Lab 4 - Version Control and Linear Regression

> ❗ Important
>
> This lab is due Friday, July 28 at 11:59pm.

Learning Goals

- Learn the basic usage of Git/Github and use it to download and upload assignments.
- Use tidymodels framework to build a linear model and estimate regression parameters
- Visualize your linear model

Monday:

- Setup Git Account and Github Access properly
- Learn to access assignments on Github
- Learn the commands of commiting (saving) and pushing (uploading)

Thursday:

- Perform exercises concerning linear regression.

> ❗ Important
>
> This Github Setup/Demonstration part of the lab is **not a part of your grade in this course**. You are only graded for the exercises concerning linear regression. However, the information contained in the Github/Git section is crucial for your success in this course,

since that will be the mode of HW/Lab dissemination.

What is Git and Github?

Git is a so called version control system (you can think of it as a fancy "Track Changes" in Microsoft Word) and GitHub is the home for your Git-based projects on the internet (like DropBox/Google Drive).

If you havent' seen git before, the terminology might be confusing.

Below is a relevant glossary:

- Repository (or repo): a folder
- Cloning: copying a folder from the internet to your computer
- Committing: saving your files on your computer
- Pushing: Uploading your saved files on your computer to the internet

The workflow generally goes as follows:

- There will be a repository on github containing the qmd files and data that are associated with an assignment/lab.
- You would "clone" the repo onto your RStudio Interface.
- You would work on your assignment/lab on RStudio, periodically "committing" to save your work.
- When you are done, you "push" your changes onto Github for the instructor to see.

Before we get started with that, we need to do some setting up (you only have to do this one time).

One Time Setup Part 1: Linking RStudio to Github

You will authenticate GitHub using SSH. Below are an outline of the authentication steps; you are encouraged to follow along as your TA demonstrates the steps.

> **Note**
>
> You only need to do this authentication process one time on a single system.

- Type `credentials::ssh_setup_github()` into your console.
- R will ask "No SSH key found. Generate one now?" You should click 1 for yes.

- You will generate a key. It will begin with "ssh-rsa…." R will then ask "Would you like to open a browser now?" You should click 1 for yes.
- You may be asked to provide your GitHub username and password to log into GitHub. After entering this information, you should paste the key in and give it a name. You might name it in a way that indicates where the key will be used, e.g., `sta199`).

You can find more detailed instructions here if you're interested.

One Time Setup Part 2: Configure Git

There is one more thing we need to do before getting started on the assignment. Specifically, we need to configure your git so that RStudio can communicate with GitHub. This requires two pieces of information: your name and email address.

To do so, you will use the `use_git_config()` function from the `usethis` package. (And we also need to install a package called `gert` just for this step.)

Type the following lines of code in the **console** in RStudio filling in your name and the email address associated with your GitHub account.

```
devtools::install_github("r-lib/gert")

usethis::use_git_config(
  user.name = "Your name",
  user.email = "Email associated with your GitHub account"
  )
```

For example, mine would be

```
devtools::install_github("r-lib/gert")

usethis::use_git_config(
  user.name = "Ed Tam",
  user.email = "tam.edric@gmail.com"
  )
```

You are now ready interact with GitHub via RStudio!

Clone the repo & start new RStudio project

- Click on the assignment link that Ed posted on Sakai announcements. Accept the assignment.

- Click on the green **CODE** button, select **Use SSH** (this might already be selected by default, and if it is, you'll see the text **Clone with SSH**). Click on the clipboard icon to copy the repo URL.
- In RStudio, go to *File* *New Project* *Version Control* *Git*.
- Copy and paste the URL of your assignment repo into the dialog box *Repository URL*. Again, please make sure to have *SSH* highlighted under *Clone* when you copy the address.
- Click *Create Project*, and the files from your GitHub repo will be displayed in the *Files* pane in RStudio.
- Click *Lab4.qmd* to open the template Quarto file. This is where you will write up your code and narrative for the lab.

Committing changes

Now, go to the Git pane in your RStudio instance. This will be in the top right hand corner in a separate tab.

If you have made changes to your Quarto (.qmd) file, you should see it listed here. Click on it to select it in this list and then click on **Diff**. This shows you the *diff*erence between the last committed state of the document and its current state including changes. You should see deletions in red and additions in green.

If you're happy with these changes, we'll prepare the changes to be pushed to your remote repository. First, **stage** your changes by checking the appropriate box on the files you want to prepare. Next, write a meaningful commit message (for instance, "updated author name") in the **Commit message** box. Finally, click **Commit**. Note that every commit needs to have a commit message associated with it.

You don't have to commit after every change, as this would get quite tedious. You should commit states that are *meaningful to you* for inspection, comparison, or restoration.

Push changes

Now that you have made an update and committed this change, it's time to push these changes to your repo on GitHub.

In order to push your changes to GitHub, you must have **staged** your **commit** to be pushed. click on **Push**.

Now let's make sure all the changes went to GitHub. Go to your GitHub repo and refresh the page. You should see your commit message next to the updated files. If you see this, all your changes are on GitHub and you're good to go!

> **!** Important
>
> The graded section of this lab begins here

Parasite Similarity Analysis

Parasites can cause infectious disease – but not all animals are affected by the same parasites. Some parasites are present in a multitude of species and others are confined to a single host. It is hypothesized that closely related hosts are more likely to share the same parasites. More specifically, it is thought that closely related hosts will live in similar environments and have similar genetic makeup that coincides with optimal conditions for the same parasite to flourish.

In this lab we will see how much evolutionary history predicts parasite similarity.

The Data

Today's dataset comes from an Ecology Letters paper by Cooper at al. (2012) *"Phylogenetic host specificity and understanding parasite sharing in primates"* which can be found here. The goal of the paper was to identify the ability of evolutionary history and ecological traits to characterize parasite host specificity.

Each row of the data contains two species, `species1` and `species2`.

Subsequent columns describe metrics that compare the species:

- `divergence_time`: how many (millions) of years ago the two species diverged. i.e. how many million years ago they were the same species.
- `distance`: geodesic distance between species geographic range centroids (in kilometers)
- `BMdiff`: difference in body mass between the two species (in grams)
- `precdiff`: difference in mean annual precipitation across the two species geographic ranges (mm)
- `parsim`: a measure of parasite similarity (proportion of parasites shared between species, ranges from 0 to 1.)

The data are available in `parasites.csv` in the data folder.

Packages

We'll use the **tidyverse** package for much of the data wrangling and visualization.

```
library(tidyverse)
library(tidymodels)
```

Exercise 1

- Part A: load the data and save the data frame as `parasites`.
- Part B: Let's start by examining the relationship between `divergence_time` and `parsim`. Based on the goals of the analysis, what is the response variable?
- Part C: Visualize the relationship between the two variables. (no regression line needed) Give no more than 1-2 sentences describing any patterns you see.

Exercise 2

Now, we'll model this relationship.

- Part A: Fit a simple linear regression model on `divergence_time` and `parsim` based on your answer in Question 1B. Report the coefficients.
- Part B: Interpret the slope and the intercept in the context of the data.
- Part C: Recreate the visualization from Exercise 1, this time adding a regression line to the visualization. What do you notice about the prediction (regression) line that may be strange, particularly for very large divergence times? (no more than 1-2 sentences)

Exercise 3.

Since `parsim` takes values between 0 and 1, we want to transform this variable so that it can range between $(-\infty,+\infty)$. This will be better suited for fitting a regression model (and interpreting predicted values!)

- Part A: Using mutate, create a new variable `transformed_parsim` that is calculated as `log(parsim/(1-parsim))`. Add this variable to your data frame. Note: `log()` in R represents taking the **natural log**.
- Part B: Visualize the relationship between `divergence_time` and `transformed_parsim`. Add a regression line to your visualization.

- Part C: Fit a simple linear regression model on `divergence_time` and `transformed_parsim`. Report the coefficients.

Submission

> **Warning**
>
> Before you wrap up the assignment, make sure all documents are updated on your GitHub repo. We will be checking these to make sure you have been practicing how to render and push changes.
> You must turn in a PDF file to the Gradescope page by the submission deadline to be considered "on time".
> Make sure your data are tidy! That is, your code should not be running off the pages and spaced properly. See: https://style.tidyverse.org/ggplot2.html.

To submit your assignment:

- Go to http://www.gradescope.com and click *Log in* in the top right corner.
- Click *School Credentials* → *Duke NetID* and log in using your NetID credentials.
- Click on your *STA 199* course.
- Click on the assignment, and you'll be prompted to submit it.
- Mark all the pages associated with exercise. All the pages of your lab should be associated with at least one question (i.e., should be "checked"). *If you do not do this, you will be subject to lose points on the assignment.*
- Do not select any pages of your .pdf submission to be associated with the *"Workflow & formatting"* question.

Grading

Component	Points
Ex 1	10
Ex 2	15
Ex 3	15
Workflow & formatting	10
Total	**50**

Note

The "Workflow & formatting" grade is to assess the reproducible workflow. This includes: - linking all pages appropriately on Gradescope - putting your team and member names in the YAML at the top of the document - committing the submitted version of your `.qmd` to GitHub - Are you under the 80 character code limit? (You shouldn't have to scroll to see all your code). Pipes `%>%`, `|>` and ggplot layers `+` should be followed by a new line - You should be consistent with stylistic choices, e.g. only use 1 of `=` vs `<-` and `%>%` vs `|>` - All binary operators should be surrounded by space. For example `x + y` is appropriate. `x+y` is not.