**The 7 Best Practices to Move Your Machine Learning Projects into Production Faster Using Python**

Practical advice for designing machine learning algorithms ready for production



Photo by [Matt Ridley](https://unsplash.com/@mattwridley" \t "_blank) on [Unsplash](https://unsplash.com/s/photos/production?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText" \t "_blank)

**Motivation**

Deploying in production is **hard** without proper infrastructure, tools and knowledge. In most cases, the data scientists contribute mainly to the experimentation, analyzing the data and selecting the Machine Learning (ML) algorithm that meets best the user requirements. So far everything looks great. The client is happy and impatient to have its algorithm working in production. The magic word is **production.**Everyone assumes that production is the easiest part. And yet unfortunately the road is still long. Less than 30% of ML (e.g [AI Adoption in the Enterprise 2022](https://www.oreilly.com/radar/ai-adoption-in-the-enterprise-2022/) ) projects end up running successfully in production acquiring business value. To move quicker to production, this article proposes a list of the 7 **best practices** to be considered by data scientists from the very beginning of the project.

**Best practices**

**1. Don’t mix too many programming languages**

Avoid mixing too many languages (Scala, Python, Java, SQL, Bash) in your ML projects without a strong reason. Adhering to a single programming language enhances the collaboration between your team members and reduces the time spend on maintenance.

If your choice is Python, stick with it.

For instance, in my case, all the ML projects mixing a couple of languages such as Bash, Python, Java, and Scala end up being too difficult to maintain as the team was most confident with Python. In the end, we had to even abandon the project or rewrite the modules in Python. An effort that paid off as we end up spending less time on maintenance and the team had more time to focus on ML and code quality.

**2. Version your code**

Teach your data scientist colleagues to version their work (notebooks included!). Versioning helps to track all the code changes. In the experimentation phase, we use to make evolutions and corrections on a daily basis and due to a wrong manipulation or a rollback, it is highly probable to lose a part of our work. Tracking all the code changes over time makes code manipulation far easier. I advise you to start using **[Git](https://git-scm.com/" \t "_blank)**for versioning because is the most popular version control framework, powerful, and easy to use. In my case, the data scientists that started to use code versioning turned out to be far more productive and much easier to collaborate with.

Please find below a simplified Git overview.

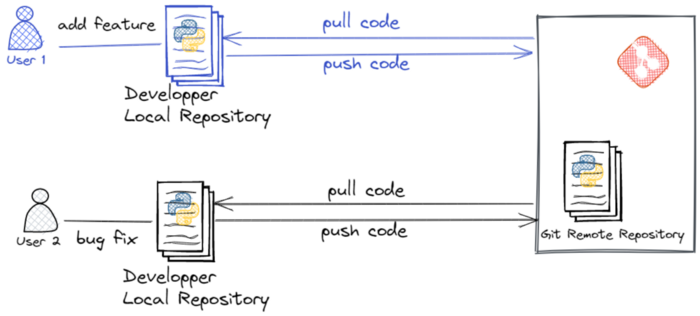


Image by the author

**3. Use Python virtual environments**

*What is a virtual environment? It's a Python tool that ensures dependency management and project isolation.*

Whenever I have to work on a task in Python, first, I make sure that I create a virtual environment rendering the development environment completely isolated from my distribution. In this way, I can easily share my application with the team without worrying about dependency conflicts. Note that isolation is a must in a production environment.

The picture below shows an outline of Python’s virtual environments.

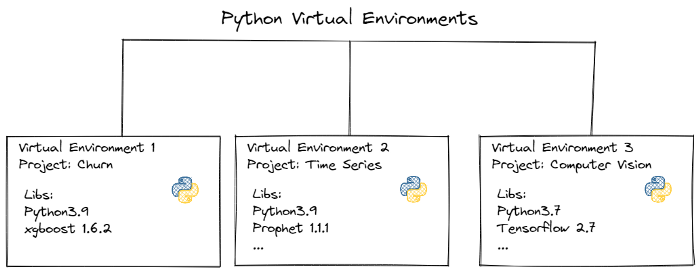


Image by the author

Check the**[second step in my previous article](https://medium.com/towards-data-science/how-to-set-up-custom-vertex-ai-pipelines-step-by-step-467487f81cad)**,itexplainshowto create a virtual environment and activate it in Jupyter.

**4. Define a project structure**

Don’t create each time a project from scratch. Propose a template project for your team. It helps both the data scientists and data engineers colleagues to contribute and switch fast between ML projects. It enables code organization and helps with code debugging. For example, I have been part of teams where each ML project had a distinct structure, it was very difficult to switch quickly between projects, and of course, the newcomers were confused. Since the team is using a unique project structure the process is transparent and effortless.

Feel free to use the open-source **[cookiecutter](https://drivendata.github.io/cookiecutter-data-science/" \t "_blank)**template generator. The usage is pretty straightforward.

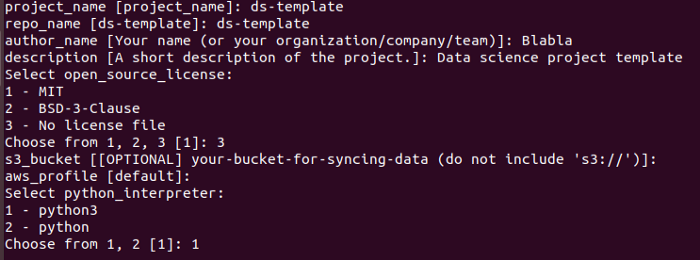
Install the package with

pip install cookiecutter

and start a new project with the command:

cookiecutter <https://github.com/drivendata/cookiecutter-data-science> .

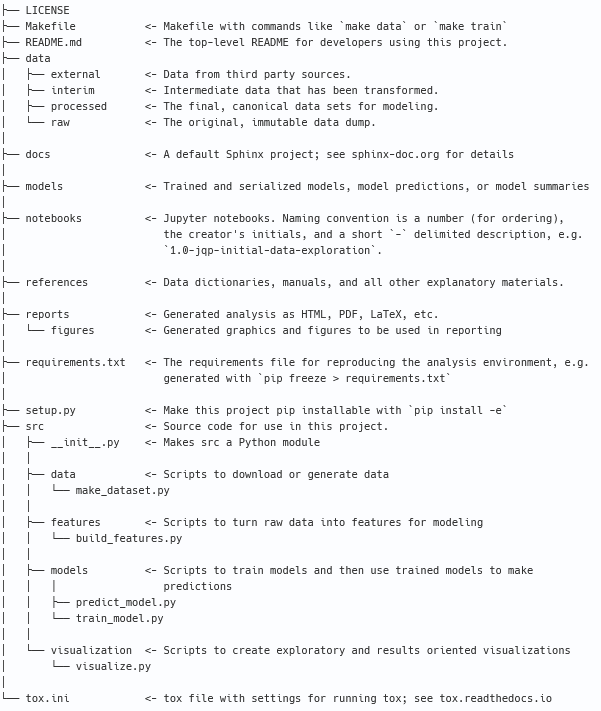
Cookiecutter will ask you to mention the project name, repository name, author, team, organization, license type, and the python interpreter you plan to use. See below my configuration.



Go to the generated directory:

cd ds-template

The generated project structure looks like the one below:



<https://github.com/drivendata/cookiecutter-data-science>

**5. Identify the Python open-source libraries**

I advise spending some time choosing the python libraries convenient for your ML project. Choose stable, secure, and common libraries. I coped with situations in which the team diverged due to library choices based on affinities and not necessarily on the team’s strategy. Hence, I end up losing time maintaining in-production deprecated or unstable libraries used just because they had cool features.

Please find hereafter my recommendations:

**Code analysis & formatting:**

[*F****lake8***](https://pypi.org/project/flake8/)*: performs static code analysis checking your code style against*[*PEP8*](https://peps.python.org/pep-0008/)*, errors, and cyclomatic complexity.*

[***Pylint***](https://pypi.org/project/pylint/)***:****performs static code analysis looking for errors, checking name conventions, code smells, and makes refactoring suggestions.*

[***Black***](https://pypi.org/project/black/)*formats the code.*

**Testing your code:**

[***Pytest***](https://pypi.org/project/pytest/)*is a powerful and easy-to-use testing framework.*

[***Coverage***](https://pypi.org/project/coverage/)*measures the testing coverage.*

**Security analysis:**

[***Bandit***](https://pypi.org/project/bandit/)*looks for common security issues turning the code into an abstract syntax tree (AST).*

[***Safety***](https://pypi.org/project/safety/)*checks Python dependencies for known*security*vulnerabilities.*

[***Clair***](https://pypi.org/project/clair/)*looks for security vulnerabilities in docker containers.*

**6. Define a simplified Git workflow for all projects**

*What is a git workflow? It defines the strategy for managing your Git branches.*

Complicated branching systems require mastering the mechanism of versioning which is difficult to achieve at the team level. Consequently as often the data scientist lack code versioning experience I suggest using a simplified strategy to manage the Git branches as the one below :

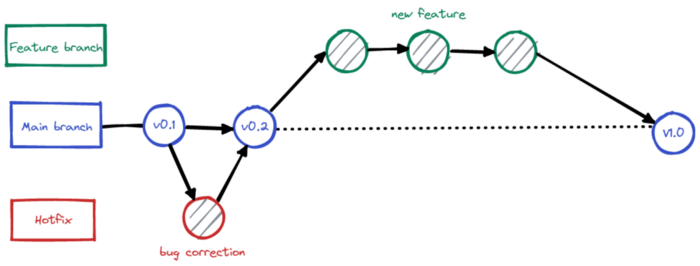


Image by the author

***Main branch***— is the default branch. It is the first branch created when a new Git repository is generated. In the older version of Git you may find the name “*master*”. Each time a code new feature or a bug is merged to the main branch it is recommended to generate automatically a tag (version). The branch contains production-ready code that can be released automatically in production.

***Feature branch****&****HotFix branch****—*are temporary branches. TheFeature branch is meant to develop the features of the product.The HotFix branch isused to quickly correct bugs. Once the development is ready the code should be merged into the main branch.

Implementing a simplified git workflow helped my team to leverage Git consistently and deliver efficiently and faster (fewer git conflicts to manage).

**7. Enforce Python Code Quality**

*Code quality stands for how maintainable and functional is the developed code.*

Code quality is very important in the short and long run. Being aware of the code quality practices helped me to deliver ready-for-production ML code easy to maintain and debug.

To enhance the code quality you should consider the following recommendations:

*Respect the*[***PEP 8***](https://peps.python.org/pep-0008/)*style and naming conventions.*

***Avoid long lines of code****.*

*Try to have****readable and modular code****.*

*Us****e [docstrings](https://peps.python.org/pep-0257/" \t "_blank)****.*

***Avoid using generic names****such as data\_XXX, dictionary\_XX etc*

*Use****[logging](https://docs.python.org/3/library/logging.html" \t "_blank)****and replace*print*with*logs*.*

***Use linter and autoformaters.****Personally, I appreciate the linters*[***Flake8***](https://flake8.pycqa.org/en/latest/)*/****[Pylint](https://pypi.org/project/pylint/" \t "_blank)****and the autoformaters****[Kite](https://kite.com/" \t "_blank)****/****[Tabnine](https://www.tabnine.com/" \t "_blank).***

***Document your code.******[You can refer to my tutorial](https://medium.com/towards-data-science/document-your-machine-learning-project-in-a-smart-way-35c68aa5fc0e)****for more details.*

**Recap**

To accelerate the process of moving ML projects to production it’s essential to be aware of the industry's best practices and to use them.

In this article, I have discussed the importance of using a common programming language, code versioning, a standard project structure, virtual environments, a simplified git workflow, well-known Python libraries, and Python code quality standards.