**Data analytics principles at an AI startup**

Rules for a scientific approach to data analysis and machine learning



Photo by [Tim Gouw](https://unsplash.com/@punttim?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com/?utm_source=medium&utm_medium=referral" \t "_blank)

Working for a health startup in the SaaS space, my role as a data scientist is largely concerned with generating reproducible analytics and constructing informative machine learning models. Whilst this seems to be a fairly standard role, I think that there’s more tension in how to approach it than some outsiders (and even data scientists themselves) might think.

Data science often swings between two extremes: ensuring that the output of the work is polished, or focusing on the code which generates that output. Unfortunately, this imbalanced approach leads to problems on both sides. Only thinking about the end product can reduce correctness of code and reproducibility, and cause generate significant technical debt. Conversely, pedantically writing code like a software developer can reduce the agility of a data science team, and prevent them from quickly iterating on experiments to find “good-enough” solutions. Here, I aim to outline the process used by my company’s data science team to strike an appropriate balance between the two extremes.

**Project templates reduce overhead**

The people at [Cookiecutter data science](https://drivendata.github.io/cookiecutter-data-science/" \t "_blank) argue that our work should be both correct and reproducible. Striking this balance is difficult, and to this end they advocate for using a standard template across projects, which can be modified as needed. The main arguments for this are:

* Raw data should be immutable, and hence data should be stored in a structured way that means you only process raw data, and don’t overwrite it.
* Whilst notebooks are great for exploration, they’re not so good at reproducing analyses. To this end, notebooks should be separated from scripts. This creates a symbiotic and self-sustaining relationship: code from notebooks can be refactored into scripts, and scripts can be used to enable further analysis and exploration in notebooks.
* Structure allows easily reproducible computational environments.

This organisational structure also lends itself to documentation. When each project has the same structure, people know where to look for code. They spend much less time figuring out how a repo is laid out, which is tiring and eats away at effort that could be devoted to contributions.

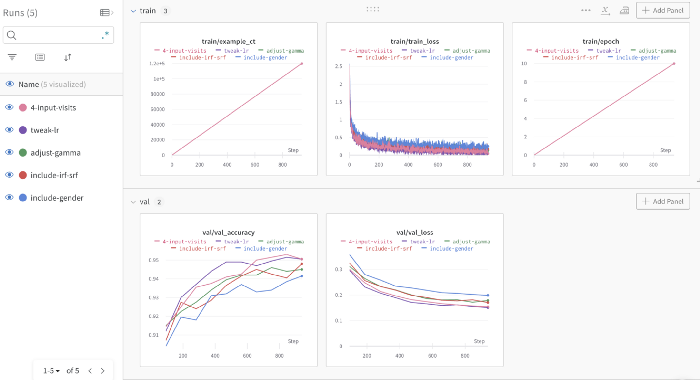


The README for one of my projects using the Cookiecutter template (image by author).

**Use an experiment tracker**

Most people initially struggle to get used to the idea of version control for machine learning models. I know I did. When I first started applying deep learning to problems at work, most of my results were saved haphazardly. I would have lists of CSVs with names like *model\_run\_3\_tweak\_lr.csv*, and another whole list of different datasets with different feature engineering. Things got very messy very quickly.

Fortunately, I discovered [Weights and Biases](https://wandb.ai/home" \t "_blank), which allows teams to iterate quickly over different models, continuously evaluate against standard metrics, and perform hyperparameter tuning. This leads to better reproducibility and a better understanding of the process that has gone into producing a model. I relied on WandB heavily when writing a paper that involved predicting vision over long periods of time, simply because of the number of hyperparameters I had to tune.



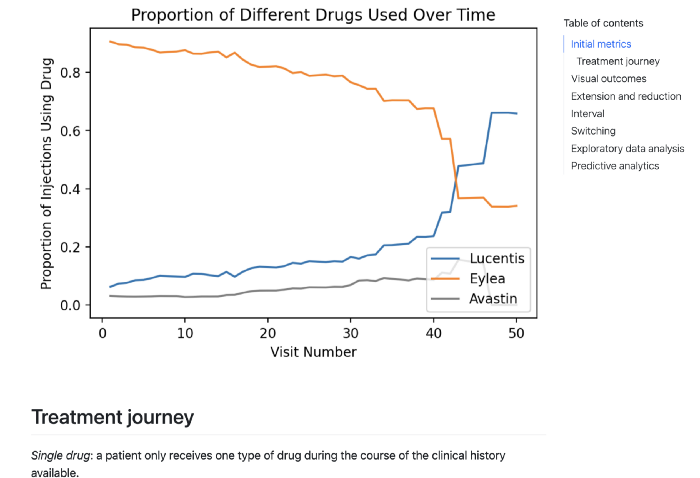
An example WandB dashboard I used to track experiments when predicting patient vision (image by author).

**Use the most accessible format, not the trendiest**

One of the problems I’ve faced at work for a while is how to present retrospective and live data to customers of our product. The two types of data are stored in different ways. Previously I was juggling many different methods of condensing analytics, such as Plotly dashboards, Metabase pages and plain copy-paste to Google Docs. This, as you might guess, was not tenable.

I got really excited one day when I wrote a script to take in a customer’s retrospective data, produce a huge number of CSVs with various sorts of results, ship these CSVs over to an R-script which would produce some nice ggplot figures for me, and then back to yet another script which collected all of these and dumped them to a folder in Google Drive. I thought it was nifty, but I realised too late that this didn’t have much utility for our company. One of the analytics people still had to go and move everything from the folder into a nice document.

After trying a number of other solutions, I settled on a very simple one. I’d still have my analytics scripts, but I’d call them from a notebook (which itself was copied from a template). After ensuring that the customer’s data was in the appropriate format, I’d type the file path into the notebook, and this would generate everything I needed, in the notebook itself (figures, tables etc.). I’d insert some comments, and use Quarto to convert the Jupyter notebook to an interactive HTML document, and hide the code. My first retrospective report at the company took me about 3 months. Now, I can generate retrospective, standardised data reports in less than five minutes. Keep it simple.



A sample of an anonymised HTML report generated using Quarto (image by author).

**Admit when a solution isn’t working**

A perfect example of this came a few months ago when I was training an NLP model to extract customer data from free text. The original model using dummy data was very good: after fine-tuning a DistilBERT question-answering model from HuggingFace, I got above 90 in both F-score and exact match across all questions. However, as more real world data started to come in, it started to suck up my time:

* I had to label at least a few instances of data to give the model something to learn
* It was highly specific jargon, so unsupervised learning didn’t work well
* The model seemed unwilling to learn all the outliers present in the data

I ended up getting fairly poor results, even after over a month of hard work. I decided to go old-school and write a bunch of hard-coded rules to extract our information. This was a much better solution, and I wish I’d just done it sooner rather than getting enticed by the allure of large language models and transfer learning.

**Conclusion**

Obviously, these aren’t all-encompassing rules, but are simply good guidelines that I’ve picked up on over the last few years. I’m sure I’ll continue to add to them as I write more spaghetti code, apply larger neural networks to basic problems, and fumble my way through future projects. But that’s how you learn!