Analysis of the main sources of variation in completing the Getting and Cleaning Data Coursera Project

Siruo Wang

Department of Biostatistics, Johns Hopkins Bloomberg School of Public Health, 615 North Wolfe Street, Baltimore, MD 21205, USA

Abstract

As data analysis skill has become a more and more important skill for students, we are interested to know how students are performing data analysis in R. We have retrieved 20,824 repositories for the Getting and Cleaning Data Coursera project from Github, scrapped their R script for data analysis, and save the scripts as a complete list. By looking into each student's R script, we summarize the main sources of variation in students completing the Coursera project such as the drop rate for the data analysis Coursera classes, comment and space lines usage, library and function usage.

Introduction

Coursera is an educational online platform which contains thousands of courses taught by instructors from universities and institutions around the world. Components of Coursera courses include but not limited to online recorded video lectures, graded assignments, and discussion forums open to the community. The goal of Coursera is to provide a low cost, open access, and short time training platform to enhance education in specific areas. Among the many courses being taught in the Coursera platform, data analysis is one of the popular areas as data analysis skills are required more often when students are looking for job openings.

In the Coursera platform, tens of thousands of students are taking data analysis online classes. Each of the data analysis class must contain a fundamental but crucial part: learn to get and clean data. Since majority of the data analysis needs to be performed on a tidy dataset, students who are taking these classes are required to submit a final project, Getting and Cleaning Data Project, to their Github repository. Although students share the same goal of preparing a tidy dataset that can be used for further data analysis, they have different preferences in providing methods to complete the ultimate goal. Therefore, we want to explore all students' scripts for completing the goal of Getting and Cleaning Data project, and summarize main sources of variations in solution to the completion of the project.

In Methods section 1, we outline procedures of retrieving all available repository names for the Getting and Cleaning Data Project from Github. In Methods section 2, we outline procedures of scraping each students' R script for performing the analysis under the whole set of repository names retrieved from section 1. In Methods section 3, we outline the detailed steps of conducting analysis on students' scripts to the Getting and Cleaning Data project. In Results section, we demonstrate the results of main sources of variation in students completing the Getting and Cleaning Data project. In Discussion section, we discuss the limitations of our current methods and further steps to continue the analysis.

Methods

1. Retrieving repository names

As students are taking the data analysis classes on Coursera and constantly pushing their final projects to Github, the number of repository results increases every minute. Updated to October 7, 2017, there are currently over 31K repository results available under the Getting and Cleaning Data project. For each repository result, it is composed of a username and followed by the course title. For example, the repository name is in the following format:

username/GettingAndCleaningDataCourseProject

Our goal is to retrieve a complete set of available repository names in the above format by scraping the Github website using the "gh" [1] R package. To ensure we can retrieve the most repository names at a time, we set up the Github API authorized key and save it securely under the .gitgnore file. By using the secure Github API authorized key, we are able to pull at most 1K repository names at a time. After we reach the 1K limit, we will receive API access error and deny our further requests. In order to fix this problem, we incorporate the R Command Sys.sleep(60) in our code of retrieving repository names. The command stops the pulling request to Github for 60 seconds between retrieving each 1K repository names.

Although we fix the API access error problem, we run into another Github pulling data issue. In every 60 seconds, we send a request to pull 1K repository names from Github. In theory, we should retrieve all available repository names if we give our code enough time to run. However, Github returns the same 1K repository names to us by default. It is being said that we are retrieving the repetitive1K repository names every time we send the pulling request. A solution to this issue is to firstly sort the Github repository names according to a type of mechanism, and then retrieve the repository names based on the sorting order.

After consulting with the instructor of the Coursera data analysis course, Jeff Leek, we notice that the data analysis Coursera course was created on December 2015 for the first time. We believe that the final Getting and Cleaning Data project should be submitted to Github after the Coursera course was created and taught online. Therefore, we decide to use the repository created date as our sorting mechanism to better retrieve data. Since we are certain that there are no more than 1K repositories created within 14 days period, we can use the "lubricate" [2] R package to break up days into14 days period starting from 2015-12-01 to 2017-09-25. Then, in every 60 second, we send request to Github to pull repository names within each 14 days period. Allowing the code to run for 1 hours, we are able to pull 5K different repository names instead of the total 31K repository names available on the Github website.

We conclude that one reason for not getting the whole set of repository names could be that students created a repository even before the Coursera course starting date, and later changed the repository name to be the course name after taking the data analysis class. If this is the case, we can improve our date sorting mechanism by setting up a longer time range. After we manually explore a random set of repositories, we decide to trace back our repository created date as early as 2008-01-01 since we do not find many repositories created before 2008. Now we create 237 date periods from 2008-01-01 to 2017-09-25, each of which contains 14 days. Requesting repositories from Github using such date mechanism, we are able to retrieve 20,824 repository names in total when running our code for 4 hours.

2. Scraping R scripts for the getting and cleaning data Coursera project

After we retrieve the 20,824 repository names from Github, we want to look closely to students' R scripts under their repository names. In order to successfully complete the data analysis class, students are required to submit the only R file named *run_analysis.R*, which contains their code to prepare a tidy data set. Therefore, we plan to scrape students' repositories and look for this R script.

First, we create an empty list named *rfile_list* which has length 20,824. Second, we use the "gh" [1] R package to look for a file with *.r* or *.R* extension under each of the 20,824 repositories using the following format:

GET /search/code?q=repo:repository_names+extension:r

When we try to retrieve the url path to the R script under each repository name, we find that some urls to the R file contain special characters such as space, which lead to access errors. In our code, we have handled majority of the special cases to successfully retrieve urls. The urls we have retrieved have the following format:

https://raw.githubusercontent.com/ repository_name/master/run_analysis.R

Third, we use read Line function in R to read in each student's R script line by line and save the script as an element in the *rfile_list*. Note that when we cannot find any R file under a student's repository, we put *NA* instead of the R script into the list.

Similar to requesting repository names from Github, when we request urls to R scripts, we may run into Github API access error if we frequently send in pulling request. In our code, we handle this issue by stopping our code for 5 seconds each time we read in a student's R script.

By conducting the above steps, we are able to retrieve a complete list of students' R scripts under each student's repository name after we let our code scrapping Github website for 30 hours. Now with a complete list of R scripts for 20,824 repository names, we can further perform analysis on summarizing main sources of variation in how students complete the Getting and Cleaning Data project.

3. performing analysis

Since we retrieve the R scripts of the Getting and Cleaning Data project for all 20,824 repositories, we have date from the whole population. We are unable to conduct any statistical tests on the whole population, but we are able to do enough observational analysis on the R scripts to summarize main sources of variation in people completing the project.

Although there are tens of thousands people are taking the data analysis classes on Coursera, we would like to know how many of them actually complete the course. Using our data, we are able to calculate the dropout rate by counting how many elements in the *rfile_list* are *NA*s. Because the *run_analysis.R* script is required for the Getting and Cleaning Data final project, if we cannot find the R script, we consider that student drops out from the course at some point.

We are also interested to know people's preference of using comments and space lines. It is often the case that students tend to use many space lines to make their code look longer, but the pure code may not be as many as it appears. We calculate the number of comment lines and space lines that each student uses and compare the pure code length to the full code length for the whole population.

Preparing a tidy data set usually involves importing necessary libraries and applying useful functions, but people have different preference on their library and function choices. It is interesting to know what libraries and functions are used more often than the others. Therefore, by scrapping each students R script, we summarize what distinct libraries and functions are being used in the whole population, and how frequently each library and function are used across students. Note that people import the same library differently. For example, library(dplyr), library("dplyr"), library('dplyr'), SuppressMessage(dplyr), etc. are indicating the usage of the same package, but students import them in a slight different way. When parsing the R script as text into R, our code handles this issue and recognize them as the same library usage.

Results

We calculate the dropout rate for the data analysis Coursera class is 0.058. There are 1211 out of 20,824 students who do not have the R script for the Getting and Cleaning data final project.

We compare the length of students' full R scripts to the length of R scripts that have comment and space lines removed. Figure 1 below indicates the mean difference of code length before and after we remove the comment and space lines for the whole population. From Figure 1, we observe plenty outliers in our population which contain thousands lines of codes in their R scripts. In Figure 2 below, we remove the outliers with code length greater than 300, which indicates a clearer difference in mean before and after we remove the comment and space lines.

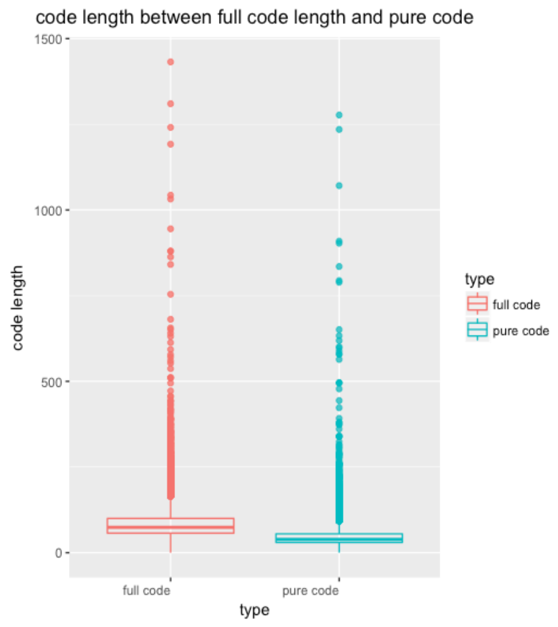

Figure 1

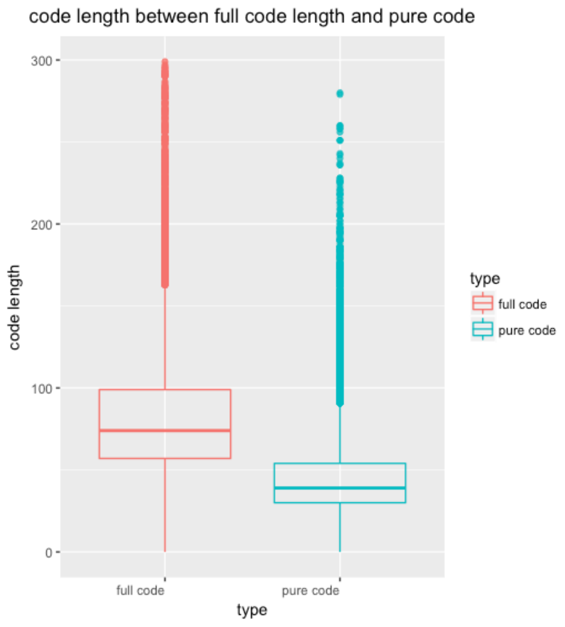

Figure 2

We look at the library usage and function usage preferences by summarizing what distinct libraries and functions appear in the whole population, and count the frequency of each. In Figure 3 and

Figure 4, we plot the top 30 frequently used R packages and R functions. We notice that *dplyer*, *plyer*, and *reshap2* are the most frequently used R package, and *Read.table*, *names*, *gsub*, *c* are the most frequently used R functions that students have in their R script to process the raw data and prepare a tidy data set.

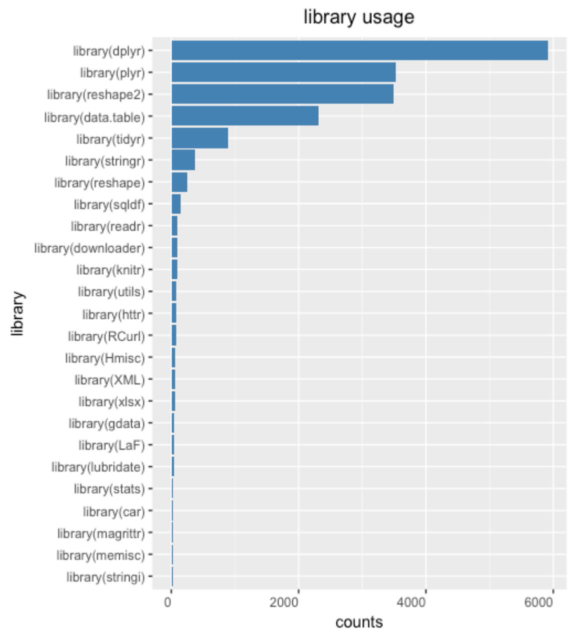

Figure 3

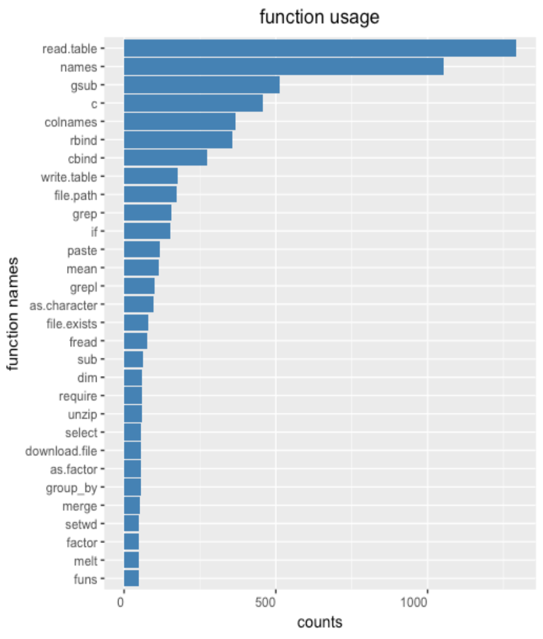

Figure 4

Discussion

In Methods section 1, we outline the procedures of retrieving repository names from Github based on 14 days period for the Getting and Cleaning data project. Because we use our R code to pull repositories created from 2008-01-01 to 2019-09-25, we are able to retrieve 20,824 repository names rather than all 31K available repositories for the Getting and Cleaning data project, which is a limitation of our program. If we are willing to sacrifice more time in running code to search for repositories created before 2008, we are likely to retrieve more repository names than what we currently have.

In Methods section 2, we outline the procedures of scrapping students' R scripts under their repository names retrieved from Methods section 1. By conducting the outlined steps, we are able to find and scrape majority of students' R scripts. However, there are two limitations in our code. One limitation is that we have only handled majority of the special characters within the urls to R scripts but not all of them. It is possible that we save some students' R scripts as *NA* because we are unable to catch the special characters in their urls to R scripts. Another limitation is that we only retrieve R scripts from master branch and discard R scripts from other branches by using the following url:

https://raw.githubusercontent.com/repository_name/master/run_analysis.R

Reference

1. Jennifer Bryan and Hadley Wickham (NA). gh: 'GitHub' 'API'. R package version 1.0.1.https://github.com/r-lib/gh#readme

2. Garrett Grolemund, Hadley Wickham (2011). Dates and Times Made Easy with lubridate. Journal of Statistical Software, 40(3), 1-25. URL http://www.jstatsoft.org/v40/i03/

Cite Stephen's code in getting urls

Cite Andrew's code in using lubricate R package