Ten simple rules for writing and sharing computational analyses in Jupyter Notebooks

Rule A, Birmingham A, Zuniga C, Altintas I, Huang S-C, Knight R, et al. 2019

Irene Amato & Anisbel León

15.06.2021

Jupyter Notebook

- Computational notebooks as Jupyter Notebook: interactive computing environment combining executable code and descriptive text in a single document [1]
- “Literate programming”: programs should be similar to works of literature [2]
- A chronological progression of markdown cells, code cells and output cells [3]
- Facilitate presentation of data analysis + keep track of variants explored [3]

→ **Reproduce + share**

Additional background

- The paper contains a set of rules that indicate good habits for organizing scientific (not only computational) research and sharing it, even when you (still) don’t know how to code.

- Good to know:
  - Code repository service such as Github
  - General concepts: markdown, modularize, dependencies, version control

Aims of the paper

- Address challenges and opportunities given by computational notebooks

- Reproducibility in computational research can be achieved if all files are contained in a single “computational narrative”

- BUT: Special care is needed!

Methods

- Review of literature on reproducible analyses in computational research

- Git Repository with annotated example of notebooks

https://github.com/jupyter-guide/ten-rules-jupyter

- a repository with some tutorials

https://github.com/jupyter-guide/jupyter-guide

The 10 rules

Rule 1: Tell a story for an audience

- Introduce the topic

- Describe your steps: what, how, why + start documenting your thoughts asap

- Interpret the results

https://github.com/jupyter-guide/ten-rules-jupyter

Predict Fold Type of a Protein from Protein Sequence

The notebooks in this directory demonstrate and apply the "Ten Rules for Reproducible Research in Jupyter Notebooks". Throughout the notebooks we refer to some the rules we applied.

For example, this notebook demonstrates:

Rule 1: Tell a Story for an Audience. This notebook was developed to learn how to apply a simple machine learning model to predict protein features based on protein sequences.

Rule 3: Use Divisions to Make Steps Clear. We broke the workflow into separate notebooks and use this top-level notebook to explain and organize the workflow.

Rule 7: Build a Pipeline. This notebook describes the entire workflow from data preparation, feature calculation, model fitting, to prediction. The modularity makes it easy to replace one of the steps, for example, use a different method to calculate features or apply a different machine learning model.

Introduction

Proteins have four different levels of structure – primary, secondary, tertiary and quaternary. Secondary structure describes the geometry of segments of a protein chain. The most common secondary structure elements are:

- Alpha helices
- Beta sheets

We can classify proteins into three major fold classes based on their predominant secondary structure content:

- alpha: contains predominantly alpha helices
- beta: contains predominantly beta sheets
- alpha+beta: contains alpha helices and beta sheets

Goal

This notebook demonstrates how to create a reproducible record using a machine learning model. We train the model to predict the fold class of a protein given its amino acid sequence using a representative set of 3D structures from the Protein Data Bank.

Run the following notebooks and explore how we applied the Ten Simple Rules.

1. Create Dataset

First, we create a dataset with protein secondary structure information obtained from 3D protein chains.

Run the following notebook to extract secondary structure information from a representative set of protein chains downloaded from the RCSB Protein Data Bank and assign a fold class to each protein chain.

1-CreateDataset.ipynb

This notebook saves the dataset in the file `./intermediate_data/foldClassification.json`.

Rule 2: Document the process, not just the results

- Document all your explorations, even the rejected ones
- Clean, organize, annotate - do it while exploring
- Do not manually edit figures with desktop publishing tools

Rule 3: Use cell divisions to make steps clear

- 1 cell = 1 meaningful step
- Label cells with markdown headers (table of content) + code comments
- Avoid long cells (< 100 lines)
- Series of notebooks + top-level index notebook

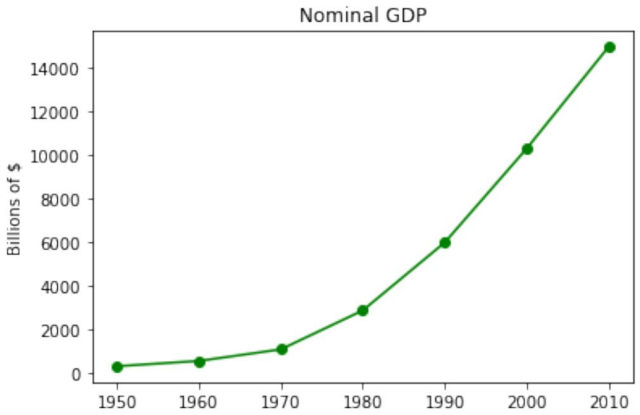
1 Make a simple plot in one cell

Load the libraries and modules in a separate cell at the beginning of the file.

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# and so on
```

Define some variables and plot them in a single cell.

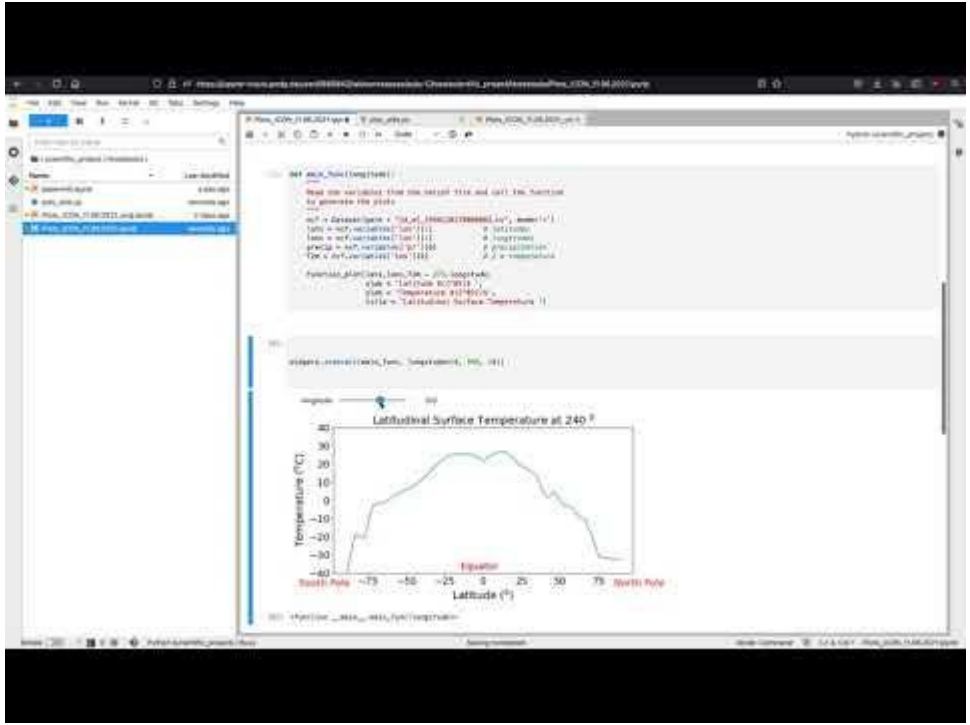
[2]:
```
years=[1950, 1960, 1970, 1980, 1990, 2000, 2010]
gdp=[300.2, 543.3, 1075.9, 2862.5, 5979.6, 10289.7, 14958.3]

# create a line chart, years on x-axis, gdp on y-axis
plt.plot(years, gdp, color='green', marker='o', linestyle='solid')

# add a title
plt.title("Nominal GDP")

# add a label to the y-axis
plt.ylabel("Billions of $")
plt.show()

# [example taken from [6]]
```



Rule 4: Modularize code

- Turn your code into a module, package, or library.

- Tie widgets to functions



Rule 5: Record dependencies

- Manage your dependencies using a package or environment manager like pip or Conda

- Conda's environment.yml or pip's requirements. txt files

```
$ conda env create --file ~/scientific_project/scientific_project.yml
```

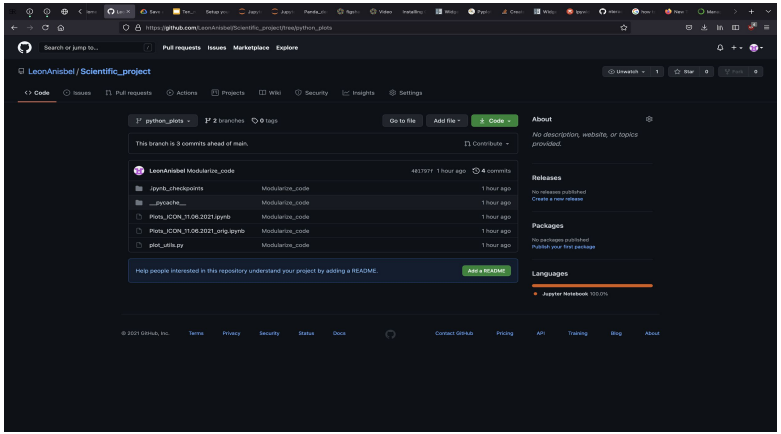
- Always conduct your work in an environment created only from these dependencies to ensure you do not add undocumented dependencies.

Rule 6: Use version control

- Due to the interactive nature of notebooks accidental changes or important content delete is easy.

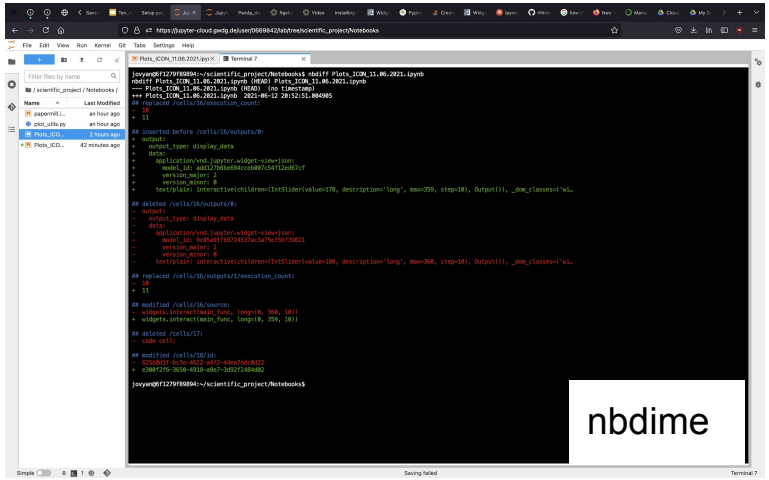
Use Git or GitHub

(https://github.com/LeonAnisbel/Scientific_project/tree/python_plots)



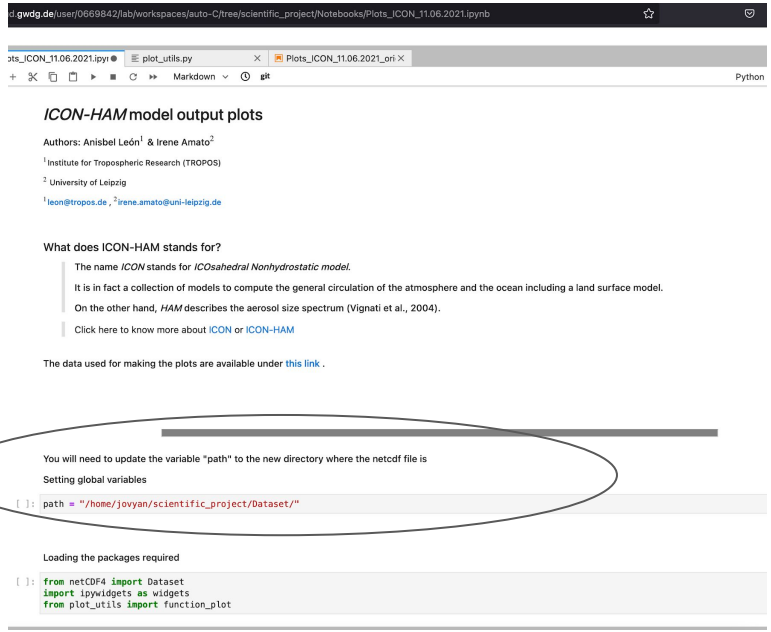
Notebooks store both code and extensive metadata

Jupyter about each cell as a text file in the JSON format



Rule 7: Build a pipeline

- Place key variable declarations at the top of the notebook
- Avoid manual interventions
- Restart your kernel and rerun all cells regularly
- Parameterized your notebook, use papermill (https://github.com/nteract/papermill)
- Link your analysis pipeline steps via a Makefile that allows for complete noninteractive execution of the entire pipeline.
- Remember to remove any introduction, interpretation, or conclusion text that is not universally applicable



Rule 8: Share and explain your data

- Figshare [https://figshare.com/]

- Zenodo [https://zenodo.org/]

https://figshare.com/articles/dataset/2d_ml_19991201T000000Z_nc/14770305

2d_ml_19991201T000000Z.nc (3.14 MB)

2d_ml_19991201T000000Z.nc

Cite | Download (3.14 MB) | Share | Embed | + Collect | •••

Dataset posted on 11.06.2021, 16:05 by Anisbel Leon

2D Meteorological variables (Output of the ICON-HAM model)

FUNDING

PhD. studies as part as AC3 project

HISTORY

- **11.06.2021** - First online date, Posted date

REFERENCES

- https://figshare.com/account/home

USAGE METRICS

23 views | 2 downloads

CATEGORIES

- Atmospheric Aerosols

KEYWORDS

Atmospheric dispersion modeling

LICENCE

CC0

EXPORTS

Select an option

About | Features | Tools | Blog | Ambassadors | Contact | FAQs | Privacy Policy | Cookie Policy | T&Cs | Sitemap

figshare. credit for all your research.

https://jupyter-cloud.gwdg.de/user/0669842/lab/tree/scientific_project/Notebooks/Plots_ICON_11.06.2021.ipynb

File Edit View Run Kernel Git Tabs Settings Help

Filter files by name

/ scientific_project / Notebooks /

Name	Last Modified
papermill.i...	an hour ago
plot_utils.py	2 hours ago
Plots_ICO...	2 hours ago
Plots_ICO...	an hour ago

Plots_ICON_11.06.2021.ipyn | Terminal 7

Code | git

ICON-HAM model output plots

What does ICON-HAM stands for?

> The name *ICON* stands for *ICOsahedral Nonhydrostatic model.*
>
> It is in fact a collection of models to compute the general circulation of the atmosphere and the ocean including a land surface model.
>
> On the other hand, *HAM* describes the aerosol size spectrum (Vignati et al., 2004).
>
> Click here to know more about ICON or ICON-HAM

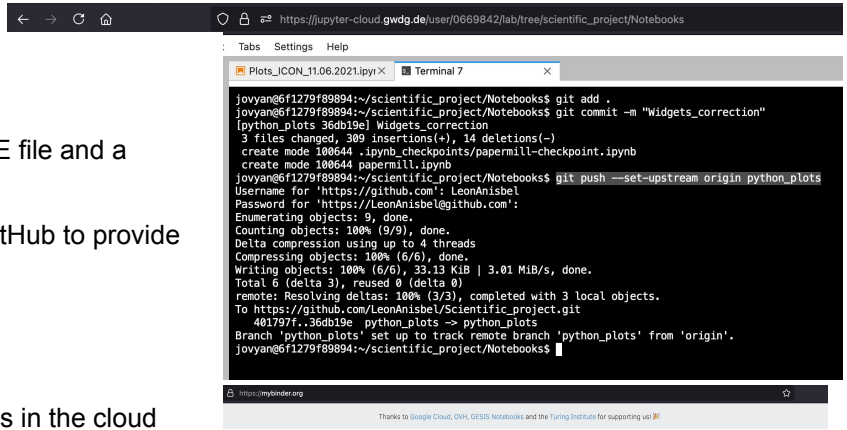
The data used for making the plots are available under this link .

You will need to update the variable "path" to the new directory where the netcdf file is

Setting global variables

[4]:
```
path = "/home/jovyan/scientific_project/Dataset/"
```

Rule 9: Design your notebooks to be read, run, and explored

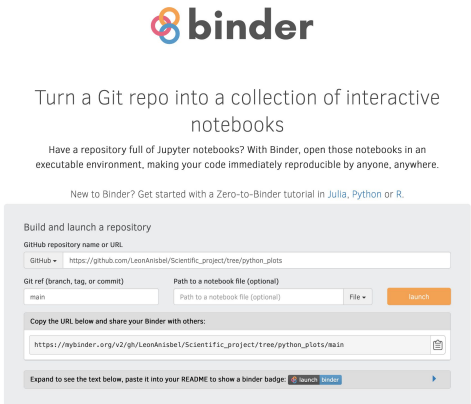


<u>Read</u>

- Store your notebooks in a public code repository with a clear README file and a liberal open source license
- Use Nbviewer (https://nbviewer. jupyter.org/) or push a notebook to GitHub to provide static views of your executed notebook online

Run

- Use Binder to provide a zero-install environment to run your notebooks in the cloud (https://mybinder.org/) or a Docker image



Explore

- Use ipywidgets (https://ipywidgets.readthedocs.io/en/stable/)
- Use cell-structure and functions to make it easier to extract sections enabling future users to introduce their own changes

Rule 10: Advocate for open research

- Become an advocate in your lab or workplace in promoting its effective use
- More people use it every day

Jupyter Notebooks shared publicly on GitHub
(https://github.com/search?l=Jupyter+Notebook&q=extension%3Aipynb+nbformat_minor&type=Code)

- more than 3 million (by December 2018)

- 5,828,664 (by June 11th 2021)

+ 57 696 in 2 days

- 5,886,360 (by June 13th 2021)

Conclusions of the paper

- Notebooks ease precise documentation of complex workflows,

- But they also complicate it because of interactivity.

- Advantages: annotation of the analysis, organization of the code, ease of access and reuse

Other perspectives

Run bits of code in arbitrary order is *weird* [4]

- “Hidden state” given by out-of-order execution, deletion of cells or edit without executing
- Interactivity: confusing unless you run the cells precisely in order (but %history)

Reproducibility is difficult to obtain [5]

- Notebooks do not, by themselves, ensure replicability.
- The authors of [5] were able to successfully execute only one of the ~25 notebooks that they downloaded.
- Code cannot be easily rerun if you don’t freeze dependencies, share data, describe computing environment.

Still far from literate programming [3]

- People use more cell structure rather than markdown explanations.
- 50% users (of 45): scratch pads, cells expected to be preliminary -> not annotated
- Cells containing alternative approaches simultaneously in view: keep a clutter of old iterations

Exploration vs. explanation [7]

- Descriptive text is not evenly distributed across notebooks
- 43.9% (of 1.23 M notebooks): evident non-linear execution order

Our personal conclusions

- Well-documented code easy to share
- Ability to run bits of code in arbitrary order: difficult to handle
- We see critiques as further tips to better use this tool
- Important: clean code + take care of version control

Future directions

Publication citations - 34

- Nust, D. et al. (2020) Ten simple rules for writing Dockerfiles for reproducible data science.
- Peikert, A. and Brandmaier, A.M. (2021) A Reproducible Data Analysis Workflow With R Markdown, Git, Make, and Docker

- Sahneh, F. et al. (2021) Ten simplerulestocultivatetransdisciplinarycollaborationin datascience.
- Moreno-Indias I, et al. (2021) Statistical and Machine Learning Techniques in Human Microbiome Studies:Contemporary Challenges and Solutions
- BodnerK, et al. (2021) Ten simple rules for tackling your first mathematical models:A guide for graduate students by graduate students.

- Jupyter Notebooks shared publicly on GitHub more than 5 millions

- Autocomplete is not working well + no real-time type-checking and linting [4]

References

1. Rule A, Birmingham A, Zuniga C, Altintas I, Huang S-C, Knight R, et al. 2019. Ten simple rules for writing and sharing computational analyses in Jupyter Notebooks. PLoS Comput Biol 15(7): e1007007. https://doi.org/10.1371/journal.pcbi.1007007
2. Knuth DE. Literate programming. 1984. The Computer Journal. 27(2):97–111.
3. Kery MB, Radensky M, Arya M, John BE, Myers BA. 2018. The Story in the Notebook: Exploratory Data Science using a Literate Programming Tool. CHI ’18 Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems. New York: ACM. https://doi.org/10.1145/3173574.3173748
4. Grus, J. I Don’t Like Notebooks. JupyterCon. New York, NY. 2018. Available from: https://docs.google.com/presentation/d/1n2RlMdmv1p25Xy5thJUhkKGvjtV-dkAIsUXP-AL4ffI/edit#slide=id.g362da58057_0_1
5. Woodbridge M, Sanz D, Mietchen D, Mounce R. Jupyter Notebooks and reproducible data science. 2017. Available from: https://markwoodbridge.com/2017/03/05/jupyter-reproducible-science.html
6. Grus J. 2019. Data science from scratch: first principles with python. O'Reilly Media.
7. Rule A, Tabard A, Hollan JD. 2018. Exploration and Explanation in Computational Notebooks. CHI ’18 Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems. New York: ACM. https://doi.org/10.1145/3173574.3173606