Best Practice

Md Tariqul Islam
University of Campinas, Brazil

Reproducibility in research, specifically in computational research, is getting a lot of momentum these days. The diversity of standards and tools relating to reproducible research is vast and quite impossible to portray in as single. However, the term and concept of best practices help correlate with typical reproducible research key elements and basic rules when it comes to individual works. The best practices dictate the recommended course of action, specifically what one should need to do or not do for both while preparing any work for reproducible and reperforming any original work. The motive of this best practice documentation to share my experience in the context of reproducible research-related tools utilities while performing this work. The observation will be in the formate of do's and don't's of the reproducible research-related tools associated with code development, data, documentation, workflow, and distribution key elements.

The observations illustrated as the following orders: :

1. **Workflow**
2. **Code Development**
   - *IDE*
   - *Automation*
   - *Version Control*
3. **Data Handling and Sharing**
   - *Data Hoarding and Manipulation*
   - *Data Sharing Repository*
4. **Documentation**
   - *Executable Paper*
   - *Scientific Paper*
5. **Distribution**

1 Workflow

"Think first, run later" to believe that quote. I started my work thinking about what I should do and how to achieve it. First of all, I have initiated a roadmap illustrating an initial workflow for the entire work. Since the work related to machine learning makes it a data-intensive computational research project, I designed the workflow considering three stages: 1. data acquisition; 2. data pre-processing; 3. data analysis. I took into consideration the free online diagram editing tool named **Draw.io** to design the workflow. After creating the initial workflow, I proceeded to perform the work step by step. The first stage consists of collecting initial data from a primary source (testbed) and ending the stage by generating raw data as a final result. The second stage involves processing or clearing the data produced in the first stage so that it is acceptable for use as an input to the third stage. The final stage consists of evaluating the performance of specific algorithms, visualizing the results, and drawing a scientific conclusion. Such workflow also helps me to create the directories for my project. During the execution of each of the

steps, I gradually updated my initial workflow, adding essential notes in each of the stages blocks. Thus I end up with the final workflow. Indeed I have created two workflows, three-stage workflow for original paper and four-stage workflow integrating new executable stage for the executable document..

1.1 Online Diagram Editor: Draw.io

The Draw.io is a free online graphical editor where we can develop simple to complex diagrams and flowcharts. Besides, It aligns with Google Drive and Github.

1.1.1 Do

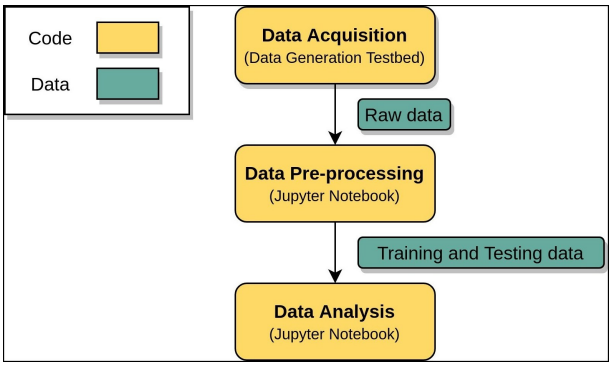
- While creating a new diagram, connect it with Google Drive account. It will help keep track of the diagram changes, and we can revert to any previous version of the diagram.
- While export diagram as JPEG try to use zoom option at least 400% to get better quality

1.1.2 Don't

- While export diagrams as JPEG or PDF don't use the grid option. Otherwise, It will create white squares in the background.

2 Code Development

Workflow organization helps me to maintain the structure of code and data elements. I have gone through three steps to developing the codes of my entire project. In the first steps, I have to build a testbed to produce raw data; next level, I developed a program to pre-process the raw data, and the last step developed a program to analyze the pre-processed data. When it comes to writing or developing code, we must first select a coding environment or an IDE based on our preference. In my case, I choose the most popular Jupyter Notebook as my IDE to write my code and execute it. After writing the codes except for data acquisition, all steps perform on the same Jupyter Notebook IDE. Simultaneous while developing codes, I used version control tool Git to keep track of all of the changes. Besides, using Jupyter Notebook as IDE and Git as version control, I have also gone through using scripting language as automation and Unix Terminal as doing some interactivity to complete all the code section steps. In the following, I will explain the reasons for using each of the tools and cover some best practices for writing code that's easy to read and use.



2.1 IDE: Jupyter Notebook

Reproducibility needs to make a runnable research code available. Donald Knuth has introduced a programming paradigm named literate programming to make codes more understandable for others. This approach involves merging code with adequate text and images at the same document. Project Jupyter, coined the term literate computing for the model used by Jupyter Notebook.

2.1.1 Do

- We can install Jupyter notebook, followed by the anaconda link, which integrates data science libraries (matplotlib, NumPy, Pandas). We can also install the Jupyter notebook following the official documentation link. In this case, we have to install other data science libraries (matplotlib, NumPy, Pandas) separately.
- Create a project at first include adequate directories according to workflow pipeline. It will ease the further process to complete the project.
- Put all imports, import x or library(x) at the top of the notebook.
- Make variable names logical and human-readable.
- Use a cell with a text explanation to comment on code. More information available in section 4.
- Keep track and record the versions of the libraries during the deducting of the work. This information will help others who wish to reproduce the same work.
- Data prepossess, and analysis code section has an element of randomness. At each step, I have used the same random seeds to control the randomness. Thus, it helps others to get the same result while reproducing the work.

2.1.2 Don't

- Be aware of the path selection while installing anaconda. Please do not use a different path as it not recommended for this work.
- The new code updates of a notebook that is already executed by another notebook will not be considered unless restarting the kernel.

2.2 Automation: Scripting and UNIX Terminal

Whenever possible, rely on the automation to execute the programs instead of manual procedures. Such manual methods are not only inefficient and error-prone, but they are also challenging to reproduce. While working at the UNIX terminal, manual execution of the program can usually replaced by using standard UNIX commands or small custom scripts. Such a script useful for streamlined automation and documentation of the research process, e.g., editing files, install the environment setup, moving input, and output between different parts of your workflow. We can describe actions in a script and run them smoothly. The most common scripting languages are Bash, Sh, Python, Powershell, MSDOS, PHP, Tcl, Perl, etc.

2.2.1 Do

- Automate the entire environment setup and emulation process by scripting. In my work, a single bash script automates my whole environment setup, and two simple python files automate the entire network emulation scenario. .

2.2.2 Don't

- Do not underestimate this automation process. Other wishes It will be tedious to reproduce the work.

2.3 Version Control: Git and Github

While developing the code of the project, it is highly recommended to use version control tools to record code changes over time and later recall a specific version. There are a bunch of version control tools. Among them, git is the most popular and convenient to use. Mainly git has four main features: 1. keep track of changes to code. 2. synchronizes code between different contributors 3. test changes to code without losing the original. 4. revert back to old versions of code. The well-known word git repository is a .git/ folder inside a project. This repository tracks all changes made to files in your project, building history over time. On the other side, Github and Gitlab websites are the most popular in the coder's community provide storing project git repositories and give access to the others to use. Here I will recommend you use Github for being the most popular and convenient for supporting Jupyter Notebook.

2.3.1 Do

- We should start using git and Github while we began to organize the workflow of your work.
- Go to the Github website and open a profile if don't have it. Next, create a new Github repository, then clone it to local machine and create the adequate directories according to workflow pipeline
- Use the git commit commands to take a snapshot of work and push back to them Github again.
- Keep doing the same whenever any changes made and want to push back to Github.
- If you do have the concept of git and GitHub, follow the link to learn basic git.
- Try to use a git tag. It helps identify specific release versions of your code.
- If you are planning to share your project and code for public access, you should use a license.

2.3.2 Don't

- Do not forget to add a README file in your Github repository. README file to your repository to tell other people why your project is useful, what they can do with your project, and how they can use it.
- Don't skip using git from the beginning of your project. Otherwise, it will be tedious to make the entire work reproducible.

3 Data Handling and Sharing

"Data sharing may not be the answer to everything, but data hoarding is the answer to nothing." Harlan Krumholz. After organizing the structure of the project, we have to get ready for data available to share. From data provenance to sharing, we should be aware of maintaining a few good practices.

3.1 Data Hoarding and Manipulation

During my work, I used python program to automate both data generation and preprocess steps. Here I have present in general good practice regarding this as follows:

3.1.1 Do

- Always avoid manual data manipulation. Always try to use a custom script or program while preprocessing data from raw data and export data set in CSV format as the final output.
- Try to maintain versioning of your data, though I did not practice this in my work correctly. I have pushed my final exported CSV file to my Github repository.
- Always work from a copy of your data. Keep a separate copy of your raw data that you never, ever touch. Work from copies of it.

3.1.2 Don't

- Don't skip the documentation on tracking history about data generation, specifically raw data provenance and data preprocessing to defend your conclusion.
- Do not store any valuable experimental data on personal devices, laptops, or computing devices used for experiments.

3.2 Data Sharing

Here is my thoughts format of Do and Don't regarding sharing data:

3.2.1 Do

- It would help if we shared the repository of the raw and processed data.
- While sharing data, we should consider the ethical, practical and legal aspects from our research perspective
- To make our shared dataset citeable.We can consider the open-access repository platform like zenodo.

3.2.2 Don't

- Don't share your data from your institutional cloud storage. Your account might no longer be available after university left. In my work, I shared my raw data from my institutional cloud storage. I should choose another open-access cloud storage platform like zenodo to avoid this kind of inconvenience.

4 Documentation

While documentation of my entire work, I took consideration two-aspect- documentation for executable paper, and documentation for a scientific paper. My initial act was to follow a 'one size fits all' approach. Using one particular method, I can document my entire work, providing the re-executing feature of my performed work and exporting the whole task as a single scientific PDF formate. Unfortunately, I did not get such a solution yet. Though Jupyter Notebook is most prominent for documentation of research work and there were many attempts tried to export the notebook as PDF formate. But these exported pdf unable to beige like actual scientific PDF format. So I split my documentation approach using two platforms: For documentation of executable paper, I have chosen Jupyter Notebook, and of scientific writing, I have chosen Overleaf, an online-based latex editor.

4.1 Executable Paper: Jupyter Notebook

I have already given a bit of insight into it in the code development section. The easy documentation structuring by the Markdown feature makes the Jupyter notebook more worthy of use. The markdown supported the following options - Headings, Blockquotes, Code section, Mathematical Symbol, Line Break, Bold and Italic Text, Horizontal Lines Ordered and Unordered List, internal and External Link, Table, Image. Thus all create readable analyses of work, including the code, images, comments, formulae, and plots.

4.1.1 Do

- Try to make documentation of your executable work in linear order for text, image, and code execution. Otherwise, it will create an issue for reproducibility.

4.1.2 Don't

- Do not need to show the entire code of your work in an executable paper notebook. If can put your codes in other directories as a separate notebook and use "%run ./pathe/file" in your executable paper notebook to load and execute corresponding code

4.2 Scientific Paper: Overleaf

A collaborative cloud-based LaTeX editor used for writing, editing, and publishing scientific documents.

4.2.1 Do

- Chosen an appropriate template (e.g., IEEE) of your work and then import it.
- Try to use the same format of adding figure, table, references according to the template. Otherwise, you will get an unwanted error.
- In this document, keep the same sequence as your executable notebook maintains with proper scientific writing rules.

4.2.2 Don't

- Don't download the pdf until you are not satisfied or it has not same sequence order of executable paper notebook

5 Distribution

Form the observation while producing other folks' work, the most challenging part is to set up the environment in a local machine. Indeed sometimes dependency hell makes the environment setup quite impossible though the exact information of software packages version was available for reproducing the work. Hence to avoid dependency hell and facilitate the execution of work, the entire computing environment with all dependencies intact must be grouped in a single package using a virtual machine or Docker. Initially, I have wrapped my whole work in a virtual machine, but Docker seems more promising in the context of the distribution of the environment.

5.1 Environment: VirtualBox

It acts as a computer inside a computer. In the subsequent part, I tried to add a few Do and Don't regarding enclosing the entire work in a virtual environment.

5.1.1 Do

- Install operating systems as VM image at the virtual box. Follow the link how to do this.
- Do your all computational work inside the VM image
- To keep the VM image lightweight, use the file-sharing option between VM and host machine. Thus, you can write and read anything from a local computer without storing it inside the VM.
- Export VM image in .ova format with giving a version number. You can track all versions of .ova formate VM image and share only the most appropriate one you think.

5.1.2 Don't

- Do not wrap your computing environment with everything. Just keep, which is enough to reproduce your result from the pre-processed dataset. VM images are prone to taking more memory, So it's awkward to share such large VM images and impracticable for others.
- Don't bind yourself to use the only virtual machine. In the context of the environmental distribution, Docker is more lightweight than a virtual machine and easy to interact with it. Learn Docker from here. I am also trying to learn it and use it in next.