Problem Set 1:Getting Started Key

Claire Duquennois

Group Member 1: Chean Shea

Group Member 2: Mayowa Idowu

Group Member 3: Milan Stefanelli

Group Member 4: Kiersten Kochanowski

In additions, to the instructions given on the problem set, you may find the "GitHub instructions" course document helpful.

Preliminary: Set up your GitHub and Git.

Git and GitHub are often used in conjunction but are really two distinct things. Git is basically a version control software that is based on your local computer. GitHub is a webservice where you have an account that allows you to do version control via the website, which becomes particularly interesting when you are working on collaborative projects and want to avoid the issue of how to synchronize and manage changes to a document when multiple people are working on the same document simultaneously. In theory you could use only one or the other but many people use them in conjunction.

To get a big picture overview of exactly what you can do with Git and Git hub, I would recommend taking the time to watch the Git and GitHub for Poets series of youtube videos https://www.youtube.com/playlist?list=PLRqwX-V7Uu6ZF9C0YMKuns9sLDzK6zoiV.

Getting Started with GitHub

This is quite straightforward:

1. Go to https://github.com/.
2. Click Sign up.
3. Choose a username, an email that GitHub will use to contact you and set you password
4. Respond to the invitation email to activate your account.

Getting Started with Git

Git is often used in conjunction with GitHub as a way to work on GitHub projects locally offline and then reintegrate your work back into a GitHub project. While there is a visual interface you can use to run Git, most programmers seem to prefer to run it directly by typing in Git commands through a terminal prompt.

On MACs: You can type Git commands directly into your Terminal (you can find it in Application/Utilities).

On Windows: You will want to install Git Bash. (See the following link for a video of the installation and some examples of commands https://www.youtube.com/watch?v=J_Clau1bYco.)

1. Go to git-scm.com and download the appropriate .exe file for your operating system.
2. Run the .exe file and install Git. In the installation, the only default you may want to change is to select the option to "Use Git and optional Unix tools from the Windows Command Prompt".

3. Once installed, launch Git Bash and you will see a terminal window in which you can execute Git commands.

You can use the terminal window to navigate through the folders in your computer's directory and then, using the proper commands, you will be able to use this terminal to push and pull materials from your computer onto github and vice versa. This will allow you to work on a github project on your local machine without having to be constantly online.

Individual Homework Tasks:

The following list of tasks are designed to get you started with Git and GitHub and get you set up with the course resources. Everyone in the group should complete these tasks individully on their own computer.

Clone the course repository to your computer

All of the lecture materials for this course are available in a repository on GitHub. Navigate to https://github.com/claireduq/MQE_Causal_Inf to view the course repository. This is where you will find the most up-to-date content for the course.

So that you have full access to this content you will:

1. Fork the course repository to your GitHub account
2. Clone this forked repository to your computer
3. Set up the cloned folder on your computer to syncronize with my course repo

Fork the course repository to your GitHub account

Let's copy the course repo onto your own GitHub page so that you can freely work on it.

To do this you want to fork (ie copy) the entire course repo

1. Navigate into the course repository
2. Click the **Fork** button in the upper right corner

You should now have a copy of the repo on your account.

If you would like to rename the repo:

1. Open the repo
2. Go to the settings tab
3. Type in a new name and click Rename.

Clone this forked repository to your computer

git If you want to have access to the course files when you are offline, you will need to clone (ie copy) the files in this repo to your local machine:

1. Make an empty folder named, for example, `MQE_Causal_Inf` where you will put the files you are going to pull off GitHub (preferably in Dropbox or someplace where it will be backed up).
2. Open your command terminal (or Git Bash on Windows).
3. Navigate via your command terminal to the new folder by typing

   `$ cd /c/Users/ *******/Dropbox/MQE_Causal_Inf`

   followed by enter.

   Note: the directory path will be different for you! To easily get your directory path, you can simply drag and drop the folder into your terminal window.
4. Check that you are in the right folder by typing

   `$ pwd`

   and you should see the directory path to the folder you want to be using.
5. Suppose your GitHub user id is `myuserid` and your repo you would like to clone is named `MQE_Causal_Inf`. In your github repo, click the green code button and copy the link it gives you. Most likely it will look like `https://github.com/myuserid/MQE_Causal_Inf.git` (Notice, the url is just the web address with .git added to the end.)
6. In the terminal type `git clone` and copy the url it gave you

   `$ git clone https://github.com/myuserid/MQE_Causal_Inf.git`

   followed by enter.

You should now find that the folder you created has a copy of all the files that are in the course repo.

Set up syncronizing with the course repository

To make sure the course folder on your computer reflect the most recent changes made in an upstream repo (my course repo that you originally forked from) you need to configure the folder on your computer so that you can easily `fetch` any changes in the upstream repo:

1. Go to the terminal window and navigate to the git folder by typing

   `$ cd /c/Users/ *******/filepath`
2. List the current configured remote repository for your folder: [] `$ git remote -v`

   `> origin  https://github.com/YOUR_USERNAME/YOUR_FORK.git (fetch)`

   `> origin  https://github.com/YOUR_USERNAME/YOUR_FORK.git (push)`

   (This is telling you where your coputer will get any online changes from (fetch) and where it will send any local changes to (push). Right now, it is probably set to fetch changes from your repo and push changes to your repo. We want to change this so that it fetches updates from the my course repo (not yours).)
3. We want to change this. Specify a new remote upstream repo (my course repo) that will be synced with your fork

   `$ git remote add upstream https://github.com/claireduq/MQE_Causal_Inf.git`

4. Verify the new upstream repository you've specified for your fork.

   ```
   $ git remote -v
   > origin  https://github.com/YOUR_USERNAME/YOUR_FORK.git (fetch)
   > origin  https://github.com/YOUR_USERNAME/YOUR_FORK.git (push)
   > upstream  https://github.com/claireduq/MQE_Causal_Inf.git (fetch)
   > upstream  https://github.com/claireduq/MQE_Causal_Inf.git (push)
   ```

5. You can now `fetch` changes to the course repo and have them reflected on your local machine with the command

   ```
   $ git fetch upstream
   ```

6. Check out your fork's local master branch.

   ```
   $ git checkout master
   ```

7. Merge the changes from upstream/master course repo into your local master branch. This brings your fork's master branch into sync with the upstream repository, without losing your local changes.

   ```
   $ git merge upstream/master
   ```

   Notes:

   - Git may send you to a weird window asking you to write a commit message. You can escape this with `Shift+:` followed by Q.

You can repeat steps 5-7 anytime you want to make sure your local folder is up-to-date with my repository.

Propose an edit to a course document

There are two ways to do this. You could do this directly as a **pull request** on the GitHub platform, or you could write the edit in the folder on your local machine, push it to your forked repository, and then submit it as a **pull request** to my repository. Though more complicated, we will practice the second method because ultimately you will be coding on your local machine and then merging your changes into your group project repository so we want to practice these steps here.

In the course reopsitory there is a document called `GitHub_names.txt`. I would like you to add your first name and username to this document by:

1. Making the edits on your local computer
2. Pushing your local changes to your GitHub repository
3. Submitting a pull request to my repository asking me to accept your edits.

Making the edits on your local computer

Open the `GitHub_names.txt` file in the folder you cloned onto your computer. Add your First name and GitHub user name to the list and save the document.

Pushing your local changes to your GitHub repository

We now want to have these edits reflected in your GitHub repository.

1. Go to the terminal window and navigate to your cloned directory by typing

   `$ cd /c/Users/ *******/filepath`

---

2. **WARNING:** At this point you want to check if GitHub knows who you are. Type

   `$ git config --list`

   and it will spit out a bunch on information. Look and see if your GitHub username and the email you used to sign on to GitHub with are listed. If they are all is well. If they are not listed type

   `$ git config --global user.name "janedoe"`

   `$ git config --global user.email janedoe@pitt.edu`

   Make sure you set these to your github user name and the email you used to sign up with GitHub.

---

3. Check the status of your edits. Type

   `$git status`

   You should see:

   - if you have modified files with edits that have not yet been commited.
   - if you have any new files that do not exist in the GitHub repo

4. If you have new files type

   `$ git add .`

   to add all new files to GitHub

5. To prepare your modifications to be committed, type

   `$ git commit -a -m "Jane Doe's changes"`

   where the -a argument tells git to commit all the modifications and -m adds a message to you commit (a good practice when working on jointly edited files). Now if you type

   `$ git status`

   you will see that the files you changed are ready to be pushed to github so that they are reflected there.

6. Now you can type

   `$ git push origin master` or just `$git push`

   which basically tells git to push your changes from your remote "origin" computer to the "master" branch on github.

7. You can now look at your repository on github and check that the edits you made appear in your files and the files history.

Submitting a pull request to my repository asking me to accept your edits.

Now that your edits appear in your GitHub repository, you need to propose those changes to the owner of the original repo (me). You will do this via a pull request. Note: This will generate a pull request for ALL of the commits (changes) you made to any file in your forked repo since you originally forked it.

1. Navigate to your forked repo
2. Click the **Pull requests** tab.
3. Click the green **New pull request** button
4. Comparing change: you will now be shown a page that shows you how your edits fit into the existing repo. Check to see the status of your edits:

    a. If it says "Able to merge":

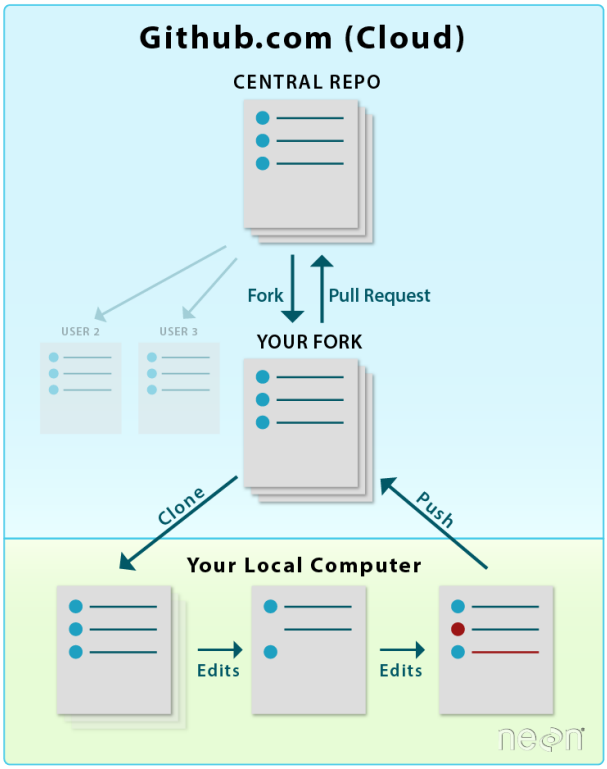
    - Click the green button **Create pull request**
    - In the pull request window, add a description/name for your edits in your pull request
    - Click **Create pull request** at the bottom of the page You have now sent me a pull request. I will review you edits and decide if I want to integrate them into my file.

    b. If it says "Can't automatically merge":

    - This means that your edits conflict with some other changes in the file. You can still create a pull request but I may need to make some modifications before integrating your changes.
    - Click the green button **Create pull request**
    - add a description/name for your edits
    - Click the green **Create pull request** button at the bottom of the page You have now sent me a pull request. I will need to review your edits, probably have to make some changes since there is a merge conflict, and decide if I want to integrate them into my file.

**Congratulatons! You have now practiced most of the key actions needed to work collaboratively with GitHub and Git. The way group projects are run on this platform is illustrated in the image below. Your group will have a central repository in which you are all collaborators. Each group member (other then the one hosting the central repository) will have their own fork (copy) of the repository on their own GitHub account. This copy will be cloned to your local machine (using Git) and then set up so that it syncs with the upstream central repo.

When you want to work on your group assignment you will open the Git command window and sync your local folder with the central repo so that any changes your groupmates made are reflected on your local machine. You can then do your coding and work on your local machine, and once you know that your code runs, you will push your changes to your fork, and then submit a pull request and merge you change to the central repo. For more detailed step by step instructions, and instructions on some other actions you may need to use, check out the GitHub_instructions document in the course folder.**



Group Homework Project

Setup

On GitHub, have one group member generate a private project repository for your group named `PS1_name1_name2_name3` where the usernames are the first names of your groupmembers. You will download the problem set .Rmd file from canvas and upload it to your groups repository. You will then add collaboradors to your repository who will create their own forks of the repository.

Group "leader": To set up a private repo:

1. Log onto you GitHub account
2. Click on the **Repositories** tab
3. Click on the green **New** button which will create a new repository from scratch
4. Name your repository "PS1_name1_name2_name3"
5. Select "Private"
6. Select "Add a README" file
7. Click the green **Create repository** button

Congratulations! You have create your repository! A couple things to note: Private repos will not show up on your main home site. To navigate to them you need to go to the Repository tab.

Group "leader": Adding the Problem set file.

1. Download the Problem Set .Rmd file from Canvas

2. Navigate into the private PS1_name1_name2_name3 repo
3. Click the **Add** file button and select Upload file in the drop down menu
4. Drop the .Rmd file you downloaded into the upload space
5. Write a commit message like "adding problem set Rmd file"
6. Click the green **Commit changes** button

You should now see the .Rmd file appear in the repository and if you select the file you can see the .Rmd code you will be working with.

Group "leader": Adding collaborators

1. Navigate into the private PS1_name1_name2_name3 repo
2. Click on the **Settings** tab
3. Click on the **Manage access** tab in the left menu
4. Click on the green **Invite collaborator** button
5. Enter your teammates username in the popup box, select the correct user and invite them
6. Repeat for all of your teammates, our course TA and I

Group team members: Becoming a collaborator on a private repo

1. Once invited, you will receive an invitation via email to collaborate on the repo (it could be in the updates tab in gmail).
2. Click view the invitation to be sent to the repo
3. Once in the repo, click the green **Accept invitation**
4. You can now view the private repo your leader set up.

Group team members: Fork the private repo

If you want to be able to use Git in order to clone the repo to your computer's files you will need to generate a fork of the repo (see instructions above).

Group workflow on GitHub:

Now that you all have repos with the problem set, you will want to:

1. Use Git to clone your repos to your local machines (see instructions above).
2. Work on your contribution to the code and answers on your local machine
3. Make sure your edits run by knitting the .Rmd file
4. Remove all files other than the .Rmd file from your local repo folder
5. Use git to push your changes to your GitHub repo (see instructions above).

If you are a team member:

Submit a pull request to incorporate your contribution into the main project repository **AND** Accept/Merge the edits you proposed your repo into the main project repository (Since you are collaborators on this repo you can authorize and merge pull requests into the main folder, you do not need the repo owner (group leader) to do this.) You can do this by

a. Navigate to your forked repository that contains the changes you want to integrate into the main repo
b. Click on the **pull request** tab
c. Click the green **New pull request** button

d. Click on the green **Create pull request** button
e. Name your pull request
f. Click on the green **Create pull request** button
g. From here there are two possibilities:

- If the pull request is able to Merge:
  - If you want to see the detail of what edits were made, click on pull request name to look over the changes and then navigate back to the main pull request page
  - Click the green **Merge pull request** button
  - Click the green **confirm merge**
  - Your changes have now been added to the main file
- If there are conflict in the pull request:
  - There will be a warning sign informing you of a conflict.
  - Click the **resolve conflicts** button
  - GitHub will open an editable window that contains your file in which the conflicting part(s) of the file is highlighted by a red bracket. Within these brackets the proposed changes are bracketed by the lines [«««< master] and [=======] and what is currently in your file is bracket by [=======] and [»»»> master]. With all this information, you can edit your file within this window until you are happy with it.
  - Click the **Mark as resolved** button (upper right of the file window)
  - Click the green **commit merge** button
  - Click the **I understand, update master** button in the pop up window
  - Click the green **Merge pull request** button
  - Click the green **Confirm merge** button
  - Your changes have now been added to the main file

Note: If you are the group leader, things could be changing in your repo without you realizing it! Keep an eye on your repo history so you know what happened while you were away. You may also want to be cautious when pulling repo changes to your local machine since there may be changes you didn't know about. (Maybe keep a back up of your own work on your local computer for your records?)

7. Next time you want to work on the assignment, make sure you syncronize your forked repo with the group leaders repo so that you are working on the most up to date version of your group's code (see instructions above). `$git pull` any updates to your local machine and repeat steps 2-6.
8. Once the main group .Rmd file is finalized, you can compile it to html, pdf or word for submission.

Hints: I would recommend only keeping track of the .Rmd file on GitHub, until the very end when you will produce your final html, pdf or word file. This will make it so that when you deal with any conflicts in pull requests, you only need to edit the .Rmd file, and don't have to worry about the other files.

When you are working on your local machine, do make sure to knit the .Rmd file frequently to make sure your code is compiling correctly and that you can produce the html or pdf files. However when it comes time to push your changes to GitHub, I would recommend deleting all the extra files that R created so that you only push the .Rmd file and only have to worry about merging the .Rmd file when you make pull requests to your group leader.

Now you are set up to work cooperatively on an assignment using GitHub. The remainder of this assignment consists of a simple coding and plotting exercise to practice collaborating with Github.

Omitted Variable Bias Simulation (Group excercise)

For this problem we are going to generate simulated data, with known characteristics, so that we can explore how the omission of a key variable biases our estimates.

To do this in R markdown we must first tell R to interpret the text we write as code. We do this by embedding a code *chunk* in the document. The following is a code *chunk* that only consists of a comment. If you are reading this in the .Rmd file, notice the chunk starts and ends with three ‘ marks. The first marks are followed by {r NAME} where NAME is the name of the code chunk (all chunks must have different names). This tells R markdown that the text inside the chunk should be interpreted as R code (not written text). Notice there are also some icons at the top right of the chunk. The first allows you to set the chunk settings (which you can also do by writing in the settings in the {} at the begining of the chunk). The chunk settings will tell R markdown whether you want the output, the warnings, the code... of the chunk to be visible in the final document. The second icon tells R to run all the code above the current chunk and the last to run the code in the current chunk.

```
#This is a comment, alone in a chunk.
```

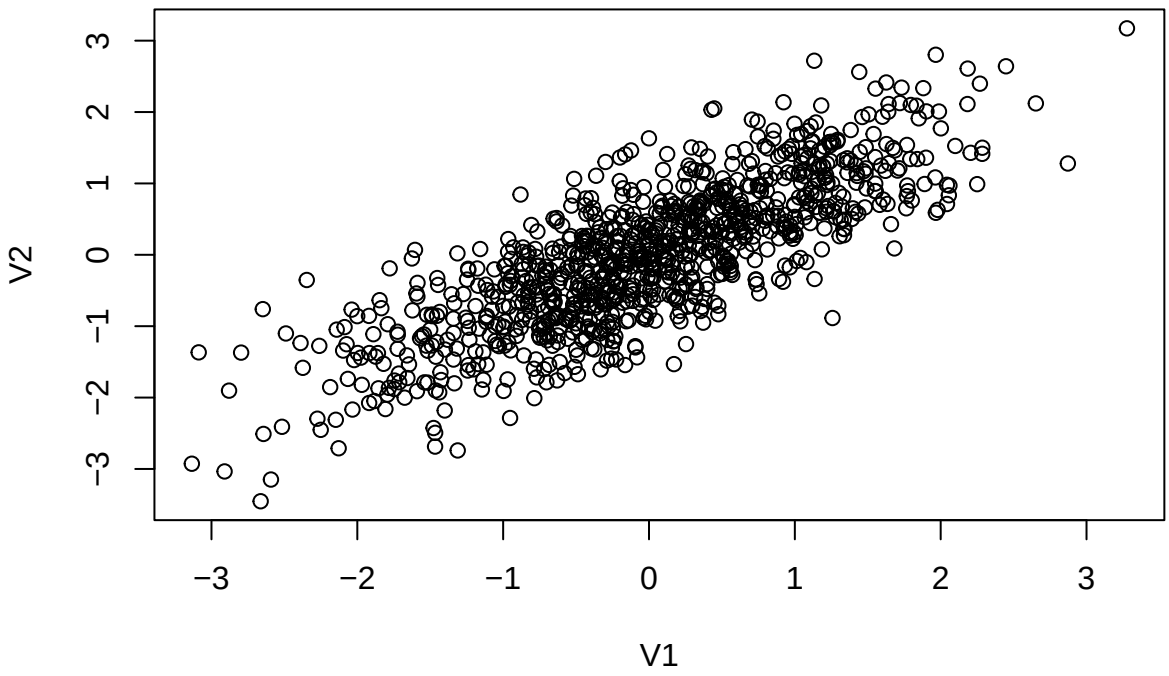
In the following chunk I generate simulated data consisting of two correlated variables. Notice the chunk setting is set to supress the warning messages that are generated when R loads a package.

```
library(MASS)
library(ggplot2)

out <- as.data.frame(mvrnorm(1000, mu = c(0,0),
                      Sigma = matrix(c(1,0.8,0.8,1), ncol = 2),
                      empirical = TRUE))
cor(out)
```

```
##     V1  V2
## V1 1.0 0.8
## V2 0.8 1.0
```

```
plot(out)
```



Next I generate a randomly distributed error term and I calculate the outcome variable which depends on both V1 and V2 and some noise:

$$Y = \beta_1 V_1 + \beta_2 V_2 + \epsilon$$

```
out$error<-rnorm(1000, mean=0, sd=1)

#The data generating process
B1<-3
B2<-6

out$Y<-out$V1*B1+out$V2*B2+out$error
```

TO DO: For the questions below write the needed code and a written response to the question.

Question 1: Write a chunk in which you regress Y on V_1 and V_2. Are your estimates of β_1 and β_2 biased?

Answer: No our data was randomly generated (within parameters) so their coefficients are not bias

```
View(out)
M1 <- lm(data=out, Y ~ V1 + V2)
summary(M1)
```

```
##
## Call:
```

```
## lm(formula = Y ~ V1 + V2, data = out)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.9621 -0.7060 -0.0009  0.6659  3.8658
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.02985    0.03243   0.921    0.358
## V1           2.88536    0.05407  53.364   <2e-16 ***
## V2           6.07374    0.05407 112.333   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.025 on 997 degrees of freedom
## Multiple R-squared:  0.9859, Adjusted R-squared:  0.9859
## F-statistic: 3.48e+04 on 2 and 997 DF,  p-value: < 2.2e-16
```

Question 2: Write a chunk in which you regress Y on V_1 only. Is your estimate of β_1 biased?

Answer: Yes, our beta 1 coefficient is biased due to OVB. By omitting V2, our beta 1 coefficient has upward bias (larger than it would be with V2 in the model: 7.87 vs 3.045).

```
M2 <- lm(data=out, Y~V1)
summary(M2)
```

```
##
## Call:
## lm(formula = Y ~ V1, data = out)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -11.905  -2.691  -0.015   2.598  11.581
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.02985    0.11977   0.249    0.803
## V1           7.74435    0.11983  64.629   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.787 on 998 degrees of freedom
## Multiple R-squared:  0.8071, Adjusted R-squared:  0.807
## F-statistic:  4177 on 1 and 998 DF,  p-value: < 2.2e-16
```

Question 3: Generate a new variable Y_{adj} such that $Y_{adj} = Y - \beta_2 * V_2$. Then regress Y_{adj} on V_1. Is your estimate of β_1 biased? Can you explain why/why not?

Answer: No, beta one is not biased in this model. We can see that the coefficient is very similar to model #1 where beta two was also included (3.072 vs 3.045). We can also affirm this mathematically given the relationship below

Y-B2V2 = B1V1 (Model #3), which is equal to Y=B1V1 + B2V2 (Model #1)

```
Y_adj <- ((out$Y)-(B2*out$V2))

M3 <- lm(data=out, Y_adj ~ V1)
summary(M3)
```

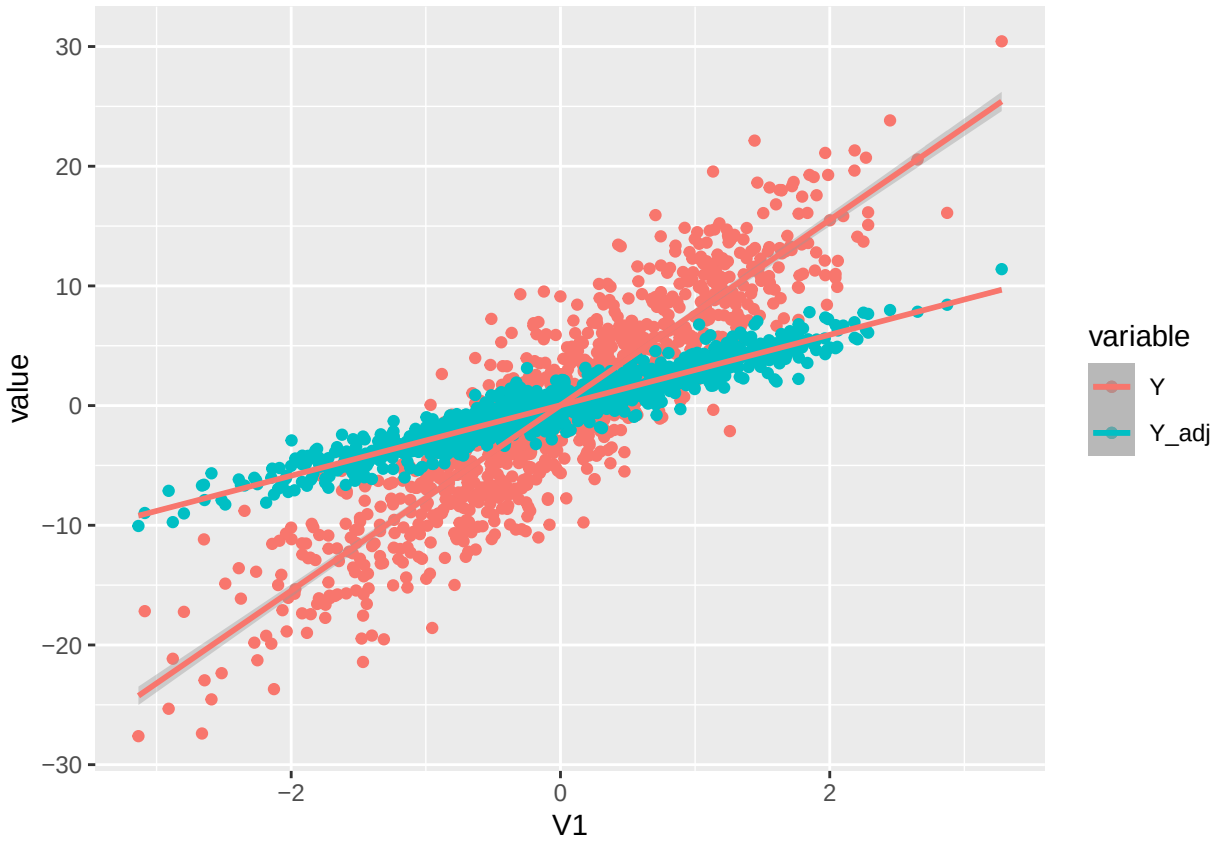
```
##
## Call:
## lm(formula = Y_adj ~ V1, data = out)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0057 -0.7155  0.0052  0.6601  3.8335
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.02985    0.03244    0.92    0.358
## V1           2.94435    0.03246   90.72   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.026 on 998 degrees of freedom
## Multiple R-squared:  0.8919, Adjusted R-squared:  0.8917
## F-statistic:  8230 on 1 and 998 DF,  p-value: < 2.2e-16
```

Question 4: The code below generates a scatter plot and regression line for the relationship between V_1 and Y as well as V_1 and Y_{adj}. Submit an improved visualization of this data. Hint: you will need to delete the # to get the code to run

```
plotted<-ggplot(out, aes(V1, y = value, color = variable)) +
   geom_point(aes(y = Y, col = "Y")) +
   geom_point(aes(y = Y_adj, col = "Y_adj"))+
   geom_smooth(method='lm', aes(y = Y, col = "Y"))+
   geom_smooth(method='lm', aes(y = Y_adj, col = "Y"))

plotted
```

```
## `geom_smooth()` using formula 'y ~ x'
## `geom_smooth()` using formula 'y ~ x'
```


30
20
10
0
−10
−20
−30
value
−2
0
2
V1
variable
Y
Y_adj

```
library(stargazer)
```

```
## Warning: package 'stargazer' was built under R version 4.1.1
```

```
##
## Please cite as:
```

```
##  Hlavac, Marek (2018). stargazer: Well-Formatted Regression and Summary Statistics Tables.
```

```
##  R package version 5.2.2. https://CRAN.R-project.org/package=stargazer
```

```
visual1 <- stargazer(M2, M3, type="text", header=FALSE, title="OVB vs No OVB",
                     style="qje",
                     column.labels = c("Model w/OVB bias", "Model w/o OVB bias" ),
                     covariate.labels = c("B1", "Intercept"),
                     dep.var.caption="Outcome (Y)")
```

```
##
## OVB vs No OVB
## ================================================================================
##                                          Y                     Y_adj
##                                  Model w/OVB bias    Model w/o OVB bias
##                                         (1)                   (2)
## --------------------------------------------------------------------------------
## B1                                    7.744***              2.944***
##                                        (0.120)               (0.032)
##
## Intercept                              0.030                 0.030
##                                        (0.120)               (0.032)
##
## N                                      1,000                 1,000
## R2                                     0.807                 0.892
## Adjusted R2                            0.807                 0.892
## Residual Std. Error (df = 998)         3.787                 1.026
## F Statistic (df = 1; 998)          4,176.962***          8,230.041***
## ================================================================================
## Notes:                              ***Significant at the 1 percent level.
##                                      **Significant at the 5 percent level.
##                                      *Significant at the 10 percent level.
```

Submission instructions:

1) Make sure the final version of your assignment is uploaded on GitHub in both html and Rmarkdown format.