Data Science for AAE

AAE 718 – Summer 2025

Worksheet 6

GitHub and Markdown

1 Reading

- GitHub - This worksheet, I guess. I'm also providing a large number of links. In general it can be surprisingly difficult to find a GitHub tutorial for somebody that knows nothing.
- Markdown - I summarize fairly well, but for more depth here is a link.

2 Daily Goals

- Understand how to use Git and GitHub
- Understand pushing, pulling, branching, pull requests and forking
- Merge conflicts
- Markdown
- GitHub, Markdown, and you
- PDFs

3 Git and GitHub

By default Git is a local program, this means it lives on only your machine. But that's not how we share code. Hence GitHub, which is a cloud based Git implementation with many nice features. GitHub is *not* the only cloud based Git service, there are many. However, GitHub is widely regarded as the best. GitHub is also now owned by Microsoft, which has certain advantages when working with the US government[1].

3.1 What is Git and why should you care?

Imagine you are working on a large coding project. This project has several (potentially hundreds) of files with thousands of lines of code. Moreover, people are using this code in their work. If you start making changes it's possible you'll break something vitally important and then people will be mad at you.[2]

This is where Git comes into play. You can tell Git to *branch* your code. This will leave your old code unaffected and you can make your changes in a safe environment. When you're confident everything works as expected Git will allow you *merge* your changes.

3.2 Getting set up

You'll also need a GitHub account. Click this link and get started. I would highly recommend making this using a personal account. You can always add other email addresses. As an example, I have been using GitHub for many years and I've lost email addresses from old institutions, if I had used one of those it would be quite annoying to recover my GitHub.

GitHub will probably also prompt you to set up the mobile app. This is a good idea for the dual authentication. GitHub security is *very* important. If your account gets compromised people can add malicious code into your projects. This is bad.

There are many ways to interact with GitHub. Let's detail two of them.

[1] Microsoft is the official cloud provider for the US government.

[2] This should be avoided at all costs.

3.2.1 GitHub Desktop

Really all you need is GitHub Desktop. Here is a link to download what you need, you can also just Google it's all the same.

3.2.2 VSCode

This is how I interact with Git. I have a number of extensions that make this easy. On the left bar there is a 'Source Control' menu. I'm not going to go over getting this installed and set up, you have Google as much as I do.

3.3 An Overview

First, this may seem relatively abstract at the moment as the practical examples will be below. But I want you to get an idea of what's happening. You can always reread this later. I'll also put this into a paper format with useful definitions highlighted.

Definition 3.1 (Repository)**.** A *repository* is your code. When something references a *repository* they are talking about your GitHub, but you can also have local repositories. These are sometimes referred to as a *repo* for short.

I have a large number of repositories in GitHub, here is an example. This is the main repository for WiNDC, the thing I do for a living. Repositories can be either private or public and you can add people repositories so they can make changes.

Definition 3.2 (Clone)**.** When you *clone* a repository, you download it to your computer.

Cloning a repository is the first step to making changes to a repository *that you own*. You can clone any public repository, but you may not be able to change that code on GitHub.

Definition 3.3 (Fork)**.** A *fork* is a copy of somebody elses repository in your personal GitHub.

If you fork a repository, that is a copy that you own. You can clone that repository and make changes.

Definition 3.4 (Commit)**.** A *commit* is registered change to the code. You are required to include a small comment when you commit.

At it's core, Git is a sequence of commits. The power of Git is that you can easily trace back and undo a sequence of commits. In general, it's better to have many small commits vs one huge commit.

A commit is a local change only. If you commit some code on your machine, nobody will see it on GitHub.

Definition 3.5 (Push)**.** A *push* uploads your commits to GitHub.

Definition 3.6 (Fetch)**.** A *fetch* downloads commits from GitHub.

Definition 3.7 (Pull)**.** A *pull* fetches changes and makes them available to you.

In theory there is a difference between fetch and pull, but in practice it acts the same. Plus, whenever you push to a repository you'll also be pulling. In GitHub Desktop it's a single button.

Definition 3.8 (Pull Request)**.** A *Pull Request* takes all of the commits you've made and requests they be pushed into a "different" repository.

Pull requests are a GitHub feature, and they are incredibly useful. First, different is in quotes in the definition because I routinely pull request from my personal repository into itself. We'll discuss why this is so useful later.

When you open a pull request, you're requesting that the owner perform a pull from your repository. This can be dangerous for the owner, what if you're inserting malicious code? To counteract this GitHub has a built in code-review feature. You can inspect line-by-line every proposed change, you can even leave

comments on individual lines of code. The owner of the repository may request you make changes, or might make the changes themselves.

Pull requests are how people contribute to open source software.

Definition 3.9 (Merge)**.** A *merge* is anytime code from one repository is being combine with another repository.

In the best case scenario, you won't have an issue merging. However, it will happen that code from one repositories conflicts with code in another, this is a merge conflict and they can be difficult. There are tools to help mitigate this, we will only scratch the surface.

Definition 3.10 (Branch)**.** A *branch* is a parallel version of your code. As you switch branches your code will change to reflect the new branch.

Branches are incredibly useful. If you have a project and you're working on several new features, each feature should live in a branch. This means you can focus on just your change and won't get distracted if another feature breaks something.

When you are finished with your branch, you open a pull request to your main branch so you can merge the changes. You can then delete your branch. The pull request process will generate automated patch notes based on your pull request. I can't overstate how invaluable this is. Here is an example, if you scroll down to "Merged Pull Requests", you'll see what I'm talking about.

3.4 How I typically use Git/GitHub

If I'm working on a project that's just me or has just started, I push commits directly to my main branch. A small code base makes this easy.

Once my project starts to get large, I'll make branches. In general, you should only branch from your main branch. Otherwise you can get issues where things get out of sync. Of course, this is not a rule and people do this all the time. But when you're first learning, it's best to keep things simple.

If I have an idea and want to test it, I'll make a local branch and just play with my idea. If it works out, I'll push the branch to GitHub and start the PR[3] process. If not, I can just switch back to my main branch and everything is back to normal.

3.5 Practical Tutorials

This may be seen as a cop-out on my part, but there are WAY better tutorials than I could every write.

- First contributions in GitHub Desktop
- Same as the above, but using tools directly in VSCode. This is what I do, but isn't necessary.
- Lots of tutorials. This is a lot of tutorials. You should do at least:
  - Introduction
  - Markdown (also below)
  - Review Pull Requests
  - Merge Conflicts

3.6 .gitignore

The last important thing to know: Git only likes raw text files. Uploading binary files (like PDFs or some image files) is discouraged. However, sometimes your code will automatically generate these types of files which means Git will see them. The solution is *.gitignore*.

In your local git repo there will be a file called *.gitignore*. Anything in this file will be automatically

[3]Pull Request

ignored by git[4]. You can put any regular expression in this file[5]. If you want to ignore all PDFs just put

```
*.pdf
```

in gitignore and save it. You may need to commit only the gitignore file, but then you should see the PDFs disappear from the git commit screen. Unless you've already uploaded a specific PDF, then it'll still be captured. There is another process to remove it which we won't cover here.

When you create a new empty repository there is typically a gitignore option that will create a default file for whatever language you're using. VSCode has an extension that will create a gitignore file for basically any language.

3.7 README

Every Git repository should have a README.md file, in fact you can should one in every subdirectory too. On GitHub if you scroll down on any repository you'll see text describing the repository, this text is in the README file. The README file is *markdown*, we'll discuss markdown in two sections.

3.8 Summary

These tools should get you started on GitHub. You may not find this useful at the moment, but someday you might. Git can be useful anytime you're working with and share raw text files (like code as opposed to a PDF). Being able to sync changes with other people is huge.

GitHub is *way* deeper than we are going in this course. I have a Julia project set up in GitHub that whenever I push a change, GitHub will run tests in several operating systems and different versions of Julia. It will also build my documentation and push it to a website.

If you want to learn more about Git/GitHub, Google is a great tool. I've written this worksheet because most of the tutorials I've seen don't explicitly define these things, which is annoying. You should now be set to do most of what you need to do.

4 Markdown

Markdown is a markup language, it's name is kind of a pun. You might be familiar with other markup languages, like HTML. Markdown is widely used because of its simplicity and how it makes things look.

In terms of complexity, Markdown sits between Google Docs[6] and LaTeX. Google Docs (or Microsoft Word) is a "what you see is what you get" editor. Whatever you type is what you'll see in the final product. This is nice for basic editing, but images can be a nightmare. LaTeX, if you're unfamiliar, is almost the exact opposite, what you type is independent of the final product. For example, I've written this document in LaTeX. I've loaded several packages to get this format. I could easily change the package and have the document look different.

Markdown is slightly harder to use than Google Docs, but easier than LaTeX. You don't need to load packages to format your document, but your document won't look pretty right away.

Markdown is ideal for quick typing that needs to look "good enough", you probably wouldn't write a research paper in it but it'll look better than Word. Plus, because it's all raw text it's better with Git.

4.1 Markdown basics

Luckily, Markdown is very simple. Open a new document and save it as "Something.md"[7]. And just type words into it. In VSCode, if you're in a markdown file, there is a button in the upper right of the window

[4] Hence the name.
[5] We'll cover these later.
[6] You can actually write using Markdown in Google Docs. It's great
[7] No need to be literal here, just the .md is important.

that looks like two rectangles and a magnifying glass. Hover over it and it should say "Open Preview", click it and it'll display your markdown.

Let's add a header, type

```
# Header 1
```

Your preview should auto-update and you'll see a nice bold header. You can use multiple #'s for smaller headers.

```
# Section
## Subsection
### SubSubSection
```

You can already see the difference between what you type and what gets displayed. As you get used to typing like this you'll find it's faster and more clear then whatever you need to do to get a header in Google Docs[8].

Let's say you want to type a few paragraphs. You'll be surprised when hitting enter at the end the line doesn't work. Try it, type a few paragraphs. The display should be a single run on paragraph. To rectify this, put an empty line between paragraphs.

```
This is a
single paragraph.

This is a new paragraph.
```

4.2 Math

Markdown steals most math from LaTeX. This is really important because LaTeX does math better than anything. Let's say you wanted to type the equation for standard deviation,

$$\sigma = \sqrt{\frac{\sum (x_i-\mu)^2}{N}}$$

in Markdown this is

```
$\sigma = \sqrt{\frac{\sum (x_i-\mu)^2}{N}}$
```

If you've never seen a LaTeX equation, this is scary. Let's break it down

1. Wrap math in dollar signs $. This is the math indicator.
2. Special names are prepended with \. Greek letters, functions, etc.
3. `\sqrt` is a function, the { ... } are where the square root starts and ends.
4. `\frac` is a function with two inputs, numerator and denominator. You should examine the code to see this.

You can make equations more prominent with two dollar signs $$

```
$$\sigma = \sqrt{\frac{\sum (x_i-\mu)^2}{N}}$$
```

I'm not going to go into depth here, but I'll give you a few examples that will get you started.

```
$$\sum_{n=0}^{10} x_n$$

$$F = G\frac{m_1m_2}{r^2}$$
```

[8] Try the above code in Google Docs. It works and is great, seriously

```
$$\int_{-1}^{1} (4-x^2) - (x^2-2x)\,dx$$

$$X = \left\{\begin{array}{cc}
    x^2, & x<1 \\
    e^x, & \text{otherwise}
\end{array}\right.$$
```

4.3 Various Markdown things

4.3.1 Lists

Numbers lists are just number.

```
1. Point 1
    1. Sub point
2. Point 2
7. Point 3
```

Note that the numbers you give may or may not be important, depending on what is displaying the markdown. If you notice Point 3 is numbered 7, but in the display it's numbered 3.

Unnumbered lists are either - or *

```
- Point 1
    - Sub point
- Point 2
- Point 3
```

4.3.2 Code blocks

There are two ways to do this, inline or as a block. For inline, use backticks

```
`function name`
```

For a code block use indentations.

```
    def f(x):
        return x**2
```

4.3.3 Links

Want to link to a URL?

```
[Google](https://www.google.com)
```

You can also use this to link to specific files on your computer. Replace the URL with a path to that file.

This also works for intra-document links.

```
# My Section


[Link to My Section](#my-section)
```

VSCode should popup a list of your section when you type the #. If it doesn't, it's probably an extension I have installed. Search the extensions for "Markdown" and you'll find a number of them.

4.4 Text Formatting

Want to **bold** some text?

```
**This is bold**
```

```
__also bold__
```

The second is two underscores.

How about *italics*?

```
*This is italics*

_also italics_
```

How about strikethrough[9]?

```
~~This is italics~~
```

4.4.1 Images

Basically a link to the image on your computer, but there is an extra twist.

```
![Some Text](path/to/image.png)
```

Many file formats are supported, but vector graphics are, in general, the best.

4.4.2 Tables

These can get complex, but at their core they look like

```
| Variables | Value | Another Column |
| --------- | ----- | -------------  |
| x         |  10   |  I wanted three columns |
| y         |   5   |                |
```

The second row must have the format `| --- |` with at least three dashes. This defaults to left aligned, for center aligned change it to `| :---: |` and for right aligned `| ---: |`. The spacing in each cell is arbitrary, you can make them as wide or narrow as you please. But for your sanity, make them consistent. My example drives me crazy because of the third column in the third row. You don't need to do anything special to end the table, just an empty line.

4.4.3 Exporting PDFs

It's possible to export your markdown file as a PDF. I use an extension in VSCode, but I'd recommend Google to find a solution.

Writing homework for this is quite difficult. Most of these problems can't reasonable be graded by myself. We're going to have to go with the honor system for most of these.

For Problems 1 – 4 you'll be creating a repository and using it throughout.

Problem 1 (5 pt) Create a GitHub account (if you don't already have one) and download GitHub desktop. If you're feeling adventurous you can also get Git working directly in VSCode[10].

1. Create a new empty repository on your PC
2. Add some files to it, code from a previous worksheet would be fine
3. Commit these files and push the repository to Github
4. Go on Github and see your files

Problem 2 (5 pt)

[9] Not easy to do in LaTeX

[10] This is how I use Git

1. Create a branch of your repository, you can do this directly in Github desktop[11]
2. Make some changes to your code and commit
3. Publish your branch to Github, basically do a push
4. You can now see your branch on Github
5. In your local repository, change back to your main branch and read through your code. You should see the changes you've made have disappeared.

This is the purpose of a branch. Make changes without destroying working code.

Here is a use-case example. Let's say I gave a problem where you get 5 points for having a working solution, but 10 points for the fastest solution. You should try to solve it first and get the 5 points, but then you want to try for the 10. You don't want to lose the code for the 5 points, so you make a branch.[12]

Problem 3 (5 pt)

1. Create a pull request to update your main branch with your new branch. You do this on GitHub.
2. Go to the Pull Requests tab and examine your PR. You should see four tabs, Conversation, Commits, Checks, Files Changed.
3. Conversation - You'll see your commits and comments. Make a new comment.
4. Commits - A better view on the commits
5. Checks - You won't see anything because we aren't covering checks. In my work I have GitHub configured to automatically run a number of tests. This screen will tell me if they have passed or failed.
6. Files Changed - You can actually see the changes and make comments on specific lines of code. You can mark a file as "Viewed" to collapse it. This is the first step of a quality Code Review. This is an information dense screen, but it fairly self-explanatory you should familiarize yourself with it.
7. Leave a comment on a few lines of your changed code. If you hover over the code you'll see a blue plus, you can click and drag to select multiple lines.

Problem 4 (5 pt)

1. Open an issue in your GitHub repo (in the issues tab)
2. Call it whatever you want and write a description. A good description in an actual repository[13] will have a brief description of the problem, a minimal working example, and, if you can, a potential solution. Here is an issue I wrote for Julia JuMP, you can see this is now a closed issue with lots of discussion.
3. Link your issue to your pull request. This is easy to do, in your Pull Request you can either edit the description or add a new comment and type a # and it should pop-up a list of all the issues. If you know the number you can type it, you can see the number on the issue page.

Issues are great. If they are linked to a pull request, when you merge the PR you can automatically close the issue. Plus if you release your code GitHub will list the new PRs and the issues they close, with links to each.

[11] Probably covered in the tutorials

[12] Don't say "I'll just copy/paste my old code somewhere else". You will go confused eventually, especially if you're working in a team or have 5 copies. I'm speaking from experience working with someone who will make a small change and email it to you and then you have 7 copies of code with slight differences and you don't know which one is correct. Then you get to combine these into a single thing, but there is NO documentation on the changes or what has changed. So you do this and you miss things, because how would you know all the differences? Then your whole work flow slows down because the code no longer works and you can't remember what the original file was, was it "code-new", "code-original", "code-new2"? You have no idea. Just force this person to use GitHub and make branches.

[13] In other words, you don't need to do this here

Problem 5 (5 pt) Here is a repository I created link to the repo. You have view access to this, because it's a public repository, but you can not edit it.

1. Fork this repository to your account
2. Add your name to the table in `roster_names.md`, follow the given format
3. Open an issue on my repository with some name and description
4. Open a PR using your fork and link your issue in the description.
5. I'll comment on your PR and have you do more work.

For this problem you'll write your solutions in Markdown and submit both the markdown file and a PDF generated from the markdown.

Problem 6 (5 pt) Find a tool to export a markdown file as a PDF. You'll get these points by submitting a PDF.

Problem 7 (5 pt) Give each problem a section header and create a table of contents with links to each problem.

Problem 8 (5 pt) Recreate the following table. You don't need to match the formatting, just data. Although you should be mindful of the justification (left, right, center) of each cell.

Variable	Value	Margin
X	1.0	0
Y	1.1	0
Z	.8	-10
PX	1.0	0

Problem 9 (5 pt) Find an interesting image and include it in your file.

Problem 10 (5 pt) Recreate the following equations in Markdown. If you are unfamiliar with LaTeX equations, this problem may be difficult. Just ask, I'll help.

1. $f(x) = \sin(x)$, I don't want $f(x) = sin(x)$
2. $U(x, y) = (\alpha x^\rho + (1 - \alpha)y^\rho)^{1/\rho}$
3. $\frac{\partial U(x,y)}{\partial x}$. This uses `\partial`
4. There is some magic here. `\left(` and `\right)` are useful

$$H = \frac{M}{p_H + \left(\frac{\alpha \cdot p_H}{1-\alpha}\right)^\sigma}.$$

5. This one might be hard.

$$p_U = \left[\sum_i \theta_i \left(\frac{p_i}{\bar{p_i}}\right)^{1-\sigma}\right]^{1/(1-\sigma)}$$