1 Introduction

In the following text, I want to present four of my key coding principles. I deliberately do not merely list many small rules, but instead, I want to describe four more general concepts that guide me while coding and concisely explain why they are important to me. My exact coding conventions are more a result of these principles and of course sometimes vary between languages or projects. A universally accepted and comprehensiv style guid with precise rules which I try to adhere to in Python is for example the *PEP 8 – Style Guide for Python Code*[1] or for R the *Google's R Style Guide*[2]. The last paragraph is about the areas in which I want to improve myself in the future.

2 Coding Philosophy

Code should be modular, reusable, extendable and maintainable

This well-known principle should always be kept in mind. Especially important to me is keeping everything modular - from the structure of the project, over the scripts, to the functions - and reusable. Many data analysis code chucks are easily transferable to other projects if they are written well which can save a lot of time.

Start with a simple working implementation and go from there

I always start my projects with a relatively small dataset and the most basic functionality. While keeping the target in mind is of course essential, building the most straightforward implementation first and going from there helps me immensely tackling complex problems. This rule also applies to code optimization. In the past I often found myself optimizing code in the wrong places wasting time and introducing unnecessary complexity. In contrast, now I try to keep the code simple first and only optimize the parts with real bottlenecks that justify the added time and code.

Be aware of code smell

Code smell describes characteristics of code that indicate a deeper problem.[3] For example, duplicate code or large functions with many parameters instead of a few smaller, more modular functions. Once I learned this concept, I immediately started recognizing code smell in my programs and tried to fix it. Bad code is often error-prone and causes more work in the long run than fixing it once.

Check the code as you go

This of course ties into the last point, but especially when doing complex data analytics, it is sometimes hard to immediately notice if the code is working correctly. Everything can run and look perfectly fine but maybe a critical formula is wrong, or specific cases get mixed up. For that reason, I always try to incorporate checks into my code. Furthermore, if possible, I also try to perform some sanity checks with a calculator or another simpler approach.

3 Areas of improvement

First of all, I want to use this practice to think about my coding style repeatedly in the future. Therefore, I will try to keep and expand this as a dynamic document on my GitHub where future collaborators can quickly get an impression about my coding style. Furthermore, on a more practical side, I want to incorporate more unit testing into my code and in general increase my documentation standard. For this reason, I will also upload all my future projects to my GitHub[4] as this will force me to include well written documentation.

[1]https://www.python.org/dev/peps/pep-0008/
[2]https://google.github.io/styleguide/Rguide.xml
[3]https://martinfowler.com/bliki/CodeSmell.html
[4]https://github.com/Amglex