Video Link

CPP 528 Lab-02 Part 2: Build a Variable Filter

Courtney Stowers, M.S.

Arizona State University

Audio Transcript (video duration: 21 minutes, 10 seconds):

Hello everyone and welcome to Lab 02 – Data Wrangling.

This week we will be discussing portability and version control using RStudio Projects and the Github Desktop. If you do not already have Github Desktop installed on your computer, you can visit desktop.github.com. In RStudio, you may also receive a request to install Git. This will allow RStudio to connect with Github Desktop. You can download Git by visiting git-scm.com/downloads. Both of these links will be posted to the Canvas and Github Pages sites.

To begin this week's assignments, visit our course Github Page and click on the Course Schedule. This will then allow you to scroll down to Week 2 – Data Management where you will find the Overview, Lecture, Reflection Topic, and Lab tabs.

When you click the Lecture tab, you will be taken to the Getting Started page. This page will provide you with background information on the data that we will be using to study Neighborhood Change. Specifically, we will be using Census Data. The Census Data that we will use are the Longitudinal Tracts Database.

Within the lecture, you will find background material that gives you more details on the data as well as good data practices. Most importantly, you will want to pay attention to the sections on portability and version control. Here, you will find lectures as well as step-by-step tutorials instructing you on how to set up your RStudio Project and your Github Desktop account. In this video, I will show you how to put these tools into action, but these videos will give you more details on how to get them set up.

Once you're set up, you will then want to visit the lab page. Here you will find a link to the step-by-step instructions as well as an overview of the lab. The data section will provide you with the downloads you will need of the Census Longitudinal Tabulated Database (LTDB) files as well as the codebook with details on the American Community Survey. It also provides helpful instructions on how to set up your directory structure in your private Github repository. We will go over this in the video.

Once you have your structure set up, you can then continue to read more details about the lab as well as the final project. There are great tips on documentation that you will also want to implement in your set-up.

Once you are done reviewing all of this material, you can then click the Lab 2 Instructions link. This is where you will receive step-by-step details on how to complete the lab. The first part of the lab, the Data Concordance, is fairly straightforward. You can download the data that you will use for the course,

as well as the codebook. To set up the data that you will need for this lab, you will want to copy and paste the definitions from the codebook into your spreadsheet.

Once you have completed that step, you will then move on to the slightly more challenging Part 2 where you will build a variable filter. I will show you how to do that in this video.

So, to get started, we will open up RStudio. Hopefully you have already followed the steps in the other video to set up your project to get started on working in RStudio. If not, you can select the blue box up here to start a new project and set up version control via Github.

Once you have done that, you should then receive a folder with all of the files from your private Github Repository already set up here. This is where you will want to begin creating new folders to complete the set up of your directory.

All of the details on your repository and your set up are available, once again, under Lab 02. The folders listed higher up in the structure are the parent folders and the items that fall under each folder are listed here.

As an example, you can see this set up here on RStudio. I've created my data folder and there are sub-folders here titled raw, rodeo, and wrangling. This is where my data will live and where I can access it to complete my labs. Similarly, I also gained access to some other items that were automatically included in the private Github repository.

For this lab, most importantly, we'll want to take a look at the analysis folder. In this folder, we will find a `utilities.R` file. You can go ahead and click on that file to open it. Once it's opened, you will discover that there is a function listed inside of this file.

The function is named today and it has the ability to calculate the date that the function is run. I'll show you shortly how this relates to part 2 of our lab.

As you can see here in the overview of part 2 of the lab, the `utilities.R` file is highlighted. This is the file where you will import your functions to use throughout the duration of your final project.

So let's get started. Following the initial steps listed in the lab, we'll want to set up our R environment. The project should already be set up to use `renv`. If you have any issues setting up that app—that package, I'm sorry—you can follow the videos listed on the lab website. Once you have that installed, you should be all set to go.

You may want to use some specific packages within your lab. You can list them in a library code chunk within your RMD file. In order to do this, you'll want to select `File > New File > RMarkdown`. This will allow you to create the file you need for your lab.

Once this is complete, you'll want to save it in your labs folder. You can see mine listed here under Wk01 (Week 01).

Next, you'll want to load the data that we will be using for the lab. You can find code for the first two files listed here on the Lab website. You can simply copy and paste.

You can also choose to read in some of the other data that we downloaded. Following the instructions listed on our data structure, you can store these files under the `rodeo` folder. This is where your data dictionary will live. The additional files will be stored in the `raw` folder.

Here, I have read in the files using `read_csv()` and I've added a few options such as indicating that there are column names and choosing not to show the column file types.

Once you have your data loaded, the next thing you're going to want to do is import your functions.

Once again, your course repository came pre-installed with a utilities file. You can navigate to this under `analysis > utilities.R`. When you click the file, it will open here.

If you recall from earlier, you can find the today function already listed as an example. So, how can we pull this file into our lab so that we can use the functions?

You'll want to use the `source()` function as well as the `here` package and then you can list the directory location of the file: `"analysis/utilities.R"` .

You can also choose to create a new .R file with functions of your own. To do this, once again, you will want to select `File > New File` and then you can select `R Script`. This will give you a blank file that you can fill out.

I created mine as `helper_functions_cs.R`. Here, I have a variety of functions created with examples of how to use them. I'll show you shortly how to use these in your lab.

To begin, we will try the today function. We'll want to run the code to pull these files into our lab. Once we do this, we'll see a functions section pop up in our global environment with a list of all the functions that we have available as well as the inputs for the functions listed here.

For example, we can find the today function and that it doesn't require any inputs. So we can just run it by typing today and two parentheses [`today()`]. When we click the run button, we get today's date.

Now, if you recall from the lab, we're asked to create a variable filter. This will allow us to search through our files and create filters that filter variables either by theme or group. Specifically, you'll want to be able to filter by the category fields that you created in your data.

To get an idea of what the categories should look like, we'll preview our data here.

Here you can see in the `dd` table that we created of our data or if we go here, we can see that it's the `ltdb-data-dictionary.csv` file that we loaded and saved into dd.

We now have categories such as as id, tract attribute, age-race variables, age variables, race, nationality, socioeconomic status, nationality, poverty-race so forth and so on. Your categories may be different depending on how you chose to group the various variables that you had in your data set. You're free to name them however feels best for you, but they should be easily identifiable and they should give a general idea of what the variable means.

For my example, in my `helper_functions_cs.R` file, I created a keyword search. The keyword search uses the `grepl()` function to search through the definition column of my data set and return the root, root2, category, and definition columns of the dd file.

So, in other words, it will search through the definition column for a keyword of my choice. This will filter out the data and provide me with the specific variables relevant to the topic that I am interested in.

So, let's see how this works. In my `helper_functions_cs` file or in your `utilities.R` file, you'll want to create a brief overview of how your function works. So first, you'll want to give it a name and you can add four dashes after the name to flag it on the bottom of RStudio.

So here I gave an example of keywords and how to use them. So for example, if I think there's going to be a space before my keyword, I'll leave a space before the search term. I can search for a word that starts at the beginning of a sentence—indicated by this little triangular symbol here. Or I can use a myriad of other grepl search pattern options.

I'll include a link to examples of these search patterns on Canvas and the Github pages site.

I can also create a series of multiple search terms and store them in a keywords variable.

Now I'll create my function and name it `keyword_search`. So using `function()`, I'll list the requirements: data and keywords. Next, I'll write out my function below and how I want it to work. Here I am using the `grepl` package and implementing keywords that I want it to search in the definition column. Next, I want it to return a data set with the root, root2, category, and definition columns.

Let's try this as an example. So I'm searching for the word family within the definition column of my data set. So here.

So I'll return searches that have the word family in them. Once again, based on the helper functions here, my function is named `keyword_search()` so I'll list it here and then press enter. And now, I'm returning columns that have family or words that are similar to family listed within my search and within the definition column of the data.

So now I can find that I have a family variable that discusses total families in population, a variable for families who have poverty status, a variable for families with a female head, a variable for families with children in poverty, a variable for the percent of female-headed families with children, and a variable for the percent in poverty, families with children, and a variable for the percent of multi-family units.

Similarly, I can try another function called `filter_data()` and I can find this function with my flag. Here, I give an indication that the `tidyverse` package is needed in order for this code to work and then I list example search terms.

The filter data function works similarly to my keyword function. However, this one actually filters out the data. I use the filter option to filter out categories based on the search terms that I enter and then it returns my filtered data.

Let's see how this works.

I will comment out the keyword search function and I will allow my `filter_data()` function. Once again I'll enter my data set, `dd`, as the first option in the function because if we refer over here to the global environment we can find the filter_data function requires a data set as well as the specific search terms we need to filter our data.

Video Link

So first I'll enter dd for the data set and then I'll choose to filter by the term `"age".`

If we return to our functions list here, we can see that this will be filtering out the category to have age.

Once again looking at the data set, we can see age and all of the different options that are available for category.

So now we'll run our function.

Here we can see all of the options that have age. It gives us a total of 12 rows.

We can also change this to `"^age"` to search for options that have age at the start of the category, but we see that this doesn't return any options for us. Then we can try `"*age*"` to see what that gives us. Once again, there are no results.

So why does it not work?

Well, if we return to our function; unlike the keyword search that used grepl(), this function specifically searches for the category to be an exact match so adding grepl-like pattern matching does not work because it's looking for an exact string.

To make your filter more flexible, you may want to implement the `grepl()` function so that you can filter your data, but you have more of a keyword option where you can search for a pattern that's similar and not as strict.

You can play around with these function as many times as you want and strategize different ways to help search your data.

If you have any questions at all, please feel free to reach out to me via email or on the Github Issues Board.

Thank you and I hope that you enjoy this lab!