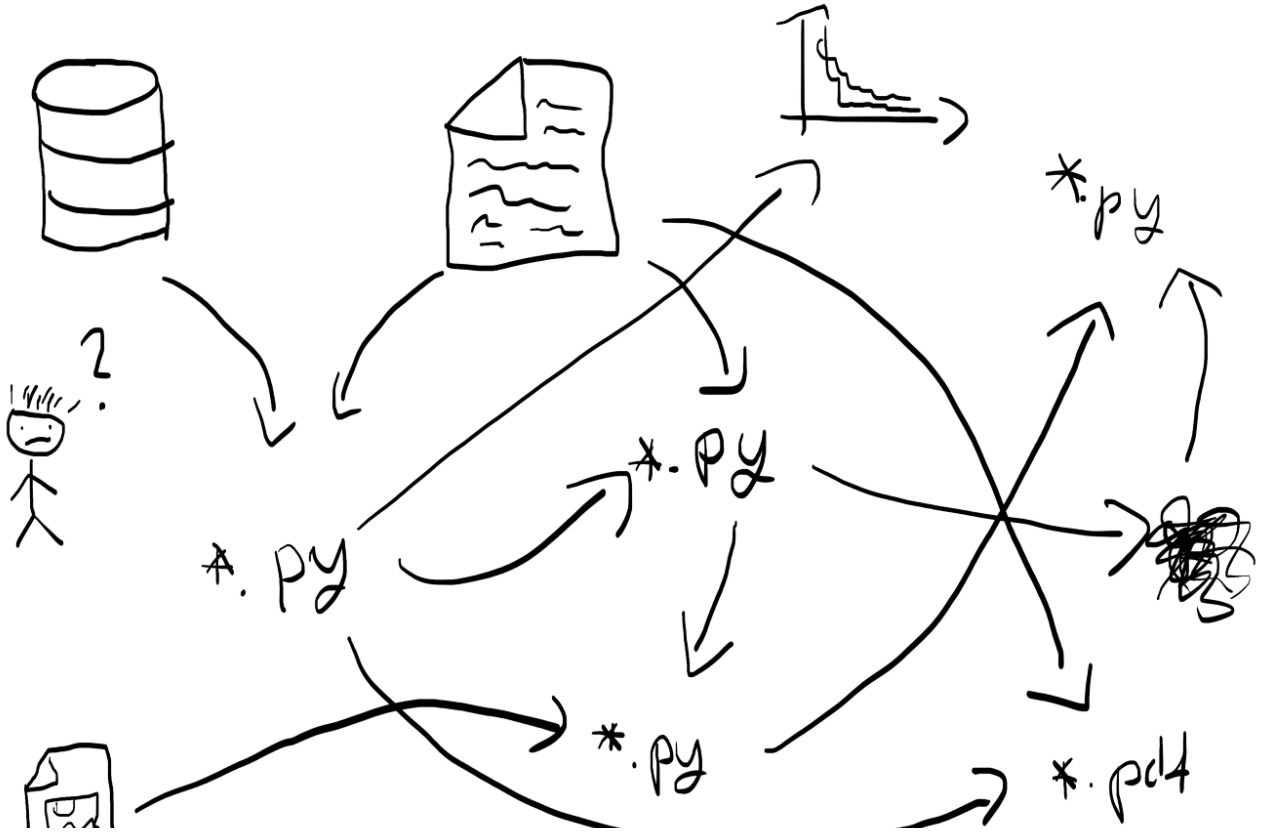
Mateusz Bednarski Follow

May 20, 2017 · 8 min read

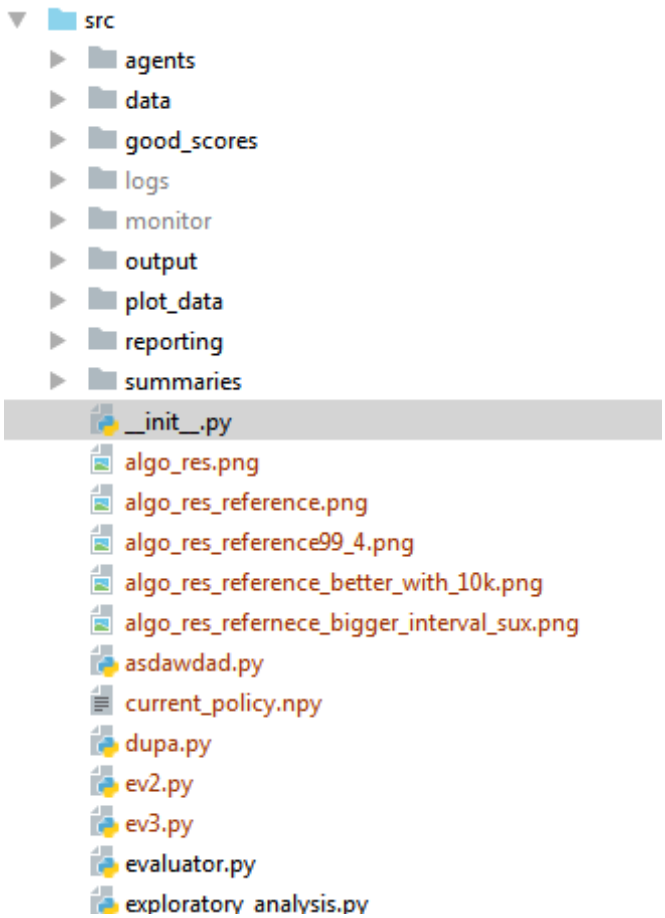


How am I supposed to run this from the beginning?

You start with a brand new idea for the machine learning project. First of all you download the dataset. Then perform some kind of preprocessing — possibly multi step because task is sophisticated. You create a bunch of models, some of them perform better while other ones worse. Oh shit, I overwrite my best model. No problem — will train it again. Wait. What were the parameters? Oh, got it. I'm mostly done — just generate some nice plots and that's all. Maybe should I refactor a little? Nah, everything is fine, moreover — I could break something as it's quite fragile.

A few months later you want to back, to it and maybe slightly modify to work with new data. Piece of cake. Git clone mycoolproject and… your mind went blank. What am I supposed to execute? In which order? What the hell *System cannot find path specified*? And from where data did previously come? Shit, I can't reproduce previous results :c Why?

> *Chaos is the only true answer*



Looks similar? I bet you experienced similar scenario, especially if you are not much experienced in machine learning. And if you are not working in models that will be eventually deployed — so you would be kind of "forced" to do things the right way. On the other hand, you do not want to spend a

lot of time on structuring your project. Here I will present you simple approach.

Prerequisites

But before we start I made a few assumptions about You ;)

- You are quite familiar with machine learning (very basic level is enough)
- You are quite familiar with Python (I'm using Python 3)
- You want to learn something ;)
- You have Anaconda installed
- You use Windows or Linux. I have not tested everything on MacOs but I do believe it should work. If you have issues with installing `make` on Windows let me know in comments.

OK, so what are benefits of presented approach?

- You will be able to perform whole process from grabbing the data to generate reports by one command

- You will be able to reproduce your work
- You will thank yourself after a year when come back to project
- Others will be able to understand your code (also potential employers)

I'll show everything as a complete, small project.

New repository

At the beginning, after cloning new repo from GitHub we have following structure:

```
LICENSE  README.md
```

Let's start with creating reproducible environment. Create file called `environment.yml` We can start with following content:

```
dependencies:
 - python=3.5
```

```
  - numpy
  - scipy
  - scikit-learn
  - jupyter
  - requests
```

As the project will grow up we will keep this file up to date. But now let's create an actual env:

```
conda env create -f environment.yml -n chaos
```

Shit, I forgot to include jupyter package. No worries. Just update `environment.yml` with jupyter and run

```
conda env update -f environment.yml -n chaos
```

And let's check if it works

```
activate chaos
jupyter notebook
```

Looks fine.

Downloading data

First step is to download the data. But we will not do it by copying from Downloads directory . Instead: we can create python script in

`src/data/download.py`

> *Of course, creating script instead of just wget is a lot of over complication but I did this with the purpose of providing educational example ;) You can imagine that grabbing the data is more complicated and requires scripting.*

```
import requests


url = 'https://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data'
filename = 'data/raw/iris.csv'
```

```
    with open(filename, 'wb') as ofile:
        response = requests.get(url)
        file.write(response.content)
```

And try to run it

```
(chaos) λ python src/data/download.py
Traceback (most recent call last):
  File "src/data/download.py", line 6, in <module>
    with open(filename, 'wb') as ofile:
FileNotFoundError: [Errno 2] No such file or directory:
'data/raw/iris.csv'
```

Aaaaand it does not work. Because directory `data/raw` isn't there. Before we create it, hold on for a second. Here is great place to stop and think for a while. Firstly: our code should be robust to non-existing directories. Next, we should not create any directory by hand. If your colleague would be forced to create directory by hand every hour on error during saving results, he won't be happy. Also, calling `mkdir(exist_ok=True)` always when we refer a directory might not be the best idea.

What is more, downloading data is very first step in our workflow — and should be repeatable.

So let's start fixing this issues. At the beginning create directory structure for data. There would be three of them:

- `data/raw` - here all raw data exists. This directory should be considered as read only - just leave what we got as it is.
- `data/processed` - data after whole preprocessing, merging, cleaning, feature engineering etc.
- `data/interim` - intermediate format between raw and processed. Not raw and also not ready yet.
- `data/external` - any data we consider as being external. E.g dictionaries, synonyms.

Flow for the data

We said, that we need a way to enforce existing of this directories And it's simple way of doing this:

```
mkdir -p data/raw data/interim data/external data/processed
touch data/external/.gitkeep data/raw/.gitkeep data/interim/.gitkeep
data/processed/.gitkeep
```

Git does not store empty directories. By creating an empty `.gitkeep` file we can enforce directory persistence. By default, data should not be stored in git repository — so that we need to adjust `.gitignore`:

```
# exclude data from source control by default
/data/
```

But this makes data directories invisible for git. So It won't create them on clone. To do so, we have to force add `.gitkeep`

```
git add -f data/external/.gitkeep data/processed/.gitkeep
data/interim/.gitkeep data/raw/.gitkeep
git add .gitignore
git status
```

We should see something like:

```
c:\code\overcome-the-chaos (master)
(chaos) λ git status
On branch master
```

```
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   .gitignore
        new file:   data/external/.gitkeep
        new file:   data/interim/.gitkeep
        new file:   data/processed/.gitkeep
        new file:   data/raw/.gitkeep

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        environment.yml
        src/
```

If so, we can

```
git commit -m "Empty data directories"
```

Since now, I will not commit to git explicitly — you can do it whenever you want.

Workflow

OK, we solved issue with directories. Let's move on to workflow. Downloading is very first step and should be written correctly. But what does it mean? Parameters should be possible to pass by command line arguments. For that we will use click library. Add to `environment.yml`

```
- click
```

And update env.

Script can look like this:

```
import requests
import click


@click.command()
@click.argument('url')
@click.argument('filename', type=click.Path())
def download_file(url, filename):
   print('Downloading from {} to {}'.format(url, filename))
   response = requests.get(url)
   with open(filename,  'wb') as ofile:
       ofile.write(response.content)
```

```
if __name__ == '__main__':
    download_file()
```

Whooah, so far we definitely not simplifying things.

To explain: we define command with two parameters and *click* will handle this for us. If we type:

```
python src/data/download.py --help
```

We will see:

```
Usage: download.py [OPTIONS] URL FILENAME

Options:
  --help  Show this message and exit.
```

Looks nice. For actual downloading we have to call:

```
python src/data/download.py "https://archive.ics.uci.edu/ml/machine-
learning-databases/iris/iris.data" data/raw/iris.csv
```

Make

Now we are ready to automate workflow. We will use … **GNU Make**. No. I'm not joking. This tool suits very well for our needs. Also writing Makefiles is not as scary as you might think. Interesting fact: in practice this is very portable solution between Linux, Mac and Windows.

Our first workflow will consist of three steps.

- `clean` - will remove all generated output
- `all` - will run whole pipeline from beginning to the end
- `download` - will download the data

All depends on download so we have a graph of dependencies

Dependency graph for simple workflow. Arrow means "all depends on download"

Of course, this graph will grow up in the future. The nice thing about make is, that it will handle most of the complexity by itself.

Create `Makefile` and add first target

```
clean:
 rm -f data/raw/*.csv
```

This is quite obvious — remove all downloaded data. Let's check it

```
make clean
rm -f data/raw/*.csv
```

And `iris.csv` is gone

> *If you have error like:*
>
> *Makefile:2: *** missing separator. Stop.*
>
> *Make sure that your editor did not replaced tab with multiple spaces.*

Next target would be download

```
data/raw/iris.csv:
  python src/data/download.py
"https://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data" $@
```

Before the colon is target name — usually file that would be created (I will back to this soon). After colon — dependencies for the task. As we can see

downloading is first step in our workflow so does not have any prerequisites. There is only one command in order to “build” iris.csv — calling download script. This crazy `$@` means “first target in a rule”. We have only one target (data/raw/iris.csv) so it will be inserted here. We can make a test drive:

```
make data/raw/iris.csv
python src/data/download.py "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data" data/raw/iris.csv
Downloading from https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data to data/raw/iris.csv
```

And indeed, file has been downloaded:

```
tree data
data
├── external
├── interim
├── processed
└── raw
    └── iris.csv
```

Let’s try to execute it one more time:

```
make data/raw/iris.csv
make: 'data/raw/iris.csv' is up to date.
```

Make knows that target already exists and there is no need to rebuild it.

There is only one piece left — `all`. We define it as follows:

```
all: data/raw/iris.csv
```

This means: “In order to build all you need to have data/raw/iris.csv”

Let’s see it in action:

```
make clean
make all
python src/data/download.py "https://archive.ics.uci.edu/ml/machine-
learning-databases/iris/iris.data" data/raw/iris.csv
```

```
Downloading from https://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data to data/raw/iris.csv
tree data
data
├── external
├── interim
├── processed
└── raw
    └── iris.csv
```

It works :)

More on Makefile

Just three more things to add. When you type just make it will execute first target. So it is reasonable to make all first target in file . Secondly, you can use variables — for example to store URL

```
IRIS_URL = "https://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data"


data/raw/iris.csv:
  python src/data/download.py $(IRIS_URL) $@
```

Finally, targets that do not create files, should be marked as `.PHONY`

```
.PHONY: all clean
```

If you want know why, try creating files with names either `all` or `clean` ;)

Here is the final Makefile:

```
.PHONY: all clean

IRIS_URL = "https://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data"

all: data/raw/iris.csv

clean:
  rm -f data/raw/*.csv

data/raw/iris.csv:
  python src/data/download.py $(IRIS_URL) $@
```

This is the end of first part. In next ones I will show you how to further structure machine learning project and how to extend whole pipeline.

If you want to hear about something specific feel free to leave a comment.

Stay tuned.

Code repository is here: https://github.com/artofai/overcome-the-chaos

Link to second part: https://medium.com/@mbednarski/structure-and-automated-workflow-for-a-machine-learning-project-part-2-b5b420625102

Machine Learning | Workflow Automation | Data Science | Python

668 claps

WRITTEN BY

Mateusz Bednarski

Follow

AI enthusiast. Focused mostly on NLP and good software engineering practices for machine learning projects. Currently working at Roche.

Towards Data Science

Follow

A Medium publication sharing concepts, ideas, and codes.

See responses (3)

More From Medium

A Better Way To Become A Data Scientist Than Online Courses

Chris in Towards Data Science

Coding Mistakes I Made As A Junior Developer

Chris in Towards Data Science

Data Classes in Python

Halil Yıldırım in Towards Data Science

Machine Learning Engineer vs Data Scientist (Is Data Science Over?)

Jason Jung in Towards Data Science

Sorry, Online Courses Won't Make you a Data Scientist

Lesser known Python Features

Ramshankar Yadhunath in Towards Data Science

James Briggs in Towards Data Science

5 Books That Will Teach You the Math Behind Machine Learning

Tivadar Danka in Towards Data Science

When not to use machine learning or AI

Cassie Kozyrkov in Towards Data Science

Discover Medium

Welcome to a place where words matter. On Medium, smart voices and original ideas take center stage - with no ads in sight. Watch

Make Medium yours

Follow all the topics you care about, and we'll deliver the best stories for you to your homepage and inbox. Explore

Become a member

Get unlimited access to the best stories on Medium — and support writers while you're at it. Just $5/month. Upgrade

---

Medium

About Help Legal