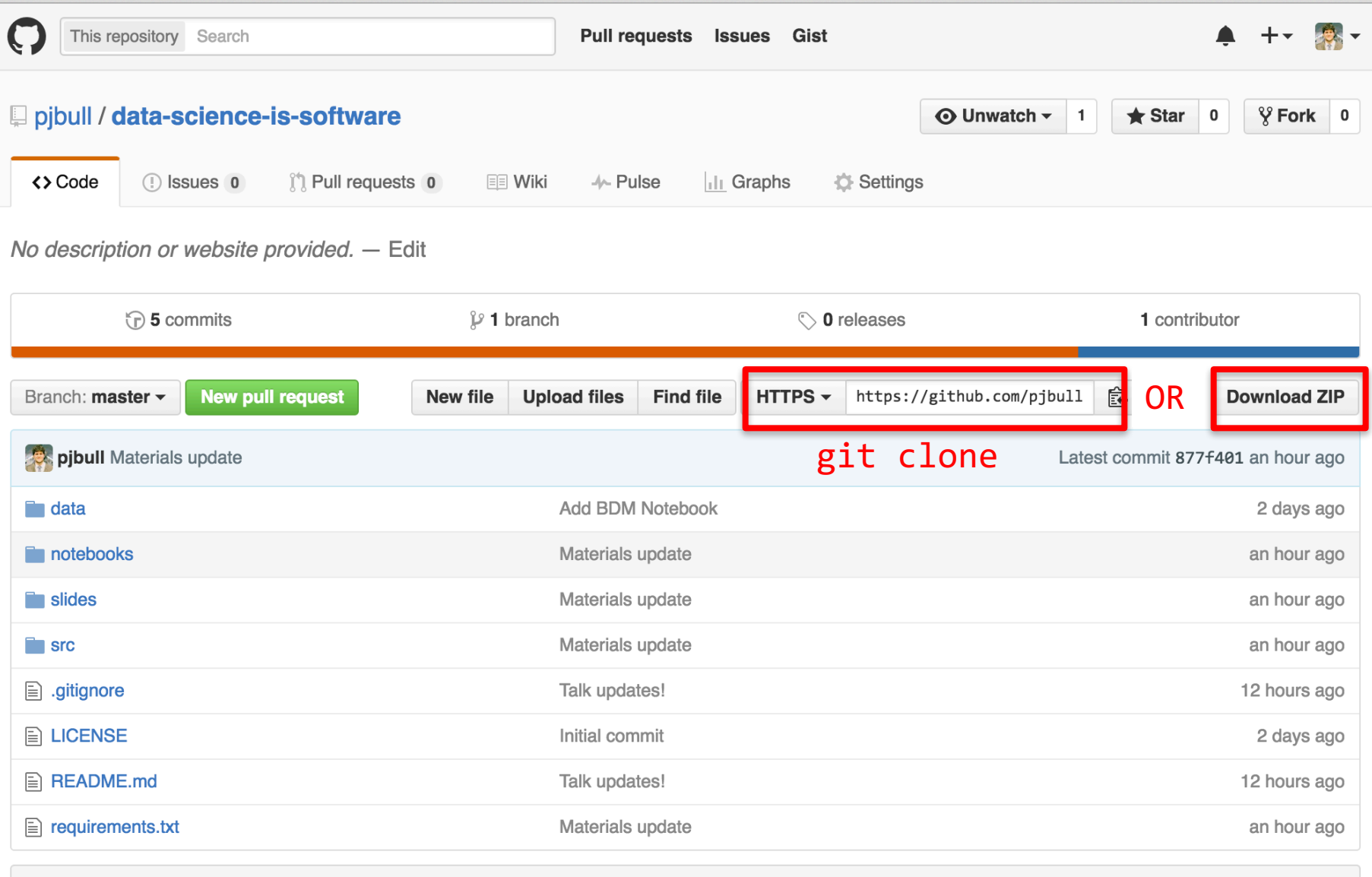
Data Science is Software:

Developer #lifehacks for the Jupyter Data Scientist

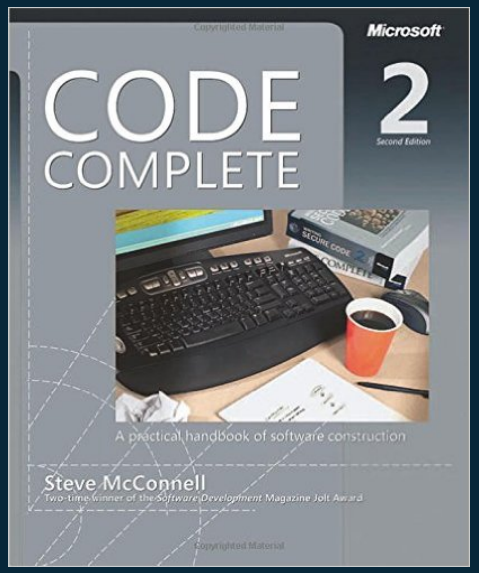
PETER BULL
@drivendataorg / @pjbull

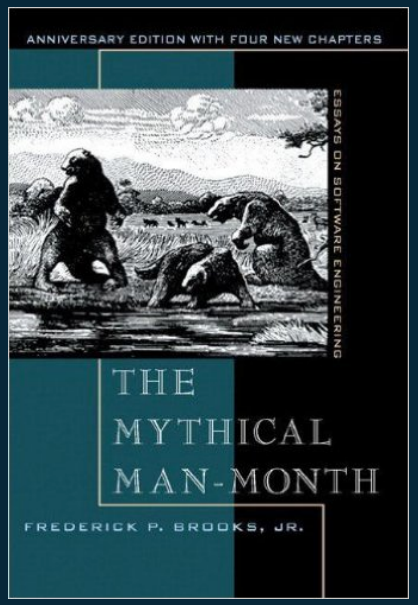
bit.ly/jupyter-lifehacks

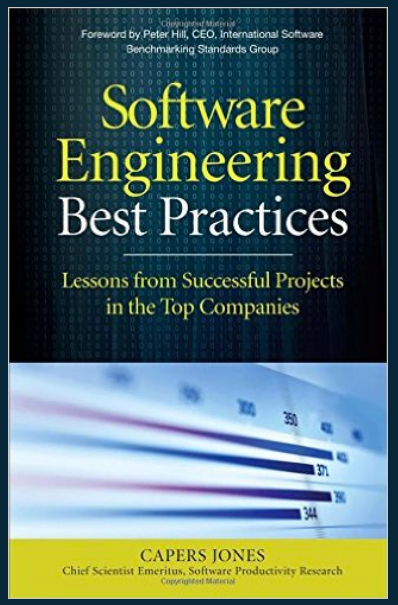


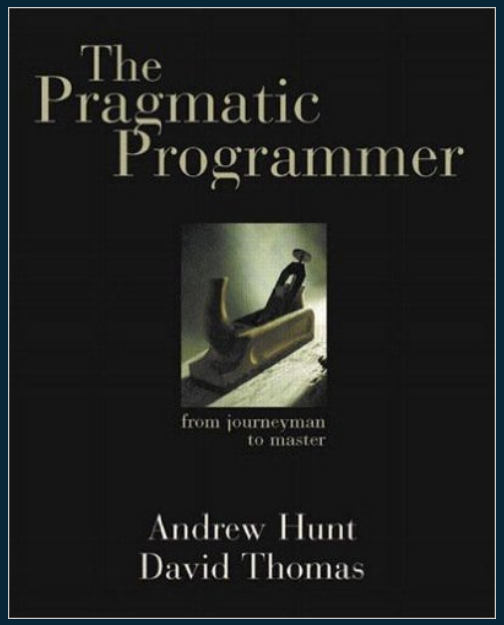
“It's easy to enhance a FORTRAN compiler to compile COBOL as well; it's just a SMOP.”

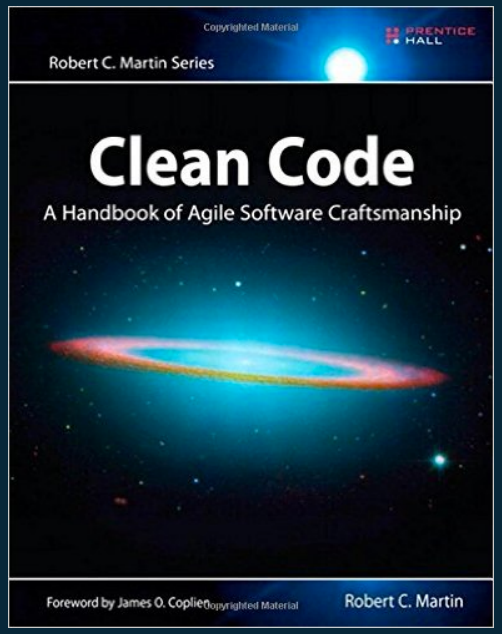
— SMOP entry in The Jargon File (a comprehensive compendium of hacker slang)

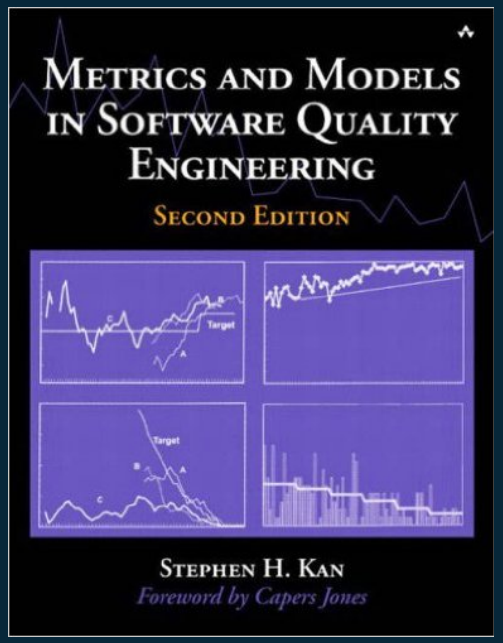

Microsoft
CODE COMPLETE
2
Second Edition
A practical handbook of software construction
Steve McConnell
Two-time winner of the Software Development Magazine Jolt Award


ANNIVERSARY EDITION WITH FOUR NEW CHAPTERS
ESSAYS ON SOFTWARE ENGINEERING
THE MYTHICAL MAN-MONTH
FREDERICK P. BROOKS, JR.


Foreword by Peter Hill, CEO, International Software Benchmarking Standards Group
Software Engineering Best Practices
Lessons from Successful Projects in the Top Companies
CAPERS JONES
Chief Scientist Emeritus, Software Productivity Research


The Pragmatic Programmer
from journeyman to master
Andrew Hunt
David Thomas


Robert C. Martin Series
PRENTICE HALL
Clean Code
A Handbook of Agile Software Craftsmanship
Foreword by James O. Coplien
Robert C. Martin

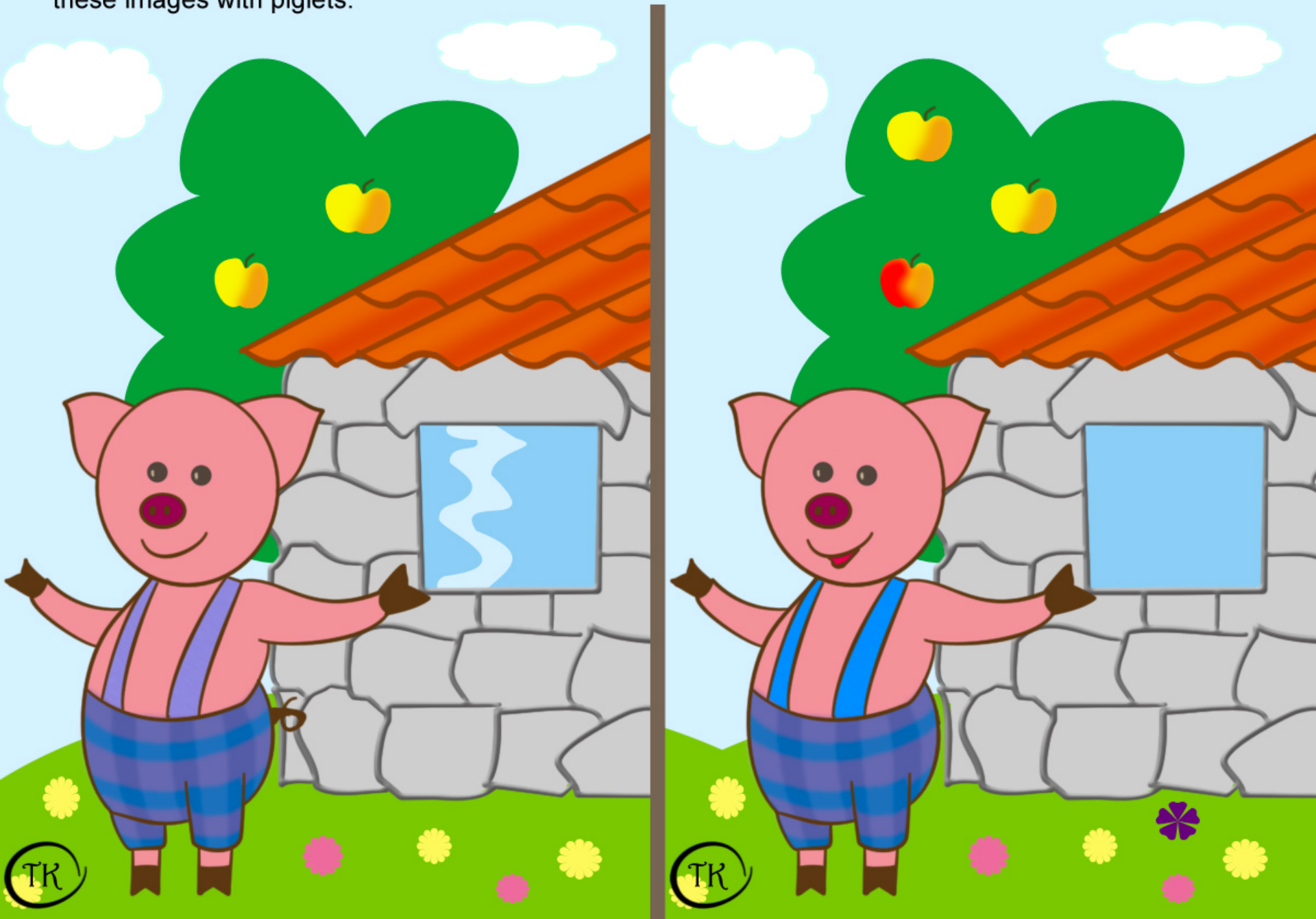

METRICS AND MODELS IN SOFTWARE QUALITY ENGINEERING
SECOND EDITION
STEPHEN H. KAN
Foreword by Capers Jones

1

This is my house

Spot 7 differences between these images with piglets.

http://www.everydayok.com




MRE
MEALS, RE
WITH FLAMELES
PRODUCT OF U.S.A
12
MEALS
THIS CASE CONTAINS TWE
EACH MEAL IS SEALED IN AN AIRTIGHT
• ENTRÉE • SIDE DISH • BREAD • SPREAD
• CONDIMENTS • INSTANT COFFEE • SPO
• STORE AT OR BELOW 72°F FOR 5 YEAR SHELF LIFE
• SEE BOTTOM FOR DATE OF MANUFACTURE
MRE
THREE-COURSE
MEAL, READY TO EAT
WITH FLAMELESS RATION HEATER
MEAL KIT
SUPPLY


jupyter

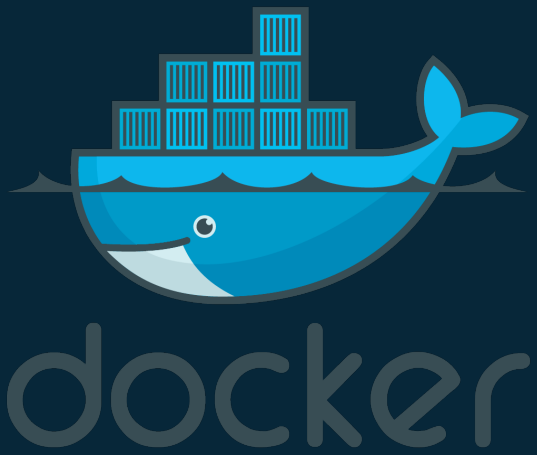
WATERMARK

VIRTUALENV

VIRTUALENVWRAPPER

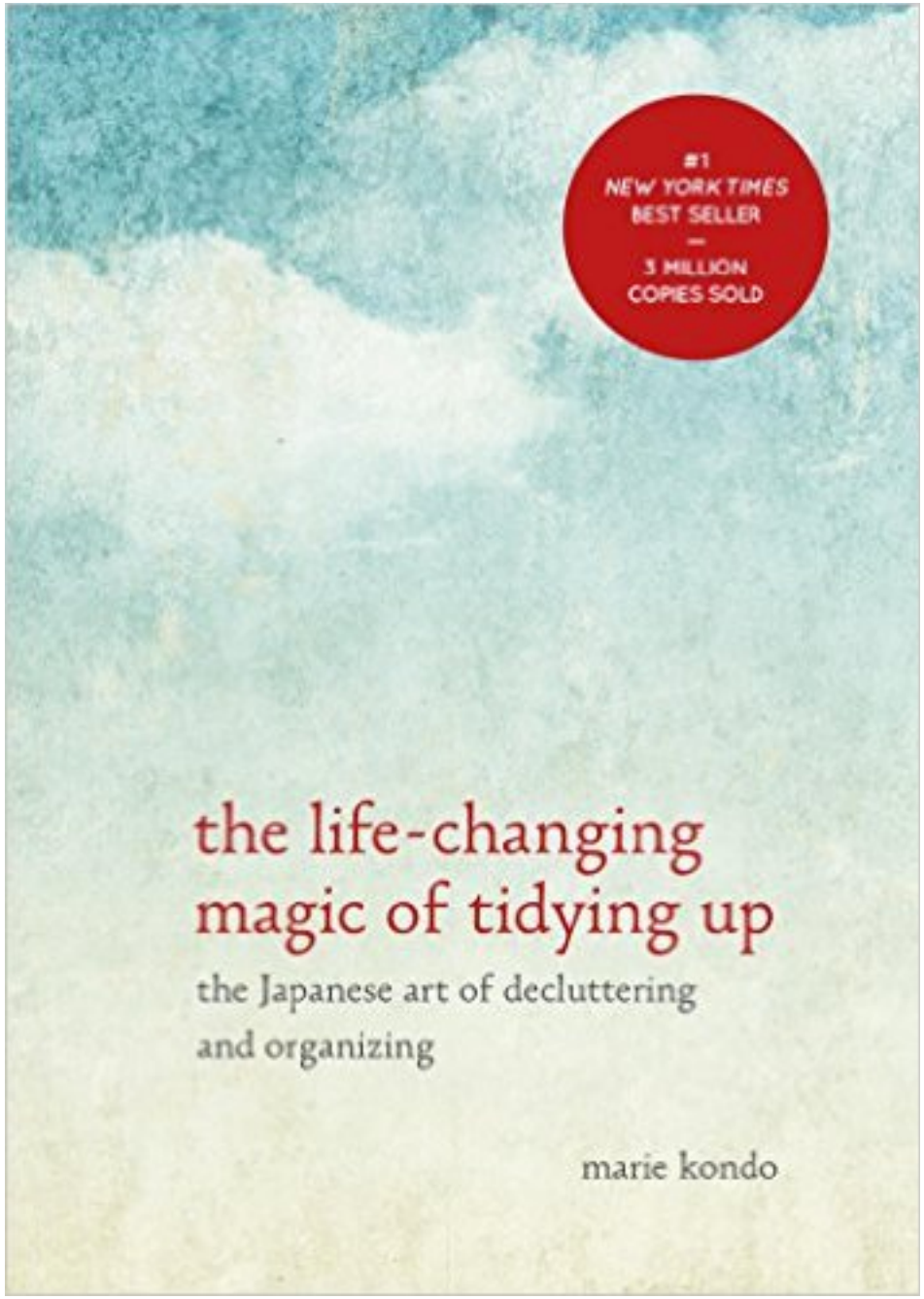
PIP REQUIREMENTS.TXT

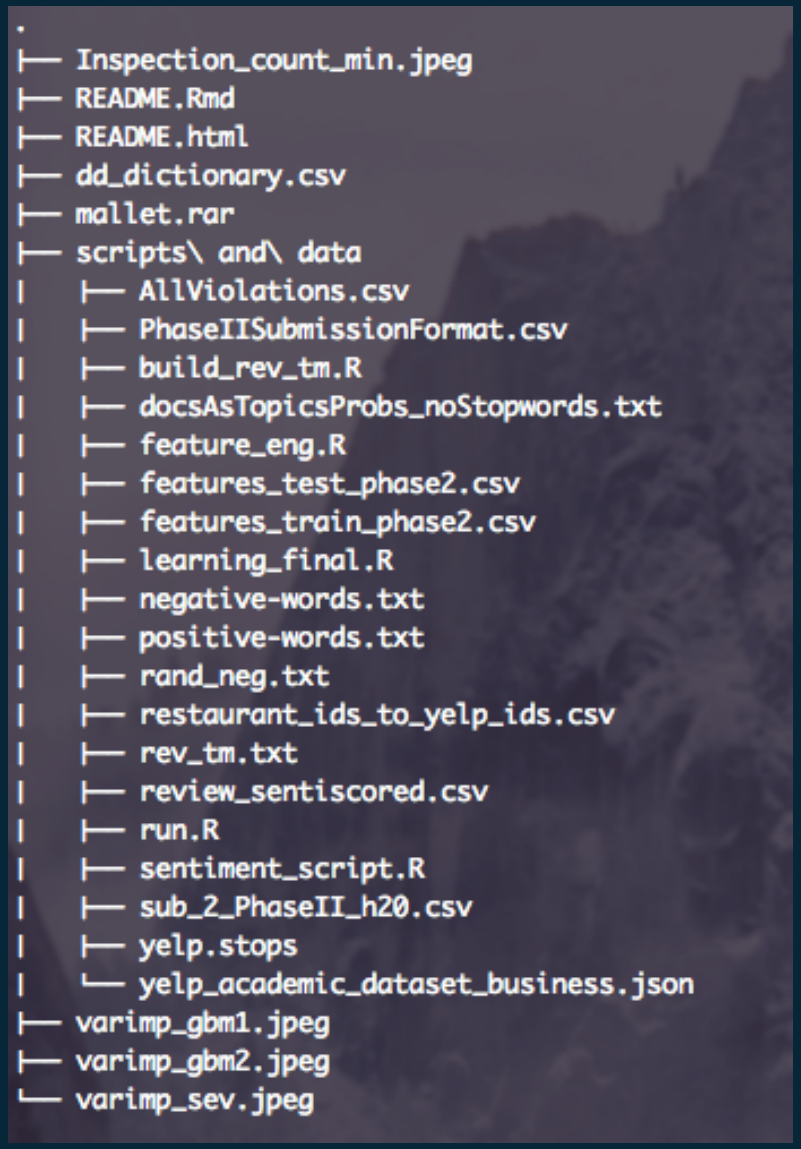
Other options (for more complex environments)

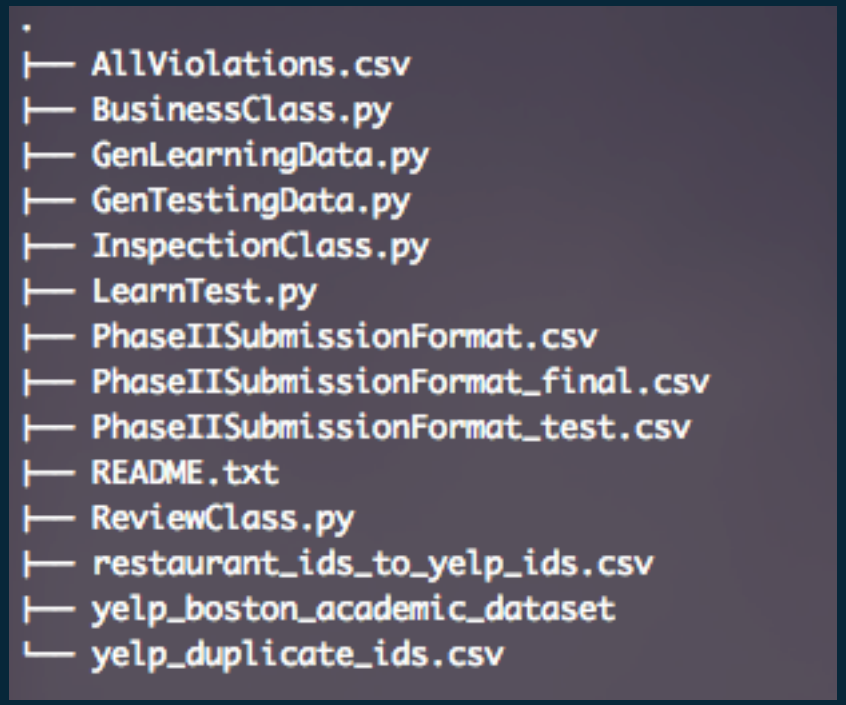


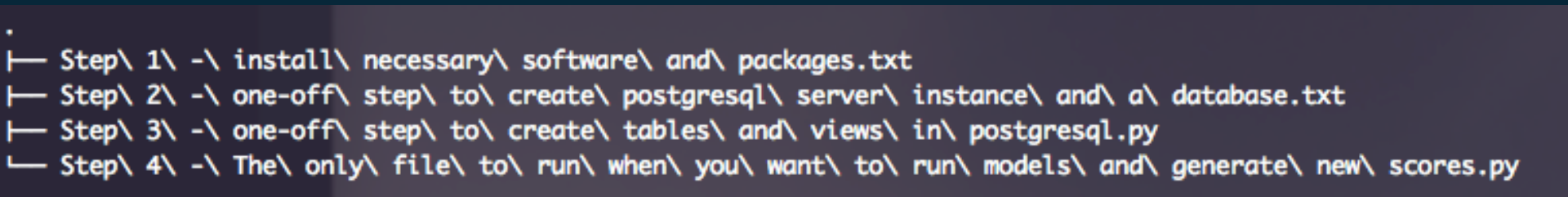
2

The Life-Changing Magic of Tidying Up

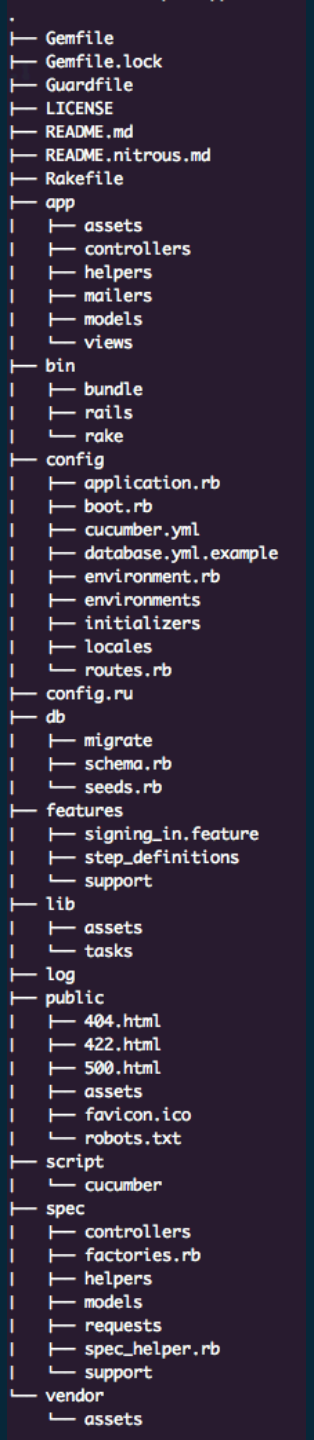

#1
NEW YORK TIMES
BEST SELLER
—
3 MILLION
COPIES SOLD
the life-changing
magic of tidying up
the Japanese art of decluttering
and organizing
marie kondo


.
├── Inspection_count_min.jpeg
├── README.Rmd
├── README.html
├── dd_dictionary.csv
├── mallet.rar
├── scripts\ and\ data
| ├── AllViolations.csv
| ├── PhaseIISubmissionFormat.csv
| ├── build_rev_tm.R
| ├── docsAsTopicsProbs_noStopwords.txt
| ├── feature_eng.R
| ├── features_test_phase2.csv
| ├── features_train_phase2.csv
| ├── learning_final.R
| ├── negative-words.txt
| ├── positive-words.txt
| ├── rand_neg.txt
| ├── restaurant_ids_to_yelp_ids.csv
| ├── rev_tm.txt
| ├── review_sentiscored.csv
| ├── run.R
| ├── sentiment_script.R
| ├── sub_2_PhaseII_h20.csv
| ├── yelp.stops
| └── yelp_academic_dataset_business.json
├── varimp_gbm1.jpeg
├── varimp_gbm2.jpeg
└── varimp_sev.jpeg

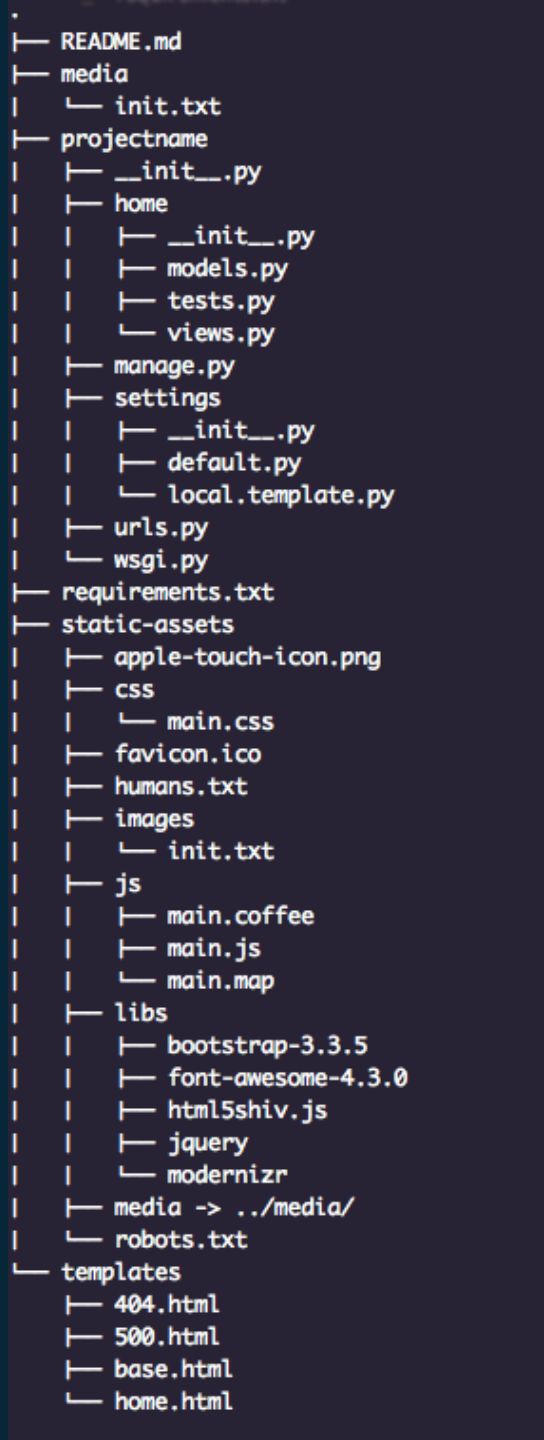

.
├── AllViolations.csv
├── BusinessClass.py
├── GenLearningData.py
├── GenTestingData.py
├── InspectionClass.py
├── LearnTest.py
├── PhaseIISubmissionFormat.csv
├── PhaseIISubmissionFormat_final.csv
├── PhaseIISubmissionFormat_test.csv
├── README.txt
├── ReviewClass.py
├── restaurant_ids_to_yelp_ids.csv
├── yelp_boston_academic_dataset
└── yelp_duplicate_ids.csv


.
├── Step\ 1\ -\ install\ necessary\ software\ and\ packages.txt
├── Step\ 2\ -\ one-off\ step\ to\ create\ postgresql\ server\ instance\ and\ a\ database.txt
├── Step\ 3\ -\ one-off\ step\ to\ create\ tables\ and\ views\ in\ postgresql.py
└── Step\ 4\ -\ The\ only\ file\ to\ run\ when\ you\ want\ to\ run\ models\ and\ generate\ new\ scores.py

Ruby on Rails Application



Django Application



Cookiecutter Data Science

A logical, reasonably standardized, but flexible project structure for doing and sharing data science work.

Why use this project structure?

> We're not talking about bikeshedding the indentation aesthetics or pedantic formatting standards — ultimately, data science code quality is about correctness and reproducibility.

When we think about data analysis, we often think just about the resulting reports, insights, or visualizations. While these end products are generally the main event, it's easy to focus on making the products *look nice* and ignore the *quality of the code that generates them*. Because these end products are created programmatically, **code quality is still important**! And we're not talking about bikeshedding the indentation aesthetics or pedantic formatting standards — ultimately, data science code quality is about correctness and reproducibility.

It's no secret that good analyses are often the result of very scattershot and serendipitous explorations. Tentative experiments and rapidly testing approaches that might not work out are all part of the process for getting to the good stuff, and there is no magic bullet to turn data exploration into a simple, linear progression.

That being said, once started it is not a process that lends itself to thinking carefully about the structure of your code or project layout, so it's best to start with a clean, logical structure and stick to it throughout. We think it's a pretty big win all around to use a fairly standardized setup like this one. Here's why:

drivendata.github.io/cookiecutter-data-science

```
├── 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`
│
├── 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.testrun.org
```

3

Edit-run-repeat: Stopping the cycle of pain


jupyter

AUTORELOAD, PDB, DEBUG

Q, MODULES

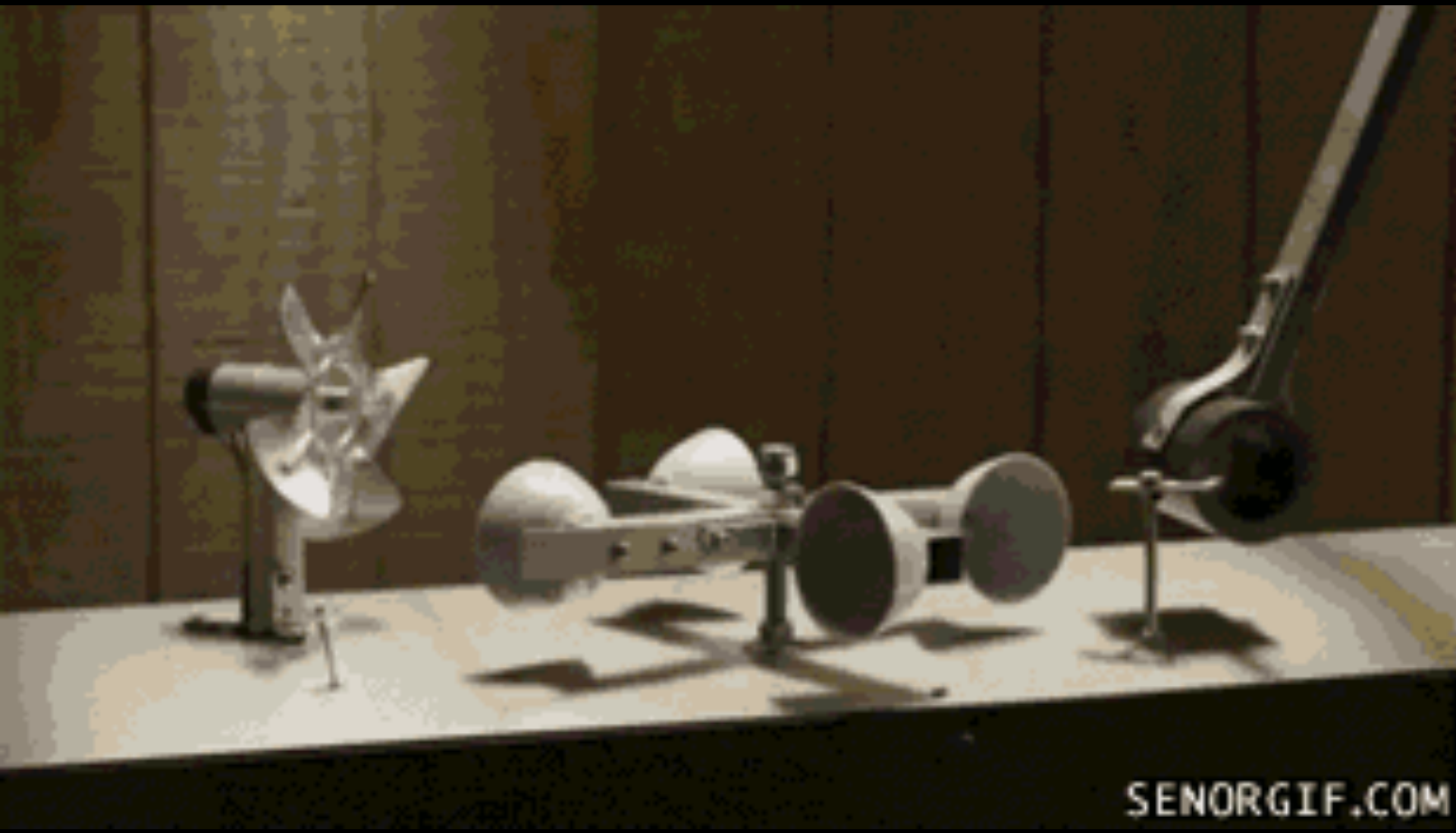
ASSERT, UNITTEST

ENGARDE, NUMPY.TESTING

Next-level code inspection


Town
8200
10
Save
Load


jupyter


SENORGIF.COM


jupyter


jupyter

COVERAGE.PY

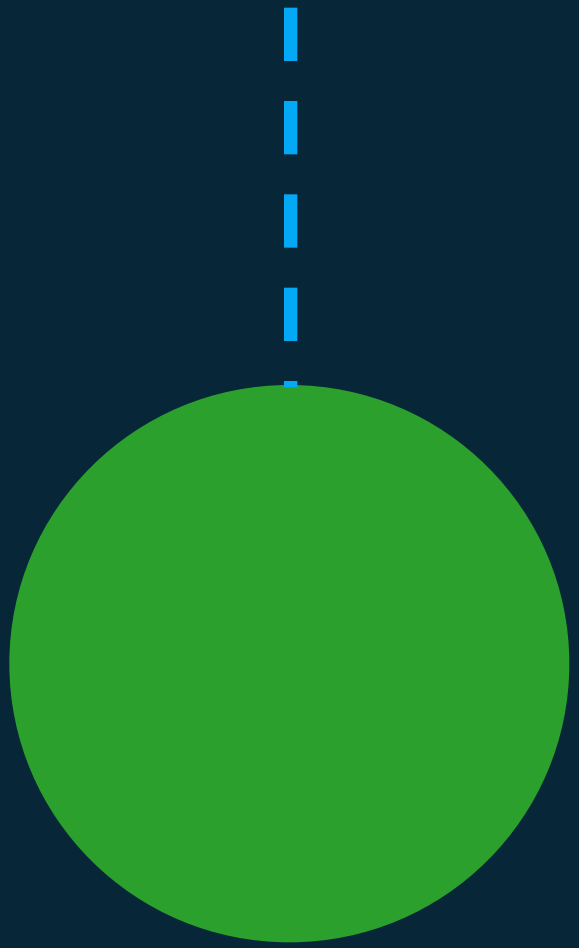
%PRUN

PYCHARM

FLAKE8 LINTING FOR SUBLIME

Not Covered

Version Control	git
Code review	git branch + pull requests
Branching strategy	GitHub Flow
Issue tracking	GitHub Issues + waffle.io
Automating workflows	Make
Documentation Generator	Sphinx
Docstrings	Sphinx ReST
Style guide	flake8
Notebook diffing	nbconvert to .py on save
Configuration isolation	config.py



Questions?

DRIVENDATA
@drivendataorg / @pjbull
peter@drivendata.org