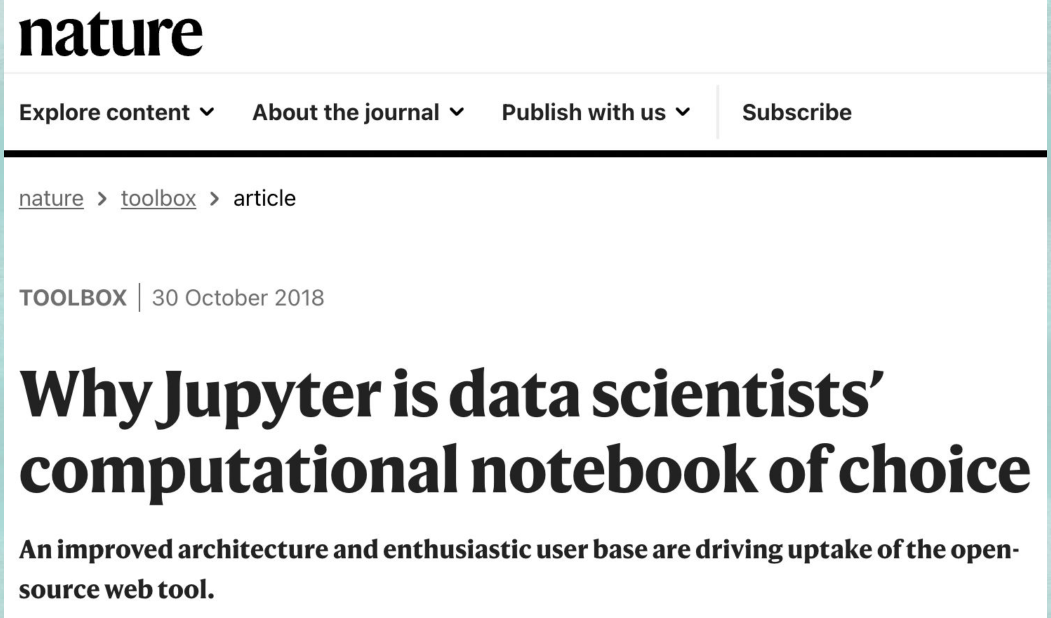
Making Jupyter Notebooks Less Awful

Laura Richter, PyConZA 2023

"de facto standard"

"Interactive computing"



TOOLBOX | 30 October 2018

Why Jupyter is data scientists' computational notebook of choice

An improved architecture and enthusiastic user base are driving uptake of the open-source web tool.

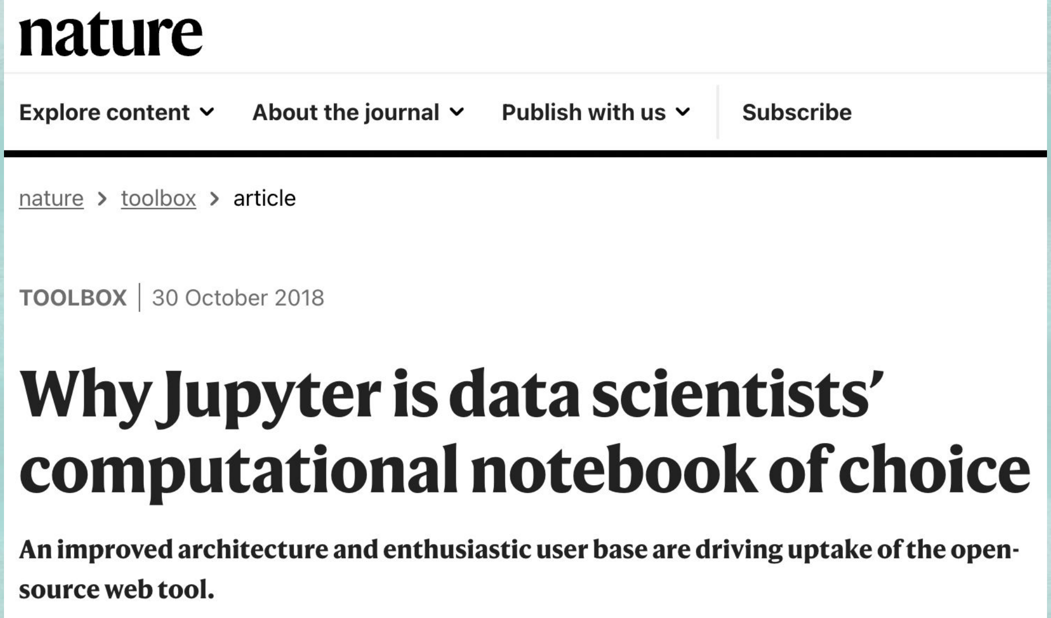
"computational narrative"

"killer app for teaching computing"

"powerful connections between topics, theories, data and results"

"move the computer to the data"

"Such tools foster computational reproducibility by simplifying code reuse."



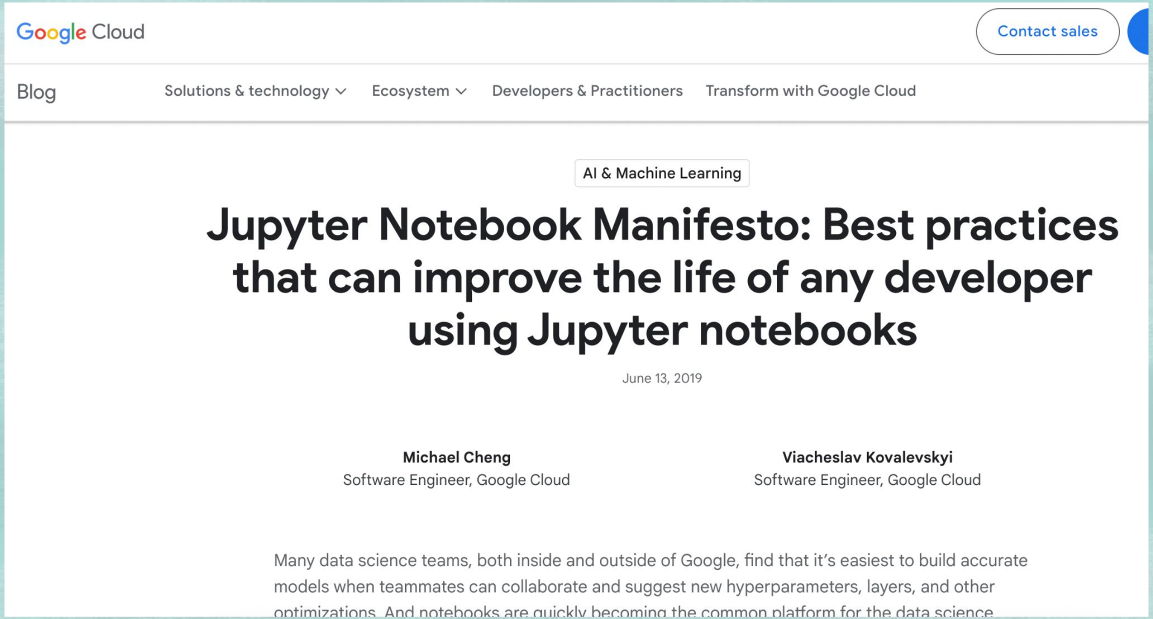
TOOLBOX | 30 October 2018

Why Jupyter is data scientists' computational notebook of choice

An improved architecture and enthusiastic user base are driving uptake of the open-source web tool.

"Jupyter notebooks also encourage poor coding practice … by making it difficult to organize code logically, break it into reusable modules and develop tests to ensure the code is working properly."

"notebooks do require discipline"



Jupyter Notebook Manifesto: Best practices that can improve the life of any developer using Jupyter notebooks

June 13, 2019

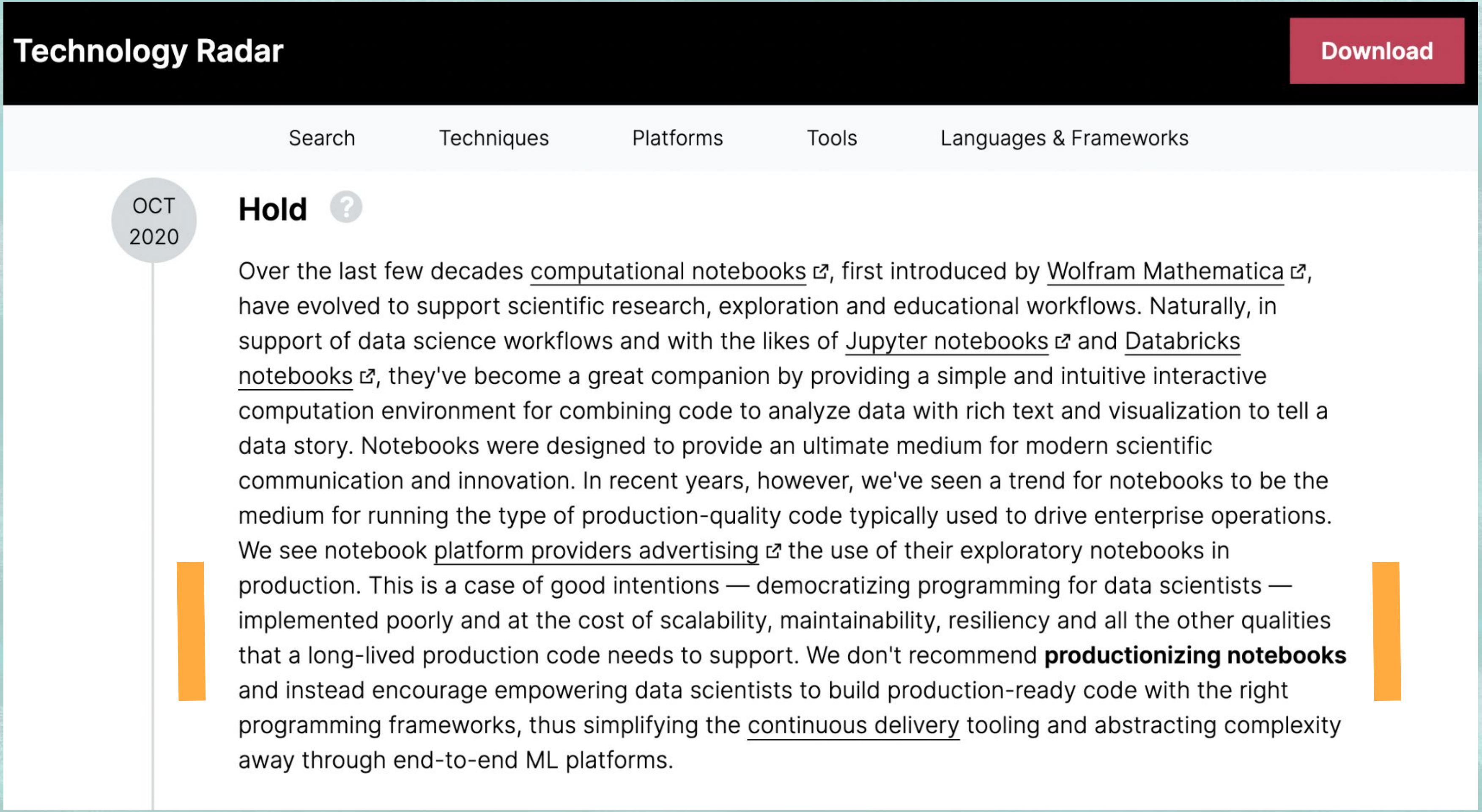
Michael Cheng
Software Engineer, Google Cloud

Viacheslav Kovalevskyi
Software Engineer, Google Cloud

Many data science teams, both inside and outside of Google, find that it's easiest to build accurate models when teammates can collaborate and suggest new hyperparameters, layers, and other optimizations. And notebooks are quickly becoming the common platform for the data science

"Because Jupyter Notebooks are a relatively recently-developed tool, they don't (yet) follow or encourage consensus-based software development best practices."

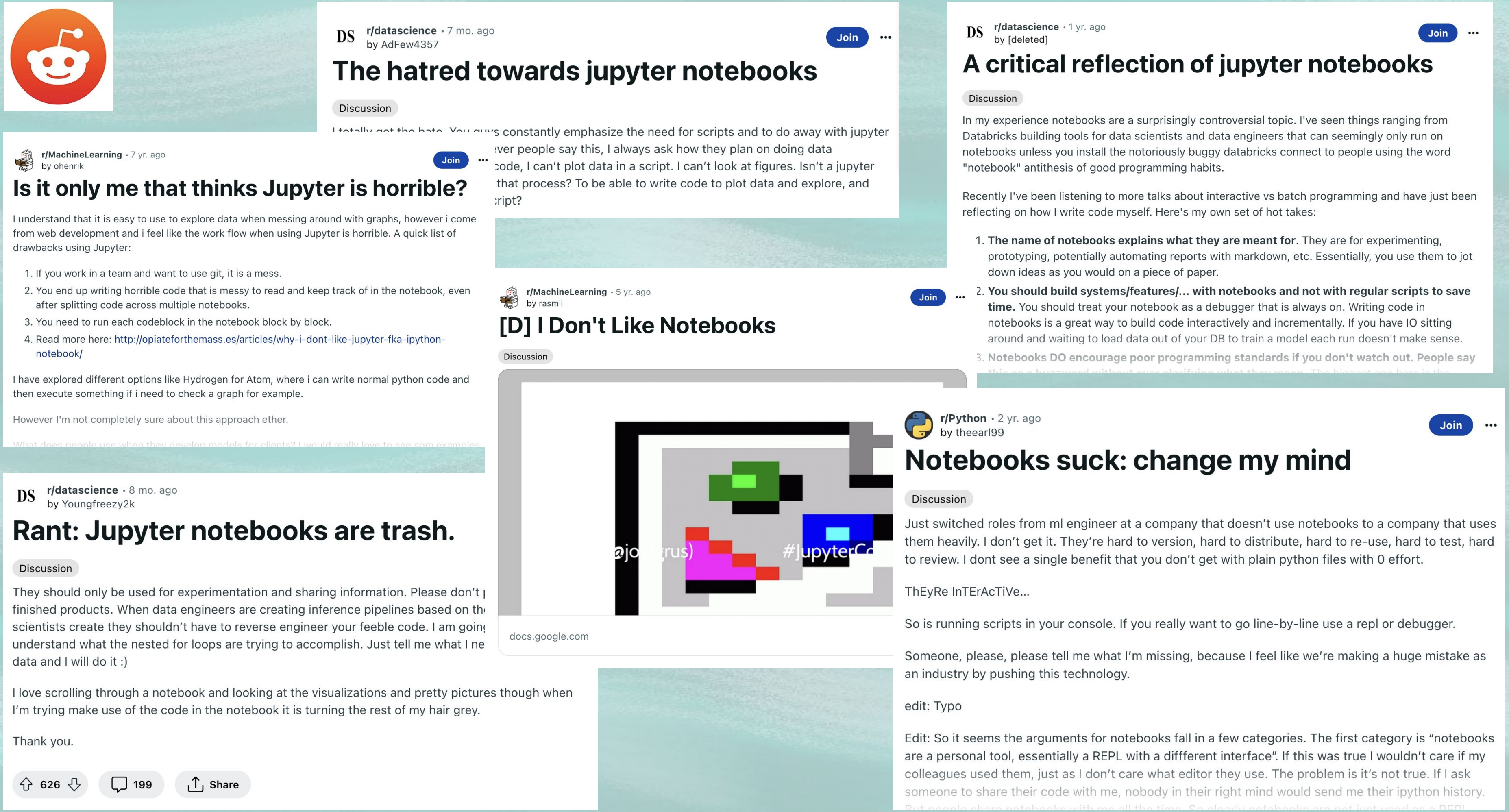
"Data scientists, typically collaborating on a small project that involves experimentation, often feel they don't need to adhere to any engineering best practices."



OCT 2020

Hold

Over the last few decades computational notebooks, first introduced by Wolfram Mathematica, have evolved to support scientific research, exploration and educational workflows. Naturally, in support of data science workflows and with the likes of Jupyter notebooks and Databricks notebooks, they've become a great companion by providing a simple and intuitive interactive computation environment for combining code to analyze data with rich text and visualization to tell a data story. Notebooks were designed to provide an ultimate medium for modern scientific communication and innovation. In recent years, however, we've seen a trend for notebooks to be the medium for running the type of production-quality code typically used to drive enterprise operations. We see notebook platform providers advertising the use of their exploratory notebooks in production. This is a case of good intentions — democratizing programming for data scientists — implemented poorly and at the cost of scalability, maintainability, resiliency and all the other qualities that a long-lived production code needs to support. We don't recommend **productionizing notebooks** and instead encourage empowering data scientists to build production-ready code with the right programming frameworks, thus simplifying the continuous delivery tooling and abstracting complexity away through end-to-end ML platforms.



r/datascience • 7 mo. ago
by AdFew4357

Join ···

The hatred towards jupyter notebooks

Discussion

I totally get the hate. You guys constantly emphasize the need for scripts and to do away with jupyter
ever people say this, I always ask how they plan on doing data
code, I can't plot data in a script. I can't look at figures. Isn't a jupyter
that process? To be able to write code to plot data and explore, and
ript?

r/datascience • 1 yr. ago
by [deleted]

Join ···

A critical reflection of jupyter notebooks

Discussion

In my experience notebooks are a surprisingly controversial topic. I've seen things ranging from Databricks building tools for data scientists and data engineers that can seemingly only run on notebooks unless you install the notoriously buggy databricks connect to people using the word "notebook" antithesis of good programming habits.

Recently I've been listening to more talks about interactive vs batch programming and have just been reflecting on how I write code myself. Here's my own set of hot takes:

1. **The name of notebooks explains what they are meant for**. They are for experimenting, prototyping, potentially automating reports with markdown, etc. Essentially, you use them to jot down ideas as you would on a piece of paper.
2. **You should build systems/features/... with notebooks and not with regular scripts to save time.** You should treat your notebook as a debugger that is always on. Writing code in notebooks is a great way to build code interactively and incrementally. If you have IO sitting around and waiting to load data out of your DB to train a model each run doesn't make sense.
3. **Notebooks DO encourage poor programming standards if you don't watch out. People say** this as a buzzword without ever clarifying what they mean. The biggest one here is the

r/MachineLearning • 7 yr. ago
by ohenrik

Join ···

Is it only me that thinks Jupyter is horrible?

I understand that it is easy to use to explore data when messing around with graphs, however i come from web development and i feel like the work flow when using Jupyter is horrible. A quick list of drawbacks using Jupyter:

1. If you work in a team and want to use git, it is a mess.
2. You end up writing horrible code that is messy to read and keep track of in the notebook, even after splitting code across multiple notebooks.
3. You need to run each codeblock in the notebook block by block.
4. Read more here: http://opiateforthemass.es/articles/why-i-dont-like-jupyter-fka-ipython-notebook/

I have explored different options like Hydrogen for Atom, where i can write normal python code and then execute something if i need to check a graph for example.

However I'm not completely sure about this approach ether.

What does people use when they develop models for clients? I would really love to see som examples

r/MachineLearning • 5 yr. ago
by rasmii

Join ···

[D] I Don't Like Notebooks

Discussion

docs.google.com

r/datascience • 8 mo. ago
by Youngfreezy2k

Rant: Jupyter notebooks are trash.

Discussion

They should only be used for experimentation and sharing information. Please don't p
finished products. When data engineers are creating inference pipelines based on th
scientists create they shouldn't have to reverse engineer your feeble code. I am going
understand what the nested for loops are trying to accomplish. Just tell me what I ne
data and I will do it :)

I love scrolling through a notebook and looking at the visualizations and pretty pictures though when I'm trying make use of the code in the notebook it is turning the rest of my hair grey.

Thank you.

626 199 Share

r/Python • 2 yr. ago
by theearl99

Join ···

Notebooks suck: change my mind

Discussion

Just switched roles from ml engineer at a company that doesn't use notebooks to a company that uses them heavily. I don't get it. They're hard to version, hard to distribute, hard to re-use, hard to test, hard to review. I dont see a single benefit that you don't get with plain python files with 0 effort.

ThEyRe InTErAcTiVe…

So is running scripts in your console. If you really want to go line-by-line use a repl or debugger.

Someone, please, please tell me what I'm missing, because I feel like we're making a huge mistake as an industry by pushing this technology.

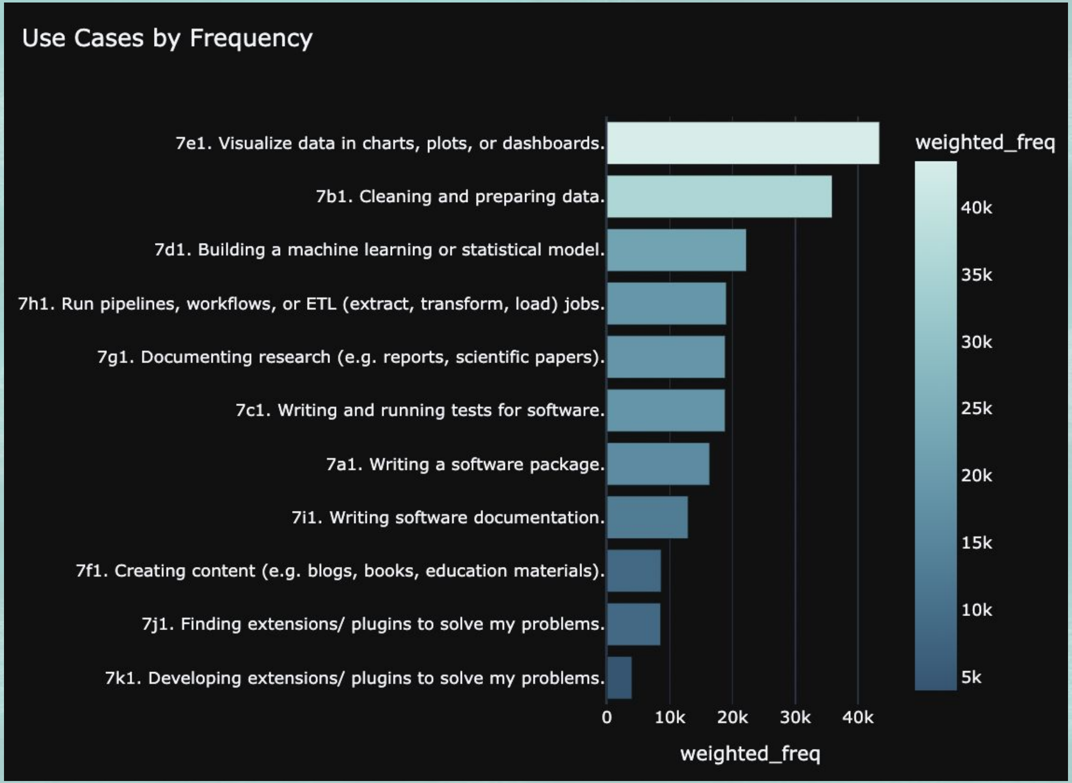
edit: Typo

Edit: So it seems the arguments for notebooks fall in a few categories. The first category is "notebooks are a personal tool, essentially a REPL with a diffferent interface". If this was true I wouldn't care if my colleagues used them, just as I don't care what editor they use. The problem is it's not true. If I ask someone to share their code with me, nobody in their right mind would send me their ipython history. But people share notebooks with me all the time. So clearly notebooks are not just used as a REPL

So why use Jupyter?

Prototype, Experiment,
Present, Teach, Learn,
Taking your compute to the data,
Anything that needs a visual result

So why use Jupyter?



2020 Jupyter Survey

When do you need good practice?

… larger project … many engineers …
… scaling … deployment …

When do you need good practice?

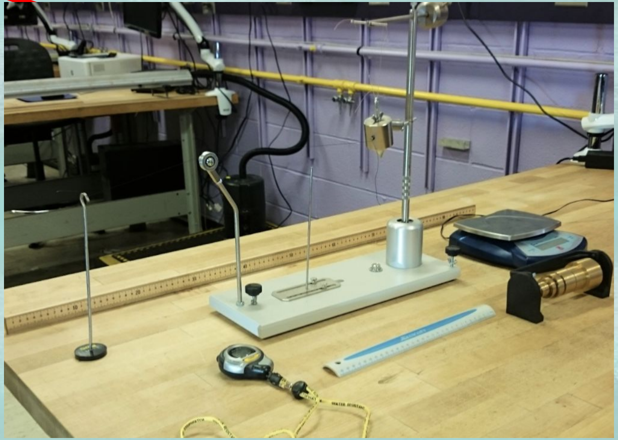
~~… larger project … many engineers …~~
~~… scaling … deployment …~~

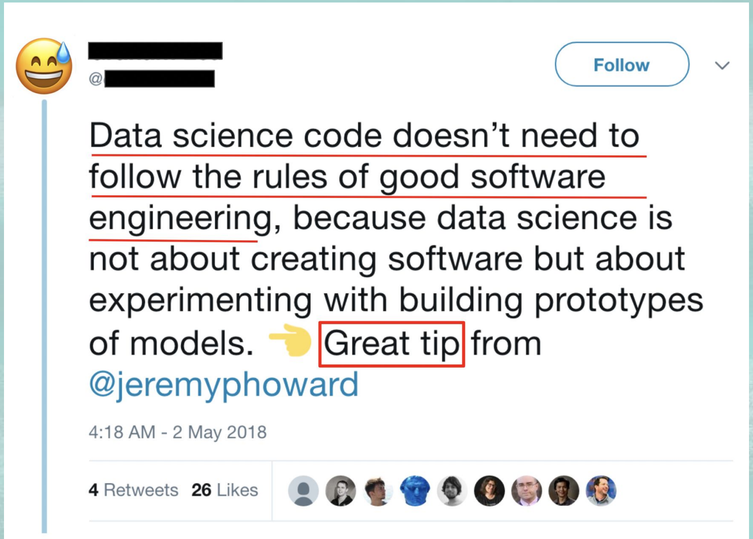
Anything you
need to be correct

When do you need good practice?

~~… larger project … many engineers …~~
~~… scaling … deployment …~~

Anything you
need to be correct





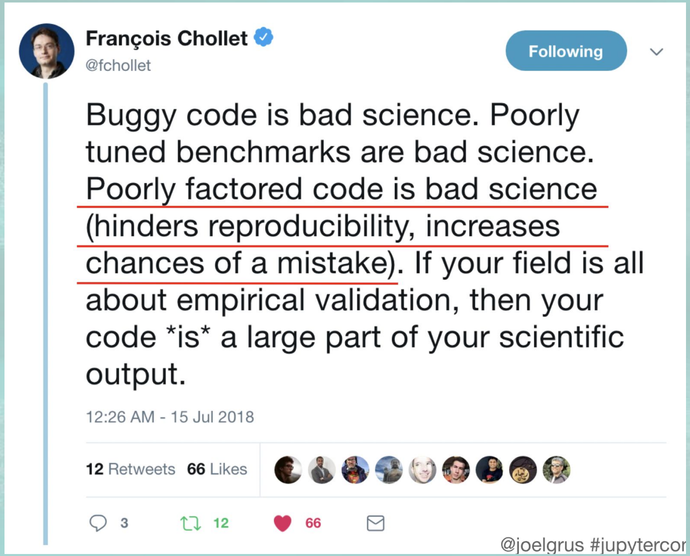
[redacted]

@[redacted]

Data science code doesn't need to follow the rules of good software engineering, because data science is not about creating software but about experimenting with building prototypes of models. 👈 Great tip from @jeremyphoward

4:18 AM - 2 May 2018

4 Retweets 26 Likes



François Chollet
@fchollet

Buggy code is bad science. Poorly tuned benchmarks are bad science. Poorly factored code is bad science (hinders reproducibility, increases chances of a mistake). If your field is all about empirical validation, then your code *is* a large part of your scientific output.

12:26 AM - 15 Jul 2018

12 Retweets 66 Likes

Stolen from "I don't like Notebooks", Joel Grus, JupyterCon 2018

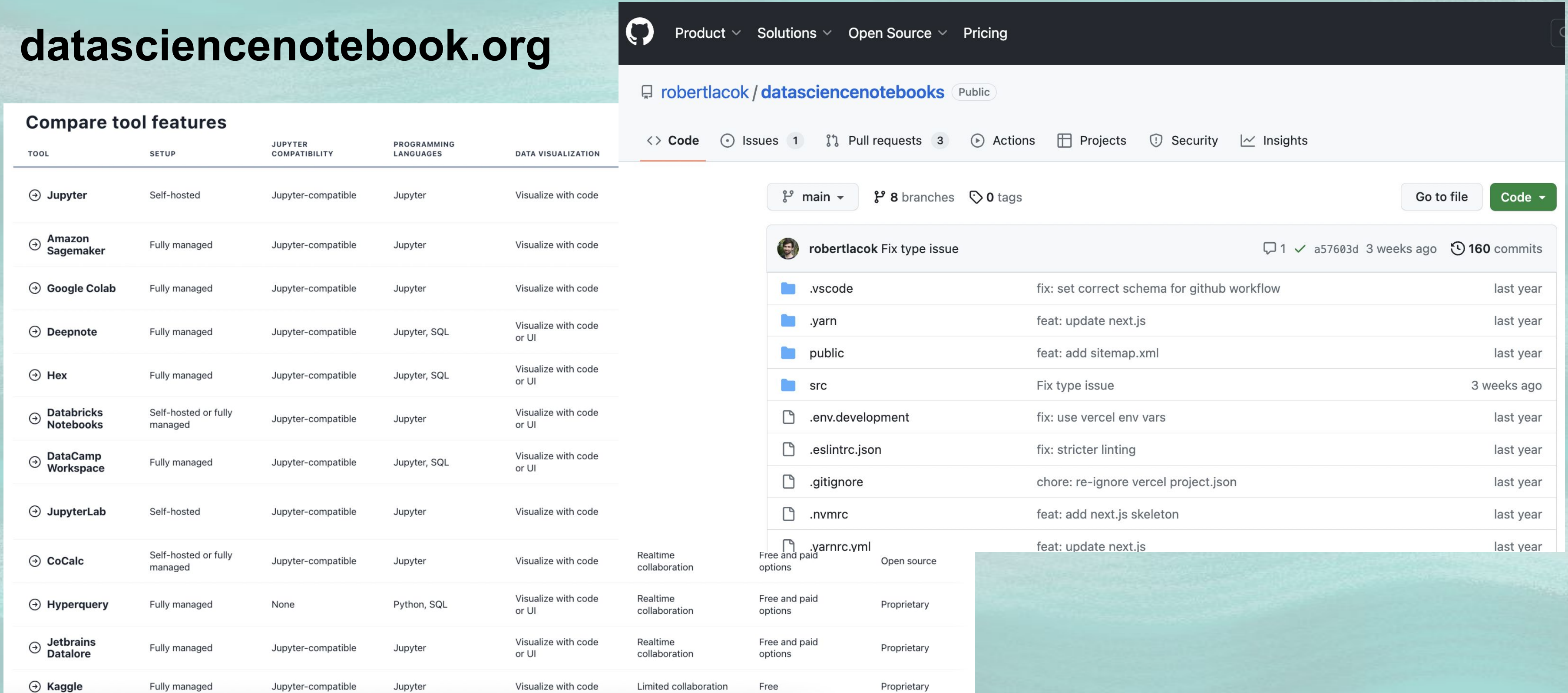
So how do we make Jupyter Notebooks more correct?

1. "Follow established software development best practices"(*)
2. Version control & Reviews
3. Testing
4. Automating testing and quality control
5. Dependency management
6. Data management

(*) https://cloud.google.com/blog/products/ai-machine-learning/best-practices-that-can-improve-the-life-of-any-developer-using-jupyter-notebooks

Where are you running your Jupyter?

datasciencenotebook.org



Compare tool features

TOOL	SETUP	JUPYTER COMPATIBILITY	PROGRAMMING LANGUAGES	DATA VISUALIZATION			
Jupyter	Self-hosted	Jupyter-compatible	Jupyter	Visualize with code			
Amazon Sagemaker	Fully managed	Jupyter-compatible	Jupyter	Visualize with code			
Google Colab	Fully managed	Jupyter-compatible	Jupyter	Visualize with code			
Deepnote	Fully managed	Jupyter-compatible	Jupyter, SQL	Visualize with code or UI			
Hex	Fully managed	Jupyter-compatible	Jupyter, SQL	Visualize with code or UI			
Databricks Notebooks	Self-hosted or fully managed	Jupyter-compatible	Jupyter	Visualize with code or UI			
DataCamp Workspace	Fully managed	Jupyter-compatible	Jupyter, SQL	Visualize with code or UI			
JupyterLab	Self-hosted	Jupyter-compatible	Jupyter	Visualize with code			
CoCalc	Self-hosted or fully managed	Jupyter-compatible	Jupyter	Visualize with code	Realtime collaboration	Free and paid options	Open source
Hyperquery	Fully managed	None	Python, SQL	Visualize with code or UI	Realtime collaboration	Free and paid options	Proprietary
Jetbrains Datalore	Fully managed	Jupyter-compatible	Jupyter	Visualize with code or UI	Realtime collaboration	Free and paid options	Proprietary
Kaggle	Fully managed	Jupyter-compatible	Jupyter	Visualize with code	Limited collaboration	Free	Proprietary

robertlacok / datasciencenotebooks Public

robertlacok Fix type issue 1 a57603d 3 weeks ago 160 commits

.vscode	fix: set correct schema for github workflow	last year
.yarn	feat: update next.js	last year
public	feat: add sitemap.xml	last year
src	Fix type issue	3 weeks ago
.env.development	fix: use vercel env vars	last year
.eslintrc.json	fix: stricter linting	last year
.gitignore	chore: re-ignore vercel project.json	last year
.nvmrc	feat: add next.js skeleton	last year
.yarnrc.yml	feat: update next.js	last year

Where are you running your Jupyter?

Yourself Local Jupyter, Local JupyterLab, JupyterHub, VS Code

Lightweight hosted Binder, Google Colab, Kaggle Kernels…

Mediumweight hosted Deepnote, Saturn Cloud, Jetbrains…

Enterprise hosted Amazon Sagemaker, Google Vertex AI workbench, Databricks notebooks, Azure ML studio, VS Code (!)...

Niche Paperspace Gradient, Cocalc, Mode analytics, naas.ai…

1. Tools for software development good practice

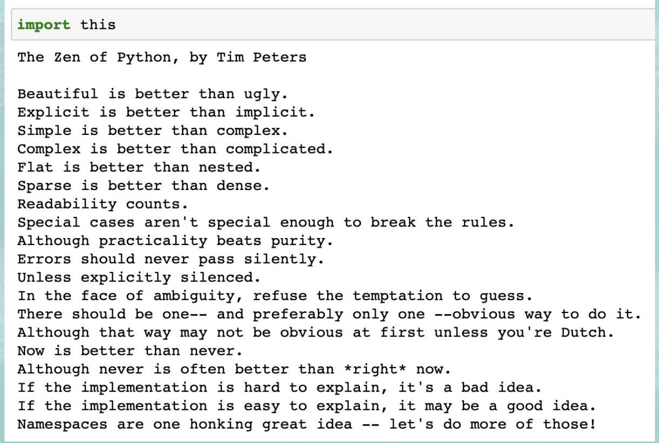
Why? My code doesn't need to be pretty. Move fast, experiment! No time for aesthetics!!

1. Tools for software development good practice

Why? My code doesn't need to be pretty. Move fast, experiment! No time for aesthetics!!
Answer: Code is read more often than it is written AND style best practice supports quality

1. Tools for software development good practice

- Write clean code (resources at the end of this presentation)
    - DRY, meaningful variable and function naming, modularity, explicit imports
- Move functions from Notebooks into modules (where appropriate)
- Python Style Guide (PEP8)
- Zen of Python



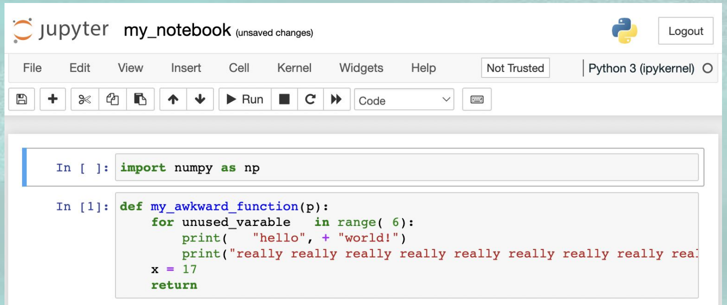
```python
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

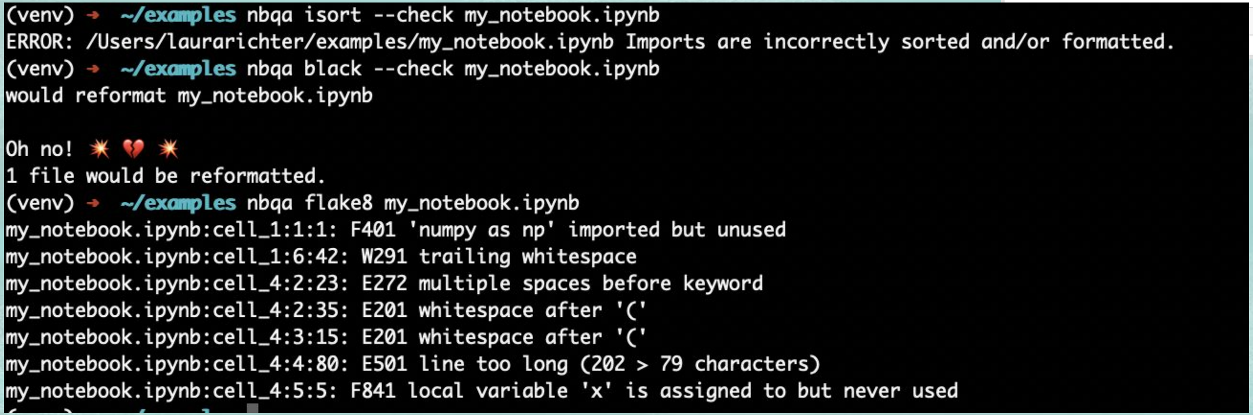
1. Tools for software development good practice

Tools:

- Code quality: **isort, flake8, black**, sqlfluff, …
- Specific to Notebooks: nbqa
- Linting and formatting



```
In [ ]: import numpy as np

In [1]: def my_awkward_function(p):
            for unused_varable   in range( 6):
                print(    "hello", + "world!")
                print("really really really really really really really really real
            x = 17
            return
```

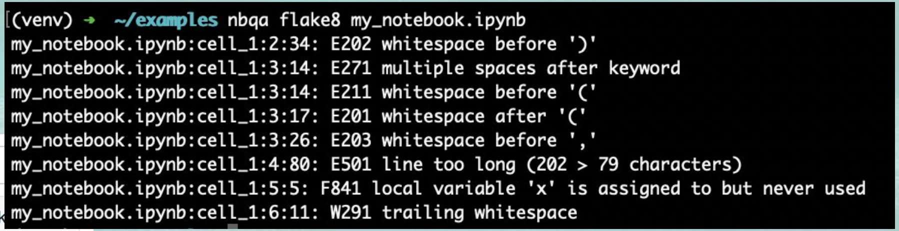


```
(venv) ➜  ~/examples nbqa isort --check my_notebook.ipynb
ERROR: /Users/laurarichter/examples/my_notebook.ipynb Imports are incorrectly sorted and/or formatted.
(venv) ➜  ~/examples nbqa black --check my_notebook.ipynb
would reformat my_notebook.ipynb

Oh no! 💥 💔 💥
1 file would be reformatted.
(venv) ➜  ~/examples nbqa flake8 my_notebook.ipynb
my_notebook.ipynb:cell_1:1:1: F401 'numpy as np' imported but unused
my_notebook.ipynb:cell_1:6:42: W291 trailing whitespace
my_notebook.ipynb:cell_4:2:23: E272 multiple spaces before keyword
my_notebook.ipynb:cell_4:2:35: E201 whitespace after '('
my_notebook.ipynb:cell_4:3:15: E201 whitespace after '('
my_notebook.ipynb:cell_4:4:80: E501 line too long (202 > 79 characters)
my_notebook.ipynb:cell_4:5:5: F841 local variable 'x' is assigned to but never used
```

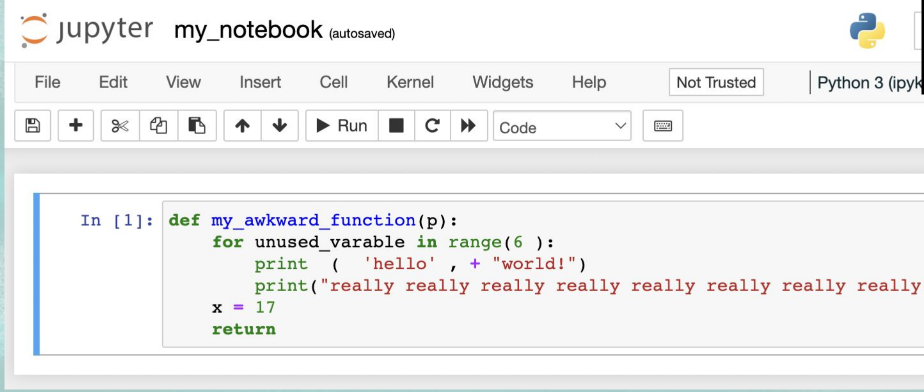
1. Tools for software development good practice

Tools:

- Code quality: isort, **flake8,** black, sqlfluff, …
- Specific to Notebooks: nbqa
- Linting and formatting



```
(venv) → ~/examples nbqa flake8 my_notebook.ipynb
my_notebook.ipynb:cell_1:2:34: E202 whitespace before ')'
my_notebook.ipynb:cell_1:3:14: E271 multiple spaces after keyword
my_notebook.ipynb:cell_1:3:14: E211 whitespace before '('
my_notebook.ipynb:cell_1:3:17: E201 whitespace after '('
my_notebook.ipynb:cell_1:3:26: E203 whitespace before ','
my_notebook.ipynb:cell_1:4:80: E501 line too long (202 > 79 characters)
my_notebook.ipynb:cell_1:5:5: F841 local variable 'x' is assigned to but never used
my_notebook.ipynb:cell_1:6:11: W291 trailing whitespace
```

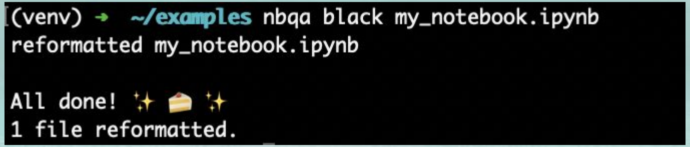


```
In [1]: def my_awkward_function(p):
            for unused_varable in range(6 ):
                print   (  'hello' , + "world!")
                print("really really really really really really really really real
            x = 17
            return
```

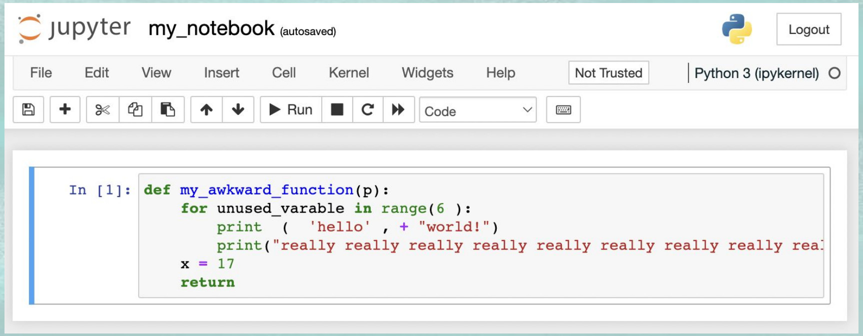
1. Tools for software development good practice

Tools:

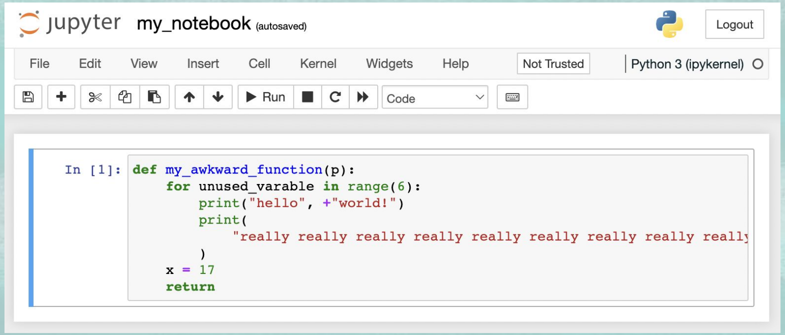
- Code quality: isort, flake8, **black,** sqlfluff, …
- Specific to Notebooks: nbqa
- Linting and formatting



```
(venv) ➜  ~/examples nbqa black my_notebook.ipynb
reformatted my_notebook.ipynb

All done! ✨ 🍰 ✨
1 file reformatted.
```



```
def my_awkward_function(p):
    for unused_varable in range(6 ):
        print  (  'hello'  , + "world!")
        print("really really really really really really really really real
    x = 17
    return
```

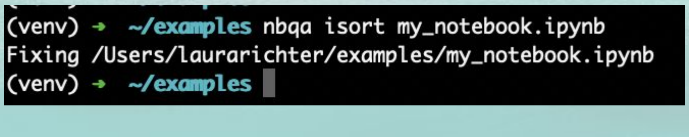


```
def my_awkward_function(p):
    for unused_varable in range(6):
        print("hello", +"world!")
        print(
            "really really really really really really really really really
        )
    x = 17
    return
```

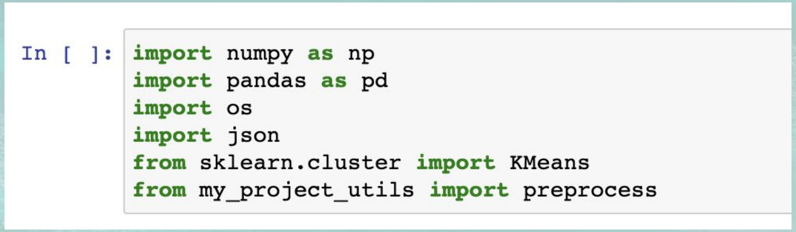
1. Tools for software development good practice

Tools:

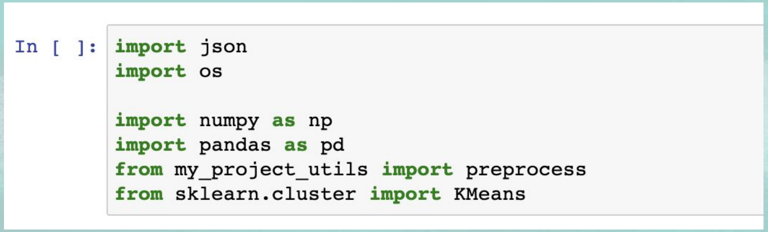
- Code quality: **isort,** flake8, black, sqlfluff, …
- Specific to Notebooks: nbqa
- Linting and formatting



```
(venv) ➜  ~/examples nbqa isort my_notebook.ipynb
Fixing /Users/laurarichter/examples/my_notebook.ipynb
(venv) ➜  ~/examples
```



```
In [ ]: import numpy as np
        import pandas as pd
        import os
        import json
        from sklearn.cluster import KMeans
        from my_project_utils import preprocess
```



```
In [ ]: import json
        import os

        import numpy as np
        import pandas as pd
        from my_project_utils import preprocess
        from sklearn.cluster import KMeans
```

2. Version Control & Reviews

Why? Waiting for reviews just slows me down!
Answer: Then you probably need to be slowed down 🙈

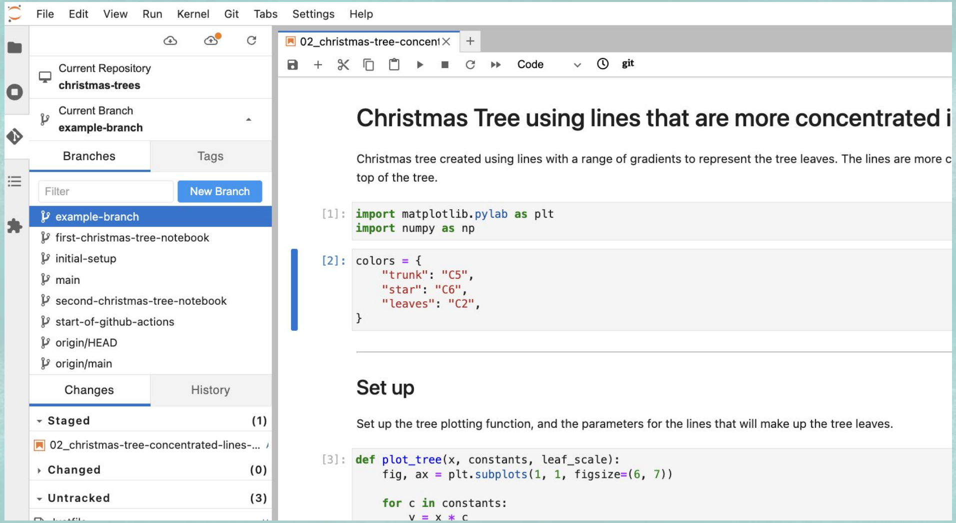
2. Version Control & Reviews

Version Control ⟺ Creating dated artifacts

2. Version Control & Reviews

Tools:

- Github
- Git tooling in VS Code, JupyterLab Git extension, hosted Notebook Git tooling



Current Repository
christmas-trees

Current Branch
example-branch

- example-branch
- first-christmas-tree-notebook
- initial-setup
- main
- second-christmas-tree-notebook
- start-of-github-actions
- origin/HEAD
- origin/main

Staged (1)
02_christmas-tree-concentrated-lines-...

Changed (0)

Untracked (3)

Christmas Tree using lines that are more concentrated i

Christmas tree created using lines with a range of gradients to represent the tree leaves. The lines are more c
top of the tree.

```
[1]: import matplotlib.pylab as plt
     import numpy as np
```

```
[2]: colors = {
         "trunk": "C5",
         "star": "C6",
         "leaves": "C2",
     }
```

Set up

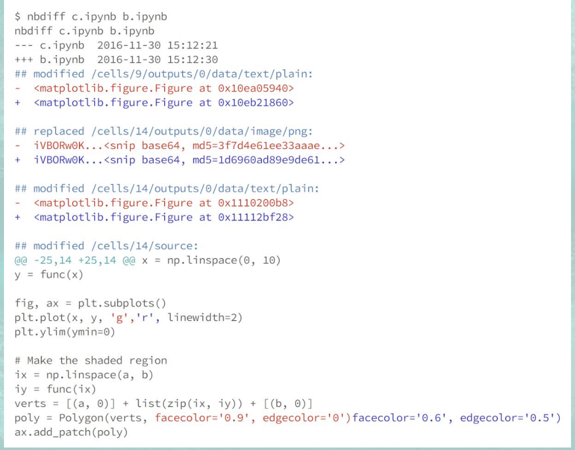
Set up the tree plotting function, and the parameters for the lines that will make up the tree leaves.

```
[3]: def plot_tree(x, constants, leaf_scale):
         fig, ax = plt.subplots(1, 1, figsize=(6, 7))

         for c in constants:
             y = x * c
```

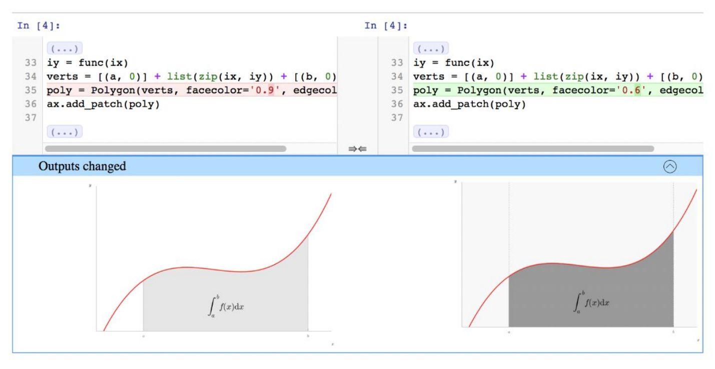
2. Version Control & Reviews

Tools:

- Github
- Git tooling in VS Code, JupyterLab Git extension, hosted Notebook Git tooling
- Git diffs:
  - nbdime



```
$ nbdiff c.ipynb b.ipynb
nbdiff c.ipynb b.ipynb
--- c.ipynb  2016-11-30 15:12:21
+++ b.ipynb  2016-11-30 15:12:30
## modified /cells/9/outputs/0/data/text/plain:
-  <matplotlib.figure.Figure at 0x10ea05940>
+  <matplotlib.figure.Figure at 0x10eb21860>

## replaced /cells/14/outputs/0/data/image/png:
-  iVBORw0K...<snip base64, md5=3f7d4e61ee33aaae...>
+  iVBORw0K...<snip base64, md5=1d6960ad89e9de61...>

## modified /cells/14/outputs/0/data/text/plain:
-  <matplotlib.figure.Figure at 0x1110200b8>
+  <matplotlib.figure.Figure at 0x11112bf28>

## modified /cells/14/source:
@@ -25,14 +25,14 @@ x = np.linspace(0, 10)
y = func(x)

fig, ax = plt.subplots()
plt.plot(x, y, 'g','r', linewidth=2)
plt.ylim(ymin=0)

# Make the shaded region
ix = np.linspace(a, b)
iy = func(ix)
verts = [(a, 0)] + list(zip(ix, iy)) + [(b, 0)]
poly = Polygon(verts, facecolor='0.9', edgecolor='0')facecolor='0.6', edgecolor='0.5')
ax.add_patch(poly)
```



In [4]:

```
(...)
iy = func(ix)
verts = [(a, 0)] + list(zip(ix, iy)) + [(b, 0)
poly = Polygon(verts, facecolor='0.9', edgecol
ax.add_patch(poly)

(...)
```

In [4]:

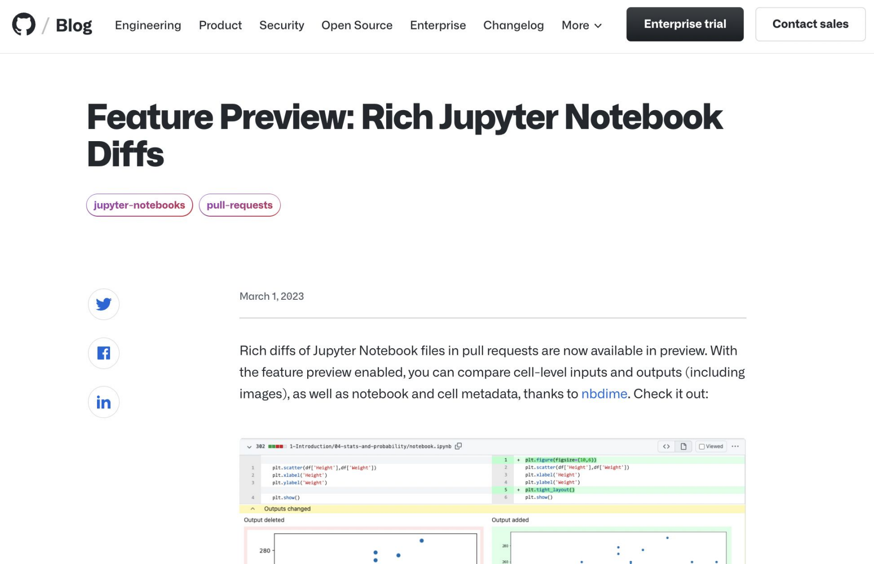
```
(...)
iy = func(ix)
verts = [(a, 0)] + list(zip(ix, iy)) + [(b, 0)
poly = Polygon(verts, facecolor='0.6', edgecol
ax.add_patch(poly)

(...)
```

content aware diffs

2. Version Control & Reviews

Tools:

- Github
- Git tooling in VS Code, JupyterLab Git extension, hosted Notebook Git tooling
- Git diffs:
    - nbdime
    - Github diffs (feature preview)



Feature Preview: Rich Jupyter Notebook Diffs

March 1, 2023

Rich diffs of Jupyter Notebook files in pull requests are now available in preview. With the feature preview enabled, you can compare cell-level inputs and outputs (including images), as well as notebook and cell metadata, thanks to nbdime. Check it out:

302 1-Introduction/04-stats-and-probability/notebook.ipynb

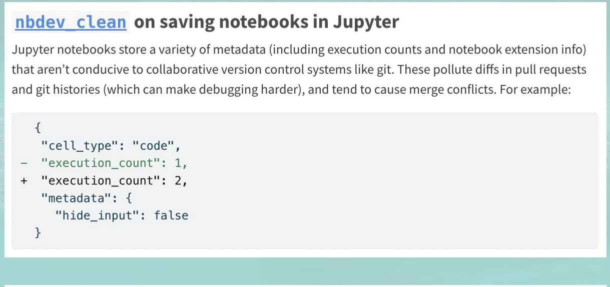
```
plt.scatter(df['Height'],df['Weight'])
plt.xlabel('Height')
plt.ylabel('Weight')

plt.show()
```

```
+ plt.figure(figsize=(10,6))
  plt.scatter(df['Height'],df['Weight'])
  plt.xlabel('Height')
  plt.ylabel('Weight')
+ plt.tight_layout()
  plt.show()
```

2. Version Control & Reviews

Tools:

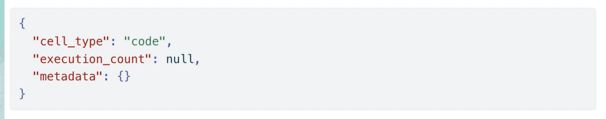
- Github
- Git tooling in VS Code, JupyterLab Git extension, hosted Notebook Git tooling
- Git diffs:
  - nbdime
  - Github diffs (feature preview)
  - nbdev



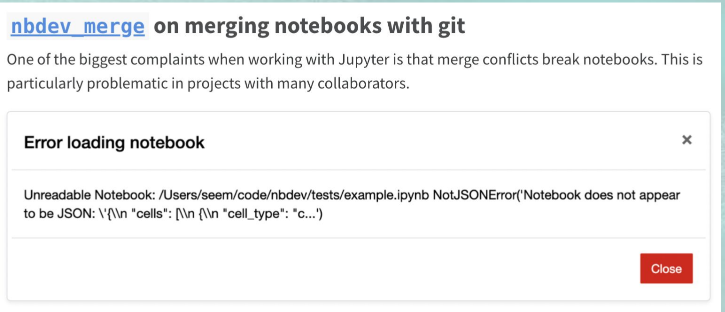
`nbdev_clean` on saving notebooks in Jupyter

Jupyter notebooks store a variety of metadata (including execution counts and notebook extension info) that aren't conducive to collaborative version control systems like git. These pollute diffs in pull requests and git histories (which can make debugging harder), and tend to cause merge conflicts. For example:

```
  {
   "cell_type": "code",
-  "execution_count": 1,
+  "execution_count": 2,
   "metadata": {
     "hide_input": false
  }
```



```
{
  "cell_type": "code",
  "execution_count": null,
  "metadata": {}
}
```



`nbdev_merge` on merging notebooks with git

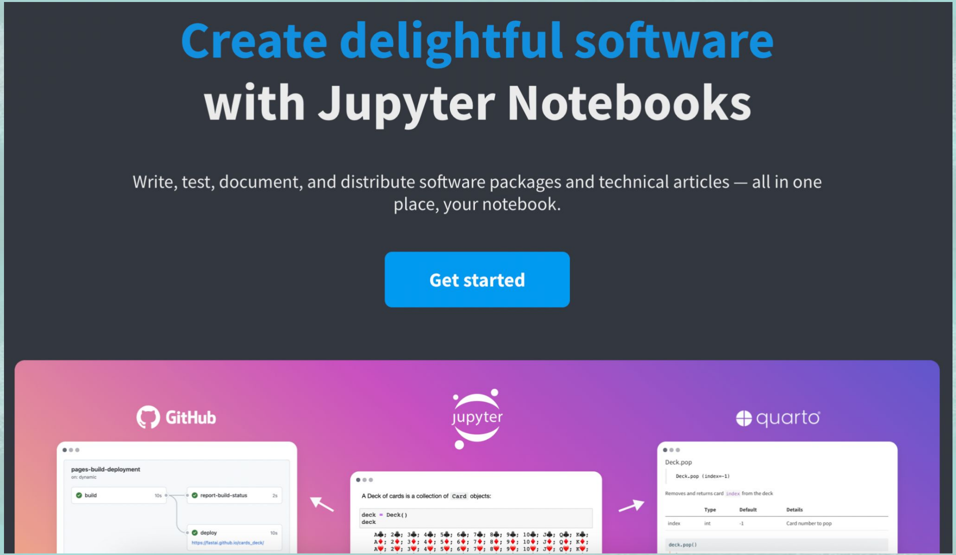
One of the biggest complaints when working with Jupyter is that merge conflicts break notebooks. This is particularly problematic in projects with many collaborators.

Error loading notebook

Unreadable Notebook: /Users/seem/code/nbdev/tests/example.ipynb NotJSONError('Notebook does not appear to be JSON: \'{\\n "cells": [\\n {\\n "cell_type": "c...')

using nbdev Jupyter save hooks

nbdev shoutout



Create delightful software with Jupyter Notebooks

Write, test, document, and distribute software packages and technical articles — all in one place, your notebook.

pages-build-deployment
on: dynamic

build 10s
report-build-status 2s
deploy 10s
https://fastai.github.io/cards_deck/

A Deck of cards is a collection of `Card` objects:

```
deck = Deck()
deck
```

```
A♣; 2♣; 3♣; 4♣; 5♣; 6♣; 7♣; 8♣; 9♣; 10♣; J♣; Q♣; K♣;
A♦; 2♦; 3♦; 4♦; 5♦; 6♦; 7♦; 8♦; 9♦; 10♦; J♦; Q♦; K♦;
A♥; 2♥; 3♥; 4♥; 5♥; 6♥; 7♥; 8♥; 9♥; 10♥; J♥; Q♥; K♥;
```

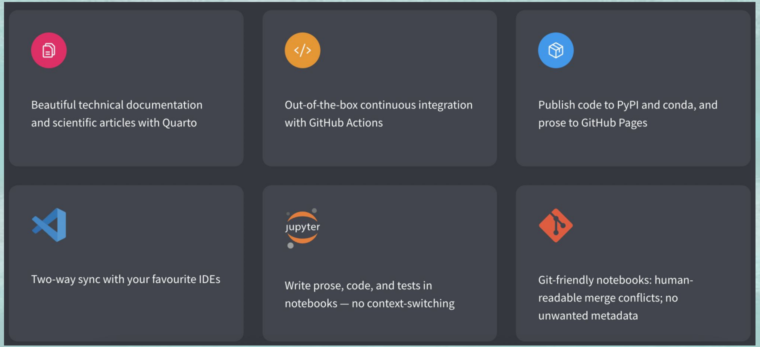
Deck.pop

```
Deck.pop (index=-1)
```

Removes and returns card `index` from the deck

	Type	Default	Details
index	int	-1	Card number to pop

```
deck.pop()
```



Beautiful technical documentation and scientific articles with Quarto

Out-of-the-box continuous integration with GitHub Actions

Publish code to PyPI and conda, and prose to GitHub Pages

Two-way sync with your favourite IDEs

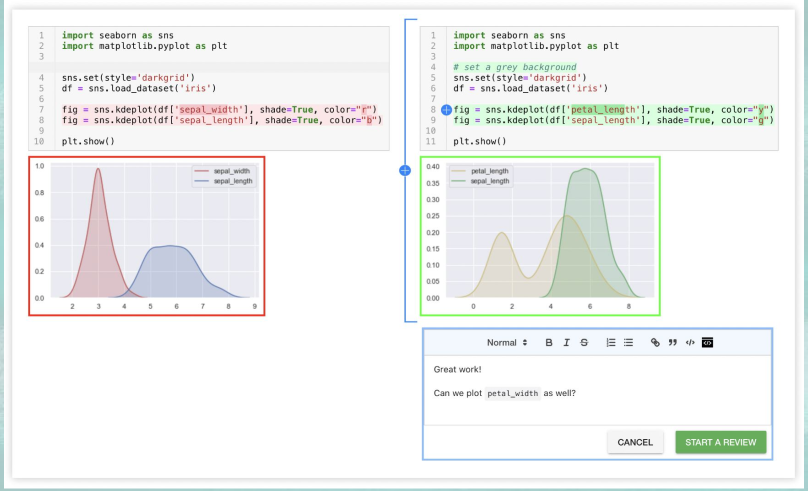
Write prose, code, and tests in notebooks — no context-switching

Git-friendly notebooks: human-readable merge conflicts; no unwanted metadata

2. Version Control & Reviews

Tools:

- Github
- Git tooling in VS Code, JupyterLab Git extension, hosted Notebook Git tooling
- Git diffs:
  - nbdime
  - Github diffs (feature preview)
  - nbdev
  - ReviewNB



```
import seaborn as sns
import matplotlib.pyplot as plt


sns.set(style='darkgrid')
df = sns.load_dataset('iris')

fig = sns.kdeplot(df['sepal_width'], shade=True, color="r")
fig = sns.kdeplot(df['sepal_length'], shade=True, color="b")

plt.show()
```

```
import seaborn as sns
import matplotlib.pyplot as plt

# set a grey background
sns.set(style='darkgrid')
df = sns.load_dataset('iris')

fig = sns.kdeplot(df['petal_length'], shade=True, color="y")
fig = sns.kdeplot(df['sepal_length'], shade=True, color="g")

plt.show()
```

Great work!

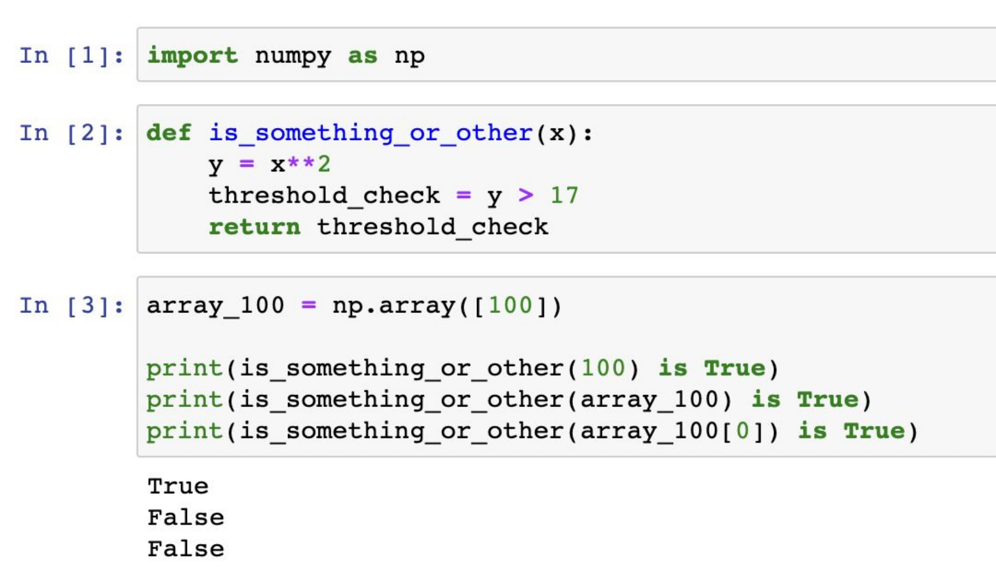
Can we plot `petal_width` as well?

3. Testing

Why? Just write your code correctly!

3. Testing

Why? Just write your code correctly!
Answer:



```
In [1]: import numpy as np

In [2]: def is_something_or_other(x):
            y = x**2
            threshold_check = y > 17
            return threshold_check

In [3]: array_100 = np.array([100])

        print(is_something_or_other(100) is True)
        print(is_something_or_other(array_100) is True)
        print(is_something_or_other(array_100[0]) is True)

        True
        False
        False
```

3. Testing

Tools:

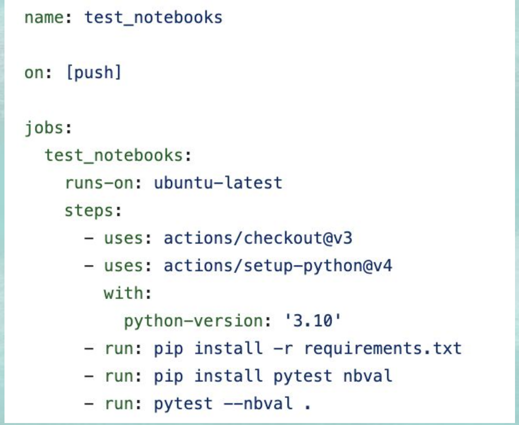
- nbval
- nbmake
- testbook
- pytest-notebook
- nbdev

4. Automating

- Automate your linting
- Automate your testing
- Automate your team norms
  - Even things file naming conventions!

Tools:

- Github actions
- Github pre-commit hooks
- Jupyter save hooks



```
name: test_notebooks

on: [push]

jobs:
  test_notebooks:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-python@v4
        with:
          python-version: '3.10'
      - run: pip install -r requirements.txt
      - run: pip install pytest nbval
      - run: pytest --nbval .
```

5. Dependency management

Why? It just adds complexity! I don't want to be spending time on this!

5. Dependency management

Why? It just adds complexity! I don't want to be spending time on this!

Answer: You'll spend time on it for sure, one way or another :)

5. Dependency management

Tools:

- virtual environments
- pinning your requirements
  - requirements.txt
  - pip-compile
- Docker
  - jupyter/base-notebook
  - jupyter/minimal-notebook

6. Data management

Why? I just slack my colleagues the csv's. It works fine.

6. Data management

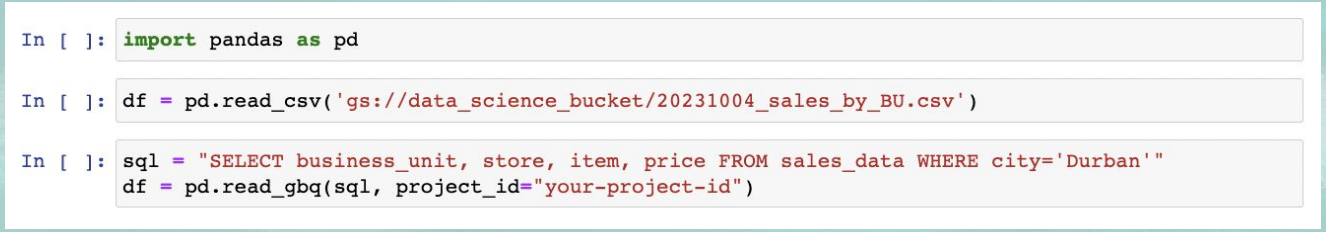
Why? I just slack my colleagues the csv's. It works fine.

Answer: Yeah, you know it doesn't really work fine :)

6. Data management

Tools:

- Cloud data in any form accessible with python:
  - AWS s3 (boto, pandas)
  - GCP GCS (google cloud python sdk, pandas)
  - GCP BigQuery (google cloud python sdk, pandas)
  - Spark cluster
  - …
- quiltdata
- Github lfs
- dvc



```
In [ ]: import pandas as pd

In [ ]: df = pd.read_csv('gs://data_science_bucket/20231004_sales_by_BU.csv')

In [ ]: sql = "SELECT business_unit, store, item, price FROM sales_data WHERE city='Durban'"
        df = pd.read_gbq(sql, project_id="your-project-id")
```

Some final thoughts

- Start with simple dependency management: virtual environments and requirements files
    - For cloud Notebooks, requirements files
- Next, auto formatting and linting (nbqa isort, nba black, nbqa flake8…)
- Next, version control (e.g. Github) and reviews
- Next, automate linting
- Next, data management
- Next, simple end-to-end testing

…

- Then, down the line…

  Dockerise? Tooling/frameworks for experiment management? Unit test for modules?

  Cookie-cutters?

  … And see what new tooling and best practice emerges!

Thank you

References

- Why Jupyter is data scientists’ computational notebook of choice: https://www.nature.com/articles/d41586-018-07196-1
- Jupyter surveys: https://github.com/jupyter/surveys
- Google Blog - Jupyter Notebook Best Practives: https://cloud.google.com/blog/products/ai-machine-learning/best-practices-that-can-improve-the-life-of-any-developer-using-jupyter-notebooks
- Notebook tool reviews: https://datasciencenotebook.org/
- I don't like Notebooks, Joel Grus, JupyterCon 2018: https://docs.google.com/presentation/d/1n2RlMdmv1p25Xy5thJUhkKGvjtV-dkAIsUXP-AL4ffI/preview?slide=id.g362da58057_0_1
- Clean code in Python: https://testdriven.io/blog/clean-code-python/
- Clean-code-python: https://github.com/zedr/clean-code-python
- Python Style Guide, PEP8: https://peps.python.org/pep-0008

References: Tools

- nbqa: nbqa.readthedocs.io
- nbdime: nbdime.readthedocs.io
- Github rich Jupyter Notebook diffs: https://github.blog/changelog/2023-03-01-feature-preview-rich-jupyter-notebook-diffs/
- nbdev: nbdev.fast.ai
- ReviewNB: www.reviewnb.com
- nbval: https://github.com/computationalmodelling/nbval
- pytest.notebook: pytest-notebook.readthedocs.io
- nbmake: https://github.com/treebeardtech/nbmake
- testbook: testbook.readthedocs.io
- Jupyter file save hooks: https://jupyter-notebook.readthedocs.io/en/4.x/extending/savehooks.html
- git hooks: https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks
- dvc: dvc.org
- quiltdata: https://github.com/quiltdata/quilt