Good practices in Python

This page is inspired by this document.

Like R, Python is "just" a programming language, so it is possible to write Python code in a text editor and run it via a terminal. Although this is feasible, it is easier to use a code editor. While it is technically possible to use RStudio as an editor for Python, I would recommend using a specialized editor such as *PyCharm*. The *notebook* format is also very popular in Python. In this case, I would recommend the `marimo` library, which allows you to convert your `.py` files into notebooks.

In addition, it is best to make sure you are working with the latest stable version of Python that is compatible with the packages you are using (e.g., `numpy`, `matplotlib`, `sklearn`). Very often, there is a delay before the latest versions of packages are compatible with the latest version of Python. It is therefore important to share the version of the packages used with the analysis.

Regarding documentation, the standard in Python is to follow the recommendations of PEP 257. The `Sphinx` library allows you to generate documentation from *docstrings*.

A uniform syntax makes code much easier to read and understand. A style guide sets out standards for uniform syntax. The PEP 8 style guide is recommended in Python.

Line breaks and indentation

Indentation is part of Python, i.e. `if...else` and `for` blocks are defined using indentation. You can use either *tabs* or *spaces*. The choice is yours, but it is important to be consistent in your code.

Naming convention

The use of single-character variable names should be avoided. Otherwise, variables, functions, methods, packages, and modules are named with lowercase letters, numbers, and the underscore `_`. Classes and exceptions should be named with uppercase letters to separate words (`UneClasse`). Constants are in uppercase.

Code organization

As with R, when the code starts to get long, it is advantageous to split it into several files. For example, you can have one file per part of the analysis (one for data cleaning, one for analysis, one for visualization, etc.). You can also create several subfolders.

Some things to do

1. Do not make comparisons to `True`, `False`, or `None`.

```
if attr:
    print("True!")

if not attr:
    print("False!")

if attr is None:
    print("None!")
```

2. Use list comprehension when possible.

```
a = [3, 4, 5]
b = [i for i in a if i > 4]
```

3. Load a file with `with`. This ensures that the connection to the file is closed once it has been read.

```
with open("file.txt") as f:
    read(f)
```

4. Use a maximum of 80 characters per line.
5. Use parentheses to wrap long strings.