**How to Write High-Quality Python as a Data Scientist**

**A Concrete Set of Skills You Should Learn!**



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

**Overview of Your Journey**

1. [Why High-Quality Code is Important](https://towardsdatascience.com/how-to-write-high-quality-python-as-a-data-scientist-cde99f582675#bada)
2. [You are Nothing Without Style — PEP8](https://towardsdatascience.com/how-to-write-high-quality-python-as-a-data-scientist-cde99f582675#6430)
3. [Variable Names — More Difficult Than it Sounds!](https://towardsdatascience.com/how-to-write-high-quality-python-as-a-data-scientist-cde99f582675#7e18)
4. [Modularize Your Code](https://towardsdatascience.com/how-to-write-high-quality-python-as-a-data-scientist-cde99f582675#b99c)
5. [Design Principles? Never Heard of it!](https://towardsdatascience.com/how-to-write-high-quality-python-as-a-data-scientist-cde99f582675#3626)
6. [Further Resouces for Code Quality](https://towardsdatascience.com/how-to-write-high-quality-python-as-a-data-scientist-cde99f582675#f712)
7. [Wrapping Up](https://towardsdatascience.com/how-to-write-high-quality-python-as-a-data-scientist-cde99f582675#d56d)

**1 — Why High-Quality Code is Important**

When starting out as a data scientist one is told conflicting tales regarding code quality. Some people say that code quality is really important. Others say that data scientists are not software engineers and adopt the following mantra:

Who cares? If it works it works, right?

When presented with the option to care about code quality or not, it is tempting to choose the path of least resistance. Learning to write high-quality code takes time and effort. Why not simply disregard code quality and have one less thing to worry about 😌

Well, as you probably suspected by the title of this blog post, I am here to tell you that **code quality for data scientists do in fact matter**. Quite a bit actually! Here are three situations where high code quality can save you countless hours:

* With bad code quality, it is easy for errors and questionable edge cases to go unnoticed. This leads later down the road to time-consuming bug fixes and, at worst, production failures. High-quality code allows you to **fail early and fail fast**.
* When newcomers arrive at a project, they have to learn the ropes of how the codebase works. This is true in data science as well as in software engineering. When the code is haphazardly written the onboarding process slows down. Certain parts of the code will be mystical and not really understandable for anyone else than the person that wrote it. And even for the person that wrote the code, it might be almost nonsensical a few months down the road. High-quality code is **understandable in the future**.
* Say you have a data science project that is widely successful. You are asked to scale out the project to meet increasing demands. The data volume and data velocity going through the data pipeline will be higher. If your code is awful, then this might not be as simple as it seems. When code is of high quality, it should be possible to scale out to new situations more smoothly. High-quality code is **extendable to new situations**.

Hopefully, you have now been convinced that high-quality code is useful. However, learning to write high-quality code is an incremental process. Your code quality will not change significantly between today and tomorrow. That’s OK. If you try to improve then the fruits of your labour will be visible for everyone to see in the long run.

I suggest that you adopt the mindset that high-quality code is something you take pride in. Almost every exceptional writer takes pride in their writing. Similarly, as a data scientist, you should take pride in your code. It is something that you have created. It is something that delivers value. As such, the code should be of high quality. Simple as that.

In this blog post, I will show get you started with some pointers for writing high-quality code in Python. Think of the blog post as a beginning rather than a complete picture. In the end, I will give you a few cool resources that can help you further on your road to code awesomeness 🔥

**2 — You are Nothing Without Style — PEP8**

**The Problem**

In most programming languages, there are many ways to solve the same problem. This encourages creativity and novel solutions to problems. However, there is also a seemingly infinite amount of ways to do the following:

* Choose whether to have spaces between parameters in function definitions. Should you write a function as def my\_function(a, b) or def my\_function(a,b) or perhaps def my\_function( a, b ).
* How should you write the function name? Perhaps in a single word def myfunction(a, b)? Perhaps in **snake-case** def my\_function(a, b)? Or maybe in **camel-case** def myFunction(a, b)? Or maybe **Pascal-case** def MyFunction(a, b)?

For every line of code, there are considerations like this. If you act randomly with regard to which choices you do, your code becomes a mess like this:

def MyFUNCTION( a,b) :  
 return a+b

Yuck! So what, then? Should your company hold internal meetings where you go through how each case should be handled? Sounds awfully boring.

**The Solution**

Fear not! In Python there is PEP8. This is the officially suggested convention for code style. For a slick-looking summary of PEP8, you can check out [Kenneth Reitz’s stylish version](https://pep8.org/).

Every Python developer should (unless your company has their own style guide) adhere to PEP8. This might seem initially like a hassle. On the contrary, once you have accepted PEP8 as the authority for most stylistic decisions, you can focus your energy on other aspects of your code. You now don’t need to think about the correct stylistic version of the code snippet above. The correct formatting is:

def my\_function(a, b):  
 return a + b

There are still plenty of decisions left up to the author of the code. Should you use [type hints](https://towardsdatascience.com/modernize-your-sinful-python-code-with-beautiful-type-hints-4e72e98f6bf1) to indicate the data types of the parameters? You should probably write a docstring to explain what the function does. How to implement the logic is still 100% up to you. You simply don’t have to put energy into thinking about stylistic choices.

I recommend that you spend 10–15 minutes browsing the [PEP8 style guide](https://pep8.org/) to pick up some improvements. You can refer back to PEP8 whenever you are uncertain of a specific choice.

**Automatic Code Formatters**

There are also automatic code formatters like [Black](https://towardsdatascience.com/tired-of-pointless-discussions-on-code-formatting-a-simple-solution-exists-af11ea442bdc) that format code automatically for you. These are awesome! However, you should know that they have limitations.

As an example, it is recommended by PEP8 to end a self-defined exception class in Python with the word Error. Hence the start of a class representing a data processing error should be written as e.g.

class DataProcessingError(Exception):  
---

This is something that automatic formatters like Black will not fix for you. By all means, use auto-formatters like Black. But be sure to know most of PEP8 by heart nonetheless.

**But I Don’t Want To!**



Photo by [Matthew Henry](https://unsplash.com/@matthewhenry?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com?utm_source=medium&utm_medium=referral)

There are some readers that will claim that code formatting and code style are unimportant. If you truly don’t see the importance of PEP8 then that is perfectly valid. Just know that many others will perceive your code as poor quality if you don’t adhere to it. Badly formatted code sets expectations like mails littered with spelling mistakes.

**3 — Variable Names — More Difficult Than it Sounds!**

Naming variables, functions, classes, modules, and packages in Python is hard work. In fact, Phil Karlton has the following famous quote:

There are only two hard things in Computer Science: cache invalidation and naming things. — Phil Karlton

Luckily most data scientists don’t have to deal with [cache invalidation](https://en.wikipedia.org/wiki/Cache_invalidation). However, you are stuck with naming things. Python packages and modules should have succinct all-lowercase names. Let’s discuss how to give good names to our darlings; *variables*, *functions*, and *classes*.

**The Easy — Conventions**

There are a few easy conventions that one should always abide by.

Firstly, **never** overwrite a built-in. As an example, it might be tempting to create the following variables:

a = 5  
b = 2  
min = 0  
if a < b:  
 min = a  
else:  
 min = b

The code above overrides the build-in function min() with a variable min. Now your program will crash if you try to use the built-in min() function. To quote The Godfather, “Look how they massacred my boy!” 😧

Secondly, there are some PEP8 conventions for variables, functions, and classes you should follow. Both variable and function names should be lowercase where different words are separated by an underscore:

my\_variable = 5def my\_function(a, b):  
 return a + b

If a variable is intended to be constant throughout the program, then you can write it in uppercase to indicate this:

MY\_NONCHANGING\_VARIABLE = 5

For classes, their names should be written with capitalized words joined together as follows:

class MyAwesomeClass():  
---

The specific choices of variable names above are not very informative. This is the hard part of naming things that we now turn to.

**The Hard—Subjectivity**

The hard part of naming things is not following the conventions described above. The hard part is choosing names that indicate in a succinct manner what the entity is (for variables and classes) or what it does (for functions). Here are some guidelines (applied to variables) I want to emphasize:

* **Pick the length carefully:** Variable names should not be too short or too long. With too short variable names you end up using abbreviations that are not understandable to everyone. Can you tell what the variables c = 7 or vprice = 5 stands for? Out of context, it is obvious how bad these variable names are. But they can be expanded to be too long as well by writing: count\_of\_how\_many\_steps\_our\_pipeline\_has = 7 and variance\_of\_the\_column\_price\_in\_our\_dataset = 5. Now it is understandable, but a nightmare to read. Try to find a fitting middle ground: How about pipeline\_steps = 7 and variance\_price = 7?
* **Don’t mention the datatype:** It is often tempting to sneak the datatype of your variable into the name. Many of us have been guilty of calling a Pandas dataframe df\_new. The idea is that, by sneaking in df as an abbreviation for a **dataframe,** you can indicate to the reader that this is a dataframe. This approach is called [Hungarian notation](https://en.wikipedia.org/wiki/Hungarian_notation) and is largely discouraged. In Python you can easily check the type of something with the type() function. If you are using type hints as well, the Hungarian notation becomes completely useless. I would suggest to rather spend your available “variable name milage” on something more useful.
* **Use intuitive domain language:** You have stored the variables broccoli, celery, and beets that count the number of each respective vegetable that are sold. You now need a container type (like a dictionary) to store all of these variables. What do you call the dictionary? Don’t worry, it is not a trick question. I don’t know about you, but I would call it vegatables. Use intuitive domain language as much as possible. The only exception is if you expect your code to be extended to cases where the domain knowledge no longer makes sense.

For functions, the same guidelines as above apply (please don’t embed the word “function” into your function names). The only thing worth noting is that functions typically do something rather than contain something. As such, functions are typically given action words (verbs) to emphasize what they do. Names such as prune\_tree() and recommend\_item() are great.

Hopefully, I have convinced you that spending some time naming your variables is worthwhile. There will be certain cases where breaking the above guidelines will be your best course of action. For instance, the acronym API (Application Programming Interface) is well-understood. Thus the (private!) variable with the name api\_passwordis perfectly fine. Use your own judgement!

**4 — Modularize Your Code**

Not all code is created equal. Some code is hard to extend. Some code can not easily be reused. Some code is hard to check for bugs. A way to make improvements in all of these areas is to **modularize your code**.

What does this mean in practice? Your code should be divided into functions and classes. These functions and classes should again be grouped into Python modules. Why is this done?

Consider code that is not grouped in any way (you might think of the code as “floating freely”). This is certainly convenient to write in the beginning. However, as the code grows it becomes more and more cumbersome.

You have to copy and paste code to make minor adjustments. Now you have to change the code in all the duplicated places if you want to extend it. God knows what you will do if a colleague asks for one of the features of your code — everything is completely interwoven. Don’t get me started on how to check for bugs in your code.

What started as a convenience has turned into a nightmare 😧

What is the solution? You should periodically refactor your code into functions and classes. Let’s take the case of functions. A function is a reusable piece of code. When your code is grouped logically into functions you can:

* **Easily reuse some of the code:** Since a function is an independent piece you can export that specific function to other settings. Maybe you have developed an awesome function clean\_missing\_values() for cleaning missing values in datasets. Now you can use that function in other projects as well (with minimal adjustments).
* **Easily check for bugs in your code:** Functions can be tested for bugs through unit tests. When writing Python, the most common libraries for writing unit tests are **Unittest** and (my favourite) **Pytest**.
* **Easily add documentation for more clarity:** With functions, you can add [docstrings](https://www.programiz.com/python-programming/docstrings) that explain what they do. You can also add [type hints](https://docs.python.org/3/library/typing.html) to functions that explain the data types of the inputs and outputs to other developers and data scientists. You will find that modularized code is a lot easier to understand once the complexity of projects grows.

Especially for data scientists, code is too often left in Jupyter notebook cells completely un-modularized. This makes all the above problems come alive if not fixed in the long run.

Don’t get me wrong. Jupyter Notebooks are a great tool for data exploration and testing out machine learning models. However, make sure that you set off a time to refactor the code into maintainable pieces (functions and classes). By doing this, you will save yourself much pain in the long run. Also, you will become the favourite person of the tester/DevOps person on the team, which is always a plus 😅

**5 —Design Principles? Never Heard of it!**

**You Don’t Use Design Principles?**

Usually, [**design principles**](https://www.geeksforgeeks.org/an-introduction-to-software-development-design-principles/) are something that software developers preoccupy themselves with. Many data scientists have the impression that design principles do not really concern them too much. To this claim, I would like to point to Adam Judge’s famous quote about design in general:

“The alternative to good design is always bad design. There is no such thing as no design.”–Adam Judge

Adam Judge is definitely talking about design from a visual point of view. Yet, I think the conclusion still stands for design principles when writing code. By the virtue of writing code, you are using design principles. You are perhaps just not using them well.

Let’s get you started with a few design principles to improve your general code quality!

**Cute Acronyms 😍**



Photo by [Gary Bendig](https://unsplash.com/@kris_ricepees?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com?utm_source=medium&utm_medium=referral)

Firstly, there are some cute acronyms that are getting thrown around. You should think of these as guiding principles to have in mind when writing code:

**KISS: Keep It Simple Silly** — This one is pretty straightforward. Try to reduce complexity whenever possible. Let’s take an example. Say you want to figure out how many non-square numbers there are between 1 and 1000. Recall that a non-square number n is just a number that can not be expressed as n = m\*\*2 for some other number m. The following code solves our problem:

non\_square\_numbers = len(set(range(1, 1001)).difference(set(map(lambda x: x \*\* 2, range(1, 1001)))))

Be honest. Although the code above solves the problem (even using cool features like sets and the map function), it looks horrible. It probably took you some time to read it. It is way too complicated. Consider the following simplification:

from math import sqrtnon\_square\_numbers = list(range(1, 1001))for n in range(1, int(sqrt(1001))):  
 non\_square\_numbers.remove(n \*\* 2)

The new code is not only clearer but also faster. I only need to run through the numbers up to the square root of 1001 since I am squaring numbers. The lesson from KISS is that sometimes it is worthwhile to make your code a bit longer if it makes it easier to understand.

**DRY: Don’t Repeat Yourself** — This principle encourages modularizing your code into functions. By doing this, you avoid code duplication. In fact, whenever you find yourself duplication code, you should probably modularize your code.

**YAGNI: You Ain’t Gonna Need It** — This principle encourages that you don’t over-engineer solutions. Let’s do a concrete example.

Say that you are writing a data pipeline where the first step is a function that reads from a SQL database. Maybe in the future, you will also need to read CSV files. So you implement that as well. And maybe in the future, you will also need to read XML. Or JSON. Or Parquet files.

The YAGNI principle in this case is simple. Don’t. You will probably not need all the extensions and you are writing code that never will be used. Instead, make sure that your code is possible to extend. But save the actual extension for when you actually have a concrete need to do so.

By following the YAGNI principle, your codebase will be smaller and there will be fewer loose threads that you need to worry about.

**The SOLID Design Principles**

A well-established group of design principles (especially in OOP) are the [SOLID principles](http://d-the-first-five-principles-of-object-oriented-design). They form a well-tested group of design principles that have been used a lot in the previous decades. The five design principles in SOLID are:

* **S** — [Single Responsibility Principle](https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design#single-responsibility-principle)
* **O** — [Open-Closed Principle](https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design#open-closed-principle)
* **L** — [Liskov Substitution Principle](https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design#liskov-substitution-principle)
* **I** — [Interface Segregation Principle](https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design#interface-segregation-principle)
* **D** — [Dependency Inversion Principle](https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design#dependency-inversion-principle)

Not all of the SOLID principles are equally useful in data science. In fact, I have seldom used Liskov’s Substitution Principle in data disciplines. Don’t get me wrong, it is a great design principle and I have tremendous respect for Barbara Liskov. However, it is definitely more useful in software engineering where OOP is heavily used. I think that the most important of the SOLID principles for a data scientist is the Single Responsibility Principle.

The Single Responsibility Principle states that functions (as well as classes) should have a single responsibility. Let’s explore with an example what this means.

Say you have a function process\_data() that loads a CSV file as a Pandas Dataframe, removes missing values, selects a few of the features, and then finally plots these features. Your function process\_data() now has four responsibilities: importing data, cleaning data, feature extraction, and plotting. Why is this bad?

For starters, it is now a lot harder to export your code to other settings. You may have written top of the notch missing value cleaning code within the function process\_data(). How do you use that code in other places where, say, feature extraction is not possible? Not so simple.

It is also harder to test your code for bugs. By doing multiple different things, there are more places for bugs to hide.

The Single Responsibility Principle would suggest breaking the function process\_data() into four functions. The functions could be called import\_csv\_data(), clean\_missing\_values(), extract\_features(), and plot\_features(). In this way, each function has a single responsibility 😃

**6 — Further Resources for Code Quality**

I have given you some starting points for improving your code quality. Where should you turn to next for more information on this topic?

* **Code Style:** There is really not that much more than learning PEP8 by heart and using an effective code-formatter like [Black](https://pypi.org/project/black/) or [autopep8](https://pypi.org/project/autopep8/). If you want to make sure that you enforce this, then ask the person doing your code review for feedback on code style. You might also want to look into the Python conventions for writing proper docstrings in [PEP257](https://peps.python.org/pep-0257/).
* **Variable Names:** My best advice here is to be conscious of the variable names you choose. Other than this, I found many interesting points in Al Sweigart’s book [Beyond the Basic Stuff with Python](https://www.goodreads.com/book/show/41392896-beyond-the-basic-stuff-with-python).
* **Modularized Code:** Writing modularized code takes effort. Consider learning more about OOP (Object Oriented Programming) if you are unsure about when (and how) to use classes. I can recommend the free YouTube series [Python OOP Tutorial](https://www.youtube.com/playlist?list=PL-osiE80TeTsqhIuOqKhwlXsIBIdSeYtc) as a good starting point. Other than this, I suggest that you get familiar with every aspect of Python functions so that you can use them efficiently. If you e.g. don’t know that Python function by default returns the value None, then your code will probably reflect that lack of knowledge.
* **Design Principles:** I gave you just a small taste of design principles. There are classical books on the subject, as well as books tailored specifically towards data science. However, I cannot recommend enough the YouTube videos by [ArjanCodes](https://www.youtube.com/c/ArjanCodes). They are of extremely high quality and give a Python-specific introduction to many design principles.

**7 — Wrapping Up**



Photo by [Spencer Bergen](https://unsplash.com/@spencerbergen?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com?utm_source=medium&utm_medium=referral)

Hopefully, you are well on your way to becoming a high-quality code producer. Don’t worry if your code is not perfect overnight. Getting high code quality is a long journey that I am still working on daily.

**Like my writing?** Check out some of my other posts for more Python content:

* [Modernize Your Sinful Python Code with Beautiful Type Hints](https://towardsdatascience.com/modernize-your-sinful-python-code-with-beautiful-type-hints-4e72e98f6bf1)
* [Visualizing Missing Values in Python is Shockingly Easy](https://towardsdatascience.com/visualizing-missing-values-in-python-is-shockingly-easy-56ed5bc2e7ea)
* [Introducing Anomaly/Outlier Detection in Python with PyOD 🔥](https://towardsdatascience.com/introducing-anomaly-outlier-detection-in-python-with-pyod-40afcccee9ff)
* [5 Awesome NumPy Functions That Can Save You in a Pinch](https://towardsdatascience.com/5-awesome-numpy-functions-that-can-save-you-in-a-pinch-ba349af5ac47)
* [5 Expert Tips to Skyrocket Your Dictionary Skills in Python 🚀](https://towardsdatascience.com/5-expert-tips-to-skyrocket-your-dictionary-skills-in-python-1cf54b7d920d)

If you are interested in data science, programming, or anything in between, then feel free to add me on [LinkedIn](https://www.linkedin.com/in/eirik-berge/) and say hi ✋