R errors, troubleshooting, and getting help

Dav King

2/3/2022

Main Ideas

- Encountering errors and misbehaving code is common at all levels of ability.
- Employing a few simple strategies and steps to follow will help you diagnose and fix code problems quickly and independently.
- Troubleshooting code takes practice.

Coming Up

- HW 2 due on Thursday, Lab 4 due at 9 AM on Friday.
- Exam will be released Friday morning, will be due on Tuesday at 11:59 PM.
- No lab on Monday.

Lecture Notes and Exercises

We will use the packages below.

```
library(tidyverse)
```

`R` or `R Markdown` will provide an error message if a problem with a computation occurs.

Encountering errors is extremely common and figuring out how to fix them is frustrating and time-consuming. But errors are helpful! They provide you with information that will help you fix your code.

In today's lecture we will introduce a step-by-step process you can follow whenever you encounter trouble with code. Additionally, we will introduce a few strategies to help you fix problematic code.

Troubleshooting steps

Step #1: Don't panic!

Errors are extremely common and fixing them quickly takes practice. Try not to panic or get frustrated.

Step #2: Find the relevant code.

Navigate to the code chunk where the error or problem occurred. If the error occurred while running a code chunk this is already done. If the error occurred while knitting, `R` usually provides a line number. If possible, navigate to the code chunk in question so you can see both the code and error at the same time.

Step #3: Read the error.

Pause and read the error carefully and in full. What does it say in plain English? **Usually**, there is enough information provided to both diagnose and fix the problem.

Below are some common errors that have enough information to fix the problem.

- `...could not find function...`

`R` can't find a function that you used. Did you include a code chunk loading the necessary packages? If you did is `eval = FALSE` included as a code chunk option?

- `...object 'some-name-of-object' not found...`

`R` can't find `some-name-of-object`. Is `some-name-of-object` spelled and formatted consistently, including correct capitalization? Was `some-name-of-object` actually created in a code chunk where `eval` is not set to `FALSE`? Is the code chunk located **above** the code chunk where the error occurred? Is it stored via `<-`? The "Environment" pane can be helpful here.

- `...unexpected 'some-symbol' in...`

Here `some-symbol` can be a comma, parenthesis, bracket, etc. This means the code you are running is not correct syntactically. Do you have an extra comma, parenthesis, bracket, or other symbol? Did you make a typo?

- `Error: attempt to use zero-length variable name`

This generally happens when you highlight the backticks in the code chunk in addition to the code. It can also occur if there is a stray `%>%` or `+` at the end of a code chunk.

- `...requires the following missing aesthetics...`

`ggplot()` is missing a necessary aesthetic. Is the `ggplot()` formatted correctly? Is the aesthetic provided consistent with what is required?

- `object of type 'closure' is not subsettable`

The above error is included because it is pretty common. It (usually) means that you tried to subset a function.

- `non-numeric argument to binary operator`

Occurs when you mix different data types in a calculation.

Often, closely reading the error will allow you to fix the problem.

Step #4: Run the code line by line.

For a code chunk with a number of lines, it is sometimes tricky to tell where exactly the error is occurring. In this situation, it is helpful to run the code line-by-line to figure out where the error is occurring.

The code chunk below has a few small errors. Let's demonstrate how running the code line-by-line helps us troubleshoot.

```
mpg %>%
  filter(class == "subcompact") %>%
  group_by(drv) %>%
  summarize(median_cty_mpg = median(cty),
            sd_cty_mpg = sd(cty),
            avg_cty_mpg = mean(cty))
```

Step #5: Examine the Documentation

If an error is from a particular function or argument, pulling up the documentation is a good step.

Documentation in `R` is extremely helpful. If you want to understand what a function does, its arguments, or examples of usage, examine the documentation. In many situations, it should be higher than the fifth troubleshooting step.

Documentation can be examined using `?` or `help()`.

- Description: a general description of what the function does
- Usage: the arguments of the function and their defaults
- Arguments: a description of each argument and what it does
- Details: details about the function
- Examples: examples of function usage

Step 6: Google

Generally you are not the first person to encounter a particular error in `R`. A well-thought out Google search can lead to others who have encountered the same or a similar problem and potentially a solution.

Include general search terms related to the error. Include quotation marks to search for an exact phrase and include aspects of the error that are unique. If the error is with a function in a particular package (`rvest`, `dplyr`, etc) include that with your search. Include `R` as a search term.

Avoid search terms that are specific to your current project. This includes datasets, variables, and aspects of your personal system (file paths, etc). You can exclude search terms using a "-" in Google.

You can also use advanced search operators. A helpful official Google link is here and an unofficial blog post is here. Check out partial search, domain search, and words by proximity.

StackOverflow is an extremely helpful site. Check out the `R` related topics with the tag `R`.

Examine a Vignette

For a broad overview of the capabilities of a package it is helpful to examine a vignette. Vignettes are "discursive documents meant to illustrate and explain facilities in [a] package" (R-Project).

Use `browseVignettes()` to see vignettes from all installed packages and `browseVignettes(package = some-package)` to see vignettes from a particular package. Use `vignette("vignette-name")` to see a particular vignette.

Let's try examining a vignette from a previous lecture.

Step #7: Post to Sakai Forums. (Not during exam!)

First (if possible) push your most recent work to GitHub.

In your Sakai Forum posting, state the following:

- What assignment and problem you are working on.
- What you are trying to accomplish.
- What issue you encountered.
- A (brief) overview of the steps you have taken to solve the issue.

Include both the code and error in a code chunk using the "Markdown editor". You can create a code chunk in the same way we do in an `R` Markdown document.

If possible, include a reproducible example of the error.

Reproducible examples

A reproducible example (reprex) is a very small, toy example used to recreate the issue for yourself and others. Often, devising a reprex helps you diagnose the issue.

Check out How to make a great R Reproducible Example and How to create a Minimal, Reproducible Example.

A reprex should be:

- minimal: use as little code as possible to reproduce the issue
- complete: should include all aspects necessary to reproduce the problem
- reproducible: should generate the issue in question

Check out the examples here and here.

The function `tribble()` is helpful for creating small, toy datasets row-by-row.

Additional Resources

- R Project Help
- Advanced Debugging
- Reprex package
- How to write a reproducible example