Creative Jupyter

Tools for literate computing, exploration, collaboration.

> Pycon Balkan 2019, 2019-10-04, 11:30-12:00, Belgrade
>
> Martin Czygan, gh@miku

About Me

- Software Developer, Data Engineer, Leipzig University Library (DE), gh/miku
- Part-time: Author, Trainer

Personal History

- IPython REPL for daily work (API exploration, data digging, ...)
- Jupyter for Python courses
- Jupyter for semi-ad-hoc analysis
- Collected material for book on Jupyter, but still long way to go ...
- Regular exposure to data related problems

Creativity in Context

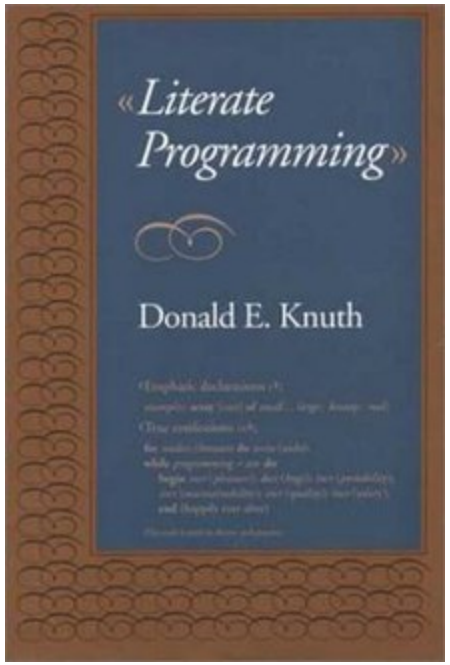
- have **material** available
- have **tools** available
- progress and **trial and error**
- comments and **collaboration**

Programming Context

Translated to programming and data context:

- (1) materials (data)
- (2) tools for exploration (languages, libraries, extensions)
- (3) experiments, allow for failure (version control)
- (4) comments and sharing (packaging, sharing, reproducibility)

Literate programming



- original paper from Knuth (1984)
- similar but different idea: write one document, generate executable and documentation from the same source
- alternative names: literate computing, interactive computing

(1) Materials

- What data you want to access?
- How do you access data in notebooks?

(1) Materials

Example data access scenarios:

> A company wants to run **business analysis** across data from different databases (articles, customers, ...) or other systems (search, API)

> Exploratory analysis in **scientific context**, e.g. weather data, ...

> Data **journalism**.

(1) Material

Options to make raw data available:

- download file and put it the repo (e.g. under `data` folder)
- have a `download.sh` script fetching larger data sets
- expose `obj.load()` or `obj.load_data()` functions on dataset helpers
- factor out an **adapter library**, that will allow to access data quickly

(1) Material

Options for other artifacts; e.g. ML toolkits (sklearn, tensorflow.datasets, keras, pytorch) include utilities for varied formats (tarball, npz, ...).

```
In [3]: _ = cifar10.load_data()
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz

In [4]: from keras.datasets import reuters

In [5]: x = reuters.load_data()
Downloading data from https://s3.amazonaws.com/text-datasets/reuters.npz
```

(1) Material Example

Example:

> Performance evaluation of various search systems: catalog frontends, search APIs. Wrote small wrapper library (about 600 SLOC), exposing objects with a common interface (e.g. a `search` method, taking a query string, returning results and metadata from 3 kinds of systems).

Quick generation of input data on the fly, then open up questions:

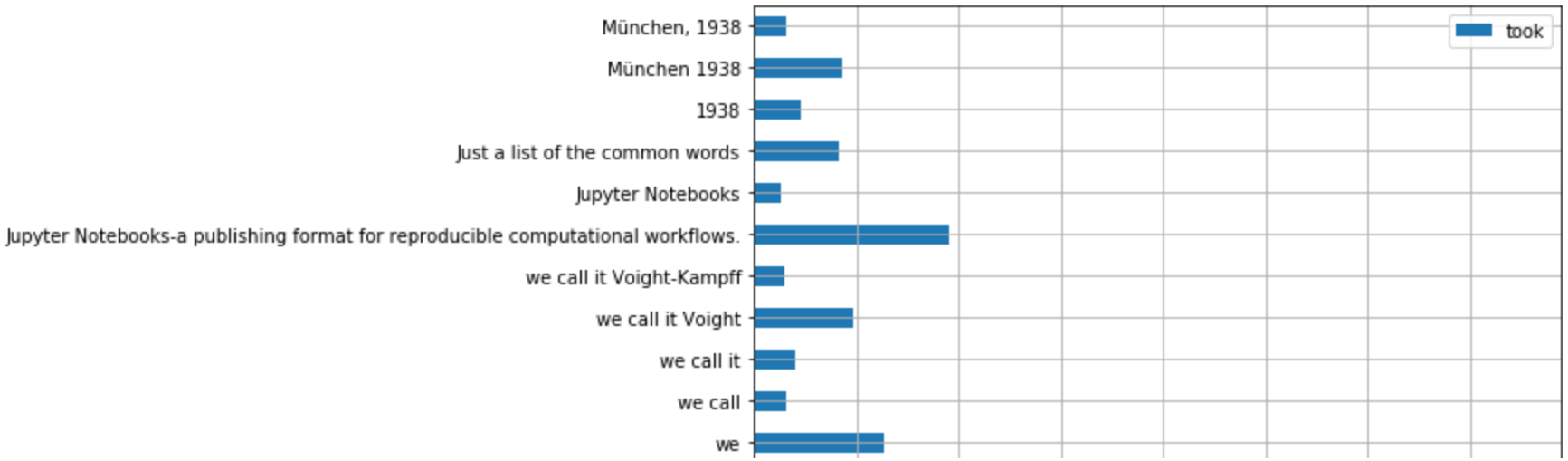
- Performance
- Query result overlaps and similarity
- ...

(1) Material Example

```
In [13]: stimes = [(q, ubl.search(q).get('took')) for q in tqdm.tqdm(queries)]
100%|██████████| 26/26 [02:03<00:00,  4.74s/it]

In [14]: df = pd.DataFrame(stimes, columns=["query", "took"])
         df.set_index("query", drop=True, inplace=True)
         df.plot(kind='barh', figsize=(8, 11), grid=True)

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x7fa6cc140fd0>
```



(1) Material Example

Advantages:

- data access module independent of notebook (e.g. usable in IPython or any other script)
- notebooks are readable documents (broader audience), no extra report needed
- adjustable

(1) Material

Generic ways to package data:

- data package, spec
- json stat

> **Data dissemination is not the business of a few anymore**. Even though the JSON-stat format can be the perfect companion for the open data initiatives of National Statistical Offices, it is suitable for all kinds of data disseminators because it has been designed with simplicity in mind.

(1) Material Demo

- Making data accessible quickly.
- Example: demos/Accessing JSON Stat.ipynb

(2) Tools

For most things, you will need an external library. Basic Python practices apply, among them: try to isolate your environment:

- documentation
- reproducibility

(2) Tools

Can be as simple as a watermark, a Jupyter magic `%` extension to timestamp a notebook along with SW and HW information.

```
%watermark
```

```
2016-01-29T22:55:57

CPython 3.5.1
IPython 4.0.3

compiler   : GCC 4.2.1 (Apple Inc. build 5577)
system     : Darwin
release    : 15.3.0
machine    : x86_64
processor  : i386
CPU cores  : 4
interpreter: 64bit
```

(2) Tools

> Make it simple to go from a repo to a notebook locally.

When using a virtual environment.

Setup.

```
$ python -m ipykernel install --user --name=myenv
Installed kernelspec myenv in $HOME/.local/share/jupyter/kernels/myenv
```

Teardown.

```
$ jupyter kernelspec uninstall myenv
```

(3) Experiments

Although notebooks are in a textual format (JSON), they are hard to version control, since there is some state associated with the execution.

(3) Experiments

There are tools to work around that, e.g. Notedown and Jupytext.

- Notedown, write markdown and convert to a notebook
- Jupytext, associate Markdown file with Notebook and sync them
- nbstripout, clean output cells

(3) Notedown

- convert between various formats (markdown, Rmd, ipynb)
- works, albeit maybe less maintained, with some forks, e.g. knitty
- demo

(3) Jupytext

- bidirectional
- uses the contents API
- allows to create plaintext workflows, using Python, Markdown and a few other formats
- programmatic access
- demo, conversion, programmatic access

(3) nbstripout

- strips output cell
- can be part a pre-commit hook
- demo

(4) Collaboration

- Run a server and use token to share a workspace (e.g. in a container)
- Run JupyterHub -- TLJH (1-100 users, support for ARM, WIP)

> JupyterHub brings the power of notebooks to groups of users. It gives users access to computational environments and resources without burdening the users with installation and maintenance tasks.

(4) The littlest JupyterHub (TLJH)

- 1-100 users
- runs on Ubuntu 18.04

(4) Binder

> Turn a Git repo into a collection of interactive notebooks.

> Have a repository full of Jupyter notebooks? With Binder, open those notebooks in an executable environment, making your code immediately reproducible by anyone, anywhere.

(4) Turn notebooks into containers

- repo2docker

> repo2docker can build a reproducible computational environment for any repository that follows The Reproducible Execution Environment Specification.

It looks for a various configuration files.

- demo, generation of a Dockerfile

(4) Install as Cloud Native application

- Kubeflow

> What is Kubeflow? The Kubeflow project is dedicated to making deployments of machine learning (ML) workflows on Kubernetes simple, portable and scalable.

> Enabling easier **notebook sharing** across the organization. Users can create notebook containers or pods directly in the cluster, rather than locally on their workstations.

- Jupyter notebooks as experimental setup
- Data volumes can be mounted
- Tries to model end-to-end workflow, up to e.g. ML model serving

(4) Export to various formats

Export Notebooks as slides, PDF and other formats with nbconvert.

- demo: PDF export

(4) Platforms

- bringing data and people together
- hide operational complexity, e.g. preparation of datasets, clusters

Examples:

- An internal data science platform based on Spark and Jupyter (blog)
- Quantopian Notebooks (https://www.quantopian.com/notebooks/survey)

> A hosted research environment with flexible data access, custom plotting, and post-hoc analysis on backtest and live trading results in an IPython Notebook.

Wrap up

- Jupyter Notebooks work standalone and have many extension points
- Both data integration and reproducibility features are getting better
- Formats available for version control
- Platforms emerge, domain-specific tasks, cloud and on premise

Thanks