# Production Data Science

## A workflow for collaborative data science aimed at production

[Filippo Bovo](https://towardsdatascience.com/@filippo.bovo?source=post_page-----3a42dee5208----------------------)

[Jul 18, 2018](https://towardsdatascience.com/production-data-science-3a42dee5208?source=post_page-----3a42dee5208----------------------) · 5 min read

Uma imagem contendo objeto

Descrição gerada automaticamente

Last year, I was working on a collaborative data science project. The project was going well, but my collaborators and I overlooked good practices and, when exploring and modelling data, we did not keep in mind that we were ultimately building a product. These oversights surfaced towards the end of the work when we automated our best model for production.

After we completed the project, I looked for existing ways to carry out collaborative data science with an end-product in mind. I could only find a few resources on the topic and what I found focused only on specific areas, such as testing for data science. Moreover, when talking to data science students, I learned that they, as well, were not taught good coding practices or effective methodologies to collaborate with other people.

I started by looking at software development practices that could be easily applied to data science.The straightforward choice was using a Python virtual environment to ensure the reproducibility of the work, and Git and Python packaging utilities to ease the process of installing and contributing to the software. Even though I knew these practices and tools, by following online data science tutorials I got into the habit of just sharing [Jupyter notebooks](http://jupyter.org/" \t "_blank). Jupyter notebooks are good for self-contained exploratory analyses, but notebooks alone are not effective to create a product. However, I did not want to ditch notebooks, as they are a great tool, offering an interactive and non-linear playground, suitable for exploratory analysis. At that point, I had exploratory analysis on one hand and productionisation on the other, and I wanted to combine them in a simple workflow.

A common workflow in agile software development alternates the development of new features and their refactoring. This cycle allows the inclusion of new features fulfilling current users’ needs while keeping the codebase lean and stable. In data science, developing new features for users is replaced with finding insights through data exploration. This observation led to the central theme of the Production Data Science workflow: *the explore-refactor cycle*.

The explore-refactor cycle, depicted in the figure above, alternates exploration and refactoring. Exploration increases the complexity of a project by adding new insights through analyses. Refactoring decreases the complexity by tidying part of the analyses into the production codebase. In other words, the production codebase is a distilled version of the code used to obtain insights. Most importantly, insights are derived partly through code and mainly through deductive reasoning. Data scientists use code like Sherlock Holmes uses chemistry to gain evidence for his line of reasoning.

Jupyter notebooks allow us to narrate our deductions using formatted text and write supporting code in the same document. However, often textual explanations are given little weight and are shadowed by long chunks of code. So, in data science, refactoring should involve both code and the text-based reasoning. Refactoring code from a Jupyter notebook into a Python package turns deductive reasoning into the protagonist of a notebook. A notebook becomes a canvas to prove a point about the data by using text, supported by code. This resembles literate programming, where text is used to explain and justify code itself. In this view, where all details are stripped away, a notebook is the combination of text and code.

In a similar light, in [The Visual Display of Quantitative Information](https://www.edwardtufte.com/tufte/books_vdqi), Tufte defines information graphics as the combination of data-ink and non-data-ink. Data-ink is the amount of ink representing data and non-data-ink represents the rest. Because it is the data-ink that carries information, data-ink should be the protagonist of information graphics. When non-data-ink steals the scene, information dilutes in uninformative content. Tufte suggests to improve information graphics by reasonably maximising data-ink and minimising non-data-ink. Similarly, when viewing a notebook as a means for reasoning, text should be the protagonist; text should not be shadowed by code. This leads to a simple rule for refactoring notebooks: text over code.

I came across a similar idea in software development: [functionality is an asset, code is a liability](https://twitter.com/kcpeppe/status/15473004648). Paraphrasing this statement, functionality is what software should offer while keeping a small codebase, because the larger the codebase, the higher the maintenance costs and the chances of having bugs. The common denominator of data-ink over non-data-ink, text over code, and functionality over code, is to work with other people in mind, that is, to care about the experience that people have when going through our work.

Similarly, the creator of Python, Guido van Rossum, noted that [code is read much more often than it is written](https://www.python.org/dev/peps/pep-0008/). Indeed, Python’s design emphasises readability. Ignoring readability, we can save an hour by not cleaning up our code, while each collaborator may lose two hours trying to understand it. If we collaborate with three people, one hour is saved and six hours may be wasted in frustration. So, if everyone does that, everyone loses. Instead, if everyone works with other people in mind, everyone wins. Moreover, when talking about other people, I do not only refer to our collaborators, but also to our future-self. The reason is that in a few months we are likely to forget the details of what we are doing now, and we will be in a similar position to that of our collaborators. For these reasons, the following principle sets the theme throughout the [Production Data Science](https://github.com/Satalia/production-data-science) workflow: *make life easier for other people and your future-self*.

Easing other people’s lives and the explore-refactor cycle are the essence of the [Production Data Science](https://github.com/Satalia/production-data-science) workflow. In this workflow, we start by [setting up a project](https://github.com/Satalia/production-data-science/tree/master/tutorial/a-setup) with a structure that emphasises collaboration and harmonises exploration with production. With this structure, we move into the first phase of the explore-refactor cycle: [exploration](https://github.com/Satalia/production-data-science/tree/master/tutorial/c-explore). Here, we use the [Jupyter notebook](http://jupyter.org/" \t "_blank) to analyse the data, form hypotheses, test them, and use the acquired knowledge to build predictive models. Once we finish an analysis, we [refactor](https://github.com/Satalia/production-data-science/tree/master/tutorial/d-refactor) the main parts of the notebook guided by the text-over-code rule. The code flows from the notebook to the production codebase and the line of reasoning becomes the protagonist of the notebook. [Exploration and refactoring are then iterated](https://github.com/Satalia/production-data-science/tree/master/tutorial/e-iterate_to_product) until we reach the end-product.