Working collaboratively

Milestone 1

Data science is a collaborative discipline. Pretty much no data scientist works alone, so neither should you! In this course you'll collaborate with teammates on the project.

The first milestone of the project, today's activity, will introduce you to the technical aspects of collaborating on a reproducible data science project that is version controlled by Git and hosted on GitHub in a repository shared by all teammates.

Yes, this means you and all of your teammates will be pushing to the same repository! Sometimes things will go swimmingly, and sometimes you'll run into **merge conflicts**.

> **Note**
>
> The word "conflict" has a negative connotation. And, indeed, merge conflicts can be frustrating. But they serve to make sure no team member inadvertently writes another team member's work. All changes that are in "conflict" with each other need to get resolved explicitly, with a commit message, which helps avoid haphazard overwriting.

Goals

The goals of this milestone are as follows:

- Pick a team name
- Collaborate on GitHub with your teammates and resolve merge conflicts when they inevitably occur
- Discuss and write up your team contract
- Start thinking about your project proposal

Team name

Pick a team name. It can be straightforward, or it can be cheeky, but it must be SFW. Let your TA know your team name asap. Once everyone's team names are in, your project repos will be created (give your TA a couple of minutes), and then you can continue with the rest of your task.

Merge conflicts

When you and your teammates work on the lines of code within a document, and both try to push your changes, you will run into issues. Merge conflicts happen when you merge branches that have competing changes, and Git needs your help to decide which changes to incorporate in the final merge.

Our first task today is to walk you through a merge conflict! First, a bit of Git review:

- Pushing to a repo replaces the code on GitHub with the code you have on your computer.
- If a collaborator has made a change to your repo on GitHub that you haven't incorporated into your local work, GitHub will stop you from pushing to the repo because this could overwrite your collaborator's work!
- So you need to explicitly "merge" your collaborator's work before you can **push**.
- If your and your collaborator's changes are in different files or in different parts of the same file, git merges the work for you automatically when you **pull**.
- If you both changed the same part of a file, git will produce a **merge conflict** because it doesn't know how which change you want to keep and which change you want to overwrite.

Git will put conflict markers in your code that look like:

```
<<<<<<< HEAD

See also: [dplyr documentation](https://dplyr.tidyverse.org/)

=======

See also [ggplot2 documentation](https://ggplot2.tidyverse.org/)

>>>>>>> some1alpha2numeric3string4
```

The `===`s separate *your* changes (top) from *their* changes (bottom).

Note that on top you see the word `HEAD`, which indicates that these are your changes.

And at the bottom you see `some1alpha2numeric3string4` (well, it probably looks more like `28e7b2ceb39972085a0860892062810fb812a08f`).

This is the **hash** (a unique identifier) of the render your collaborator made with the conflicting change.

Your job is to *reconcile* the changes: edit the file so that it incorporates the best of both versions and delete the `<<<`, `===`, and `>>>` lines. Then you can stage and render the result.

Activity

Setup

- Clone the `project` repo and open the `about.qmd` file.
- Assign the numbers 1, 2, 3, 4, and 5 to each of the team members. If your team has fewer than 5 people, some people will need to have multiple numbers.

Let's cause a merge conflict!

Our goal is to see two different types of merges: first we'll see a type of merge that git can't figure out on its own how to do on its own (a **merge conflict**) and requires human intervention, then another type of where that git can figure out how to do without human intervention.

Doing this will require some tight choreography, so pay attention!

Take turns in completing the exercise, only one member at a time. **Others should just watch, not doing anything on their own projects (this includes not even pulling changes!)** until they are instructed to. If you feel like you won't be able to resist the urge to touch your computer when it's not your turn, we recommend putting your hands in your pockets or sitting on them!

Before starting

Everyone should have the repo cloned and know which role number(s) they are.

Also, any teammates who haven't done this before should go to their Terminal and type `git config pull.rebase false` to set up their preferences for pulling.

Role 1

- Go to `about.qmd` in your project repo. Change the `[team name]` to your actual team name.
- Render the project by clicking on Render in the Build tab, commit (all changed files), and push.

> **!** Important
>
> Make sure the previous role has finished before moving on to the next step.

Role 2

- Change the team name to some other word.
- Render the project by clicking on Render in the Build tab, commit (all changed files), and push. You should get an error.
- Pull. Take a look at the document (`about.qmd`) with the merge conflict.
- Clear the merge conflict by editing the document to choose the correct/preferred change.
- Render the project by clicking on Render in the Build tab.
- **Click the Stage checkbox** for all files in your Git tab. Make sure they all have check marks, not filled-in boxes.
- Commit and push.

> **!** Important
>
> Make sure the previous role has finished before moving on to the next step.

Role 3

- Change the a name of the first team member.
- Render the project by clicking on Render in the Build tab, commit, push. You should get an error.
- Pull. No merge conflicts should occur, but you should see a message about merging.
- Now push.

> **!** Important
>
> Make sure the previous role has finished before moving on to the next step.

Role 4

- Change the a name of the first team member to something other than what the previous team member did.
- Render the project by clicking on Render in the Build tab, commit, push. You should get an error.
- Pull. Take a look at the document with the merge conflict. Clear the merge conflict by choosing the correct/preferred change. Render the project by clicking on Render in the Build tab, commit, and push.

> **!** Important
>
> Make sure the previous role has finished before moving on to the next step.

Role 5

- Change the a name of the rest of the team members and add descriptions for each person with the help of your team members. Role 5 should be the only one typing, the others should help verbally.
- Render the project by clicking on Render in the Build tab and commit. Discuss as a team what you expect to happen when you hit push. Should there be a merge conflict error or not?
- If there is a merge conflict, fix it. If not, push your changes.

Everyone

Pull, and observe the changes in your project.

Tips for collaborating via GitHub

- **Always pull first before you start working.**
- Resolve a merge conflict (render and push) *before* continuing your work. Never do new work while resolving a merge conflict.
- Render, commit, and push often to minimize merge conflicts and/or to make merge conflicts easier to resolve.
- If you find yourself in a situation that is difficult to resolve, ask questions ASAP. Don't let it linger and get bigger.

Team contract

Go to your project repository and open `contract.qmd`. As the instructions suggest, pick a teammate to be the scribe. Discuss the prompts and write up your answers. Once done, click *Render* in the build tab, commit, and push.

Grading

We will evaluate the first milestone of your project based on your participation in this activity. Each team member who participates in the activity during the lab session will earn 5 points towards their project.