How to Structure a Python-Based Data Science Project (a short tutorial for beginners)

Misha Berrien Following
Jul 24, 2019 · 7 min read ★

Photo by Markus Spiske on Unsplash

The Problem
In the middle of my last data science project, I realized that I had a real problem. After staying up late the night before to clean a particularly disorganized dataset, I woke up to discover that much of my hard work had been lost. What's worse, I had simultaneously been working in a number of different jupyter notebooks, so the code I did have was disordered and unclear. That day, as I struggled to reproduce the code that I had written the night before, I vowed then and there to learn a way to organize my projects in a way that would ensure that this particular road bump would never happen again.

The Solution
After speaking with a number of friends working in the data science field, I discovered that I needed my own data science project template. This post is broken up into three distinct parts:

- First, I'll discuss three characteristics that make a good data science project template.
- Second, I'll walk you through a short tutorial — in the form of a mini-project — in order to demonstrate how I use my project template to keep my projects organized.
- Third, I'll show you how to easily save, reuse and share your project templates.

Let's get started!

(Note: I chose the machine learning pipeline template found on the Data Science for Social Good website. You can find the organizations original explanation here along with many other useful data science resources).

. . .

STEP 1: THINK ABOUT THE PROBLEM YOU'RE TRYING TO SOLVE

After taking a look through my last project, I realized that I needed to solve a number of different (but related) problems:

1. ***Inconsistent file and variable names***: During my early days of coding, I would often return to a project a few days or even weeks later and find it difficult to remember which files to work with. I also found it incredibly difficult to remember the differences between variables and data frames with vague names such as df, df_new, df_new_final and (the dreaded) df_new_final2.
2. ***Inconsistent folder structure***: In the past, each of my project structures contained a data folder, but that was the extent of my project structure. Even within the data folder there were often a large number of datasets with vague names in different stages between raw and clean.
3. ***Wasted Time***: I found that I was wasting a lot of time re-running older scripts in order to re-clean my code before moving onto the next step.

What's worse, all of these problems were compounded when I worked with other people with different file naming conventions and ideas about how to organize a project.

STEP 2: CHOOSE A PROJECT TEMPLATE (no need to reinvent the wheel)

After talking with a few data scientist — and doing a lot of independent research — I realized that I needed to come up with a consistent data science project file structure (a project template). There is no reason to reinvent the wheel at this stage. If you google 'data science project templates', you'll find a number of great examples on the web. After reading through — and trying out — a number of project structures, I've found that the best ones all have a few things in common:

1. ***A clear data pipeline (starting with unchanged raw data)***: This means that the data is transformed in stages. For example, the raw data is fed into a notebook that transforms and saves an intermediate (cleaned) dataset. The intermediate dataset is then read into a processing notebook where feature engineering takes place. The new dataset (with features) is then saved before moving onto the next stage.
2. ***Separation of concerns (seperate files that do different things)***: In computer science, separation of concerns (SoC) is a design principle for separating a computer program into distinct sections, so that each section addresses a separate concern.
3. ***Standard file naming conventions***: In order to avoid the terrible df/df_new/df_new_final scenario listed above, it's important to build one consistent file naming system before you start coding.

Ultimately, the idea is that someone who stumbles across your project on github should be able to quickly get a sense of your file structure and be able to reproduce your results from running just a small number of scripts (for more in-depth info on the logic behind the project structure look here).

In the end, I chose to follow the project structure laid out by the people at Data Science for Social Good. Below is a slightly-modified schema of their system.

```
├── README.md          <- The top-level README for developers.
├── conf               <- Space for credentials
│
├── data
```

```
│   ├── 01_raw          <- Immutable input data
│   ├── 02_intermediate<- Cleaned version of raw
│   ├── 03_processed    <- Data used to develop models
│   ├── 04_models       <- trained models
│   ├── 05_model_output<- model output
│   └── 06_reporting    <- Reports and input to frontend
│
├── docs               <- Space for Sphinx documentation
│
├── notebooks          <- Jupyter notebooks. Naming convention is
│                         date YYYYMMDD (for ordering),
│                         the creator's initials, and a short `-`
│                         delimited description.
│
├── references         <- Data dictionaries, manuals, etc.
│
├── results            <- Final analysis docs.
│
├── requirements.txt   <- The requirements file for reproducing the
│                         analysis environment.
│
├── .gitignore         <- Avoids uploading data, credentials,
│                         outputs, system files etc
│
└── src                <- Source code for use in this project.
    ├── __init__.py    <- Makes src a Python module
    │
    ├── d00_utils      <- Functions used across the project
    │   └── remove_accents.py
    │
    ├── d01_data       <- Scripts to reading and writing data etc
    │   └── load_data.py
    │
    ├── d02_intermediate<- Scripts to transform data from raw to
    │   │                  intermediate
    │   └── create_int_payment_data.py
    │
    ├── d03_processing <- Scripts to turn intermediate data into
    │   │                 modelling input
    │   └── create_master_table.py
    │
    ├── d04_modelling  <- Scripts to train models and then use
    │   │                  trained models to make predictions.
    │   └── train_model.py
    │
    ├── d05_model_evaluation<- Scripts that analyse model
    │   │                      performance and model selection.
    │   └── calculate_performance_metrics.py
    │
    ├── d06_reporting  <- Scripts to produce reporting tables
    │   └── create_rpt_payment_summary.py
    │
    └── d07_visualisation<- Scripts to create frequently used plots
        └── visualise_patient_journey.py
```

STEP 3: LEARN HOW TO USE YOUR NEW TEMPLATE

After learning the basics of organizing a data science project and choosing a project structure, I decided to test it out on a mini-project before implementing it on a larger scale.

Libraries, packages and tools used:

- ***OS Module (os)***: The functions that the OS module provides allows you to interface with the underlying operating system that Python is running on
- ***System-specific parameters and functions (sys)***: This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter.
- ***Jupyter notebooks***: All experimental code is kept in the notebooks folder. When completed, code is put into a function and migrated to the appropriate sub-folder under 'src' and read back into a new notebook. Below is a sample pipeline for a simple project workflow.

Below is a recap of the mini project I ran with a step-by-step guide.

. . .

Jupyter Notebook 1: *20190723-mlb-data-collection-file*

I first wrote code to interact with the alpha vantage api and pull the raw dataset. After pulling the data I did two things:

First, I saved the raw data to the data/01_raw folder in my file system (raw data should never be edited, more on that here).

Second, after testing the code used to download the data in a jupyter notebook, I then wrote a function to pull the dataset seamlessly with just a few inputs. I then saved the new function in a .py file inside the src/d00_utils folder.

· · ·

Jupyter Notebook 2*: 20190723-mlb-feature-processing*

Note*: Since the data came in clean, I will skip the intermediate step of data cleaning (the product of going from raw to clean data would go into the data/02_intermediate folder) and go straight to feature engineering.*

I first imported the raw data from the 01_raw data folder.

Below is the raw data structure.

(output from msft_proc.head())

I then ran a (very naive) regression on the dataset in order to test the project structure. For this mini-project, I'm interested in understanding if the day of the week corresponds to the volume of Microsoft stock being traded. Before running the regression, I first changed the timestamp variable into a dummy variable denoting days of the week.

Below is the new dataframe structure.

(output from msft_proc.head())

Like in the last example, I then wrote a function that contains all steps in the feature engineering process. I then saved the function in a .py file in the src/d00_utils folder. Afterwards, I saved the processed data in the data/03_processed folder.

· · ·

Jupyter Notebook 3*: 20190723-mlb-model-fitting*

I first read the processed dataset into the new jupyter notebook.

I then ran a simple regression on the processed dataset.

After running the regression, I then chose a model to save to the data/04_models folder.

. . .

STEP 4: SAVE YOUR TEMPLATE (and share it with others!)

In June, github released a feature called repository templates that makes reusing (and sharing) a project file structure incredibly easy. Checkout their blog post here for more details.

AND THAT'S IT (for now)! Whichever project file structure you choose, make sure to stick with it and share it with the people you work with. Your future self will thank you.

Check out my project template on github here and try it out for yourself!

Source: For even more detailed information check out Cookiecutter Data Science.

Data Science