Preparing data for analysis and visualization in R

Using reproducible research practices

Jenine Harris
Brown School

Using comments to organize and explain code

- The meaning of the code, `states <- 29`, `states`, and `2 + states`, may seem obvious right now while they are new, but in a few weeks it might be less clear why `states` has a value of 29 and what this object means.
- One way to keep track of the purpose of code is to write short explanations in the code while coding.
- For this to work, the code needs to be written in the **Source** pane, which is opened by creating a new **Script** file.
- To create a new Script file, go to the File menu in the upper left corner of R Studio and choose "New File" from the choices.
- Then, from the New File menu, choose "R Script" (or use the shortcut command of *Control-Shift-n*).
- A new blank file will open that says "Untitled1" at the top; this is a **Script** file in the Source pane that can be used for writing code.

Code in the console vs. code in a script file

- There is a difference between writing R code in the *Console* versus writing R code in a *Script* file.
- Code written in the Console at the `>` prompt is executed immediately and not able to be edited or saved (e.g., like an instant message).
    - The History pane saves the code but cannot be saved as a code file and shared or edited.
- A **Script** file, however, is a text file similar to something written in the **Notepad** text editor on a Windows computer or the **TextEdit** text editor on a Mac computer.
    - A Script file can be edited, saved, and shared just like any text file.
    - When Script files of R code are saved they have the **.R** file extension.

Using a script file

- Open a new Script file and rewrite the earlier code (or bring it in from the history tab).
- Include information about the code in by typing comments that explain the purpose of the code.
    - Each line (or set of lines) of code is preceded by a short statement of its purpose.

```
# create an object with the number of states with
# legal medical marijuana in 2017
states <- 29

# print the value of the states object
states

# determine how many states there would be if 2
# more passed this policy
2 + states
```

Using comments to organize and explain code

- These statements are called **comments** and the practice of adding comments to code is called **commenting** or **annotation**.
- The practice of commenting or annotating is one of the most important habits to develop in R or in any programming language.
- In R, comments are denoted by a hashtag `#`, which notifies R that the text following the `#` on the same line is a comment and not something to be computed or stored.
- Comments are not necessary for code to run, but are important for describing and remembering what the code does. Annotation is a *best practice* of coding.
- When writing and annotating code, keep *two* goals in mind:
    - Write clear code that does not need a lot of comments, and
    - Include useful comments where needed so that anyone (including yourself in the future) can run and understand your code
- Clear R code with useful annotation will produce *reproducible* work, which is one of the most important characteristics of good data science.

Running code from the console or script file

- When writing code in the **Console** pressing Enter would run the code.
- When writing code in a **Script** pane, there are many ways to run the code.
- To run all the code at once, highlight all the code at once and click on *Run* at the top right corner of the **Source** pane.
- To run one line of code, highlight the one line of code or put the cursor anywhere in the line of code and click *Run*.
- The keyboard shortcut for *Run* is *Control-Enter* (or *Command-Enter* on a Mac), so putting the cursor on a line of code and pressing *Control-Enter* will run the code on that line.

Example of running some code with comments

- Try running the code from earlier to review the output.

```
# create an object with the number of states with
# legal medical marijuana in 2017
states <- 29

# print the value of the states object
states
```

```
## [1] 29
```

```
# determine how many states there would be if 2
# more passed this policy
2 + states
```

```
## [1] 31
```

Including a prolog to introduce a script file

- A prolog is a set of comments at the top of a code file that provides information about what is in the file.
- Including a prolog is a *best practice* for coding.
- There are many features a prolog can have, including:
    - Project name
    - Project purpose
    - Name(s) of data set(s) used in project
    - Location(s) of data set(s) used in project
    - Code author name (you!)
    - Date code created
    - Date last time code was edited

Example of a really fancy prolog

A formal prolog might be set apart from the code by a barrier of hashtags, like this:

```
# PROLOG   ##################################################

# PROJECT: NAME OF PROJECT HERE #
# PURPOSE: MAJOR POINT(S) OF WHAT I AM DOING WITH THE DATA HERE #
# DIR:     list directory(-ies) for files here #
# DATA:    list dataset file names/availability here, e.g., #
#          filename.correctextention #
#          somewebaddress.com #
# AUTHOR:  AUTHOR NAME(S) #
# CREATED: MONTH dd, YEAR #
# LATEST:  MONTH dd, YEAR #
# NOTES:   indent all additional lines under each heading, #
#          & use the apostrophe hashmark bookends that appear  #
#          KEEP PURPOSE, AUTHOR, CREATED & LATEST ENTRIES IN UPPER CASE,
#          with appropriate case for DIR & DATA, lower case for notes #
#          If multiple lines become too much, #
#          simplify and write code book and readme. #
#          HINT #1: Decide what a long prolog is. #
#          HINT #2: copy & paste this into new script & replace text. #

# PROLOG   ##################################################
```

Example of a basic prolog

An informal prolog might just include:

```
####################################
# Project name
# Project purpose
# Code author name
# Date last edited
# Location of data used
####################################
```

Adding the prolog to a script file

Including a prolog with the code could look like this:

```
##########################################
# Project: Foundations in Biostatistics
# Purpose: Code examples for week one
# Author: Jenine
# Edit date: July 25, 2020
# Data: No external data files used
##########################################

# create a variable with the number of states with
# legal medical marijuana in 2017
states <- 29

# print the value of the states variable
states

# determine how many states there would be if 2
# more passed this policy
2 + states
```

Naming your files

- Another formatting best practice like commenting and writing a prolog, is to include information in the file name that is a reminder of what is contained in the file.
- One commonly recommended way to do this is using `date_project_author` as the file name, for example: **20200726_chap1_jenine.R**.
    - Note that the year is first, followed by the month, and then the day
    - Also note that, when the month or day is a single digit like 2 or 7, add a 0 in front so that the file name appears in the right order in a list of files

Naming objects

- The final formatting tip to remember when coding is to use meaningful names for objects so they are easy to understand.
    - It is much easier to guess what is in an object called `states` than what might be in an object called `var123`.
- Keep in mind that some letters and words are already used by R and will cause some confusion if used as object names.
    - For example, the uppercase `T` and `F` are used in the code as shorthand for `TRUE` or `FALSE` so are not useful as object names.
    - When possible, use words and abbreviations that are not common mathematical terms.

Naming constants

- There are recommended methods for naming objects in R that depend on the type of object.
- The `states` object is a *constant* because it is a single numeric value.
    - The recommended format for constants is starting with a "k" and then using **camel case**.
    - Camel case is capitalizing the first letter of each word in the object name, with the exception of the first word (the capital letters kind of look like camel humps).
- There are two ways to rename the states constant.
    - Remember, the arrow `<-` assigns whatever is on the right of the arrow to the name on the left of the arrow
    - Use the arrow to make an entirely new object named `kStates`
    - Or, use the arrow to assign the old object of `states` a new name (no need to do both, just use the one that your brain prefers!)

```
# make a new constant object with well-formatted name
kStates <- 29

# assign the existing states object a new name
kStates <- states
```

Deleting unneeded objects

- Now `states` and `kStates` are both listed in the environment.
- This is unnecessary since they both hold the same information.
- Remove an object using the `rm()` function:

```
# remove the states object
rm(states)
```

Using a function

- This is different from what they had been doing so far since it includes a function, `rm()`, and the name of an object, `states`.
- This format is common in R, having some instruction or function with a set of parentheses like `function()`.
    - Inside the parentheses, there is typically the name of one or more objects to apply the function to, so `function(object)` is a common thing to see when using R.
    - The information inside the parentheses is called an **argument**, so the `states` object is the argument entered into the `rm()` function.
- Sometimes R functions will need one argument to work, sometimes functions will require multiple arguments.
- Arguments do not all have to be objects, some are just additional instructions for R about how the functions should work.
- Sometimes R users will call `rm()` and other functions "commands" instead of "functions" and these two words tend to be used interchangeably.

Naming variables

- Variables are measures of some characteristic for each observation in a data set.
    - For example, `income` and `voted` are both variables.
- Variable objects are named using **dot case** or **camel case**.
    - Dot case puts a dot between words in a variable name while camel case capitalizes each word in the variable name (except the first word).
- For example, a variable measuring the number of medical marijuana prescriptions filled by each of 100 cancer patients in a year could use:
    - a dot case variable name like `filled.script.month`
    - a camel case variable name `filledScriptMonth`
- Dot case is the preferred method of naming objects but camel case is frequently used and there are other variable naming conventions used by some R users.

Naming functions

- Functions are objects that perform a series of R commands to do something in particular.
- They are usually written when someone has to do the same thing multiple times and wants to make the process more efficient.
- Writing functions is a more advanced skill, but it is good to know the naming convention, which is camel case with the first letter capitalized (this is also called "upper camel case" or "Pascal case").
    - For example, a function that multiplies everything in a data set by 2 might be called `MultiplyByTwo` or something similar.

Developing your coding style

- When starting to code it is good to take a little time and develop a coding style with a consistent way of annotating code and the recommended way of naming things.
    - For example, do you prefere dot.case or camelCase for variables?
- One final code formatting tip is to limit the length of a line of code to 80 characters or shorter.
    - Set this up in R Studio by going to **Tools -> Global Options -> Code -> Display**
    - Check the box for **Show margin** and click **Apply**
    - There will now be a faint line in the window at the 80 character mark

Check your understanding

- Open a new script file (or modify the existing file if you have been following along) and create a prolog.
- Make a constant named `kIllegalNum` and assign it the value of *21*.
- Subtract 2 from the `kIllegalNum` object and check the output to find the value.

```
##########################################
# Project: Foundations in Biostatistics
# Purpose: Code examples for week one
# Author: Jenine
# Edit date: July 26, 2020
# Data: No external data files used
##########################################

# create a constant with the number of states with
# medical marijuana illegal in 2017
kIllegalNum <- 21

# subtract 2 from the constant
kIllegalNum - 2
```

```
## [1] 19
```