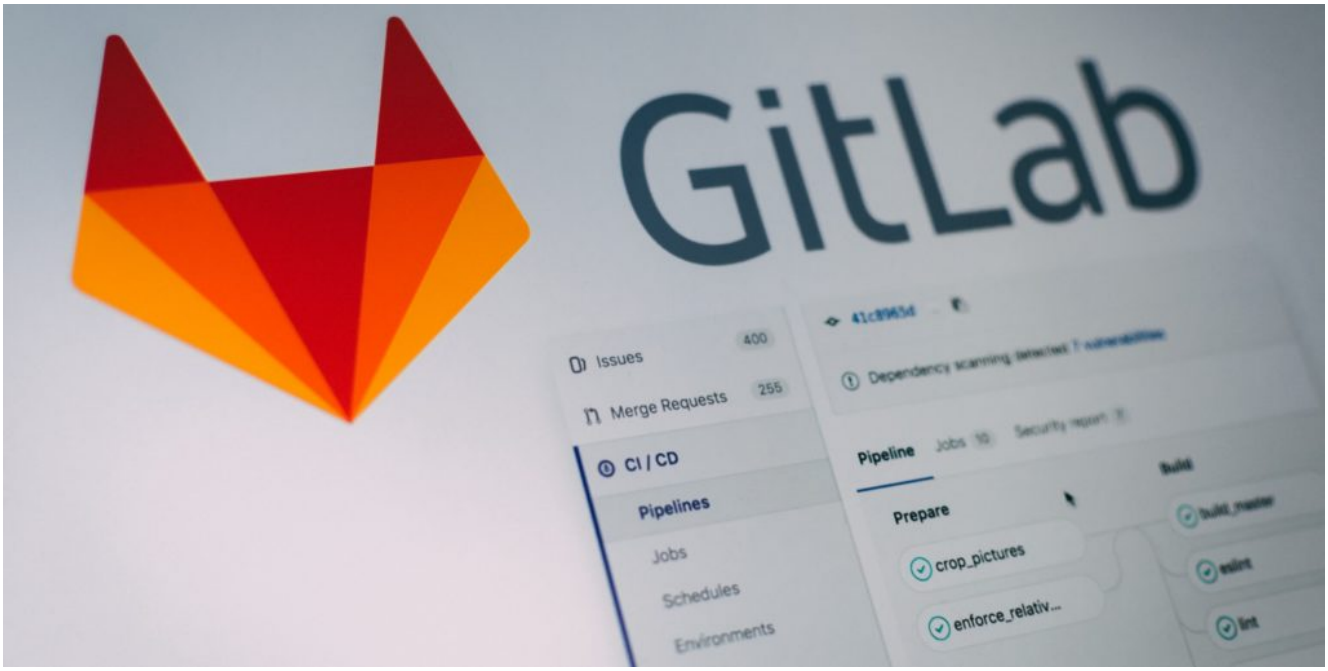
Creating Reproducible Data Science Projects



Reproducibility and Why it Matters

A Nightmare Scenario

Imagine you completed a one-off analysis a few months ago, creating a fairly complex data pipeline, machine learning model and visualisations. Fast forward to today and you have Emily, a senior executive at your company, asking you to reuse that work to help solve a similar, time-critical business problem. She looks stressed.

Now if only you could remember which copy of your model was the correct one; if you could make sense of the spaghetti code scattered throughout your Jupyter Notebooks, each with helpful names such as Untitled_1 and Untitled_2; what did the data_process function do and why are there six slightly different versions of it? If only there was some documentation!

“No problem!” you assure her, and after a few sleepless nights, during which you had to reverse engineer the entire codebase, the analysis is ready. Emily looks impressed, that promotion you’ve been waiting for might finally happen.

The next day Emily is back from presenting your findings. She's not happy – apparently there were mistakes in the analysis caused by simple coding errors that could have cost the company millions. If only you had run some tests! You apologise as she walks away muttering under her breath, you sit there and wonder if maybe you should pack up your desk before you head home for the night.

This blog article gives an overview of how we avoid this scenario by ensuring our data science projects and code are reproducible and production ready from the outset.

Why Reproducible Data Science?

Reproducible data science projects are those that allow others to recreate and build upon your analysis as well as easily reuse and modify your code. In business, reproducible data science is important for a number of reasons:

- It's not uncommon for business stakeholders to request what a data scientist thought was a one-off analysis, be repeated with different parameters. If your code is not easily adapted this will prevent you from meeting your stakeholders' new requirements within a reasonable timeframe.
- If a data science project gets good traction with your stakeholders, it will need to be productionised at scale. In most companies this means handing over your project to an engineering team to implement. Well documented production ready code will make this transition much smoother.
- Reproducibility builds trust, stakeholders are more likely to trust a model if they can walk through the analysis themselves. Well tested code is also more accurate and less likely to contain obvious programming mistakes.
- Reproducibility allows for knowledge sharing amongst data scientists and aspiring data scientists at your company. Good documentation allows others to understand the data science techniques used and reproducible code allows them to build on and reuse parts of your team's project.
- And finally, it will make your life as a data scientist much less frustrating, making you much happier and much more productive.

Below are some rules we have learnt through our experience in delivering data science projects, many of which are borrowed from the software engineering

domain. These are intended to introduce you to each of the concepts, without plunging into any individual techniques in too much detail. Hopefully this will help you think about how best to generate more reproducible data science projects in your team.

Rules for Reproducible Data Science

Use Version Control

Use a version control system such as GitHub or GitLab, to provide a remote backup of your codebase, track changes in your code and collaborate effectively as a team. Try to use git best practises, frequently committing small changes that solve a specific problem.

Even if you are working alone use a branching workflow such as Git Flow. Avoid working directly on the master branch, this is for production ready code only and most development branches will be merged to master once they are ready.

Where possible consider implementing a code review process, ensuring new code is reviewed by at least one colleague. The goal of the code review is to help catch any errors and improve the quality of the code committed to your codebase. This is generally performed before merging to a master branch and after any tests or continuous integration has been run (more on this below). Even if you are working on a project alone, it can be worth asking a colleague to have a look over your code from time to time.

Agree a Common Project Structure

Agree a common project structure for all your team's data science projects. This will enable collaboration as everyone will be familiar with where things are and aid project reproducibility.

If your team doesn't already have its own project structure, consider using tools such as Cookiecutter to generate a standard data science project folder structure for you.

If a specific projects requirements mean you need to use a different structure than your team normally uses, document the new structure in your repositories README.md file.

Use Virtual Environments

Use conda or Python's built in venv environments to keep track of your projects dependencies and Python version information. This will avoid dependency version conflicts between your projects, stopping the base development environment from becoming bloated and unmanageable.

Once your environment is fully set up, you can create an environments.yml or requirements.txt file to capture and share your projects dependencies. This allows others to quickly and easily run your code, as they can automatically install your project's dependencies in their environment. Helping avoid having to hunt down the specific package versions and libraries that your project relies on.

Clearly Document Everything

Clearly documenting your projects and code will save you time if you have to revisit the project at a later date. It will also make it far easier for others to use your code or follow and build on your analysis.

At a minimum include a README.md file at the root level of your repository. The contents will vary between projects but should include a description of the project and an overview of the methodology and techniques used.

You may want to create separate in-depth documentation for data scientists describing the statistics and techniques used. If your project produces code as a key component, consider including separate in depth API documentation. This will help others get started without having to look through your code and work it out for themselves.

The code itself should be written in a clear, self-documenting fashion, using descriptive variable names instead of names like x, y or data.

Including comments can be important for explaining sections of complicated code and can particularly useful in data science. As it is often necessary to communicate why and how you are using a certain algorithm or technique.

Include docstrings in your functions and classes, these should contain a quick description of what each function does and why. Generally, this includes a description and data types of their parameters and returned output. Your team will need to agree on a docstring style and use it for all of your projects, generally

we like to use Google style docstrings. Consider using a Python documentation tool such as Sphinx to automatically generate API documentation in HTML format from your codes docstrings.

Use Jupyter Notebooks Wisely

Jupyter Notebooks are fantastic for exploring your data, creating reproducible analysis and are an answer the broader scientific reproducibility crisis. They allow data scientists to include their original code and interactive visuals alongside a detailed research or analysis output. This allows others to not only understand your analysis but also the story of how you got there, allowing the reader to interact directly with the data and insights.

As Jupyter Notebooks allow out of order execution be careful when sharing them, check cells are in the correct execution order and that none are missing. Also make sure all dependencies are imported, ideally at the top of the notebook, try running the notebook all the way through to ensure nothing has broken. Finally, even though your code is executed by running cells individually, it is good practise to use functions to avoid a crowded and confused global namespace.

Unfortunately however Jupyter Notebooks have one big weakness, in their current form they are a poor tool for creating reproducible code. It is very difficult to work collaboratively on a notebook using version control, leading to logic being duplicated across team members notebooks. Notebook code can also be hard to test and integrate with continuous integration tools. Jupyter also lacks the features of a fully-fledged IDE such as automatic linting, error detection and usage checks.

With this in mind consider moving your core logic out of your Jupyter Notebooks and into separate importable Python module files. This will enable the sharing of code across your team, avoiding duplicate and slightly edited versions of core data science code being scattered across your teams' notebooks. Code quality will also improve as you can easily collaborate, run tests and conduct code reviews on your shared modules.

Keep Your Code Stylish

Agree coding standards and in general try to write Pythonic code in line with Python's PEP8 style guide. Using a fully featured IDE such as PyCharmor Visual

Studio Code with built in linting, will highlight any poorly styled code and help identify and syntactic errors in your code.

Using an automatic code formatter such as Black will ensure that the code in your teams projects has a consistent style, improving readability. This will also improve the quality of code reviews as diffs will be smaller and there will be less squabbles over code style, allowing reviewers to focus on the code quality instead.

Black Playground: left original code, right Black formatted code.

Test Your Code

Use a unit testing framework such as PyTest to catch any unexpected errors and test that your logic executes as expected. Where appropriate consider using Test Driven Development, this will ensure your code is error free and satisfies your requirements as you write it.

It is also a good idea to use a tool such as Coverage to measure the proportion of your code covered by your unit tests. Ideally you want your code coverage to be as close to 100% as possible, to ensure most of your code is fully tested. Python IDEs such as PyCharm have built in testing and coverage support, even automatically highlighting which lines of code are covered by your tests.

If each function in your codebase has been well tested, this improves reusability. Making it much less likely for coding errors to affect your current and future analysis results. Having a good suite of tests also ensures that you don't break anything if you need to edit or add features to your code, for example to meet a stakeholder's changing requirements. Tests reduce technical debt and make it easier for those unfamiliar with the project to work with your codebase.

Use Continuous Integration

Consider using continuous integration tools such as Travis CI or Circle CI, to automatically test your code when merged to your master branch. Not only does this prevent broken code from reaching master, it also simplifies the code review process. You can even use Black with a pre-commit hook to automatically format committed code, removing any debates over code style from the review process and ensuring a standard code style across your repositories.

On merge testing and coverage with Travis CI.

Sharing Data & Models

Unlike software engineering, data science projects produce more than just code; there are the test and training datasets, intermediate products and of course the models themselves. Version control is particularly important given the typically iterative process of finding the optimal model, allowing you to go back and tweak older models.

For projects with reasonably small datasets and model sizes, you may get away with using the same system used to version control your code for your data and models. However for projects with larger files or many iterations this may not be possible, for example GitHub has a number of size restrictions on its repositories including a max file size of 100 MB.

For larger or more complex projects consider using a cloud storage solution such as AWS S3, Azure Blob or locally hosted network storage to store your model and data. This can be combined with DVC a version control system designed to effectively version control the output of machine learning projects, without pushing your large data and model files to GIT.

Try to avoid manual data manipulation in your project, this can be invisible to others unless you document the exact process and therefore impossible for them to reproduce. Take care also to use relative paths in your code when working with local datasets and consider using modules such as pathlib for platform agnostic file paths.

Data Pipeline Management

Try to make your data pipeline code modular, breaking your pipeline into modules for each discrete process and unit testing each of them. For larger more complex pipelines consider using a workflow management tool such as Spotify's Luigi or Apache Airflow to execute your Python modules as chained batch jobs in a directed acyclic graph. This will make your pipeline more scalable and handle failures, dependency resolution and visualization.

Although not all these rules may apply to your data science projects, I hope this article has contained some useful ideas and has inspired you to think about how to improve the reproducibility of your data science projects.

Thanks for reading!

Written by: Justin Boylan-Toomey – Data Scientist at Sword Venture

Want to read more that Justin has written follow him here:

https://towardsdatascience.com/@justinboylantoomey

https://medium.com/@justinboylantoomey

Home