Tips to write production-level code

1. **Keep it modular**
   1. Break the code into smaller pieces each intended to perform a specific task
   2. Group these functions into modules (or python files) based on its usability
2. **Logging and instrumentation**
   1. The main purpose of LI is to record useful information from the code during its execution to help the programmer mainly to debug if anything goes awry and also to improve the performance of the code (such as reduced execution times).
   2. Logging should be minimal containing only information that requires human attention and immediate handling
   3. Instrumentation should record all other information left out in logging that would help to validate code execution steps and work on performance improvements.
3. **Code optimization**
   1. Space and time complexity
   2. Simplify loops and recursion
4. **Unit testing**
   1. There are three stages in code cycle – development, staging and production
   2. To be able to identify different issues that may arise we need to test our code against different scenarios, different datasets, edge and corner cases.
   3. It is inefficient to carry out this process manually every time we want to test the code which would be every time we make a major change to the code. Hence opt for Unit testing which contains a set of test cases and it can be executed whenever we want to test the code.
   4. We need to debug the code and then repeat the process until all test cases are cleared off. Python has a module called unittest to implement unit testing.
5. **Compatibility with ecosystem**
   1. Your code is not going to be a standalone function or module. It will to be integrated into company’s code ecosystem and your code has to run synchronously with other parts of the ecosystem without any flaws/failures.
   2. it is like a chain, the new chain-link should lock-in with the previous and the next chain-link otherwise the process fails
   3. discuss with the relevant team about the requirements before beginning the development process
6. **Version Control**
   1. Git — a version control system is one of the best things for source code management.
   2. The process in simple terms “modify and commit”. Every time we make a change to the code, instead of saving the file with a different name, we commit the changes — meaning overwriting the old file with new changes with a key linked to it. We usually write comments every time we commit a change to the code.
7. **Readability**
   1. Appropriate variable and function names. The variable and function names should be self-explanatory. When someone reads your code it should be easy for them to find what each variable contains and what each function does, at least to some extent. Try not to exceed 30 char for variable names and 50–60 for function names.
   2. Doc string and comments
      1. Doc string — Function/class/module specific. The first few lines of text inside the function definition that describes the role of the function along with its inputs and outputs. The text should be placed between set of 3 double quotes.

def <function\_name>:

“””<docstring>”””

return <output>

* + 1. Comments - The need for comments will be considerable reduced if we give appropriate names to variables and functions — the code will be, for the most part, self explanatory.

1. **Code review**
   1. Get code review by your peers for improving your coding skill.
   2. Do not send all scripts for review at once. Learn from the mistakes of first script, revise them in the following scripts and then send for review.

Tips to make code readable from Kaggle Dr. Richard:

* **Put all your imports, import x or library(x) at the top of your notebook**. This is especially helpful for reproducing work because installing packages tends to be the biggest hassle, and getting all that work out of the way up front means that you're not blindsided by it later on.
* **Make sure to use white space and use it consistently.**I think of writing code a little like writing a paper: white space can serve like the breaks between paragraphs to help you group similar concepts together and help your readers follow the logical flow of the text.
* **Break up long lines at logical places.** For example, if you have a function that takes a lot of arguments, you can try inserting a line break after each argument. Long lines that wrap in a viewer are very difficult to read and understand. In both Python and R, I also really, *really* dislike the use of ; to write multiple statements on the same line.
* **Make your variable names logical and human readable.** If you've parsed the text of the Rubáiyát of Omar Khayyám and saved the parse trees in a variable called "x\_298", as a newcomer to the project I'll have no idea what's in that variable. Something like "rubaiyat\_parse\_trees", on the other hand, clearly tells me what's in this variable.
* **Comment your code!** Especially for research code, when your intended audience may not be as fluent in your language of choice, I err on the side of more and more explicit comments. One of my proudest technical accomplishments is the fact that one of my co-authors was actually able to learn R based on my commented research code.
* **Use consistent patterns of capitalization.** The prevailing pattern in data science tends to be lower\_case\_with\_underscores for functions and variables, but as long as you're consistent it doesn't matter too much.

Tips from Chaitu

1. Make three types of Files – general\_functions.py, specific functions (data related functions in one file, model related functions in one file) and main.py (that executes the code exactly the way you teach someone). The main.py file imports the functions from general\_function.py and specific functions files
2. Separate analysis and main code files, or do the analysis at the end of main code file
3. Use classes if possible
4. Write the code structure on paper first - First write main steps/functions, then write sub-functions within each main functions. Call these functions from relevant files.
5. Avoid using lists for large dataset as they are not scalable. Use dictionary.
6. Read codes on Github!

Reference

<https://towardsdatascience.com/how-to-write-a-production-level-code-in-data-science-5d87bd75ced>

<https://www.kaggle.com/rtatman/reproducible-research-best-practices-jupytercon>

<https://github.com/zhixuhao/unet>

In this post, I summarize some resources to create a standard data science project template and to effectively manage data science codes.

Resources:

1) <https://drivendata.github.io/cookiecutter-data-science/>

2) <https://www.kaggle.com/rtatman/reproducible-research-best-practices-jupytercon>

Three things we need to manage in a data science project

* + 1. Environment
    2. Data
    3. Code
    4. Results

Standard Data Science Project Template



How to structure your data science project

Structuring a data science project is an important step to logically proceed in your research and to effectively communicate your findings with colleagues. In this post, I summarize some resources to logically structure a data science project which will be helpful for both a newbie and a professional.

1. Create a standard folder structure
2. Organize your data and code
3. Make your code readable