Tips & Tricks

These are some general tips and tricks for programming and data science. They are considered extra to the class in general - you need not follow them to succeed in the class. They are comments that will apply to the projects, rather than the assignments.

Read code

The hardest part of starting is not knowing what you don't know. Seeing what other people do and how they do can be finding solutions that you didn't know how to search for.
How to do this:

- go to GitHub, pick keywords and search for interesting repos
- Look at the source code for functions and classes that you use. Use the '??' magic command after a function (in Jupyter notebooks) to see source code.
- Find your anaconda folder and scroll through packages and their source code.

Write all your code like a human is going to read it.

Code always serves two purposes - it specifies a set of instructions for a computer to follow, and it is a set of instructions for a human to understand what the computer is being asked to do. Keep this in mind, and write code with a human reader in mind. Most code (for this class, in research, and definitely in industry) will indeed be read by some, or perhaps many, humans.
How to do this:

- Documentation. Comment your code!
- The best code is not necessarily the shortest number of lines, or cleverest implementation - prioritize clarity over line efficiency.
- If you write code that is quick and effective, but that is obscure as to how it works, also add a commented out longer-but-clearer version for others to follow the logic.

Learn & Follow Standard Procedures and Best Coding Practices

Standard procedures, style guides, and best practices exist for a reason - they are usually experience driven protocols developed to increase efficiency, portability, shareability and user-experience.
How to do this:

- Read and follow Python's style guide, pep8: https://www.python.org/dev/peps/pep-0008/
- Industry standards require code be version controlled, profiled, and tested. Although you are not expected to necessarily be doing these things in this class, and not everything applies to every project, it is worth knowing what they are:
  - Version Control: https://en.wikipedia.org/wiki/Version_control
  - Testing: https://en.wikipedia.org/wiki/Test_script
  - Profiling: https://en.wikipedia.org/wiki/Profiling_(computer_programming)

Plan your coding projects

Decisions made early on in a project can have significant ramifications as the codebase grows. Start with a clear plan about what you are trying to do and how you want to do it. This can be done at many levels - when thinking of a project, a module, a function or a line of code.
How to do this:

- Consider the specifications of your problem / project / function. What are the input/ output requirements, and how quick does it need to be? Write out an outline of these specifications, and revisit it as you work on your project.
- For larger projects / functions: sketch them out, in pseudocode.
    - And/or use special functions like ‘break’, ‘continue’, and ‘pass’ so that you can get skeleton of real code that will run, even if doesn’t actually do anything. This lets you map out a functional structure, and then go back and fill in the details.

Know when to pause

Tech culture may glorify coding sprints, long hackathons, and all-nighters, and though there may be place for that, real progress comes from sustainable work over the long term.
How to do this:

- Sleep, eat, nap, play, rest, go outside.
- Look through hands-on materials, and get started on assignments and the project early, so if you get stuck, you have time to pause, ask for help, and come back to it.
- Some things suddenly seem much clearer after you take a minute to get away from them for a bit and come back to them. Try to recognize when it might be useful to step away and come back to it later.