**This Quick and Easy 7-Step Checklist Will Help You Write Better Python Code for Data Science**

**This checklist will help you write clean and easy-to-manage Python code**



Photo by [Austin Lowman](https://unsplash.com/@shutter_hunter?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com?utm_source=medium&utm_medium=referral)

I never really understood why software engineers or data scientists from computer science backgrounds would complain about the code that data scientists write until I began reading the code…

Then, I discovered the kryptonite of the data scientist.

[“Data science isn’t a field that naturally stems from computer science”](https://towardsdatascience.com/software-engineering-best-practices-for-data-scientists-4c199ede6e03), and it’s reflected in the wide variety of backgrounds that data scientists hold. Generally coming from unrelated fields, such as medicine, engineering, the arts, business, and more, data scientists rarely have computing-based backgrounds on which to support their knowledge of writing code.

More often than not, the first code that a data scientist writes is through an online course that will give them a kickstart into the field. Not only that, but most data scientists begin by learning Python, which is a language that will let a new coder get away with a whole heck of a lot.

Therefore, data scientists are often at a disadvantage when it comes to writing code that can become apparent when it’s committed to the production environment.

The [importance of writing good code](https://towardsdatascience.com/software-engineering-best-practices-for-data-scientists-4c199ede6e03) becomes apparent when you go to work for a company that doesn’t have the budget to hire both data scientists and software engineers. In those scenarios, data scientists are responsible for creating production-ready code. Therefore, you must develop a system to help you write clean, easy-to-manage code.

Whether you’re a new data scientist just starting out (it’s always a good idea to start good habits early!) or an old hand who could do this in their sleep (it’s never a bad idea to knock the rust off once in a while), this checklist will help you write better Python code for data science.

**1. Does your syntax follow Python conventions?**

If you come from a computer science or software engineering background, you may code that you code with an accent.

Just like speaking a new language or second language, you’ll always speak with an accent that makes it just clear enough to a listener that it’s not your primary language.

The same thing can happen when writing code in a language that you’ve just picked up.

However, if you begin writing code using an accent and conventions from a different programming language you know well, it’s possible that your code [won’t necessarily understand what you’re asking it to do or it will do what you’re asking in a very inefficient manner](https://towardsdatascience.com/how-to-become-fluent-in-multiple-programming-languages-9f473c146b90).

In short, use Python in the way it was intended and code with it using its proper conventions and patterns.

As an extension of using proper Python syntax and conventions, you want to make sure that you’re using pre-existing libraries. Libraries such as NumPy, Pandas, Matplotlib, TensorFlow, Seaborn, SciPy, Scikit-Learn, and more, are debugged and production-ready. Furthermore, they’re becoming the standard by which many companies work, which means your team will greatly benefit if you begin using their preferred libraries.

**2. Is the code well designed?**

The first couple of years you work as a data scientist, your main goal is probably to write code that just works. Planning and design may not be concepts that cross your mind when your main goal is to produce something that runs and doesn’t break the company-wide system.

At a certain point, the design will become important to you as you begin to expand the reach of your code and begin to prepare it for the production environment.

At this point, you’ll be wanting to focus on its planning and design, how it will work with existing code, and whether or not it's organized in terms of the placement of its components.

You’ll know you’ve written well-designed code when you hear fewer swears coming from the engineering department, or when you know that the swears aren’t for you.

Some [design parameters](https://www.toptal.com/python/python-design-patterns) to focus your mind around when preparing your code are iterators and generators (which are nicely built into Python), the chain of responsibility (*every piece of code must do one, and only one, thing*), and the command pattern.

**3. Is your code readable?**

Is your code well-organized?

Is it clear and concise?

Are [variables and functions given meaningful and style guide-compliant names](https://deepsource.io/blog/python-code-review-checklist/)?

These are the questions you need to ask yourself before you publish your code.

Python is notorious for giving its users the flexibility to write spaghetti code that has little to no clarity, conciseness, or organization.

Despite its flexibility, strict style conventions must be followed to ensure that you can understand your code after a weekend away and that a new recruit can look at your code five years later and successfully refactor it.

The style guide is your best friend. Use it, and refresh yourself on its contents once in a while.

**[PEP 8 - Style Guide for Python Code](https://peps.python.org/pep-0008/" \t "_blank)**

**[This document gives coding conventions for the Python code comprising the standard library in the main Python…](https://peps.python.org/pep-0008/" \t "_blank)**

[peps.python.org](https://peps.python.org/pep-0008/" \t "_blank)

**4. Have you written code documentation?**

Your code documentation should involve two pieces:

1. Comments within the code using proper Python comment conventions.
2. Supporting documentation that elaborates on the comments.

Each new code commit should include at a minimum these [three types of comments](https://towardsdatascience.com/software-engineering-best-practices-for-data-scientists-4c199ede6e03):

1. A comment at the top of the newly committed code that gives a brief description of the goal of the code and its functionality.
2. A comment at the top of each function describing its inputs, outputs, and what logic it's performing.
3. A comment at the top of any logic or practical one-liners that aren’t necessarily easily understood so that your thoughts as the author can be organized and better understood by those looking at your code from the outside.

Supporting code documentation should be written anytime you write a substantial piece of code that will have major implications once added to the production code. This [external document should include](https://towardsdatascience.com/software-engineering-best-practices-for-data-scientists-4c199ede6e03):

* The goal and functionality of the code.
* Instructions on how to integrate your code (if necessary).
* An explanation of tricky or hard-to-understand parts of your code in long form where you take the time to describe exactly what the code does line-by-line, perhaps with a brief description of why you chose to write the code the way you did.
* Screenshots to help with debugging and troubleshooting.
* Helpful links to external supporting documentation.
* Your contact information if additional information or help is required.

**[How to Write Good Code Documentation for Data Scientists](https://towardsdatascience.com/how-to-write-good-code-documentation-for-data-scientists-c9940aebb4f0" \t "_blank)**

**[A crash course on the best practices you need to ensure that everyone understands the code you write.](https://towardsdatascience.com/how-to-write-good-code-documentation-for-data-scientists-c9940aebb4f0" \t "_blank)**

[towardsdatascience.com](https://towardsdatascience.com/how-to-write-good-code-documentation-for-data-scientists-c9940aebb4f0" \t "_blank)

**5. Has your code been optimized for the production environment?**

Data scientists are becoming increasingly responsible for making their code production ready.

This means that you need to ask yourself these questions before you commit your code:

1. Does your code scale as per its need?
2. Does your code contain profilers for reporting metrics?
3. Has your code been optimized to run at maximum efficiency?

These questions can all be answered by [keeping a few Python principles in mind](https://betterprogramming.pub/10-ways-to-improve-your-python-applications-performance-78e697e8dc2a):

* Use a profiler that delivers a constant stream of statistics and information concerning code failures and code performance — this takes the guesswork out of determining whether your code has been optimized.
* Keep your code light by only assigning and declaring the variables and functions you need to make your code work — abusing variables and functions requires unnecessary memory and CPU resources.
* Avoid loops when you can to put less burden on servers.
* [Write efficient code](https://stackify.com/20-simple-python-performance-tuning-tips/) by using alternative functions that provide improved performance, such as xrange() and join() , avoiding global variables, remembering to use multiple assignment, and using sets and unions.

**6. Have you kept it simple, stupid?**

Writing a one-liner solution to a problem is a [great party trick](https://towardsdatascience.com/software-engineering-best-practices-for-data-scientists-4c199ede6e03) if you want to impress your friends or if you want to publish a viral article on Medium.

However, you’re [opening yourself up to a world of hurt](https://towardsdatascience.com/software-engineering-best-practices-for-data-scientists-4c199ede6e03) when it comes time to debug and refactor your code.

The key here is to remember the old acronym that you probably lived by when you were first learning how to code:

**K**eep **I**t **S**imple **S**tupid.

This acronym is a great reminder that no matter how impressive your one-liner solution is to predicting the stock market, it’s not the most simple or practical solution.

Unnecessary complexity should have no place in your code, especially when it comes to a language that is already easy to keep simple. Don’t become wooed by the flexibility that Python gives you. Instead, keep your code simple, easy to read, and most importantly, easy to debug.

**7. Is your commit message well-written?**

There’s no denying that as a project drags on, your commit messages are going to get less and less informative.

Everyone has fallen into this trap at some point or another where the day has been long, the code has been flowing slowly, and the commit messages begin to look more and more like a toddler got ahold of your keyboard.

Then the next day rolls around and you find yourself with a pile of illegible commit messages that you have to sort through when you eventually find a bug that needs fixing.

The trick here is to ask yourself these two questions before submitting a commit:

1. Would you be able to understand the commit message the day after a long weekend away from work?
2. Does your commit message follow a format standard?

That’s all there is to writing good commit messages!

[**Pro Tip:**](https://deepsource.io/blog/python-code-review-checklist/)Use a commit message format that allows you to instantly understand what a commit involves. For example, use a prefix label or an emoji (using [gitmoji](https://gitmoji.dev/)) to represent the type of commit, such as:

* [FEATURE] or ✨This commit is adding a new feature to the code.
* [BUG] or 🐛 This commit is notifying of/fixing a bug.

**Key takeaways**

* Writing good Python code will increase your impact as a data scientist because you’ll be able to write production-ready code that can integrate into existing systems seamlessly with little fuss from the engineering department.
* Good Python code is kept simple and easy to read by paying special attention to planning its design to center around organization principles. Well-organized code is naturally optimized for the production environment.
* Your code should always be accompanied by a long-form version of your thought process, through comments, documentation, and commit messages, to ensure that your code can always be understood by yourself and outside users.

Subscribe to get my stories sent directly to your inbox: [Story Subscription](https://madison13.medium.com/subscribe)

Please become a member to get unlimited access to Medium using my referral link (I will receive a small commission at no extra cost to you): [Medium Membership](https://madison13.medium.com/membership)