R Notebooks for Research and Teaching - Overview

This notebook covers how to use RStudio Cloud and is intended to be a reference for researchers and teachers. Associated notebooks will use technqiues covered here and will give examples specifically related to research and teaching. To view this notebook as a PDF, select **File -> Knit Document** above.

RStudio Cloud as an Infrastructure for Research and Teaching

The goal of this lesson is to get you setup with an infrastructure for data analysis that can support both research and teaching. Setting up infrastructure can be a serious barrier both for you and your students, and we're going to be working in RStudio because it reduces many of the barriers associated with other options. Python, for example, is probably the main alternative to R for academic data analysis – however, setting up an infrastructure in python is considerably more difficult than in R.

R is a statistical programming language that is widely used in academia. It's free and supported by a strong community. **RStudio** is an IDE for R. Again, it's free and strongly recommended if you're working with R on your local machine.

RStudio Cloud lets you work in the RStudio interface but stores and processes everything remotely. This means that students have nothing to install and can all have access to the exact same environment. (If you have ever dealt with Python versions or package control, you know that this is a very nice thing).

Literate Programming and Reproducible Research

This is an **R Notebook**, which combines Markdown formatting with executable code.

Notebooks (both in R and in other languages, like Python) are based on the principle of **literate programming**. Literate programming is just the idea that, when we write code, we're writing for humans as much as the machine that will execute the code. Instead of writing a few comments within code, literate programming suggests that we write an explanation of what the program will do and why and – within that explanation – insert code as needed.

One benefit of literate programming is that it supports the goal of **reproducible research**. The idea of reproducible research comes from the sciences and is just that other researchers should be able to properly evaluate your results and/or recreate them. But reproducible research also implies that *you* are better able to reproduce and reuse your research because you won't forget how your produced your results or lose track of which figures are the most recent. Chris Hartgerink gives a quick rundown of using RMarkdown to produce reproducible research products here.

Reproducible research, from a programming perspective, generally just means sharing your data and scripts. Literate programming means that these resources will be more useful for other researchers.

While the norms of reproducible research are less common in some areas of the social sciences, the principles of literate programming and reproducible research still set you up well to:

1. Understand your own work.
2. Collaborate with colleagues.
3. Share code with students.

Principles of Notebooks

Interactive notebooks combine text and executable code – basically, they're intended for literate programming. The most well-known is Project Jupyter, which you should look into if you're interested in Python. This

lesson uses RMarkdown Notebooks in RStudio Cloud, but most notebooks share a few principles:

Markdown

The bulk of notebooks is text written for humans, and most notebooks use Markdown to do simple formatting of text. If you're viewing this notebook in the RStudio Cloud interface, for example, you're seeing # used to create headings and other marks used to insert links and bold text. When the notebook is exported, text written in Markdown is coverted, giving you visual headings, etc.

(Much) more information about the use of Markdown in R is available here.

Code Chunks

Code chunks are the executable part of notebooks; they're where the programming happens. Within the development interface, code chunks can be executed one at a time, usually by clicking a play icon within them. There's also usually an option to execute all chunks in the document.

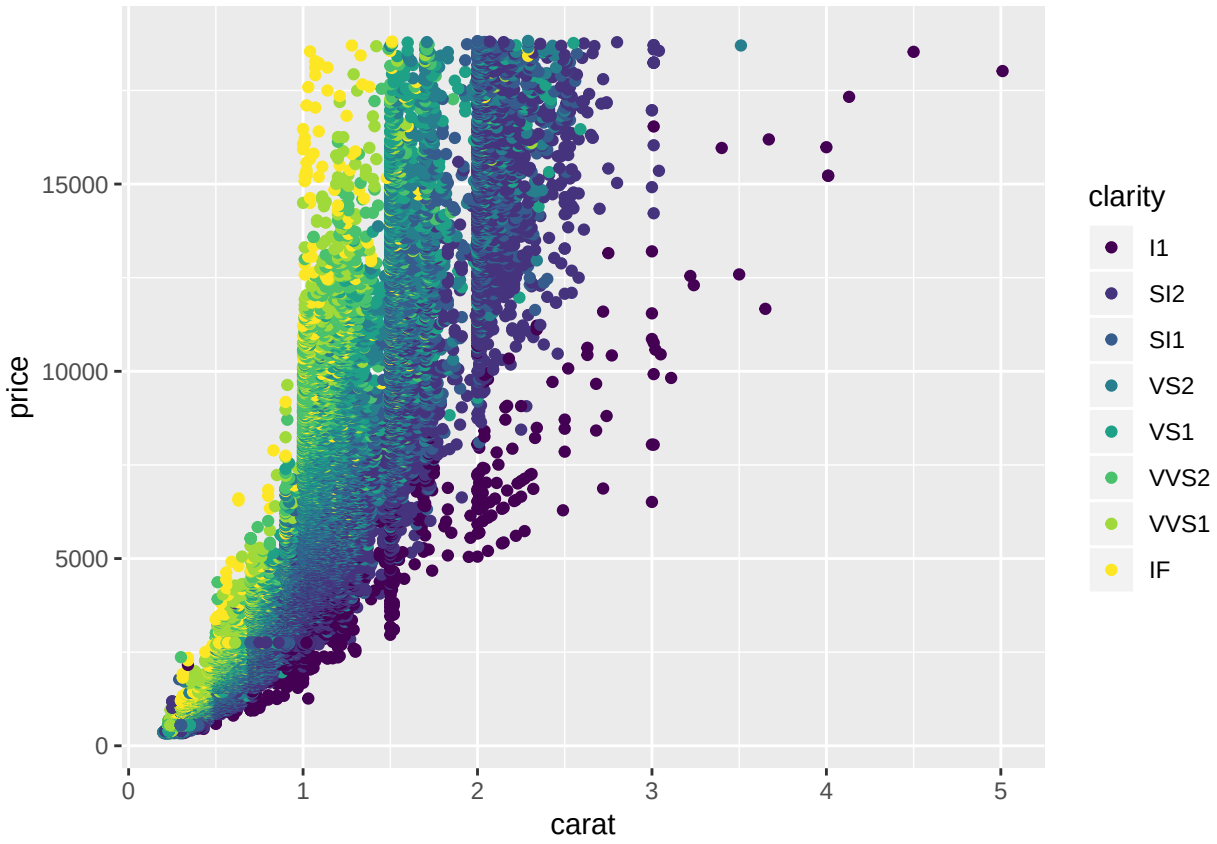
Code chunks in RMarkdown Notebooks begin with "'{r} and end with"'. For example, the following chunk just adds 2 + 2.

```
2 + 2
```

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

One important difference to note when programming within a notebook is that the output of code chunks is placed immediately after the chunk when it is run. The output is also printed in the same location when you export the notebook. This means that figures will appear in your exported document just below the code that generates them:

```
library(ggplot2)
data("diamonds")
ggplot(diamonds, aes(x=carat, y=price, color=clarity)) + geom_point()
```



Export

Notebooks are written and can be interacted with inside a development interface. For example, you're probably looking at this inside the RStudio interface. In a development interface, you can add and execute code, inspect objects and do all the things you need to while creating your document or interactively examining someone else's work.

However, notebooks also allow you to export your work in a variety of formats such as PDF, Word or HTML. This means that as you're writing your documentation and code, you're also writing a report that can be generated and shared.

RMarkdown Notebooks use a package called knitr for exporting. You can read a nice tutorial on knitr, but in most cases you don't need to know much about how it works. Just above this code panel there should be a dropdown allowing you to preview the notebook or export to PDF, HTML or Word.

Base Packages

In addition to setting up RStudio Cloud and RMarkdown Notebooks, an important part of computational infrastructure is the packages or libraries that you invest time in and encourage students to learn. R uses packages; Python uses libraries – both extend the base functionality of the language in order to complete specific kinds of tasks. For example, there are R packages from text mining, for working with dates and for creating visualizations.

Because packages take time to learn and because there are generally several packages that have similar functionalities, it's helpful to focus on packages that are well-supported and widely used. It's also helpful to have a stable set of packages that you'll teach students to work with.

In R, the *tidyverse* collection presents a simple solution to basic package selection. The tidyverse is a collection of packages that work together and reflect a core set of principles. ggplot2 and dplyr are especially widely used and provide functionality – in data visualization and manipulation – that is core to many tasks. To make sure that students always have these packages installed, you can simply install the tidyverse package in the Base Project for your course workspace (see notes below). (To install the package, select the Packages tab in the bottom right RStudio pane, click Install and search for tidyverse.)

Notes on Teaching

Setting Up Classes

RStudio Cloud has several nice options for running a course. The "Using Private Spaces in Courses and Workshops" section of the RStudio Cloud guide provides a nice background. The main ideas are:

- Create a private workspace and invite students to join.
- Create a project that is specified as the Base Project for the space. This means that any files and packages included in the base project are included in all projects within the space. You can use this to make sure students all have access to data files, for example, and you can also ensure that everyone has access to the same packages.

Creating Assignments

Assignments can be created in RStudio Cloud by editing a project's access settings. Open the project, click the gear in the upper right corner and select Access. Set the project so that everyone in your workspace can view it and then choose to make it an assignment. When students open that project, they will be informed that they are creating a copy that is theirs to work on.

Turning in assignments can be handled in a variety of ways. The simplest is to have students export their notebook to PDF, giving the instructor access to their documentation and analysis as well as their code and results.

Further Reading

- Adam Rule, Aurélien Tabard, and James D. Hollan. 2018. Exploration and Explanation in Computational Notebooks. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18). ACM, New York, NY, USA, Paper 32, 12 pages. DOI: https://doi.org/10.1145/3173574.3173606
- Çetinkaya-Rundel, Mine, and Colin Rundel. "Infrastructure and tools for teaching computing throughout the statistical curriculum." The American Statistician 72.1 (2018): 58-65.
- Grolemund, G., & Wickham, H. R for Data Science. Retrieved from http://r4ds.had.co.nz/
- Hicks, S. C., & Irizarry, R. A. (2018). A guide to teaching data science. The American Statistician, 72(4), 382-391.
- Horton, N. J., Baumer, B. S., & Wickham, H. (2014). Teaching precursors to data science in introductory and second courses in statistics. arXiv preprint arXiv:1401.3269.
- Knuth, D. E. (1984). Literate programming. The Computer Journal, 27(2), 97-111.
- Shum, S., & Cook, C. (1994, March). Using literate programming to teach good programming practices. In ACM Sigcse Bulletin (Vol. 26, No. 1, pp. 66-70). ACM.
- Stander, J., & Dalla Valle, L. (2017). On Enthusing Students About Big Data and Social Media Visualization and Analysis Using R, RStudio, and RMarkdown. Journal of Statistics Education, 25(2), 60-67.