**How to Write Good Code Documentation for Data Scientists**

A crash course on the best practices you need to ensure that everyone understands the code you write.



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

A data scientist writing good code documentation is a lot like an engineer making sure that the pillars supporting a bridge can bear the weight of the bridge itself and the passengers that use it.

Code documentation not only provides an idea of what the code does, but it also provides insight for other people into the coder’s thought process and why the code has to be written a certain way.

Everyone has had those experiences where they’ve been tossed an old piece of code and told to rework it, or worse, a piece of code that needs to be integrated with an existing process. In these scenarios, it’s an easy task to complete if there is good code documentation available. However, it can become an impossible task if no code documentation exists, leading to the loss of time, money, and possibly the destruction of a system. Hence the analogy that code documentation is as important as the pillars that support a load-bearing bridge.

Given that many data scientists don’t come from programming-based fields or don’t believe that code documentation is an essential practice because data scientists are often secluded from the rest of the organization, it’s completely normal that data scientists may not even write any code documentation at all.

So, to keep the peace between future data scientists working on your code and your engineering department should code integration be necessary, check out the following best practices you can use to write good code documentation.

**Write step-by-step notes that describe how your code works and the steps you need to make the code work.**

The notes that you write during the coding process will be the foundation upon which you develop your code documentation.

When working on the code initially, you want to begin by writing detailed notes that describe each step of your process.

These notes are then used to write your documentation and will offer the most accurate glimpse into your thought process. This is also important to avoid the issue that every single programmer in history has come across: coming back from a weekend away from your code and not remembering your original thought process from Friday that made complete sense back then but means absolutely nothing to you now.

These notes can be written in your code (most helpful, though do remember to clean them up when production time is near) or in a separate documentation file.

The key here is to write down absolutely everything that comes to mind when writing your code, especially things that you may deem inconsequential to mention though could end up being game-changers in its later usages. This includes items such as describing what each variable represents, what each function does, and the results the code should yield. You will also want to include notes on why the code has to be written a certain way and in what order the functions are used and called.

**Assume that the person reading your code knows nothing.**

Assume the person reading your code knows nothing about what the code does, how it works, and why it works.

Too many times I’ve come across a piece of code with little to no documentation where the person who wrote it very obviously assumed that the next person would know everything about it, from how it worked, to why it had to be written a certain way to keep from breaking, to what it even did.

Therefore, one of the best things you can do when writing your code documentation is to write it as if the next person has no clue about anything to do with your code.

This involves piecing together the notes you wrote while developing your code into a complete explanation of the code that the person will be looking at. As mentioned previously, this information could reside within the code (again, most helpful if coherent) or could reside in an external document.

Short-form notes should be included within the code, whereas long-form explanations or descriptions are better left in an external document, though personal preference or company policy should be the determining factor.

**Use coding conventions and best practices.**

It’s tricky to help people understand the code you write when the code you write doesn’t even follow proper coding conventions and best practices.

While this discussion is beyond the scope of this article (you can check out my full article on software engineering practices for data scientists [here](https://towardsdatascience.com/software-engineering-best-practices-for-data-scientists-4c199ede6e03)), the main idea is to make sure that you are writing clean, legible code that follows proper software engineering standards. This not only helps those after you to understand the code, but it also makes the code as production-ready as possible if and when that time comes.

The key ideas to focus on when it comes to [coding conventions and best practices](https://towardsdatascience.com/software-engineering-best-practices-for-data-scientists-4c199ede6e03) are to:

* Use descriptive variable names.
* Use functions.
* Use comments (as discussed above).
* Use a consistent coding style.
* Use the syntax conventions of the language you use.
* Use libraries.
* Keep your code DRY.

You get the idea.

Using coding conventions and best practices ensures that, if your code documentation is sub-par, someone who is comfortable with code will likely be able to understand what you have written and generally how it should work.

Additionally, this ensures that your code documentation makes sense because it will follow the form of your properly written code.

**Provide your contact information.**

You would be surprised at how useful it is to be able to call the author of the code that you are working with and ask them to explain how it works using small words.

Sometimes, the documentation just isn’t enough. Or perhaps the documentation isn’t particularly clear in a specific area. Or you need to confirm with the author that what they have produced works how it should.

Providing your contact information at the end of your code or documentation is a simple, yet effective way of ensuring that any questions or issues are directed to you, which can help save time, money, and frustration.

**Use flow-charts.**

Viewing a process graphically can be an extremely effective way of understanding how it works.

Flow charts are something I was taught back in college and still use to this day to ensure that I understand how my code should work. While some people can understand by reading, I understand through viewing illustrations or graphics.

These flow charts can be included in the external documentation for your code and should provide step-by-step instructions on how a function works, what it should return, and what should occur if something goes wrong or if a condition isn’t satisfied.

Not only will these graphics help the next person who uses your code, but they will also help you along the way to produce cleaner, more effective code.

**Practice describing how your code works so that a 5-year-old could understand it.**

When finalizing your code documentation, you want to ensure that you could explain how your code works to a 5-year-old. If you don’t have a 5-year-old handy to explain your code to, a cactus or a rubber duck work equally as well.

The idea behind this practice is to ensure that when someone inevitably has a question that isn’t answered by your documentation, that you can answer their question completely and succinctly. Additionally, this helps you check to see if you have missed any key points within your documentation or if you need to rework a description of a particular scenario, function, or result.

By reviewing your code documentation a few times before making it available, you ensure that you can explain your code as simply as possible through your literature and that you can explain it in more detail if questions ever arise.

**Key Takeaways**

* Throughout the coding process, write detailed notes that can be used as the foundation for your documentation.
* When writing your documentation, assume that the reader knows nothing about your code, how it works, and why it works. This will give you the best shot at writing well-rounded documentation that can be understood by anyone.
* Write your code using syntax conventions and best practices so that in the event your documentation doesn’t provide all of the answers, the next user will generally be able to understand your code.
* Provide your contact information so the next users can ask you questions directly. This saves time, money, and frustration for everyone.
* Use flow charts to provide graphical descriptions of how your code works.
* Practice explaining how your code works to a 5-year-old/cactus/rubber duck to ensure that you’ve included all key points in your documentation and so that you can provide further explanations if anyone comes back with a question.