

Gradescope Autograder Tutorial with R

Danny Klinenberg*

Last Updated: 2020-09-07

Contents

1	Introduction	3
1.1	Welcome	3
2	Setup	4
2.1	R and RStudio	4
2.2	Github	5
2.3	Gradescope	13
3	Layout	18
3.1	hw_template	19
3.2	manuals	20
3.3	helper_functions	23
3.4	class roster	25

*University of California, Santa Barbara. Special thanks to Victor Huang and Zihao Zheng for their tireless work on the development and implementation.

4	Every Step you Take: Making an Assignment from Start to Finish	25
5	Writing the Autograder	31
5.1	Prompt and Example Autograder	34
6	Troubleshooting	40
6.1	Figuring out Filepaths	40
6.2	Choosing the version of R	41
6.3	When the Autograder Breaks	41
6.4	Troubleshooting Student's Code	42
6.5	Writing Questions	42
7	Building the Infrastructure	47
A	Appendix	48
A.1	Version Control	48
A.2	List of Functions	51

1 Introduction

1.1 Welcome

Welcome to **Data Wrangling for Economics**! This course will focus on students learning the basics of data cleaning and manipulation through the statistical software R. All homework and exams will have a coding component. To grade this, the class is utilizing a software called **Gradescope**. This software allows universities to automatically grade assignments including code. In this class, students will submit both their code and a brief writeup simulating real world data analysis. This manual will focus on the code aspect of their submissions.

Students will upload their code to Gradescope which will grade it. In order for Gradescope to do this, the instructor must supply an **autograder** for each assignment. An autograder is a script that reads scripts and assigns a score to the accuracy. This tutorial is meant to teach the reader how to write a Gradescope autograder in R.

Working with Gradescope's autograder function requires a great deal of computer knowledge beyond that of a normal social science graduate student. To alleviate this burden, a great deal of infrastructure has been built. This infrastructure automates a lot of the technical issues saving both time and hair.

After reading through the manual, you should:

- (1) Know what R and Rstudio is as well as how to download both programs.
- (2) Know how to log into Github, clone repositories to your computer, pull the repository to your computer, and upload.
- (3) Know what Gradescope is, how to log in, and upload autograders.
- (4) Know the components of an autograder.
- (5) Be able to write your own autograder.

(6) Upload the autograder to Gradescope through Github.

(7) Confidently debug your autograder.

This manual is written for incoming TAs. It is assumed the infrastructure for the class is already in place. If you are building the course from scratch, see the *Building Infrastructure* section at the very end of the manual.

The flexibility of the infrastructure discussed allows for usage beyond **Data Wrangling in R**. While the target audience is the incoming TAs, the manual and infrastructure have uses for all individuals interested in assigning coding homework in R. Other resources exists for C++, Python and R. Please see the [community resource boards](#).

This is a working draft of the manual. All feedback is welcome. Please do not share this manual with current or potential students of **Data Wrangling in R**.

2 Setup

This section reviews the necessary programs and getting started.

This is a working draft of the manual. Please do not distribute. Please do not share this manual with current or potential students of **Data Wrangling in R**.

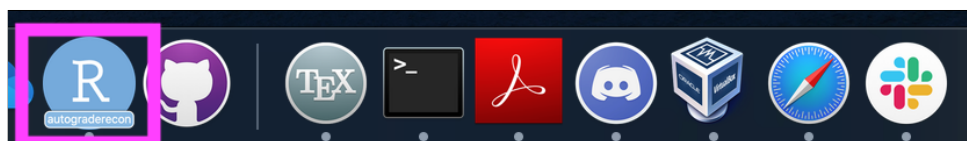
2.1 R and RStudio

R is a powerful statistical software used for data analysis. R can be downloaded [here](#). Pick the appropriate version for your device.

Rstudio is a “skin” for R. This allows the user to script, create latex documents integrated to code, and much much more! The program can be downloaded [here](#). Pick the free desktop version.

2.1.1 RProjects

R projects create environments within R. These environments include files (Rfiles, datasets, etc.) as well as saved settings. This is extremely useful for integration with Github. You'll notice you are in a project when the R banner has a logo around it:



Creating projects will be discussed in the [Github](#) section.

2.2 Github

Github is an online code sharing platform. It allows for collaboration among many individuals on the same piece of code, saving past version. All autograders for this course will be uploaded to Github. This is done for two reasons: (i) reproducibility of the course and (ii) efficiency in debugging.

We will be working in a shared repository from individual accounts. Think of this as a shared folder in Google Drive: we can all edit, upload, and download in real time.

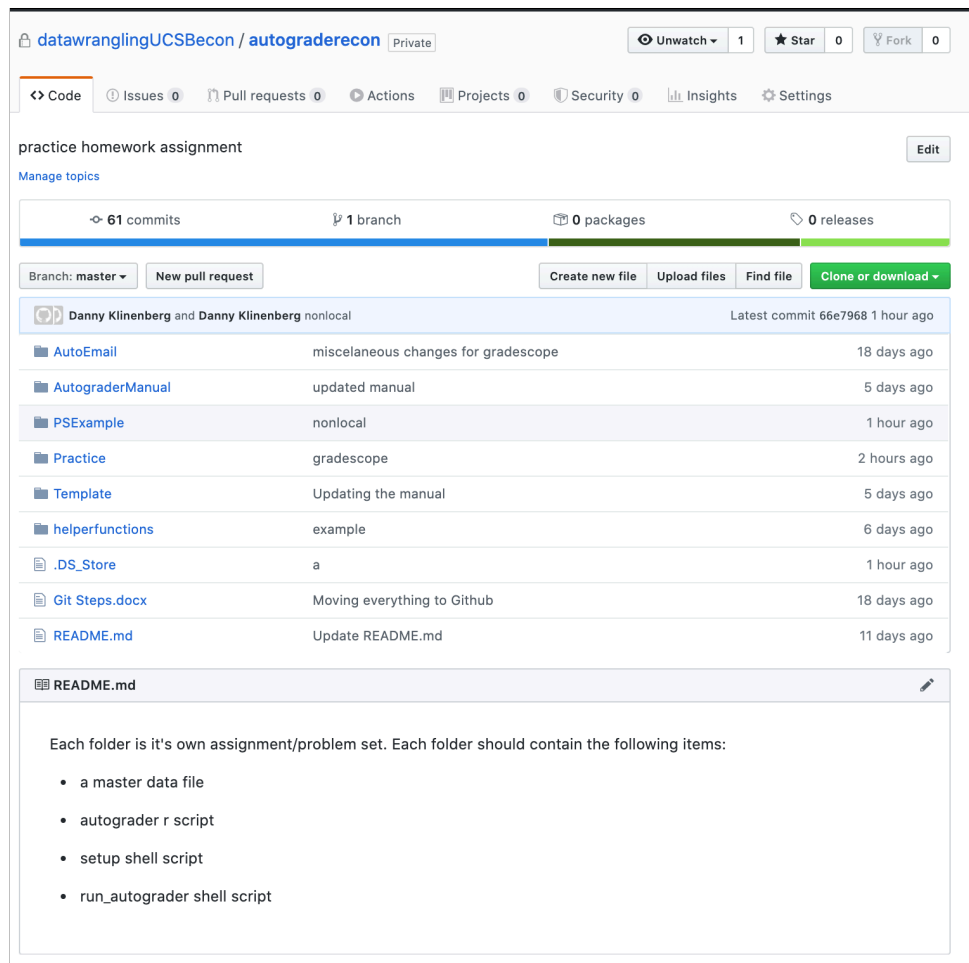
At the end of this section, you should be able to:

- log into Github
- create a clone of the repository on your computer
- be familiar with the main github commands
- upload and download from Github using RStudio

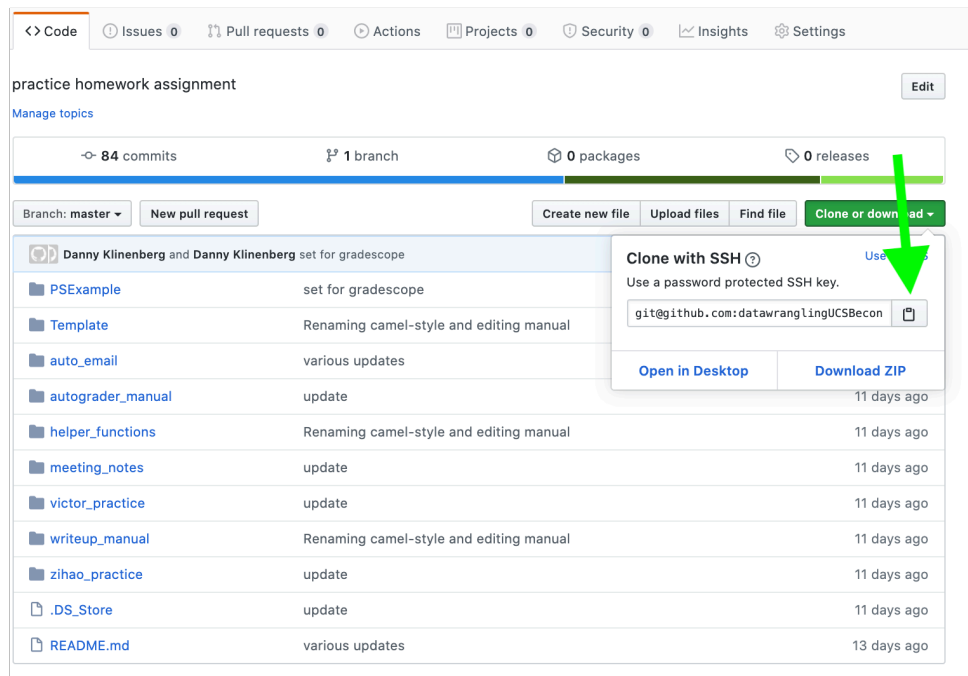
This is not an exhaustive manual. It is an introduction to Github with RStudio.

2.2.1 Getting Started

First, you will have to create a Github. Simply log onto github.com and sign up. It's free! After creating a Github, you will be invited to the Github repository. Accept the invitation. The repository will look something like this:



Click **clone or download** (the green button) then copy the *password protected SSH key* (where the green arrow is pointing).



In order to make working with Github as easy as possible, the TA will clone the repository to their home computer. This way they can do all their edits natively and then upload back to Github for everyone to use. If you are working with a PC, you will need to download [GitBash](#). Mac computers have GitBash integrated into their terminal.

- Next, open RStudio. Go to **File** and choose **New Project**.
- Choose **Version Control**.
- Click **Git**.
- paste the *password protected SSH key* into **Repository URL**. Name the project whatever you want and save it wherever you want.

Congrats! Now you have the Github repository saved onto your computer. Everything you do on your computer can be synced with the Github repository and everything on the repository can be synced with your computer! The next section will go through how that works.

2.2.2 The Basics of Git and Github

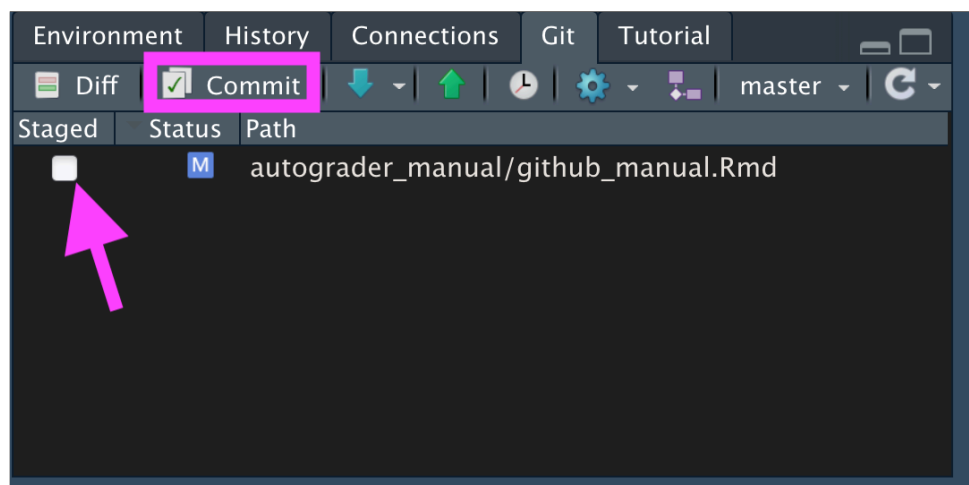
Many, many, many sources exist providing tutorials on using Github (e.g. [here](#), [here](#) and [this one](#)). This manual will focus on the Github steps relevant to the course. Please read through at least the first two links.

When we run into conflicts, check out this [link](#).

The big ideas you need to take away from the links are: `git pull`, `git add`, `git commit`, `git push`, and `version control`.

2.2.3 The Git pane

The Git pane is a GUI interface for Git. Below is a picture of the pane with some key features highlighted:



- Staged (pink arrow): Click the box to `git add`.
- Commit (pink box): This is equivalent to `git commit`.

Pro Tip: `git commit` comments should be short. Each commit should be one task. Examples of this would be writing a line in a manual, or writing one answer for an autograder. This makes it easier to retrace steps in the future.

- Blue arrow: `git pull`. Always do this before doing anything. `git pull` syncs the files on your computer with that in Github.
- green arrow: `git push`. This syncs Github with your changes on your computer.

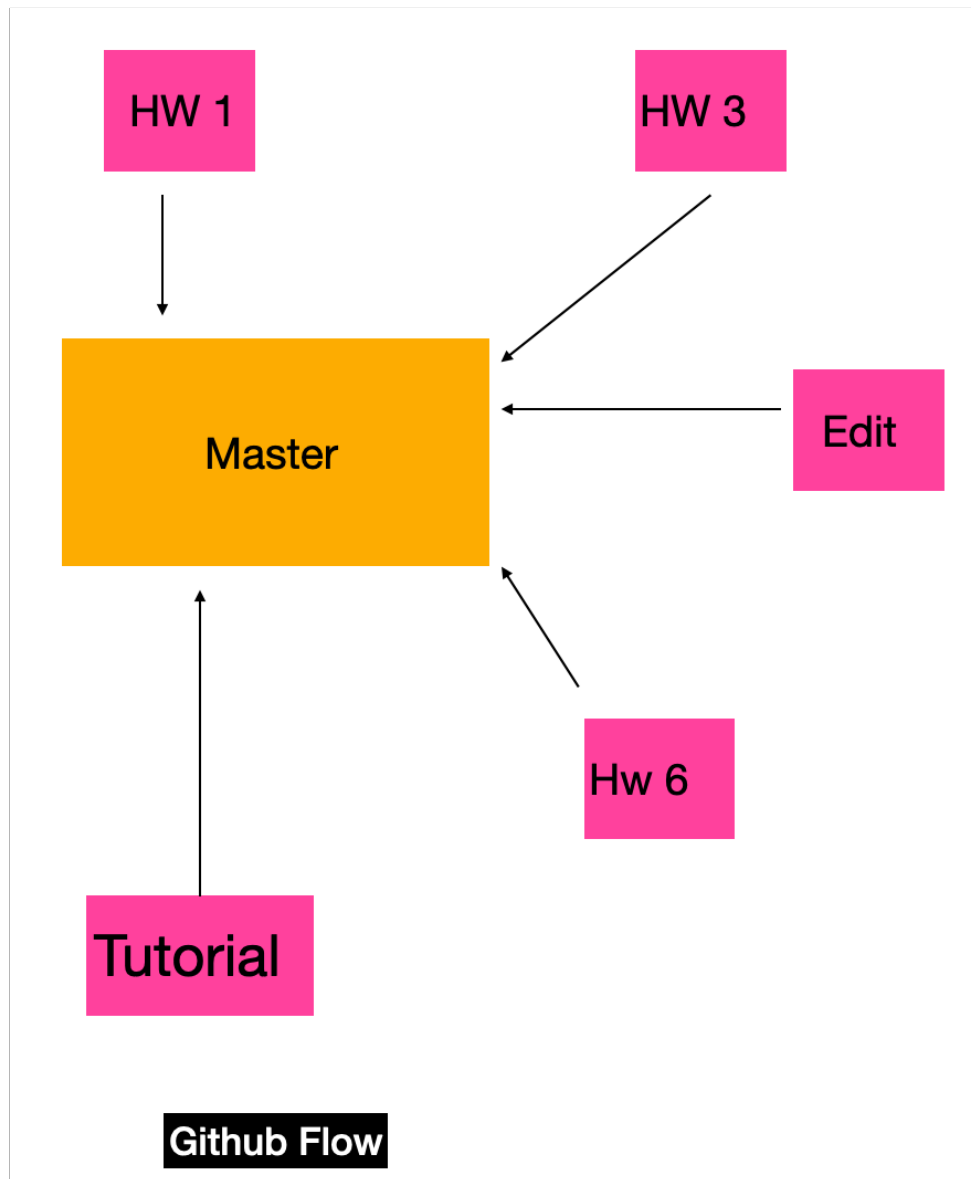
`git add` stages your changes, `git commit` allows you to comment the changes, `git push` sends your changes to Github, and `git pull` pulls all changes to Github to your computer.

Pro-tip: A git commit should equate to one task. Each git commit should be summarized in a sentence or less. This means that you will be constantly git committing and pushing to Github.

2.2.4 Version Control

This section is dedicated to working in groups. Github can be an amazing tool and an even bigger headache. The key is to maximize the benefits and invest in aspirin.

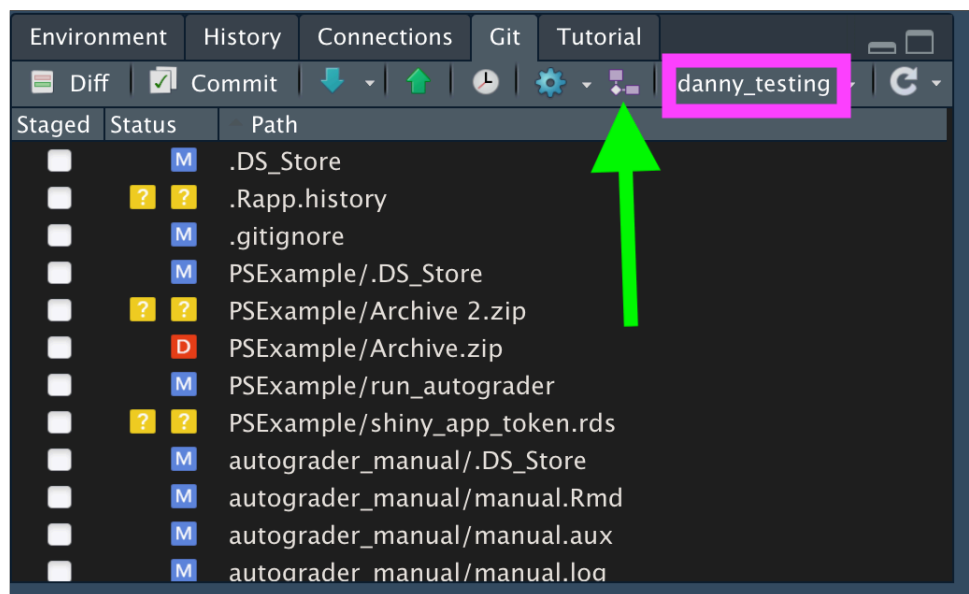
One way to mitigate the headaches is through a formalized version control approach. This manual recommends creating individual branches for each task. There are many resources discussing branching. One good place to start is [here](#). A graphical example of this is depicted below:



Notice at the center of the web is the master branch. Each pink square represents a different branch. Every time a new job needs to be done (e.g. a new homework assignment or tutorial), a new branch should be used. By having each project on it's own branch, there will be less of a chance of conflicts when uploading and pushing. In addition, this manual advocates **no one should be working directly on Master**. Actually, that should be a commandment:

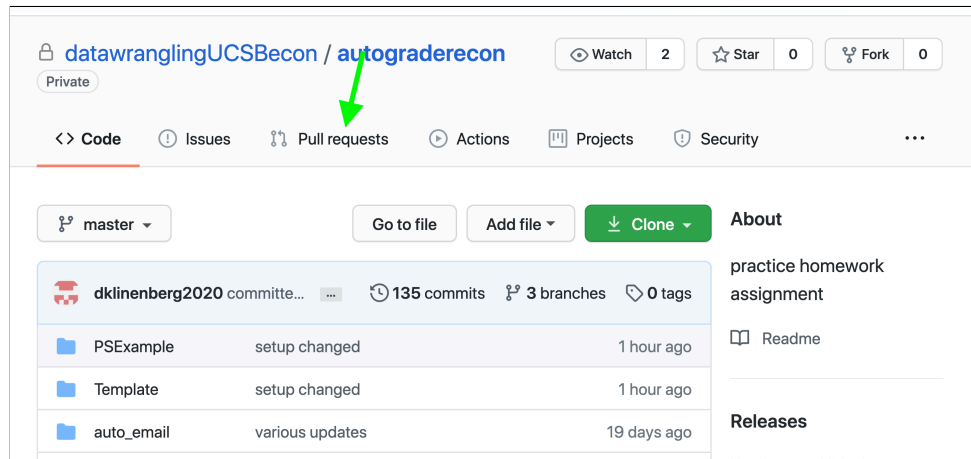
Github Commandment 3: Do Not Work Directly on Master Branch

Making branches in R is extremely useful. First, make sure you are in the Rproject. Next, click the “create new branch” button. It is pointed out by the green arrow in the following picture:



Name the branch. Notice the pink box that says *danny_testing* in the picture above? That should now be whatever you named your branch. You can click the name in the pink box and switch between branches with ease. However, be aware **YOU CAN SWITCH BETWEEN BRANCHES WITH EASE**. This means you need to be extra careful of which branch is selected.

When your work is finished on a branch, you can post a *pull request*. This is you proposing changes to Master in which the other TAs will review and approve. Pull requests can be made directly on Github:



After clicking where the bright green arrow is pointing, click the bright green button that says **New pull request**.

In conclusion, the workflow will be as so:

- 1) You are given an assignment.
- 2) Create a new branch.
- 3) Do you work as described above. Do the normal push and pull and committing.
- 4) Once you are done with your assignment, submit a **new pull request**.
- 5) Have a friend confirm the pull request. If there are no conflicts, confirm your own pull request.

To end this section, I will state a few Git commandments I've made up or found:

Github Commandment 1: Push and Pull often

If you think you are pushing or pulling too much, you're not. People will be making changes constantly and this will lead to issues if everyone isn't on the same page.

Github Commandment 2: Never, Ever, Ever, EVER git push -force

This overrides any merge conflict. That means you are saying that your stuff should override everything. Don't do that. If there's a conflict, talk with your team and figure out what's the best course of action.

2.3 Gradescope

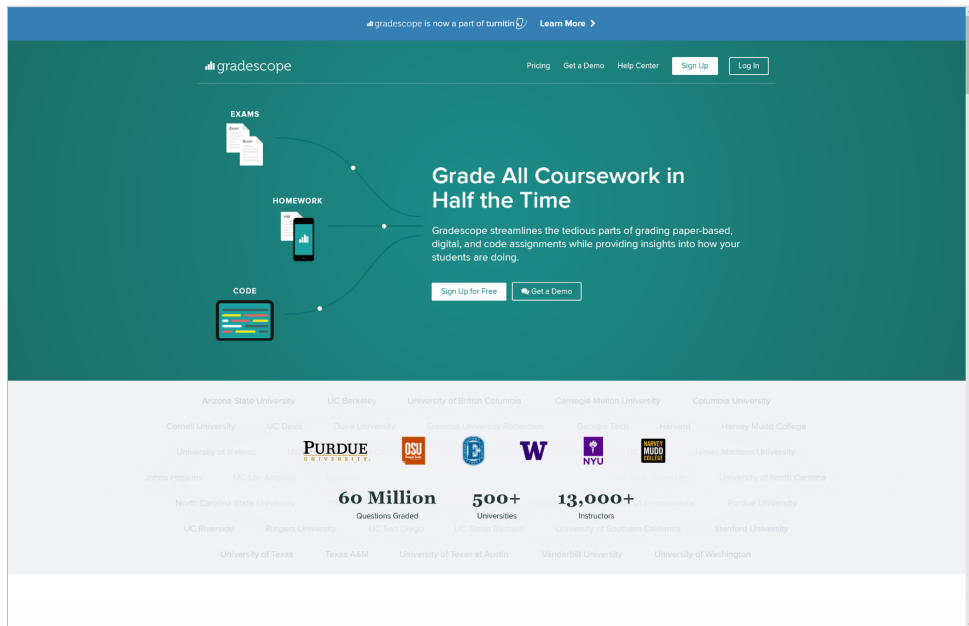
2.3.1 Intro

Gradescope is a 3rd party software designed to make grading faster and easier. Gradescope allows for all types of assignments to be graded. This class will focus on using Gradescope to grade student written assignments and code. We will be utilizing Gradescope's autograder functions. Before delving into the autograder, this section will walk the reader through the basic navigation of Gradescope. For the purposes of the manual, we will be working in the course "Practice". This section will explain:

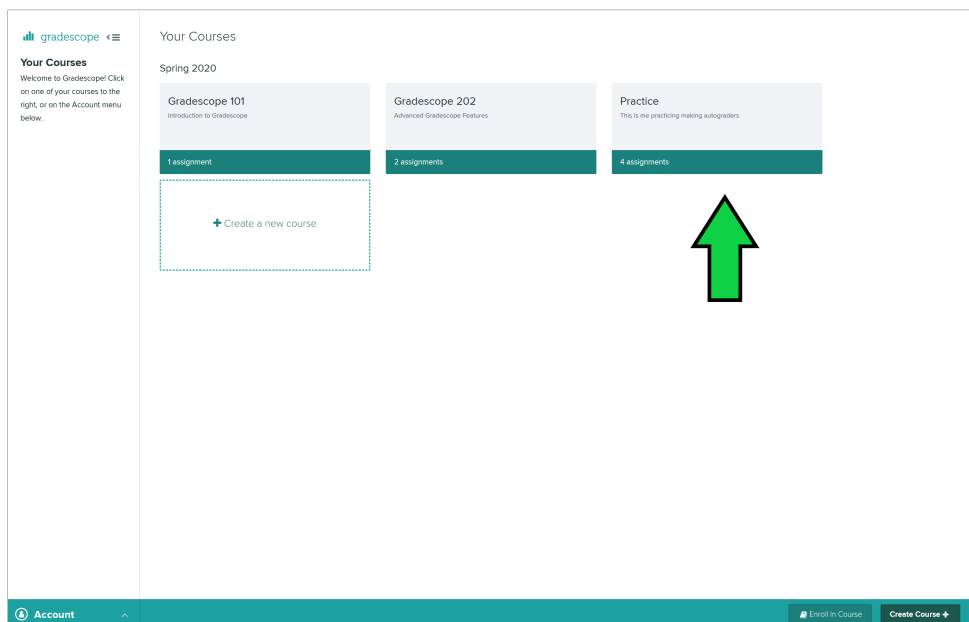
- (1) How to find and log into Gradescope.
- (2) How to navigate to assignments.
- (3) How to setup new assignments.
- (4) Where to enter in the autograder.

2.3.2 Navigating Gradescope

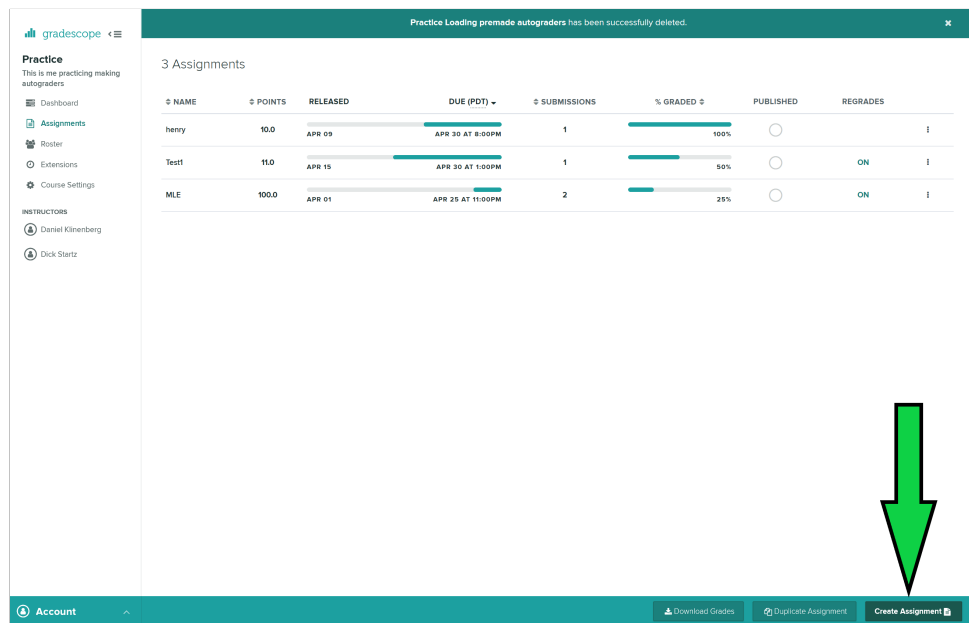
To begin, simply type in www.gradescope.com. If the user is not already logged in, they will be greeted with the following image:



Click the “log In” button in the top right hand corner. You will be redirected to a login screen. Click **School Credentials** and select UCSB. From there, you will log in using your **UCSB** login information. The page will then direct you here. Please click the relevant course (in our example, it is **Practice**):

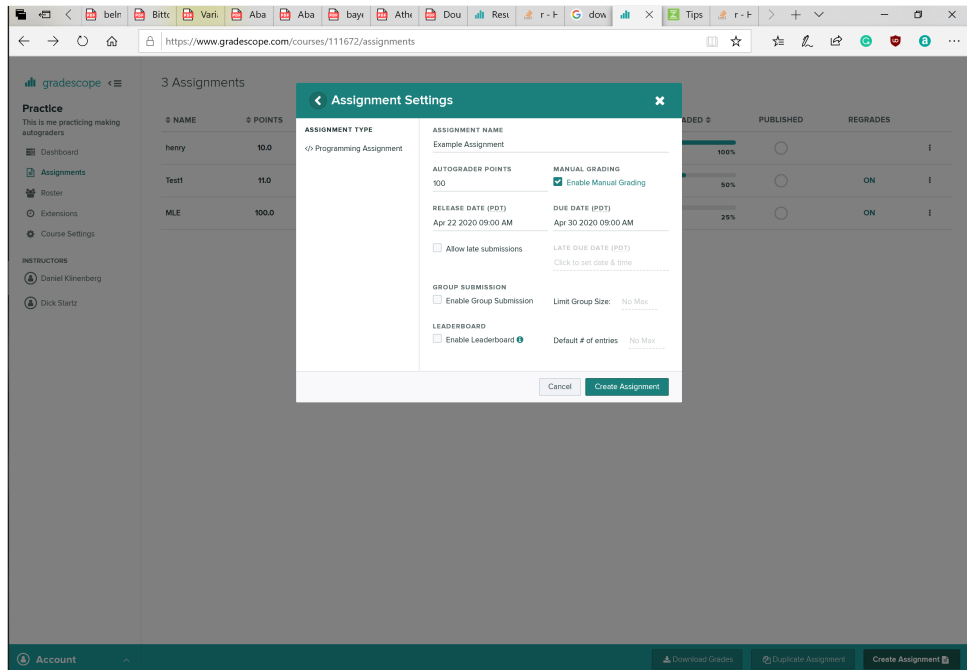


2.3.2.1 Making a New Assignment Gradescope assignments are created using the button in the bottom right hand corner:

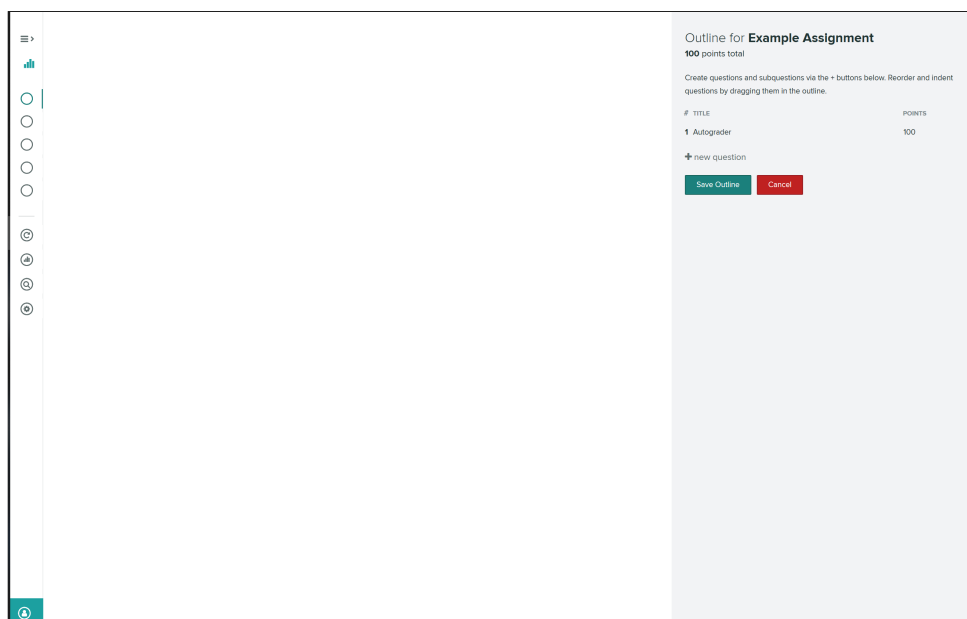


A popup box will ask what type of assignment is being created. Select **</> Programming Assignment**, then click next.

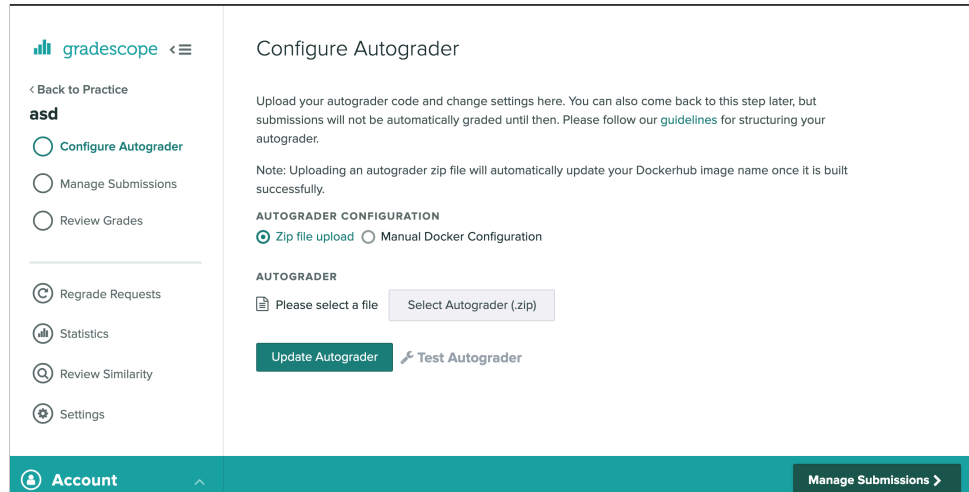
Next, a popup creating the assignment setting will appear. Please fill in the setting appropriately. This will be provided to you by the head TA in the form of a rubric or via conversation. If you do not know the setting, **ASK A FELLOW TA OR INSTRUCTOR**. The example image has been filled in for illustrative purposes:



Gradescope will then automatically open to the outline section. Notice that the autograder points are already in the top right hand corner. Add a new question titled “style” and set the point value to 0. This allows the grader to allocate points for “clean code”. By setting it equal to 0, we are saying we don’t care about the cleanliness of the code. In the future, this may become part of the rubric, but for now leave it as 0. Then click “save outline”.



Finally, we are at the upload autograder phase! This is where the reader will submit the autograder. What an autograder is and the components of an autograder are discussed in details in the following chapters. For the time being, just know the autograder comprises a zip file called “gradescope.zip” and an R file on Github. Click the “select gradescope.zip” file and upload your created autograder.



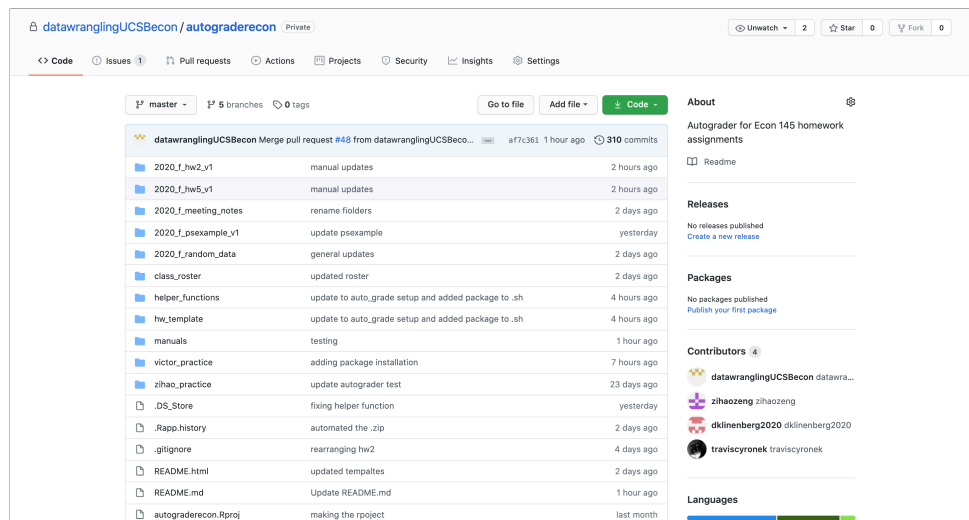
This concludes the section on Gradescope. At the end of this section, the reader should know:

- (1) How to find and log into gradescope.
- (2) How to navigate to assignments.
- (3) How to setup new assignments.
- (4) Where to enter in the gradescope.zip. The reader does not yet know what constitutes the gradescope.zip.

The following section will delve into the parts of the autograder and how to actually make one.

3 Layout

Now that R, RStudio, and Github are setup, we will go through the layout of the repository. If you have not yet set up [R](#) or [Rstudio](#) or [Github](#), please click the links and do so. In order to understand how to make a homework for Gradescope, we first must understand the Github repository and underlying functions. The Github repository will look similar to this:



Folders that follow the naming convention `year_quarter_item_versionnumber` are quarter specific folders. For example, `2020_f_hw1_v2` is homework 2 from fall quarter 2020. Each folder houses a different item. By item, I mean the files necessary for a homework assignment, lecture, or section. All these files are stored on the main Github page. There are no quarter specific folders. This is meant to keep the organization easier to follow. This is also useful for the file dependencies.

The only folders that should not follow this naming convention are:

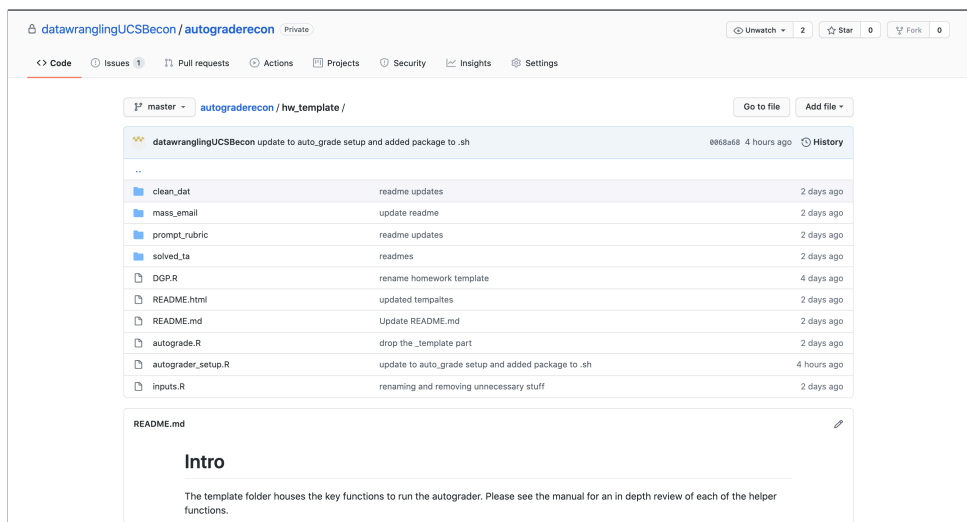
- 1) `hw_template`
- 2) `manuals`
- 3) `helper_functions`

4) `class_roster`

I will go through each of the 4 folders listed. A description of each file can be found in the [appendix](#):

3.1 `hw_template`

This is the main folder of the repository. The files here are directly used for Gradescope. Whenever a new homework needs to be made, the TA will duplicated this folder and rename it following the naming convention. For example, if it was time to make the autograder for version 1 of homework 7 in Spring 2025, the TA would duplicate this folder and rename it `2025_s_hw7_v1`. `hw_template` holds the following files:



This folder holds all the direct files for the autograder. When a new homework assignment is announced, the first step is to copy `hw_template` and rename the folder the name of the homework assignment. Then the TAs will work almost exclusively within this folder to create working autograders. More so, they will be able to work only in R files (except one step). Notice that the folder also has a sub-folder to house the prompt and rubric (`prompt_rubric`). Within that folder, there are templates on how to format the homework assignment to be compatible with Gradescope.

It is worth noting some nuances associated with writing homework for an autograder. First, you must be very specific on what everything is named. The autograder is sourcing the student's code. That means the student must all have the same:

- 1) name of their R file (e.g. 'hw1.T'),
- 2) name of their dataset (`hw1_dat.csv`),
- 3) load in the dataset with the same name using the same command (`hw1_dat <- read_csv("hw1_dat.csv")`),
- 4) name all their answers the same name,
- 5) write their PERMID. In future versions, writing the PERMID will become obsolete.

However, current development is still experimenting with using Gradescopes metadata.

Please refer to the `README` in `prompt_rubric` as well as the templates.

3.2 manuals

`manuals` holds the manuals. The main manual of interest is `manual.pdf`. This goes through all the steps of the autograder. However, there are additional manuals related to other aspects of the course such as Nectir and using a virtual machine at UCSB. See the `README.md` in the folder for more information.

Gradescope Autograder Tutorial with R

Danny Klinenberg*

Last Updated: 2020-07-29

Contents

1	Introduction	3
1.1	Welcome	3
2	Setup	4
2.1	R and RStudio	4
2.2	Github	5
2.3	Gradescope	13
3	How to Use This	18
3.1	hw_template	19
3.2	manuals	20
3.3	helper_functions	20
3.4	class roster	20

*University of California, Santa Barbara

Gradescope Manual

1 Part 1: Login and Access to Assignments

The homework assignment of this class would be posted on Gradescope. You can access this website via gradescope.com. When asked to login, use school credentials and select USCB NetID.

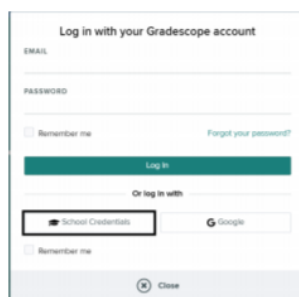
A screenshot of the Gradescope login interface. It features a title "Log in with your Gradescope account". Below this are input fields for "EMAIL" and "PASSWORD". There are checkboxes for "Remember me" and a link for "Forgot your password?". A green "Log In" button is present. Below the button, it says "Or log in with" followed by two options: "School Credentials" (which is highlighted with a red box) and "Google". There is another "Remember me" checkbox and a "Close" button at the bottom.

Figure 1: Choose School Credential Portal

A screenshot showing a list of school credential portals. The list includes: "University of Alabama at Birmingham (BannerID)", "University of Cincinnati Username", "UC Davis Username", "UCLA Legin ID", "UCLouvain Username", "University of California Santa Barbara NetID" (highlighted with a red box), "UC Santa Cruz CridID", "University of Maryland Directory ID", and "University of Missouri-Kansas City Username".

Figure 2: Login Using UCSB Credentials

nectir

[Nectir \(https://www.nectir.io/\)](https://www.nectir.io/) provides a platform for students and instructors to conveniently communicate within the class. You can discuss problems synchronously with peer students, TAs, and professors in the Nectir channels for our class.

If you don't already have a Nectir account, follow this [guide](#), or click the "Setting up Nectir" link on the Gauchospace page for ECON 145 under week 09/28-10/04, to register a new account.

Once you have an account, you can join our class channels by clicking the "Join Econ 145 Discussion Nectir" and "Join Econ 145 Announcement Nectir" links on Gauchospace under week 09/28-10/04.

Now you can access our class channels by logging on to [Nectir](#) (go to Nectir, select "UCSB Nectir" in the drop-down menu of "Launch Nectir" at the top right corner of the webpage), or simply clicking the "Go To Nectir Discussion Board" and "Go To Nectir Announcements" buttons at the top of our Gauchospace page.

For further information about how to download the Nectir app, how to reply to messages, etc., please go to the [Nectir Student Guide](#).

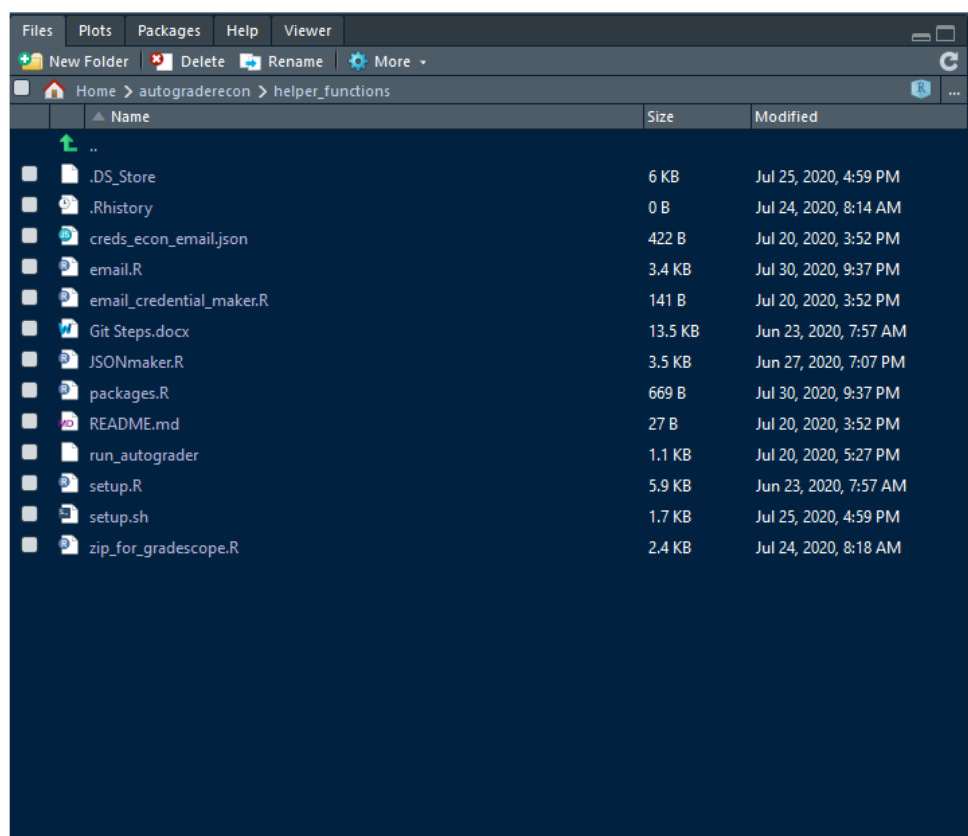
3.3 helper_functions

`helper_functions` streamline a lot of the Gradescope technicalities. These functions are meant to make writing autograders accessible to any individual who has proficient knowledge of R (and patience).

Gradescope requires output to be in a very specific JSON format. With this repository, there is no need to know what exactly that means. The R helper functions create the output in

the proper format. Under the hood, the helper functions are creating a dataframe that the TA is meant to fill in. Once they fill in the dataframe, the helper functions convert it to the appropriate .JSON format and export it in the appropriate location.

There is also `email.R`. This is used to mass email students individual datasets. In order to cut down on cheating, this course creates unique datasets for each student to analyze. In theory, a unique dataset shouldn't deter cheating. If students are writing good code, the specific data shouldn't matter. However, specific datasets do deter cheating in early coding courses where the temptation to hard code answers in is highest. The TA determines the creation of the individualized datasets, which will be discussed further in the homework example.



3.4 class roster

The automatated emailing functions rely on a class roster. The easiest way to get the class roster is to download it from the class Moodle and save it in the file. From here, the specific roster can be specified for mass emailing.

4 Every Step you Take: Making an Assignment from Start to Finish

Below is the standard workflow to start making an online assignment:

Note that this needs only be done the first time you use the Github. If you have the files on your local drive, please skip this step. Unzip the files in your desired filepath.

1: First, go to the class github repository. Then click on the green buttom on the top right corner and copy the HTTPS/SSH URL from the dropdown menu. Both actions are provide you with identical access to the Git repository. Now, Navigate to your RStudio and from the dropmenu of File, choose **New Project**. Then, choose to create project from **Version Control Repository** and then hit **Git**. Paste the URL as instructed. For more detailed instructions, please refer to *GitHub* section of this manul.

master 4 branches 0 tags

Go to file Add file Code

zihaozeng Merge pull request #78 from datawranglingUCSBecon/zihao_1

2020_f_hw2_v1	update manual	
2020_f_hw5_v1	Merge branch 'delet'	
2020_f_meeting_notes	rename fiolders	
2020_f_psexample_v1	update manual	
2020_f_random_data	general updates	6 days ago
class_roster	updated roster	5 days ago
helper_functions	update manual	16 hours ago
hw_template	Merge pull request #74 from datawranglingUCSBecon/manual_update	16 hours ago
manuals	update manual	16 hours ago

Clone with HTTPS Use Git or checkout with SVN using the web URL.
<https://github.com/datawranglingUCSBecon>
 Open with GitHub Desktop
 Download ZIP

RStudio

File Edit Code View Plots Session Build Deb

New File

New Project...

Open File... Ctrl+O

Reopen with Encoding...

Recent Files

Open Project...

Open Project in New Session...

Recent Projects

Import Dataset

Save Ctrl+S

Save As...

Save with Encoding...

Save All Ctrl+Alt+S

Knit Document Ctrl+Shift+K

Compile Report...

Print...

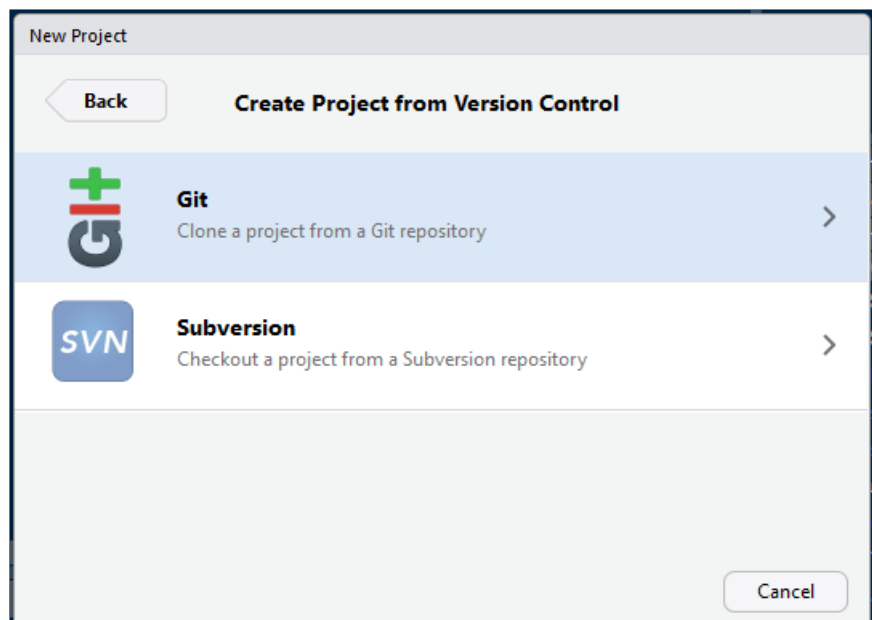
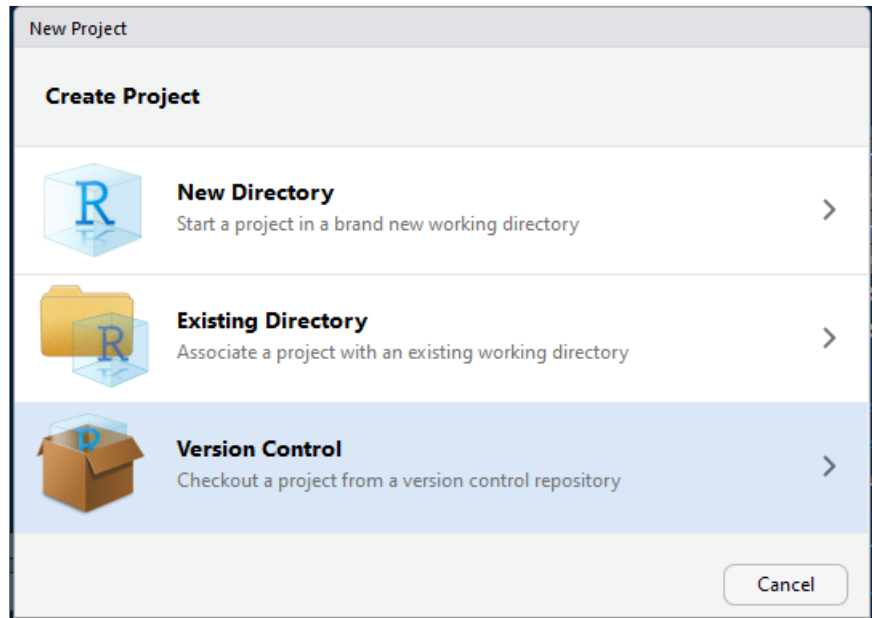
Close Ctrl+W

Close All Ctrl+Shift+W

Close All Except Current Ctrl+Alt+Shift+W

Close Project

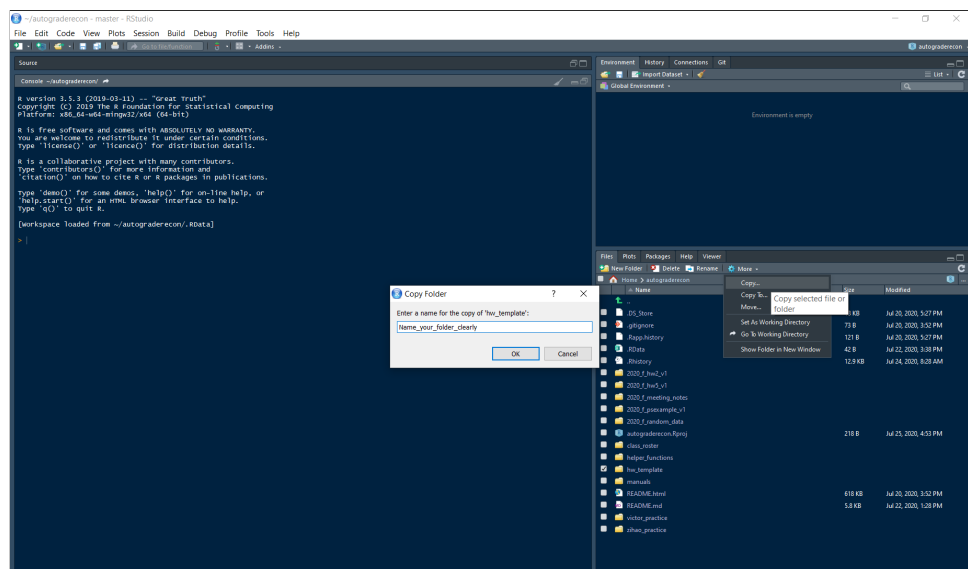
Quit Session... Ctrl+Q



2: Open RStudio and set working directory to the downloaded file. This is done by clicking the the `.rProj` item in the repository. In the example case, it is called `autograderecon.Rproj`. The rest of the operations could be mainly done in RStudio.

3: Using RStudio file browser, select `hw__template` folder by checking the box before it, then click **More** next to **Rename** in the same panel and select **copy** in the dropdown menu. Name your duplicate file following the naming convention. Please ensure that your custom

homework folder should be placed in autograderecon folder.



4: Import your messy dataset into the `clean_dat` folder. Do the necessary cleaning to the datafile (i.e. messing or cleaning it up for the students). When done, save the completed dataset in the homework folder. **Do not save the completed dataset in `clean_dat`.**

5: Go to **prompt_rubric** folder where you can edit R markdown templates for both rubric and prompts for your assignment.

6: Solve the homework assignment and save your R scripts in the `solved_ta` folder. Make sure to name it properly to avoid confusion.

7: Find **inputs.R** in the homework folder, open and edit the file according to your assignment. Please make sure the inputs are correct or else it can break the autograder.

```

1 # user inputs: CHANGE THESE TO MATCH THE ASSIGNMENT
2 rscript <- "test.R" # the name the student is supposed to give the file
3 data <- "fakedata.csv" #name of the csv file
4 datatitle <- "fakedata" #what the student is supposed to call the datasource
5 parts <- 2 # Number of problems being graded
6 Master <- "Masterdata.csv" #The name of the master data file. This should be in folder
7 DGP <- "DGP.R" # The data generating process for individual student's datasets.
8 #The filename "DGP" should have one function titled "dgp".
9

```

8: Find **autograde.R** in the folder. Input your answers and autograding scripts where necessary. For more detailed instructions, please refer to the section, **Autograder**.

9: Make the necessary adjustments to DGP.R. The file will look similar to this:

```

# This is how each student is assigned their unique
# dataset The DGP may change depending on the
# assignment. Please Please Please Please do not
# change the name of the DGP nor the inputs of the
# DGP Simply change how the dataset 'd' is created
# (line 9)

```

```

DGP <- function(masterdata = "Masterdata.csv", PERMID = PERMID,
  dataname = dataname) {
  masterdata <- read.csv(masterdata)

  set.seed(PERMID)

  #-----#

  # You can change this line however you please, Do
  # not change any other lines in the folder PLEASE
  # PLEASE PLEASE :)

```

```

d <- masterdata[sample(1:nrow(masterdata), 1000,
  replace = F), ]
#-----#

write_csv(d, dataname)
}

```

The only line that will EVER be changed is `d <- masterdata[sample(1:nrow(masterdata), 1000, replace = F),]`. If you don't want to create individual datasets, set `d <- d`. Notice that the individual datasets come from setting the seed to student's PERMID.

10: Common mistakes that breaks your autograder:

1 variable names in **inputs.R** or **autograde.R** are incorrect.

2 set *loc* to “gradescope” in autograde while working on your local files, or vice versa.

3 homework folder not placed in autograderecon.

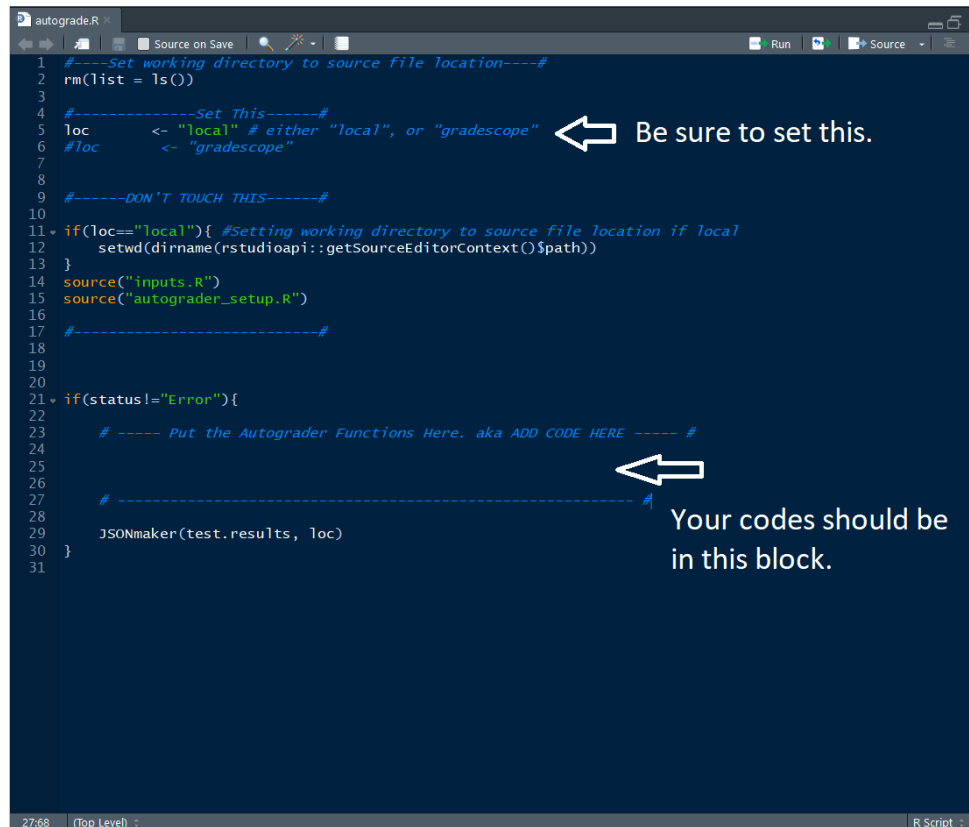
10: After finished with **autograde.R**, set *loc* to ‘local’ and run the script. the function should generate **gradescope.zip** in your folder. Uploading **gradescop.zip** to gradescope, for detailed instructions, please refer to *gradescope* section.

11: Set *loc* to “gradescope” in **autograde.R**. Save all edited files, commit and push your changes to github repository using Git Push. Detailed instructions in *GitHub* section.

12: Navigate to helper_functions folder. If not already done, get email credentials by using **email_credential_maker.R**. Make sure that your **inputs.R** file is correctly named and saved, then run the script **email.R**. For more detailed instruction, please refer to **mass email** section.

5 Writing the Autograder

1: The bulk of your autograder codes should be in **autograder.R** in the respective homework folder. Make sure that the autograder codes are placed in the right location, and *loc* is set to correct variable for either **local** or **gradescope** grading.



```
1 #---Set working directory to source file location---#
2 rm(list = ls())
3
4 #-----Set This-----#
5 loc     <- "local" # either "local", or "gradescope"
6 #loc     <- "gradescope"
7
8 #-----DON'T TOUCH THIS-----#
9
10
11 if(loc=="local"){ #Setting working directory to source file location if local
12   setwd(dirname(rstudioapi::getSourceEditorContext()$path))
13 }
14 source("inputs.R")
15 source("autograder_setup.R")
16
17 #-----#
18
19
20
21 if(status!="Error"){
22   # ---- Put the Autograder Functions Here. aka ADD CODE HERE ---- #
23
24   # -----#
25
26   # -----#
27
28   JSONmaker(test.results, loc)
29 }
30
31
```

Be sure to set this.

Your codes should be in this block.

2: Start your autograder by providing correct answers to each question. Make sure the answer variable's names are different from the ones specified in the prompts, and it is advised to comment your answers to later reference.

3: Compare your answers with the ones provided by students, and grade each questions accordingly. It is recommended to test if said variable exists before comparing. You can do so with **is.error()** function from **berryFunctions** library.

```
if (is.error(student.name)){ "Variable name does not exist" } else{
  #Test if student answer is correct
}
```

4: How to grade XXX?

1 How do you grade a tibble? Three functions can be used to compare objects like dataframe and tibble. `all.equal()` is the default function of R. It can perform crude comparison between tibble objects, but minor changes like ordering or different grouping could affect its output. `compare()` from library *compare*. It has several built-in arguments that allowed you to specify if orders, data_types, or capitalizations matter in the comparison. But this function works poorly with tibbles especially when grouping, or factor levels are different. `all_equal()` from library *tidyverse*. It is similar to `compare()` in that it allowed you specify if order, etc. matter. It performs better with tibbles that have factors. But it would not recognise 1*n dataframes as valid input. Thus, it is recommended to use `all_equal (as.tibble(df1),as.tibble(df2))`. Troubleshooting: it often happens when minor class attributes could affect the autograder, like student answer has factor as a column while your answer converted it to characters. Or differences like double versus integer. Some easy fixes for these issues are `type_convert()` from library *readr*.

2 How do you grade a function? Sadly no function in R provided tools that could compare if two functions are fundamentally different. A good approximation to this could be to provide a variety of inputs to feed into both functions and compare answers. It is advised to prepare inputs that covers multiple data types, with either valid or invalid output for more accuracy. A great example can be found on Michael Guerzhoy's [Github](#).

3 How do you grade a numeric value with potential round-offs? Though this arises less than often, it could be the case when a numeric answer requires multiple steps of calculation, and some round-offs are involved. Many methods exist to fix this

issue. One of them is to use `almost.equal()` from library *berryFunctions*. This function allowed you to do fuzzy comparison with a set scale.

4 How do you grade console outputs? Sometimes students are asked to print stuff. You can use the `sink()` function from base R to access their printed outputs. A simple way to do so is provided below. Notice that `student.output` is a character vector. Each element corresponds to a line of output from running the student's script. You can then use functions like `str_detect()` from the *stringr* package to check if a desired output does exist.

```
1 sink("temp.txt") # The outputs of the following lines of code enclosed by sink() are sun
2                 # newly created "temp.txt" in the working directory. You can name this
3                 # whatever you want.
4 source(rScript) # Run the student's script while recording the outputs.
5 sink() # Tell R we are done sinking and close "temp.txt".
6 student.output <- read_lines("temp.txt") # Read in the outputs of the student's script t
7                                           # be graded.
8 file.remove("temp.txt") # Remove the temporary output file in the directory.
```

5: Save the grades in json file. This step is mostly automatic. In `autograder_setup.R`, functions should already initiate a dataframe `test.results`, with each row corresponding to one question and four columns: *description of the question*, *the score*, *max score*, and *notes*. To save the grades, you should make to fill each cell with corresponding inputs.

```
if(status!="Error"){
  # ----- Put the Autograder Functions Here. aka ADD CODE HERE ----- #
  # Example: Suppose the student earned 5 out 10 in problem 3.
  test.results[3,] <- c('Problem 3: Example(10pt)',5,10,"This should be your feedback to the student[]")
}
```

6: Pushing to Github by Git push.

7: My autograder broke, why? If this happens to you, don't fret. This happens **A LOT**.

Try to see if you have done the following:

1 Pushed **autograde.R** before saving *loc* to *gradescope*.

2 Created the autograder in a branch other than **master** and did not merge the branches after pushing.

3 Used functions from packages that are installed but not declared, this can fixed by 'library(NameOfthePackage)'.

4 Used functions from packages that are not installed in the Virtual Machine, to fix this you should edit **setup.sh** in your **gradescope.zip**.

5 some parts of the autograder returned NA instead of correct grading, this is usually hinted by:

```
Error in JSONmaker(test.results, loc) :  
  Error: There is a missing value somewhere (NA). Please make sure all  
          values are filled in. If a student recieved 0 points please make sure  
          there is a 0 in the matrix
```

5.1 Prompt and Example Autograder

Below is an example homework prompt with accompanying autograder:

Practice Assignment

2020-06-09

Prompt

You are working for a local retail firm. They have many, many clients and are interested in some basic information about them. The firm has supplied you with the excel sheet *Masterdata.csv*. They have also asked you to do some basic analysis on the dataset. You are asked to do the following:

Basic

- 1) How many missing values are there in each column? How many missing values are there total? Put your answers in a tibble and name it **missing**.
- 2) How many unique states does the company operate in? Are there any typos in this column? If so, correct the typos. Save the corrected dataframe as **data.cleaned**. Unless otherwise told, work with **data.cleaned**.
- 3) Find the average invoice and the percent of invoices paid. Save them as **avg_inv** and **avg_inv_paid**.

1

```
#----Set working directory to source file location----#
```

```
# clear workspace
```

```
rm(list = ls())
```

```

# SET THIS!!!#

loc <- "local" # either local, or gradescope

#-----#

#-----DON'T TOUCH THIS-----#

if (loc == "local") {
  # Setting working directory to source file location
  # if local
  setwd(dirname(rstudioapi::getSourceEditorContext())$path))
}

source("inputs.R")
source("autograder_setup.R")

#-----#

if (status != "Error") {
  # ----- Put the Autograder Functions Here ----- #

  # Create the answers. Notice they aren't named the
  # same as the homework
  answer.1 <- tibble(Class = c(colnames(fakedata),
    "Total"), Num = c(colSums(is.na(fakedata)),
    sum(is.na(fakedata))))
  answer.2 <- fakedata

```

```

answer.2$State[answer.2$State == "Arisona"] <- "Arizona"
answer.3 <- mean(answer.2$Invoice..USD., na.rm = TRUE)
answer.4 <- sum(answer.2[which(answer.2$Paid ==
  "No"), 3], na.rm = T)/sum(answer.2[, 3], na.rm = T)

# Grading <--> comparing answer key answers to
# student answers Problem 1 score:
p1.score <- if (is.error(missing) == TRUE) {
  # checking to make sure they saved the name right
  test.results[1, ] <- c("Problem 1: Missing value (10 pt.)",
    0, 10, "mssing not found. Please ensure the variable is properly named")
} else if (isTRUE(all_equal(missing, answer.1))) {
  test.results[1, ] <- c("Problem 1: Missing value (10 pt.)",
    10, 10, "nice work")
} else {
  test.results[1, ] <- c("Problem 1: Missing value (10 pt.)",
    0, 10, "Try Again")
}

p2.score <- if (is.error(data.cleaned == TRUE)) {
  # checking to make sure they saved the name right
  test.results[2, ] <- c("Problem 2: Data cleaning (12 pt.)",
    0, 12, "data.cleaned not found. Please ensure the variable is properly named")
} else if (isTRUE(all_equal(data.cleaned, answer.2))) {
  test.results[2, ] <- c("Problem 2: Data cleaning (12 pt.)",
    12, 12, "nice work")
} else {

```

```

test.results[2, ] <- c("Problem 2: Data cleaning (12 pt.)",
  0, 12, "Try Again")
}

p3.score <- if (is.error(avg_invoice == TRUE) ||
  is.error(avg_invoice_paid == TRUE)) {
  # checking to make sure they saved the name right
  if (is.error(avg_invoice == TRUE)) {
    if (is.error(avg_invoice_paid == TRUE)) {
      test.results[3, ] <- c("Problem 3: Average (8 pt.)",
        0, 8, "both avg_invoice and avg_invoice_paid not found. Please ensure
      } else {
        if (avg_invoice_paid == answer.4) {
          test.results[3, ] <- c("Problem 3: Average (8 pt.)",
            4, 8, "avg_invoice not found. Please ensure the variable is properly
          } else {
            test.results[3, ] <- c("Problem 3: Average (8 pt.)",
              0, 8, "avg_invoice not found. avg_invoice_paid incorrect. Please ens
            }
          }
        } else {
          if (avg_invoice == answer.3) {
            test.results[3, ] <- c("Problem 3: Average (8 pt.)",
              4, 8, "avg_invoice_paid not found. Please ensure the variable is prop
            } else {
              test.results[3, ] <- c("Problem 3: Average (8 pt.)",
                0, 8, "avg_invoice incorrect and avg_invoice_paid not found. Please en
              }
            }
          }
        }
      }
    }
  }

```

```

    }
  } else {
    if (avg_invoice == answer.3) {
      if (avg_invoice_paid == answer.4) {
        test.results[3, ] <- c("Problem 3: Average (8 pt.)",
                               8, 8, "nice work")
      } else {
        test.results[3, ] <- c("Problem 3: Average (8 pt.)",
                               4, 8, "check avg_invoice_paid")
      }
    } else {
      if (avg_invoice_paid == answer.4) {
        test.results[3, ] <- c("Problem 3: Average (8 pt.)",
                               4, 8, "check avg_invoice")
      } else {
        test.results[3, ] <- c("Problem 3: Average (8 pt.)",
                               0, 8, "Try again")
      }
    }
  }
}

p4.score <- if (is.error(data.cleaned.split ==
  TRUE)) {
  test.results[4, ] <- c("Problem 4: Split Data (2 pt.)",
                        0, 2, "check data.cleaned.split")
} else {
  if (isTRUE(all_equal(data.cleaned.split, answer.5))) {

```

```

        test.results[4, ] <- c("Problem 4: Split Data (2 pt.)",
                               2, 2, "Well done.")
    } else {
        test.results[4, ] <- c("Problem 4: Split Data (2 pt.)",
                               0, 2, "Try again")
    }
}

# ----- #

JSONmaker(test.results, loc)
}

```

Notice that first the autograder solves the homework, then the autograder goes through a plethora of if-then statements comparing the student's submission to the correctly solved answers.

6 Troubleshooting

Many things can break when working with Gradescope. There are some key issues that are bound to happen when working with Gradescope:

6.1 Figuring out Filepaths

run_autograder relies heavily on using file paths. A good way to test if new filepaths work is in the terminal for Macs, and GitBash for PCs. Test the filepaths work locally before attempting to add them to **run_autograder**.

6.2 Choosing the version of R

Ubuntu 18.04 will by default install R.3.4.4. If your class requires a more advanced version of R, you will have to call it in the **setup.sh** file. Figuring out the syntax for this file is very difficult without a background in computer science. The best way to learn is through a virtual machine (VM) on your own computer. A useful way to do this is with [Instant Ubuntu's VMs](#). This program will allow you to create instant VMs similar to the one used by Gradescope. They also have helpful [resources](#) for commands. This environment will allow you to troubleshoot **setup.sh** syntax. The best part about this is that VMs are independent of the rest of the computer: whatever you do in the VM won't affect anything out of the VM!

6.3 When the Autograder Breaks

This is bound to happen. You upload the autograder and find out there's a mistake. Bummer! Because we're working through Github, all you have to do is edit your autograder and push it to Github. This is possible because the Gradescope virtual machine pulls the `autograder_PSEExample.R` from github every time.

Common fixes to the autograder not working on Gradescope:

- Set the `autograder_.r` to "gradescope".
- Push the autograder to the github.

When in doubt, ask a fellow TA or a friend.

A useful resource is Michael Guerzhoy [github](#). He creates autograders for functions in R. Gradescope also has a list of useful links [here](#).

6.4 Troubleshooting Student's Code

The best way to debug student's code is to download it from gradescope and experiment on your personal computer. It is easiest to create a separate branch and copy their code into the homework folder. From there, you can experiment with their code and the autograder locally. In the event there's a bug in the autograder, it is easy to fix and push. Once finished, you can simply kill the branch!

Below are some common issues run into in the past:

- Students upload their dataset using `read.csv` instead of `read_csv`. The former is from base R while the latter is from tidyverse. The slight discrepancies in loading leads to major issues in autograders. The `setup.R` function warns students if they use `read.csv` that their answers may be incorrect. We highly recommend `read_csv` for it's integration in tidyverse.
- Students use a different version of R than the autograder does (R 3.6.3). This is a major headache.
- STATA natives are used to their code stopping when there is an error. If there is an error in one of the lines in R, R keeps going. Expect to see many homework assignments with broken lines of code in the center.
- Students copy and paste in the variable names from the pdf. Sometimes they also copy in the space before (e.g. if the variable is `h1`, they type in `" h1"`). This should be one of the first things to look for, especially in the name of the `.R` file.

6.5 Writing Questions

Because the autograder requires very specific inputs, it is very easy to write questions that are not conducive to grading code. These questions include too many parts or unclear

objectives. Below is a fake assignment with poorly written questions for autograding code. This manual will go through each question, identify issues and provide suggestions to improve the question:

Practice Assignment

2020-06-09

Prompt

You are working for a local retail firm. They have many, many clients and are interested in some basic information about them. The firm has supplied you with the excel sheet *Masterdata.csv*. They have also asked you to do some basic analysis on the dataset. You are asked to do the following:

Basic

- 1) How many missing values are there in each column? How many missing values are there total? Put your answers in a tibble and name it **missing**.
- 2) How many unique states does the company operate in? Are there any typos in this column? If so, correct the typos. Save the corrected dataframe as **data.cleaned**. Unless otherwise told, work with **data.cleaned**.
- 3) Find the average invoice and the percent of invoices paid. Save them as **avg_inv** and **avg_inv_paid**.

Advanced

- 4) Create a new dataframe where the “Names” column is split into 2 columns: *First Name* and *Last Name*. Nothing else needs to be changed. Save it as **data.cleaned.split**. For part 5, use **data.cleaned.split**.
- 5) Write a function named **client_status** that inputs *last name*, *first name* and outputs:
i) state they live in, ii) number of invoices that need to be paid, iii) outstanding debt.
Here, outstanding debt means the sum of the value of the invoices not yet paid. Have your code return an error if someone enters a name not in **data.cleaned.split**. Provide a line of code that runs your function for *Jordyn Kang*.

6.5.1 Problem 1

- Notice that the students aren't told what to name their columns. This can lead to issues with reading the dataframe.
- The students aren't told if the dataframe should go up and down or left to right. The lack of clarity on the dimensionality can lead to issues.
- **Potential Solution:** Specify the number of columns and rows (if dataframes same) as well as the name of the columns.

6.5.2 Problem 2

- There are multiple parts to the question.
- Each part isn't clearly labeled. The student's don't know what to name the outcome of every problem.
- **Potential Solution:** Rewrite the question with subparts and clearly indicate what each part should be titled.

6.5.3 Problem 3

- No issues. This is a very good problem for an autograder.

6.5.4 Problem 4

- Same as problems 1 and 2.

6.5.5 Problem 5

- Is the answer the function or the results for *Jordyn Kang*? It is unclear from the writing. What should they label the output with *Jordyn Kang*?

- **Potential Solution:** Break the problem into two parts: one where they write the function and one where they save the answer for *Jordyn Kang* as `jordyn_kang`.

7 Building the Infrastructure

This section is dedicated for setting up the Github for a class FOR THE FIRST TIME. This will only need to be done once per class. This includes future iterations of the class. Inside this folder you will find the following folders:

- 1) `class_roster`
- 2) `2020_f_example_v1`
- 3) `helper_functions`
- 4) `hw_template`
- 5) `manuals`

Click the link to the public repository: [autograder_template](#). From here, it is recommended to make a private fork of this public repository or to privately clone the repository. One useful resource is [here](#) and [here](#). Name the new private fork/repository as you see fit.

After you create your private repository, enter `helper_functions`. Click `email_credential_maker.R` and follow the link. At the end of the steps, you will have a Google API token that allows for mass emails through R.

Next, enter `zip_for_gradescope_2.R`. Replace `USERNAME` with your Github username, `PASSWORD` with your Github password, and `REPONAME` with the name of your repository. After finished, save. I recommend doing this using `cmd+F` and `replace`.

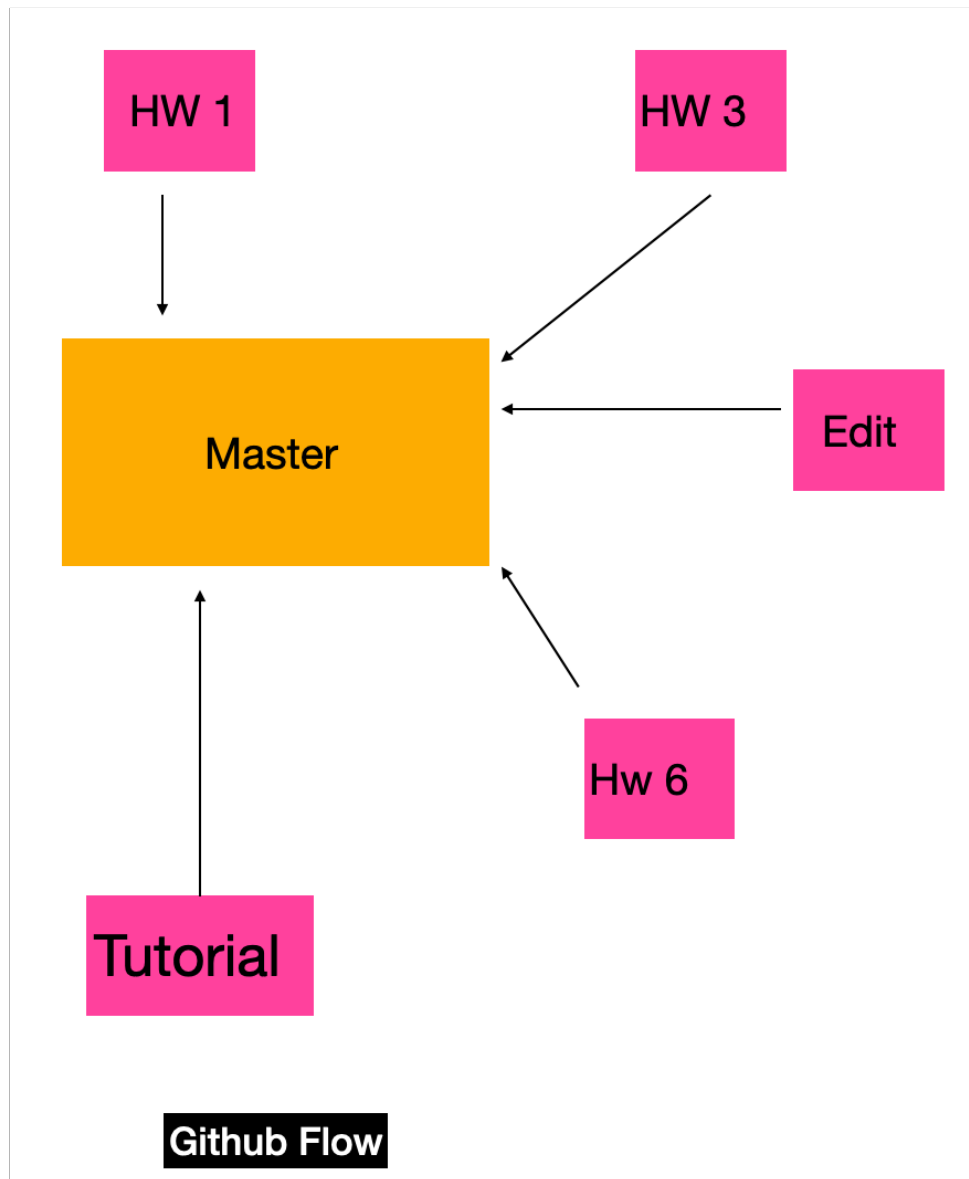
Now, your Github repository is ready to go! The repository can be used for all future quarters/semesters of the class.

A Appendix

A.1 Version Control

This section is dedicated to working in groups. Github can be an amazing tool and an even bigger headache. The key is to maximize the benefits and invest in aspirin.

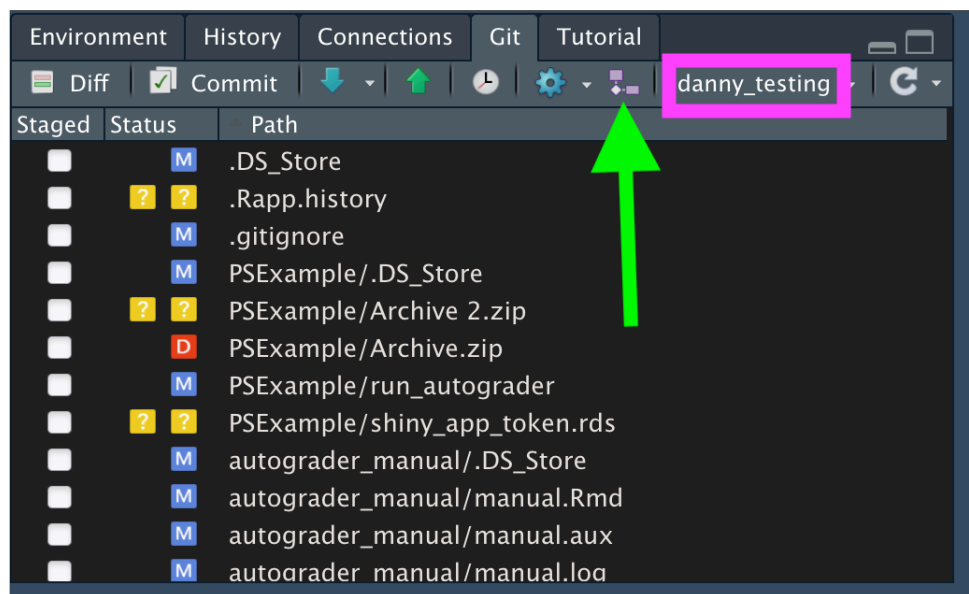
One way to mitigate the headaches is through a formalized version control approach. This manual recommends creating individual branches for each task. There are many resources discussing branching. One good place to start is [here](#). A graphical example of this is depicted below:



Notice at the center of the web is the master branch. Each pink square represents a different branch. Every time a new job needs to be done (e.g. a new homework assignment or tutorial), a new branch should be used. By having each project on it's own branch, there will be less of a chance of conflicts when uploading and pushing. In addition, this manual advocates **no one should be working directly on Master**. Actually, that should be a commandment:

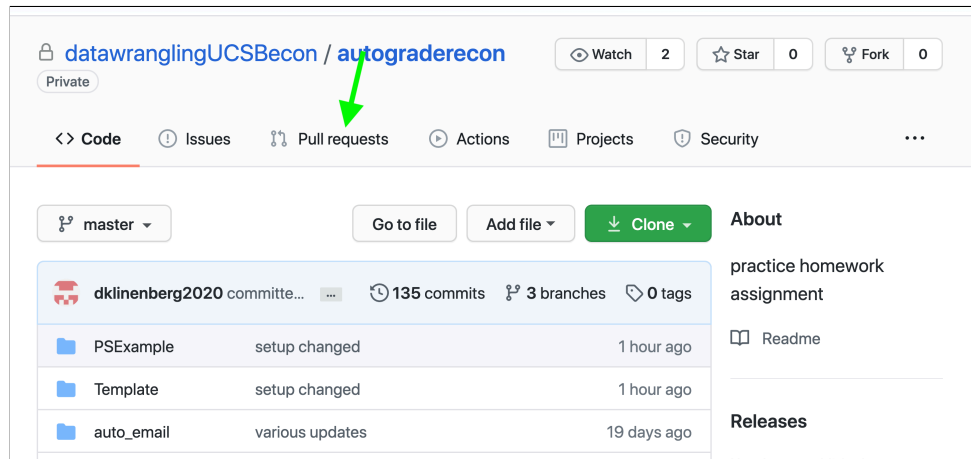
Github Commandment 3: Do Not Work Directly on Master Branch

Making branches in R is extremely useful. First, make sure you are in the Rproject. Next, click the “create new branch” button. It is pointed out by the green arrow in the following picture:



Name the branch. Notice the pink box that says *danny_testing* in the picture above? That should now be whatever you named your branch. You can click the name in the pink box and switch between branches with ease. However, be aware **YOU CAN SWITCH BETWEEN BRANCHES WITH EASE**. This means you need to be extra careful of which branch is selected.

When your work is finished on a branch, you can post a *pull request*. This is you proposing changes to Master in which the other TAs will review and approve. Pull requests can be made directly on Github:



After clicking where the bright green arrow is pointing, click the bright green button that says **New pull request**.

In conclusion, the workflow will be as so:

- 1) You are given an assignment.
- 2) Create a new branch.
- 3) Do you work as described above. Do the normal push and pull and committing.
- 4) Once you are done with your assignment, submit a **new pull request**.
- 5) Have a friend confirm the pull request. If there are no conflicts, confirm your own pull request.

A.2 List of Functions

Below is documentation on each file in `helper_functions` and `hw_template` folders:

A.2.1 `autograde.R`

Location: `/hw_template`

Description: This is the main script in the repository. `autograde.R` is where the TA will enter in all of the code to grade code. The majority of time will be spent editing this file, namely between lines 23 and 27.

Package Dependencies: `packages.R`

File Dependencies: `inputs.R`, `autograde_setup.R`, `JSONmaker`

A.2.2 `autograde_setup.R`

Location: `/hw_template`

Description: This script is created to de-clutter `autograde.R`. It sources all the helper functions and creates the dataframe `test.results` the TA will fill in. It also creates the student's individualized dataset. Finally, it removes any problem lines from the student's code. For example, students are encouraged to clear their environment before starting (e.g. `rm(list=ls())`). Doing this would remove all the saved work from the autograder breaking the whole project. `autograde_setup.R` removes this line from the student's code prior to running. In doing so, `autograder_setup.R` actually creates a separate `.R` file titled `autograde_STUDENTSUBMISSION.R` where `STUDENTSUBMISSION` is the name of their code. Then, `autograde_STUDENTSUBMISSION.R` is sourced, rather than the student's original submission, `STUDENTSUBMISSION.R`. The TA will never need to touch this file.

Package Dependencies: `packages.R`

File Dependencies: `setup.R`, `JSONmaker.R`, `packages.R`, `zip_for_gradescope.R`, `DGP.R`

A.2.3 `DGP.R`

Location: `/hw_template`

Description: This file is used to create unique datasets for each student. The only “real” line of code is line 12. Here is where the TA can specify how to create the individual datasets.

If the prompt does not require individual datasets, change 1000 to `nrow(masterdata)` and keep `replace=F`. This file is called in `autograde.R` as well as `inputs_email.R`.

Package Dependencies: `packages.R`

File Dependencies: `inputs.R`

A.2.4 `email.R`

Location: `/helper_functions`

Description: Runs all the setup to send individualized datasets. The majority of the script is the setup for `gmailR` as well as statements to catch input errors. This is never called directly by a TA. Rather, it is implicitly used when running `inputs_email.R`

Package Dependencies: `pacman`, `ggplot2`, `glue`, `gmailr`, `progress`, `tidyverse`, `quantmod`, `xts`

File Dependencies: `class_roster.csv`, `creds_econ_email.json`, `DGP.R`, `inputs.R`

A.2.5 `email_credential_maker.R`

Location: `/helper_functions`

Description: This houses a link to get `creds_econ_email.json`. Follow the link. It's quite good.

Package Dependencies: NONE

File Dependencies: NONE

A.2.6 `inputs.R`

Location: `/hw_template`

Description: All inputs for the assignment. The TA will be required to fill in these values at the beginning of every new homework. They **MUST** match the rubric. If these values do not match, then the autograder **WILL NOT WORK**.

Package Dependencies: NONE

File Dependencies: NONE

A.2.7 inputs_email.R

Location: /hw_template/mass_email

Description: A simple script used to send mass emails. The TA **WILL** edit this file filling in roster, message, subject, and prompt_name.

Package Dependencies: packages.R

File Dependencies: email.R

A.2.8 JSONmaker.R

Location: /helper_functions

Description: Creates the JSON output needed for Gradescope. The file contains the function `JSONmaker`. This inputs a dataframe that contains the score an assignment receives and location parameter (`local` or `gradescope`). It outputs a Gradescope compatible JSON file.

Package Dependencies: berryFunctions, tidyverse, jsonlite

File Dependencies: None

A.2.9 packages.R

Location: /helper_functions

Description: This runs all the necessary packages. Most of the repo relies on the same packages. Rather than having to go from file to file ensuring all the packages are up to date, someone can simply go into this folder and add the package in.

Package Dependencies: See file

File Dependencies: NONE

A.2.10 run_autograder

Location: /helper_functions

Description: One of two files required by Gradescope to build the docker. This is a template that is pulled by each homework assignment. The lines in the file are file paths. It is telling Gradescope where to go to grab which files to properly run the autograder. A TA will never need to directly work this file. `zip_for_autograder.R` was created to automate this process.

Package Dependencies: NONE

File Dependencies: NONE

A.2.11 setup.R

Location: /helper_functions

Description: Used to create the individual student dataset and fill-in matrix. TAs write autograders by filling in a dataframe, `test_results.R`.

Package Dependencies: readr, stringr, berryFunctions, tidyverse, jsonlite

File Dependencies: NONE

A.2.12 setup.sh

Location: /helper_functions

Description: One of two files required by Gradescope to build the docker. This file creates the R environment. Lines 5-9 are used to download R 3.6.3 while lines 15-31 are downloading all the necessary packages. This is also called in `zip_for_autograder.R`. A TA will never need to touch this file unless:

- 1) The version of R must be updated or
- 2) A new package needs to be downloaded.

In the case of 1), reach out to your local IT department. In the case of 2), copy line 31 and replace `tidyverse` with the name of the package you want to use.

Package Dependencies: NONE

File Dependencies: NONE

A.2.13 `zip_for_gradescope.R`

Location: `/helper_functions`

Description: Gradescope is built off of a zip file that includes two files: `run_autograder` and `setup.sh`. To limit the amount of technical details a TA would need to know, this function was written. `setup.sh` does not change for any of the homework assignments. `run_autograder` does because it references a lot of file paths. This function will update `run_autograder` accordingly, turn it into a unix executable file, and zip it with `setup.sh`. Now, the TA simply has to pull and draw the newly created zip. This file is called in `autograder.setup.R`. The TA will never need to edit this file.

Package Dependencies: `packages.R`

File Dependencies: `inputs.R`