Web Resources to Start Learning R

Basics

RStudio

Download R itself, if you haven't yet, then download RStudio before you do anything else. This is the way to code in R. It's an Integrated Development Environment (IDE) that makes everything from installing packages to find-and-replace so much easier. RStudio makes the superpowers of R accessible for normal people to use. The website itself also has some useful resources.

R for Data Science e-Textbook

An intro to R specifically for data science using Hadley's tidyverse packages (some of the most useful). It's all about exploring so it's a great primer. It's fairly interactive incorporating practical coding exercises into sensible theory. It's not really a resource book to figure out how to do particular tasks (more on that below). I very highly recommend at least reading Section I: "Explore" and highly recommend also reading Section II: "Wrangle".

Online Intro Courses

Two good ones are Duke's RFun and edX Harvard Data Science: R Basics Course. There are also paid courses with occasional discounts like the DataCamp Intro to R course.

Write some code

R Style Guide

To keep your code clean and readable, you need to follow some basic conventions. RStudio does much of this automatically but rules for naming variables and line length can help keep things consistent and easy to read. I've linked Hadley's Style Guide above for some ideas (longer form here), Google has one that's a little different if you prefer. The important thing is just to pick a set of conventions and stick to it.

RStudio Cheatsheets

RStudio puts out a bunch of cheatsheets that summarize particular functions or packages on a single back-and-front poster. I use the Data Transformation cheatsheet all the time and the ggplot2 one fairly often as well.

Stack Overflow

This is your resource to figure out how to do specific things. Since R is open-source and free, it relies on its community to provide tech support. It's a massive forum so when you can't find exactly what you're looking for there's usually an analogous example somewhere. You can make an account or just Google "stack overflow r [insert question]".

Hadley's Tidyverse
This contains documentation for some of the most useful R packages. While you can usually get this info in RStudio with the "help"/"?"/F1 function, these sites have examples and visualizations that are useful when you need more help. Click on the package then on the "Reference" link on the top right to get documentation.

Esquisse
The esquisse package add-in for R-Studio is a drag-and-drop interface for making graphs. It's an easy way to jump into one of R's best features.

Get some help from AI
Rstudio has integrations with GitHub Copilot that will

Tips

Use project directories. You might learn about "working directories" using setwd(). That might work okay for a while, but anytime you reorganize your computer files around or share your code, you'll have to edit your code. Instead RStudio lets you make "project directories" that can contain a bunch of sub-folders that are self-contained in the project.

Comment your code. You might remember what you did today but will you remember in two weeks? And could someone else understand what you did? See my example below for how to do this. I also recommend putting a title, author, date, and short description at the top of our code.

Load the packages you need at the top of your code. This helps anyone running the code to check if they need to install any new packages first and avoids weird conflicts between packages.

Know the main arguments a function takes. Every function, the things with parentheses after them like c() or library(), have particular settings called arguments. Sometimes you need to specify them (like what data to use) but other times there are defaults you can change if you want to do something differently. Click on the function and press "F1" to get the help file for that function in the bottom left panel of RStudio.

R deals with objects. For anything to be "real" in R it has to be created with the <- function. Once you create it (my_object <- 1), it will appear in the Environment tab of RStudio. Until then, it doesn't exist and R can't use it. Similarly, if you delete a line of code but don't clear the Environment (using the little broom icon), R will keep using the object that code created.

R "thinks" in vectors. R is vectorized, meaning it thinks and acts using lists of stuff it calls vectors. Lists of numbers make up a variable, lists of variables make up a table, and so on and

so forth. Everything it does is just going down a list and spitting out another list (that's why it puts little numbers like this "[1]" on its printouts).

Get familiar with different types of vectors. R has a bunch of different ways it stores data in vectors. It could be character, numeric, factor, integer or another type. R calls these classes and it only does certain things with certain classes (e.g. it only adds numeric or integer types). If you get an error, incompatible classes could be your issue.

Data frames are basically a bunch of vertical vectors. R's tables are What looks like a table to you, is what R calls a "data frame" and thinks of as a bunch of vertical vectors of the same size stacked next to each other. So, while it's easy to tell R to grab the "score" variable, it's not as easy to tell it to grab all the data points for one subject across all variables (though there are packages that make that easier) and you can't just slap columns of different sizes together like in Excel.

Use the pipe %>% function. You might find you're doing a bunch of commands on the same object. Rather than keep writing out the <- command, you can pipe objects from one command to another in a big long string: my_data <- my data %>% select(var_1, var_2) %>% group_by(var_1)

Sample code

```
# Title: Sample code
# Author: Justin Rasmussen
# Date: 2019-10-31
# Description: Basic template for new R users. Runs example tidyverse functions.

# load packages
library(tidyverse)

# import csv file
raw_data <- read.csv("filename.csv")

# tidy dataset
tidy_data <- raw_data %>%
  mutate(new_var = var_1 + var_2 ) >%> # create new aggregate variable
  select(id_num, new_var, var_3) %>% # select variables you want
  filter(id_num > 0) # filter observations you want

# group summary stats
summary_data <-tidy_data %>%
  group_by(var_3) %>%
```

```
  summarize(mean = mean(new_var),
            N = n())

# create a basic scatterplot with means
ggplot(data = tidy_df,
            aes(x = var_3,
            y = new_var)) +
  geom_point() +
  geom_point(data = summary_data,
            aes(y = mean),
            colour = 'red',
            size = 3)

# save file to csv
write.table(tidy_data, “filename_tidy.csv”, sep  = “,”)
```