Project repository structure

One way we can ensure that our ML projects are easy to understand for others is to have a standard repository structure. Following a predefined format means that one knows where to find input data, models, or notebooks at a glance. In other words: we don't have to search through a whole directory to find the specific thing for which we are looking. This makes it much easier to jump into a project and start collaborating.

A common repository structure is the Cookiecutter data science project structure, which contains all of the elements to lay a good foundation for our projects:

```
├── LICENSE
├── Makefile           <- Makefile with commands like `make data` or `make
train`
├── README.md          <- The top-level README for developers using this
project.
├── data
│   ├── external       <- Data from third party sources.
│   ├── interim        <- Intermediate data that has been transformed.
│   ├── processed      <- The final, canonical data sets for modeling.
│   └── raw            <- The original, immutable data dump.
│
├── docs               <- A default Sphinx project; see sphinx-doc.org for
details
│
├── models             <- Trained and serialized models, model
predictions, or model summaries
│
├── notebooks          <- Jupyter notebooks. Naming convention is a number
(for ordering),
│                         the creator's initials, and a short `-`
delimited description, e.g.
│                         `1.0-jqp-initial-data-exploration`.
│
├── references         <- Data dictionaries, manuals, and all other
explanatory materials.
│
├── reports            <- Generated analysis as HTML, PDF, LaTeX, etc.
│   └── figures        <- Generated graphics and figures to be used in
reporting
│
├── requirements.txt   <- The requirements file for reproducing the
analysis environment, e.g.
│                         generated with `pip freeze > requirements.txt`
│
├── setup.py           <- Make this project pip installable with `pip
install -e`
├── src                <- Source code for use in this project.
│   ├── __init__.py    <- Makes src a Python module
│   │
│   ├── data           <- Scripts to download or generate data
│   │   └── make_dataset.py
│   │
│   ├── features       <- Scripts to turn raw data into features for
modeling
│   │   └── build_features.py
│   │
│   ├── models         <- Scripts to train models and then use trained
models to make
│   │   │                 predictions
│   │   ├── predict_model.py
│   │   └── train_model.py
│   │
│   └── visualization  <- Scripts to create exploratory and results
oriented visualizations
│       └── visualize.py
│
└── tox.ini            <- tox file with settings for running tox; see
tox.readthedocs.io
```

One of the most important directories is the `src` directory, which stands for *source*. We can keep all of the scripts that you use within our project in this directory. We might have a script to download your data from somewhere, one to featurize that data, one for model training, one for model validation, and so forth. Remember how we described in lesson 4 that we like to organize our code into distinct and reusable units? In this directory, we adhere to a structure along those lines.

We use a slightly adapted version of the cookiecutter structure throughout the rest of this course. While most of our setup is standard, we divide up the `src` directory into subdirectories that match the stages in our DVC pipeline. Moreover, we won't be using some of the items such as `Makefile`, `tox.ini`, and directories such as `references` and `docs`.

Converting to a project structure

Ideally, we start with a predefined structure right at the beginning of our project. Realistically speaking, however, we often start scripting in the early stages of prototyping and only later discover that we have somehow wound up with a `train.py` file consisting of 1200 lines of code. Luckily, breaking up an existing prototype and fitting it into a clearer project structure is doable.

In module 3 we will explore how to convert a notebook into a project that adheres to the cookiecutter data science project structure.