ENCORE. A practical implementation to improve reproducibility and transparency of computational research

Supplementary Information

Example of 0_README.md file. This specific file is copied from
\ENCORE\Processing\NameOfComputation\Code

Most readme files contain three sections to guide the researcher in providing relevant and sufficient documentation. The information that is asked for is not exhaustive, and additional information might need to be provided for specific projects.

The 'Explanation' section provides general information about the type of documentation that should be provided and the reason why. The 'Instruction' section gives more specific instructions to the researcher. Finally, the file ends with a 'Template' that asks the researcher to specify a minimum amount of information. Not all requested information might always be relevant. Part of the requested information may also reside in separate documents if that is more convenient.

The explanation and instructions are not exhaustive. Much more can, for example, be said about best-practices for code development. However, this provides a starting point for the researcher without being overwhelmed. In the future, we aim to add references to relevant resources (which are now partially in the main text).

CODE

Explanation:

This directory contains the code you use (and developed) as part of this project.

Clean code

Write programs for people, not computers (code should be easily read and understood). Adopting clean code practices helps to standardize and organize software code in order to enhance readability. This allows developers to concentrate on core functionality and reduce errors. Readability helps reproducibility and transparency.

A few general rules:

- *Reduce complexity. Decompose programs into functions and modules (instead of making very long scripts).*

- *Limit number of function arguments and length of functions.*
- *Be ruthless about eliminating duplication (use functions).*
- *Always search for well-maintained software libraries that do what you need, and test these libraries before relying on them.*
- *Do not comment and uncomment sections of code to control a program's behaviour. This is a recipe for making your software irreproducible.*
- *Choose a style guide—And stick with it (see for example, https://peps.python.org/pep-0008/ or https://web.stanford.edu/class/cs109l/unrestricted/resources/google-style.html)*
- *Give functions and variables meaningful names.*
- *Make dependencies and requirements explicit.*
- *Refactor as needed (process of restructuring your code without changing its interface).*
- *Write error messages that provide solutions or point to your documentation.*

Code testing

Interrogate specific and isolated coding behaviour to reduce coding errors and ensure intended functionality, especially as code increases in complexity. Describe if software tests performed and how to re-run these tests.

***Integrated development environment** (IDE)*

Consider using an IDE (integrated development environment (IDE;)e.g., PyCharm and DataSpell from JetBrains (https://www.jetbrains.com/), RStudio (https://www.rstudio.com/)

***Write comments as you code (not afterwards).** Modern integrated development environments (IDEs) will often automatically generate documentation strings as you write code, which removes the burden of having to remember to write comments. (e.g., PyCharm, DataSpell).*

Automated documentation tools

Consider the use of documentation tools to partially automate the generation of documentation. For example, using Python docstrings (https://realpython.com/documenting-python-code/) together with Sphinx https://www.sphinx-doc.or) to automatically generate documentation.

Instructions:

- *Remove all Instructions and the Explanation once you have completed the template.*

- *Level of detail: Information provided should be sufficient for someone who was not involved in the project and/or has limited knowledge about the topic, to understand and reproduce the project.*

Software documentation

Ensure that you have properly documented your code. Not all types of software documentation on this list need to be written for a single research or support project but a combination of several should be selected depending on the nature of the project. Whether more or less documentation is needed for your project will depend on its scale and complexity. For research project you need at least to write (external) source code documentation and user documentation.

- *Requirements Specification*
- *Software Design*
- ***(External) source code documentation***
  - *Place a brief explanatory comment at the start of every program.*
  - *Document the input and output of your script/functions.*
  - *Describe what the code is doing and why.*
- *Testing Requirements*
- ***End-User Instructions (including a quick-start guide)***
  - *How to install and configure the software.*
  - *Where to find its full documentation.*
  - *Include examples how to execute the code to produce certain output. Provide a simple example or test dataset.*
  - *Under what license it's released.*
- *Include a help command for command line interfaces*

Version control your documentation. *Keep your documentation inside your Git repository along with the rest of your files. In addition/alternatively 'Read the Docs' (https://readthedocs.org/) provides an approach for hosting and automatically building documentation.*

====== TEMPLATE STARTS HERE ======

What (external) documentation is available?

Documentation files: [filename; short description of content]

How did you test the code:

Script 1: [description]

Script 2: [description]

Description of manual steps:

Relation between the scripts, and user instructions for execution: