condition ~ new_value,
    TRUE ~ NA
    )) |>
  drop_na()


# example mutations
# reverse code

# example conditions for case_when
# something %in% values
```

[1]If you're looking at R code from elsewhere (e.g. looking up help online) you may see a different pipe: `%>%`. The `|>` pipe, called the "native" pipe, was only included as a feature of base R relatively recently. Until then, the `%>%` pipe was provided by an external package (called `magrittr`. Get it?). In practice the pipes work similarly, so you can often just replace `%>%` with `|>` and it'll work fine, but it's worth being aware of.

```r
# !something %in% values
# something > 0
```

4.7 Data exploration

```r
dplyr::summarize()
```

4.8 Data Visualization

4.8.1 Using built in `plots`

4.8.2 Using `ggplot`

```r
my_data |>
  pivot_longer(everything(),
               names_to = "condition",
               values_to = "score") |>
  summarize(mean = mean(score),
            .by = condition) |>
  ggplot(aes(x = condition, y = mean)) +
  geom_col()
```

4.9 Data Analysis

4.9.1 Correlation

```r
cor(x, y)

cor.test(x, y)

cor(df)
```

4.9.2 *t*-test

```r
t.test()
```

4.9.3 ANOVA

Sometimes the function that computes a model doesn't tell us everything we typically want to know about that model.

```r
aov(DV ~ IV, data = df)
```

It is often useful to assign the model to a name, and the ask for a summary of the model:

```r
model <- aov(DV ~ IV, data = df)

summary(model)
```

4.9.4 Regression

```r
regression_model <- lm(DV ~ IV, data = df)

summary(regression_model)
```

4.9.5 Mixed-effects

5 General comments

5.1 Justify using R

Helps to set expectations. Students see it as worthwhile from the outset.

Pedagogically, using R can facilitate students' understanding of fundamental statistical concepts and applied analytic techniques. While having students simultaneously learn statistical concepts and a coding language is challenging, my experience to date shows that it is not only possible, but that the coding elements can be used to reinforce students' understanding of the conceptual side. As compared to other statistics software, such as SPSS, R has many features that make it especially well-suited for teaching. I have students use R to demonstrate basic concepts of probability and sampling, as well as running basic and advanced analyses. Equations that are introduced in class using paper and pencil can be adapted into R code and turned into custom functions, forming a concrete link between the underlying mathematical procedures and their computational implementation. By the time students are making use of built-in functions to perform sophisticated analyses they understand how the computer is applying the same mathematical procedures they have learned in class. This keeps the focus on understanding and applying the statistical concepts, rather than merely learning the correct sequence of buttons to press on a graphical user interface.

The second way in which using R brings tangible benefits is in the value of the skills it imparts. For social-scientists-in-training, R is a tool they will likely encounter at some point. The main alternative, SPSS, has been declining in popularity while R has been increasing. Beyond academia, the number of jobs which list R as a desirable or required skill vastly outnumber those which require SPSS (Muenchen, 2023). In part, R's popularity is a result of it being freely available and easily extensible, making data-analytic tasks easier. Resulting analyses are inherently documented and reproducible, which has made R code files a common format for sharing analysis scripts alongside open-access data (on psyArXiV, for example). Using R allows instructors not only to offer training in a cutting-edge tool which students can use for scholarly projects, but to make students aware of the broader context in which such analytic tools exist. This is in line with the Thinking Technologically and Digitally requirement's goal of helping "instill in students the confidence to make decisions about the adoption and use of current and future technologies in a critical and creative manner."

5.2 Grading

Students are often daunted. It does not come naturally to all.

Grade for effort rather than results.

5.3 Start with something cool

5.4 Most common problems

cover the problems and challenges that students most often run into, and how these can be avoided or even turned into learning opportunities.

5.5 Additional resources

Modern Statistical Methods for Psychology (https://bookdown.org/gregcox7/ims_psych/)

5.6 Generative AI

6 Posit Cloud

6.1 Names

There are a few different names involved:

- **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 makes RStudio
- **posit.cloud** provides a way of using RStudio in your web browser

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 building R into a course.

6.2 Differences between cloud vs local

Each cloud 'Project' is its own unique system. Additional packages installed in one project will not be.

6.2.1 Cost

No cost to students. Potentially cost to instructor/department

6.2.1.1 Tiers

Free:

$15/month instructor account.

6.3 Assignments

Packages installed in your source version will be already installed in students' copies. Avoids the need for tedious and confusing installation, and occasional errors due to differences across students' hardware/software setups.

Instructor can see all studnets' projects which were derived from the assigment. Can click into students' project and see their work, debug, etc.

7 Quarto

7.1 What is Quarto?

“An open-source scientific and technical publishing system”

Works seamlessly within RStudio

Next generation version of R Markdown.

7.1.1 Formats

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

7.2 Benefits of using Quarto over just R.

7.3 Working with Quarto documents

7.3.1 Typing text

Source vs Visual

7.3.2 Code chunks

7.3.3 Rendering

All code gets executed from scratch

7.4 Getting started

7.4.1 A single document

File > New File > Quarto Document

YAML section

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

Text.

Code chunks.

```
# hello!
```

7.4.2 A Quarto Project

7.4.2.1 Project Features

The `_quarto.yml` file.

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

Listing 7.1 `_quarto.yml`

```
project:
  type: website

format:
  html:
    toc: true
```

8 Course Materials

How I use Quarto to create problems sets to teach R.