EPFL

Tools for (computational) chemists

Simon Duerr

Outline

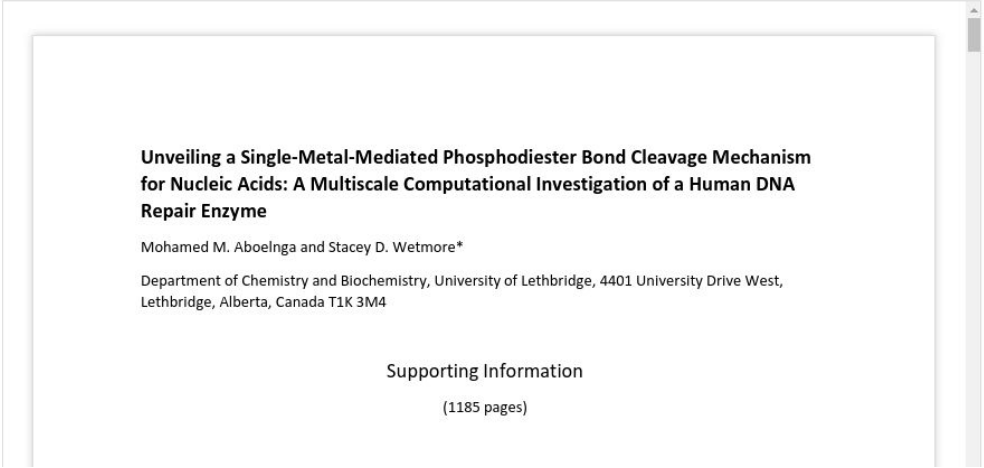
- Why care?
- Do and Don't data processing
- Simulation data processing
- Research coding basics (Code editor, Linting, Formating, Testing, Documentation, start a project, coding environments)
- Use AI productively for your research data workflows
- Sharing code (Webapps, Colab, packaging, Zenodo)

Why care?

Chemists generate a lot of data and we're required to share it in Switzerland

Unveiling a Single-Metal-Mediated Phosphodiester Bond Cleavage Mechanism for Nucleic Acids: A Multiscale Computational Investigation of a Human DNA Repair Enzyme

46 views | 34 shares | 0 downloads



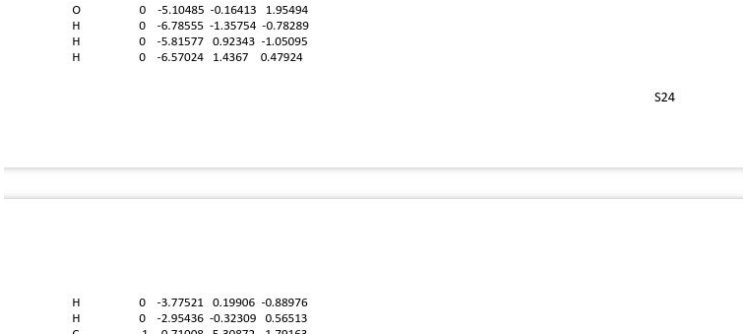
Unveiling a Single-Metal-Mediated Phosphodiester Bond Cleavage Mechanism for Nucleic Acids: A Multiscale Computational Investigation of a Human DNA Repair Enzyme

Mohamed M. Aboelnga and Stacey D. Wetmore*

Department of Chemistry and Biochemistry, University of Lethbridge, 4401 University Drive West, Lethbridge, Alberta, Canada T1K 3M4

Supporting Information

(1185 pages)



O	0	-5.10485	-0.16413	1.95494
H	0	-6.78555	-1.35754	-0.78289
H	0	-5.81577	0.92343	-1.05095
H	0	-6.57024	1.4367	0.47924

S24

H	0	-3.77521	0.19906	-0.88976
H	0	-2.95436	-0.32309	0.56513

RDM for chemists

Guidelines for Reporting Molecular Dynamics Simulations in JCIM Publications

Thereza A. Soares*, Zoe Cournia, Kevin Naidoo, Rommie Amaro, Habibah Wahab, and Kenneth M. Merz Jr.

Cite this: *J. Chem. Inf. Model.* 2023, 63, 11, 3227–3229
Publication Date: May 16, 2023
https://doi.org/10.1021/acs.jcim.3c00599

Input/output files and data availability

ARTICLE SECTIONS Jump To

JCIM now requires depositing your input files (including topologies, parameters, input files, and initial configurations) and output trajectory files in a public repository providing a DOI and provide the accession link in the manuscript. Some available repositories are Zenodo (https://zenodo.org/), Dryad (https://datadryad.org/), Nomad (https://nomad-repository.eu/), and Figshare (https://figshare.com/). If the full trajectories are too large, a selection of clustered structure representing the respective trajectories should be made available in the public repository.

Real problems from my own PhD

- People not sharing their data/models even after inquiring multiple times

- Compiled binary only for 32bit systems and environment not specified

review from 20221

Table 1. Computational tools for metal-binding site and metalloprotein prediction.

Name	Website	Related Metals	Main Algorithm	Reported Performance	Ref.
RDGB	http://www.cerm.unifi.it/home/research/genomebrowsing.html	Zn, Cu, Fe and other metals	Integration of tools for retrieval of protein domains and genome analysis	Accuracy: 89.6%, precision: 85.9%	[32]
Zincfinder	http://zincfinder.dsi.unifi.it	Zn	[a] SVM	[b] AURPC: 0.590 (local predictor) and 0.633 (gating network)	[33]
Zincpred	http://www.fos.su.se/~nanjiang/zincpred/download/	Zn	SVM- and homology-based algorithm	AURPC: 0.723 (local predictor) and 0.701 (gating network)	[34]
TEMSP	http://netalign.ustc.edu.cn/temsp/	Zn	Structure-based algorithm with a range of geometric criteria	[c] AUC: 0.945	[35]
Zincidentifier	http://protein.cau.edu.cn/zincidentifier/	Zn	A two-step feature selection method based on random forest algorithm	AUC: 0.955, AURPC: 0.829	[36]
ZincExplorer	http://protein.cau.edu.cn/ZincExplorer/	Zn	A combination of SVM-, cluster- and template-based predictors	AURPC: 0.907	[37]
ZincBinder	http://proteininformatics.org/mkumar/znbinder/	Zn	SVM model trained on PSSM-based input feature	AUC: 0.91	[38]
ZINCCLUSTER	http://www.metalactive.in	Zn	SVM-based Ligand Finder and Cluster Finder algorithms	[d] MCC: 0.798, F1-score: 0.801	[39]
ZnMachine	http://bioinformatics.fzu.edu.cn/znMachine.html	Zn	A combination of several intensively-trained machine learning models	AUC: 0.933 (SVM) and 0.910 (neural network)	[40]
HemeBIND	http://mleg.cse.sc.edu/hemeBIND/	Fe (heme)	SVM	MCC: 0.504, F1-score: 56.87%	[41]
SCMHBP	http://iclab.life.nctu.edu.tw/SCMHBP/	Fe (heme)	Based on a newly-developed scoring card method for predicting heme-binding proteins	Accuracy: 85.90%	[42]
Isph	http://biodev.extra.cea.fr/isph	Fe (Fe-S)	A penalized linear model based on machine learning approach	Precision: 87.9%, recall: 80.1% (extended model)	[43]
MetalPredator	http://metalweb.cerm.unifi.it/tools/metalpredator/	Fe (Fe-S)	Integration of existing domain-based methodology with a new approach for discovering metal-binding motifs	Precision: 85.2%, recall: 88.6%	[44]
MetSite	N/A	Fe, Zn, Cu, Mn, Ca, Mg	Artificial neural network	Mean accuracy: 94.5%	[45]
FINDSITE-metal	http://cssb.biology.gatech.edu/findsite-metal/	Fe, Zn, Cu, Mn, Ni, Co, Ca, Mg	Integration of structure/evolutionary information and machine learning approach (SVM)	Overall accuracy: 70–90%	[46]
SeqCHED	http://ligin.weizmann.ac.il/seqched	Fe, Zn, Cu, Mn, Ni, Co, Ca, Mg	A modification of the CHED algorithm and machine learning filters (decision tree classifier and SVM)	Sensitivity: 84–85%, selectivity: 82–93% (stringent filtration)	[47]
MetalDetector	http://metaldetector.dsi.unifi.it/v2.0/	Transition metals that use cysteine and histidine as ligands	A combination of different machine learning algorithms (SVM-HMM, structured-output SVM)	Precision: 60–79%, recall: 71–88%	[48]
MIB	http://bioinfo.cmu.edu.tw/MIB/	Ca, Cu, Fe, Mg, Mn, Zn, Cd, Ni, Hg, Co	Fragment transformation method	Overall accuracy: 92.9–95.1%	[49]

Table 1. Computational tools for metal-binding site and metalloprotein prediction.

Name	Website	Related Metals	Main Algorithm	Reported Performance	Ref.
RDGB	http://www.cerm.unifi.it/home/research/genomebrowsing.html	Zn, Cu, Fe and other metals	Integration of tools for retrieval of protein domains and genome analysis	Accuracy: 89.6%, precision: 85.9%	[32]
Zincfinder	http://zincfinder.dsi.unifi.it	Zn	[a] SVM	[b] AURPC: 0.590 (local predictor) and 0.633 (gating network)	[33]
Zincpred	http://www.fos.su.se/~nanjiang/zincpred/download/	Zn	SVM- and homology-based algorithm	AURPC: 0.723 (local predictor) and 0.701 (gating network)	[34]
TEMSP	http://metalign.ustc.edu.cn/temsp/	Zn	Structure-based algorithm with a range of geometric criteria	[c] AUC: 0.945	[35]
Zincidentifier	http://protein.cau.edu.cn/zincidentifier/	Zn	A two-step feature selection method based on random forest algorithm	AUC: 0.955, AURPC: 0.829	[36]
ZincExplorer	http://protein.cau.edu.cn/ZincExplorer/	Zn	A combination of SVM-, cluster- and template-based predictors	AURPC: 0.907	[37]
ZincBinder	http://proteininformatics.org/mkumar/znbinder/	Zn	SVM model trained on PSSM-based input feature	AUC: 0.91	[38]
ZINCCLUSTER	http://www.metalactive.in	Zn	SVM-based Ligand Finder and Cluster Finder algorithms	[d] MCC: 0.798, F1-score: 0.801	[39]
ZnMachine	http://bioinformatics.fzu.edu.cn/znMachine.html	Zn	A combination of several intensively-trained machine learning models	AUC: 0.933 (SVM) and 0.910 (neural network)	[40]
HemeBIND	http://mleg.cse.sc.edu/hemeBIND/	Fe (heme)	SVM	MCC: 0.504, F1-score: 56.87%	[41]
SCMHBP	http://iclab.life.nctu.edu.tw/SCMHBP/	Fe (heme)	Based on a newly-developed scoring card method for predicting heme-binding proteins	Accuracy: 85.90%	[42]
Isph	http://biodev.extra.cea.fr/isph	Fe (Fe-S)	A penalized linear model based on machine learning approach	Precision: 87.9%, recall: 80.1% (extended model)	[43]
MetalPredator	http://metalweb.cerm.unifi.it/tools/metalpredator/	Fe (Fe-S)	Integration of existing domain-based methodology with a new approach for discovering metal-binding motifs	Precision: 85.2%, recall: 88.6%	[44]
MetSite	N/A	Fe, Zn, Cu, Mn, Ca, Mg	Artificial neural network	Mean accuracy: 94.5%	[45]
FINDSITE-metal	http://cssb.biology.gatech.edu/findsite-metal/	Fe, Zn, Cu, Mn, Ni, Co, Ca, Mg	Integration of structure/evolutionary information and machine learning approach (SVM)	Overall accuracy: 70–90%	[46]
SeqCHED	http://login.weizmann.ac.il/seqched	Fe, Zn, Cu, Mn, Ni, Co, Ca, Mg	A modification of the CHED algorithm and machine learning filters (decision tree classifier and SVM)	Sensitivity: 84–85%, selectivity: 82–93% (stringent filtration)	[47]
MetalDetector	http://metaldetector.dsi.unifi.it/v2.0/	Transition metals that use cysteine and histidine as ligands	A combination of different machine learning algorithms (SVM-HMM, structured-output SVM)	Precision: 60–79%, recall: 71–88%	[48]
MIB	this version no longer available http://bioinfo.cmu.edu.tw/MIB/	Ca, Cu, Fe, Mg, Mn, Zn, Cd, Ni, Hg, Co	Fragment transformation method	Overall accuracy: 92.9–95.1%	[49]

Do and Don't data processing

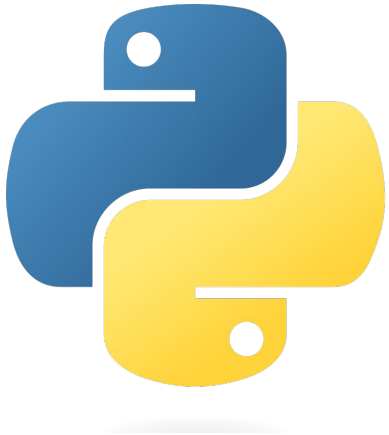
No manual file editing

No manual file copying

Clear track record from raw data to plots/tables (e.g via scripting)

Go from Python to Microsoft Word
https://rowannicholls.github.io/python/data/export_to_word.html

Programming language



Simplicity is key

Multiply 2d arrays of size (100,100)

```
# numpy
Z = X * Y
```

almost as fast

```
integer nx,ny
parameter (nx=100,ny=100)
double precision X(nx,ny),Y(nx,ny),Z(nx,ny)
do i = 1, 100
   do j = 1,100
      Z(i,j) = X(i,j)*Y(i,j)
   end do
end do
```

fast

Simulation data processing

Running a lot of computations?

Use workflow tools to keep track:

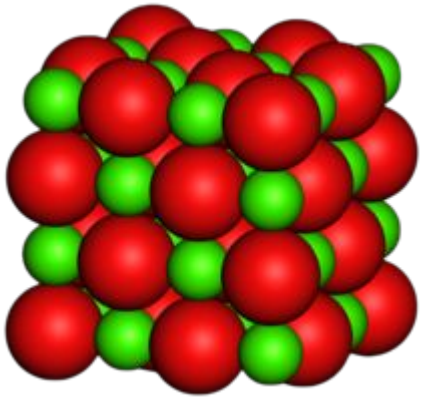
Computational Chemistry

- AIIDA
- atomate2/jobflow

Machine Learning

- Weights and Biases

Old school



Create structure in Avogadro or GaussView

Run `pw.x < MGO.opt.in > MGO.opt.out`

Extract energies `grep ! MGO.opt.out > energies.dat`

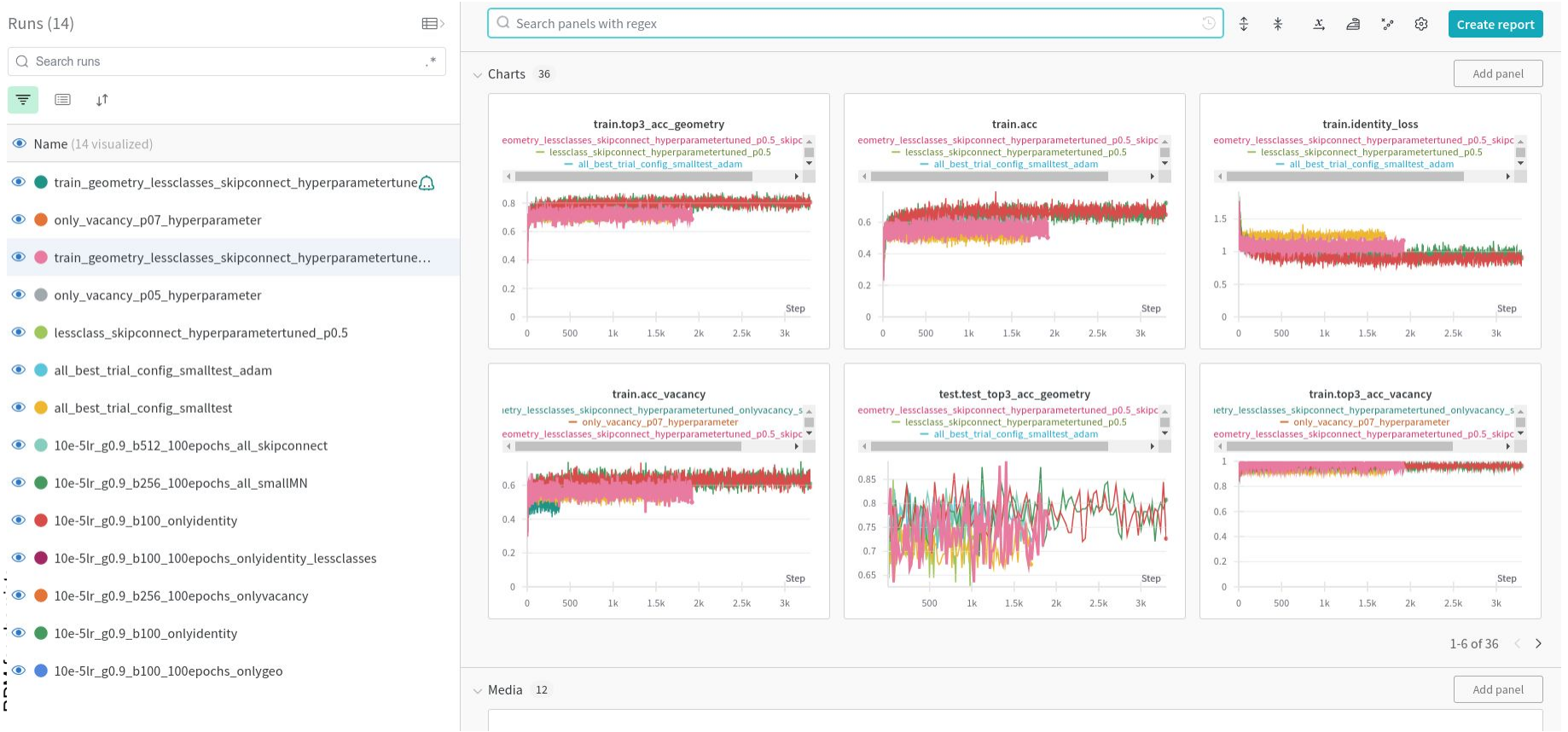
Plot them using `gnuplot` by entering the commands

```
> plot 'energies.dat' using 1:2 with lines
```

struct_opt_band_calc.py

```python
from atomate2.vasp.flows.core import RelaxBandStructureMaker
from jobflow import run_locally
from pymatgen.core import Structure

# construct a rock salt MgO structure
mgo_structure = Structure(
    lattice=[[0, 2.13, 2.13], [2.13, 0, 2.13], [2.13, 2.13, 0]],
    species=["Mg", "O"],
    coords=[[0, 0, 0], [0.5, 0.5, 0.5]],
)

# make a band structure flow to optimise the structure and obtain the band structure
bandstructure_flow = RelaxBandStructureMaker().make(mgo_structure)

# run the flow
run_locally(bandstructure_flow, create_folders=True)
```

Weights and Biases



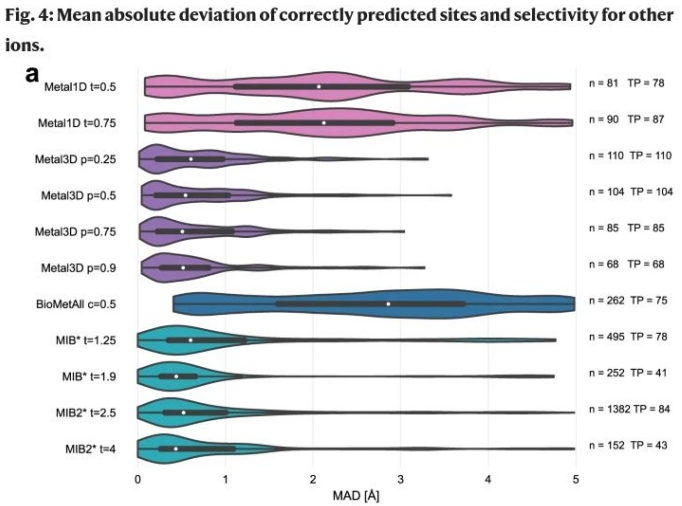
Prefer scriptable tools to GUIs (less automatable)

Prefer open software to closed software

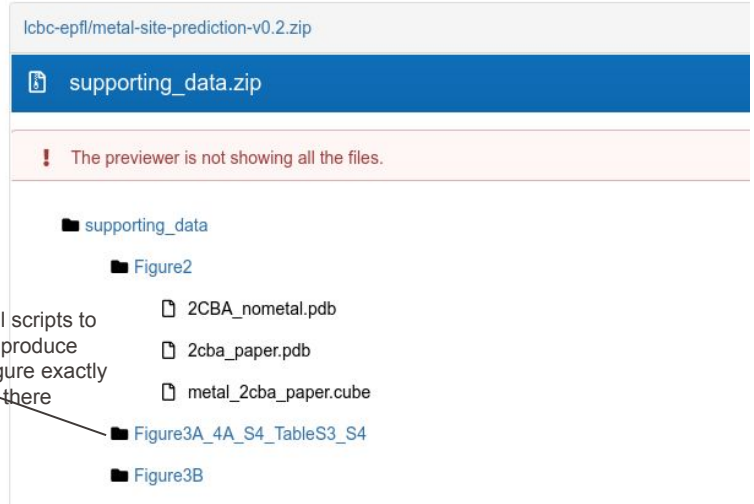
Make everything explicit instead of running single commands

Chances that you have to redo something multiple times are high.

Use automated scripts as much as possible to generate plots directly



All scripts to reproduce figure exactly is there



Let's talk about Jupyter Notebooks

jupyter Quickstart_Example (unsaved changes)

Visit repo | Copy Binder link

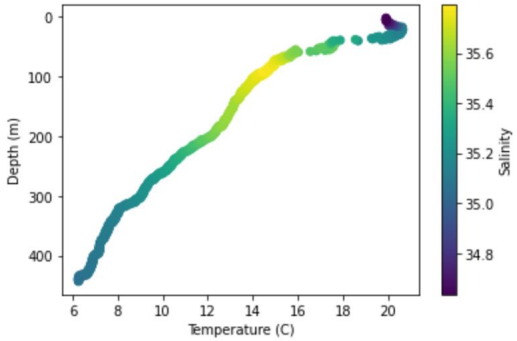
File Edit View Insert Cell Kernel Widgets Help | Trusted | Python 3

Run | Code | Memory: 197 / 2048 MB

Download | GitHub | Binder

In [25]:
```
import matplotlib.pyplot as plt
```

In [27]:
```
# Temperature vs. Depth
ax = plt.scatter(x=data['Temperature'],y=data['Depth'],c=data['Salinity']);
plt.gca().invert_yaxis(); # Flip the y-axis
plt.colorbar(label='Salinity')
plt.xlabel('Temperature (C)')
plt.ylabel('Depth (m)');
```




NOTEBOOKS ARE GREAT FOR ITERATIVE DEVELOPMENT
NOTEBOOKS ARE DANGEROUS UNLESS YOU RUN EACH CELL EXACTLY ONCE, IN ORDER


JAKE-CLARK.TUMBLR
imgflip.com


UNTITLED25.IPYNB
UNTITLED24.IPYNB
imgflip.com

jupyter madness

File Edit View Insert Cell Kernel Help Python 3

Code CellToolbar

```
In [3]: def f(x): return x + 2

In [2]: y = f(2)

In [4]: y == 4
Out[4]: False

In [5]: print(y)
        5
```

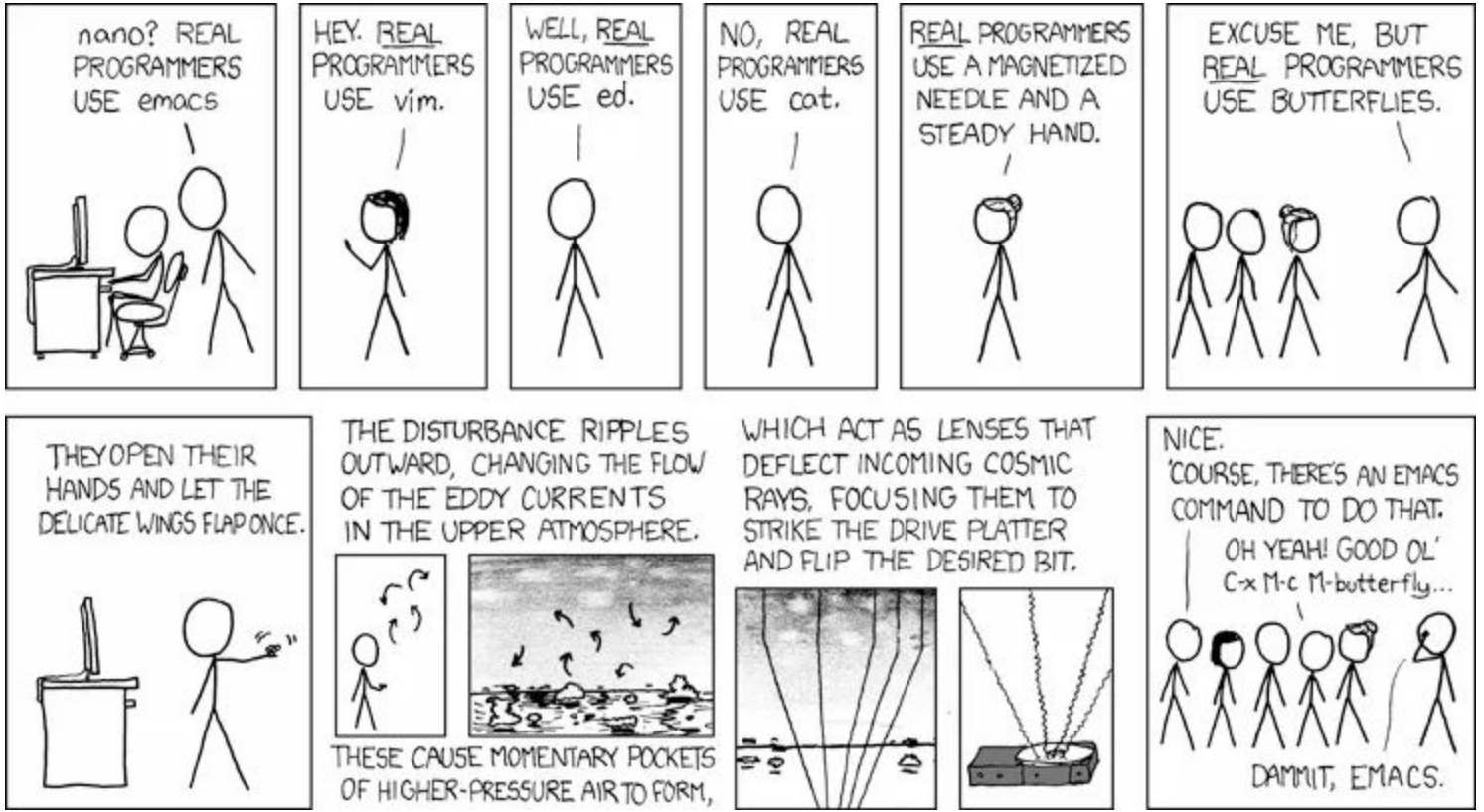
@joelgrus #jupytercon

input functions

processing functions

main program

Notebook vs. Script vs. Code


nano? REAL PROGRAMMERS USE emacs
HEY. REAL PROGRAMMERS USE vim.
WELL, REAL PROGRAMMERS USE ed.
NO, REAL PROGRAMMERS USE cat.
REAL PROGRAMMERS USE A MAGNETIZED NEEDLE AND A STEADY HAND.
EXCUSE ME, BUT REAL PROGRAMMERS USE BUTTERFLIES.
THEY OPEN THEIR HANDS AND LET THE DELICATE WINGS FLAP ONCE.
THE DISTURBANCE RIPPLES OUTWARD, CHANGING THE FLOW OF THE EDDY CURRENTS IN THE UPPER ATMOSPHERE.
THESE CAUSE MOMENTARY POCKETS OF HIGHER-PRESSURE AIR TO FORM,
WHICH ACT AS LENSES THAT DEFLECT INCOMING COSMIC RAYS, FOCUSING THEM TO STRIKE THE DRIVE PLATTER AND FLIP THE DESIRED BIT.
NICE.
'COURSE, THERE'S AN EMACS COMMAND TO DO THAT.
OH YEAH! GOOD OL' C-x M-c M-butterfly...
DAMMIT, EMACS.

Code editor

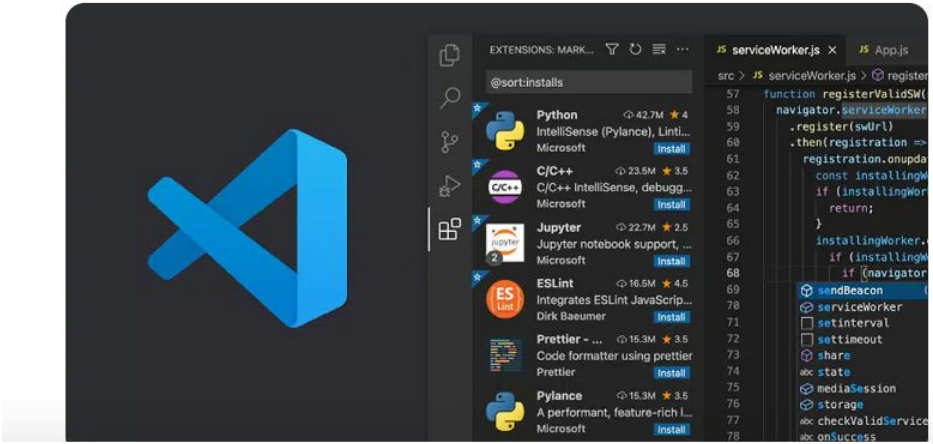
Many choices but if you don't have a strong preference for an editor:

VS Code offers:

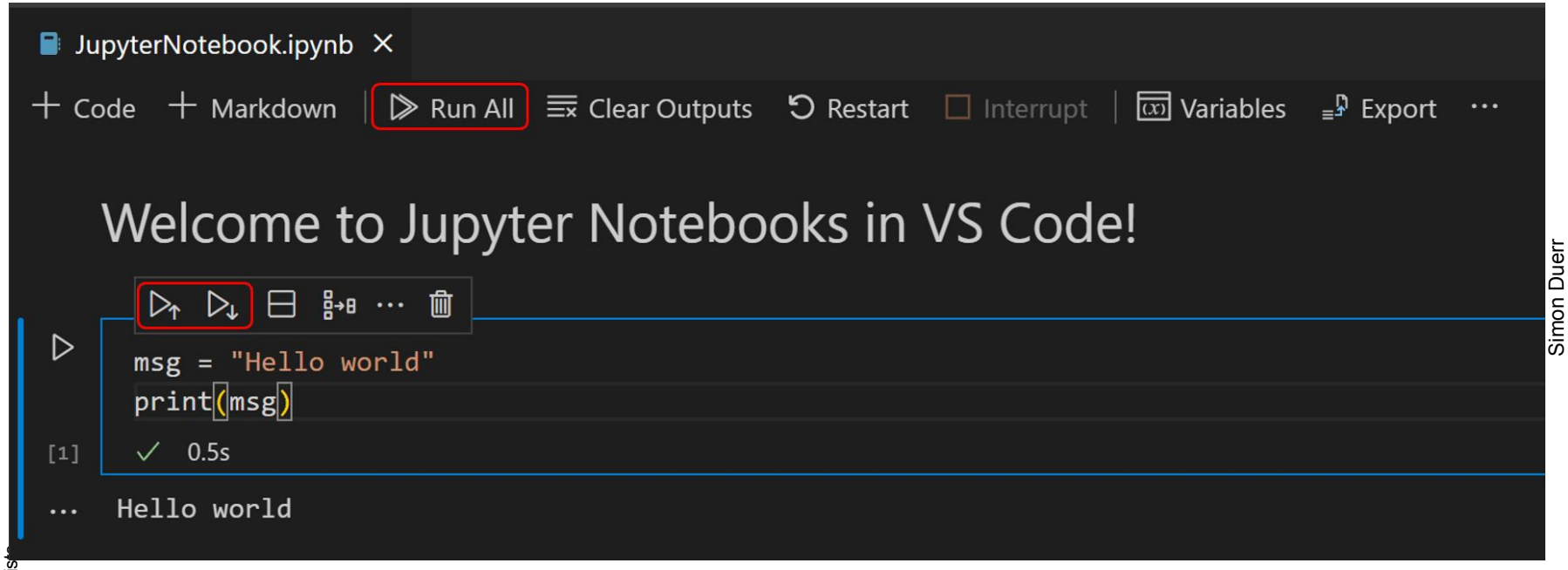
- the right abstractions,
- version control,
- inbuilt terminal,
- linting,
- formatting,
- AI completion,
- Jupyter support and
- remote capabilities

Why VS Code remains a developer favorite, year after year

Anastasija Uspenski



Notebooks in VS Code

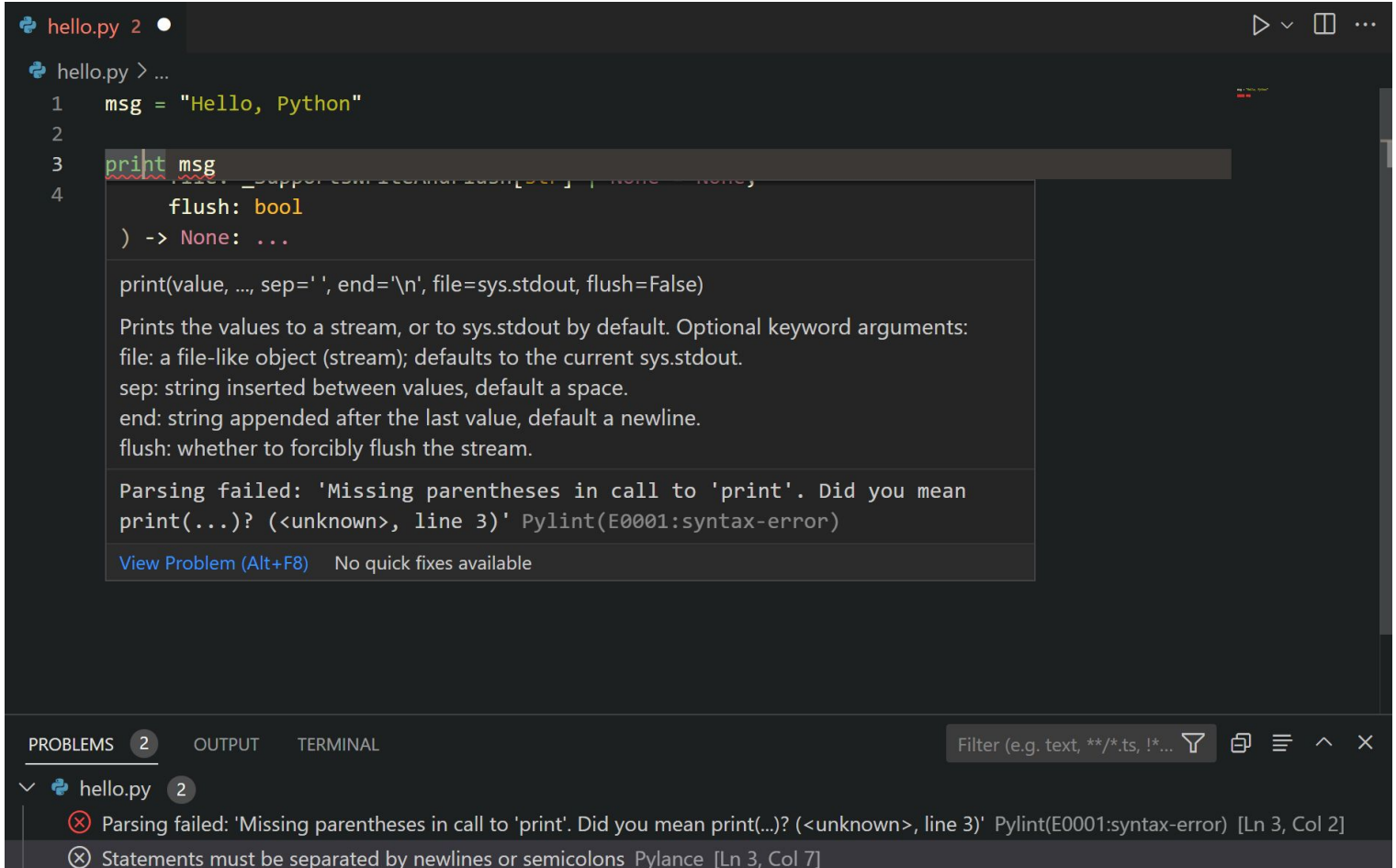


Syntax Highlighting, Linting, Formatting

Tools for coding

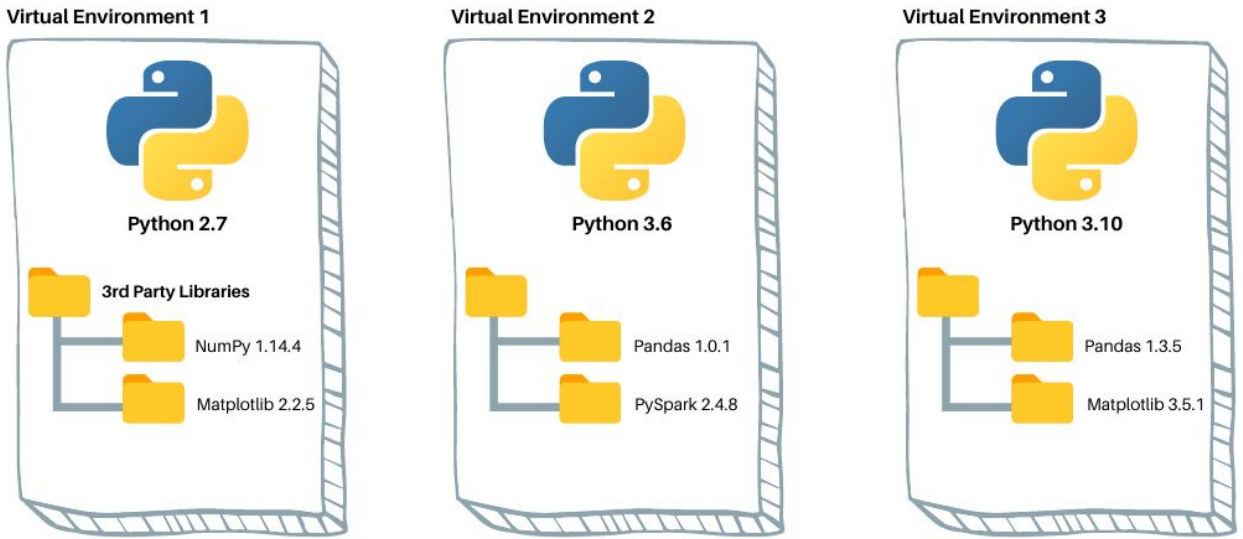
- Linting
- Formatting
- Code environments
- Version control
- Continous integration
- Documention

Linting: Automatically detect errors



Code environments

Do not install everything in one big environment



dataquest.io

Different options:

	conda	venv
Python version (e.g. Python 3.9.7)	easy	hard
Package version (e.g. nltk 3.6.2)	easy	easy

- venv/pip
  - smaller/lightweight
  - cannot change python version
  - only python packages from pypi
- conda/mamba
  - change python version
  - heavier/slower
  - install also other scientific software

Formatting

```python
from seven_dwwarfs import Grumpy, Happy, Sleepy, Bashful, Sneezy,
x = {  'a':37,'b':42,

'c':927}

x = 123456789.123456789E123456789

if very_long_variable_name is not None and \
 very_long_variable_name.field > 0 or \
 very_long_variable_name.is_debug:
 z = 'hello '+'world'
else:
 world = 'world'
 a = 'hello {}'.format(world)
 f = rf'hello {world}'
if (this
and that): y = 'hello ''world'#FIXME: https://github.com/psf/blac
class Foo  (     object  ):
  def f    (self    ):
    return       37*-2
  def g(self, x,y=42):
      return y
def f  (   a: List[ int ]) :
  return      37-a[42-u :  y**3]
```

```python
from seven_dwwarfs import Grumpy, Happy, Sleepy, Bashful, Sneezy, Do

x = {"a": 37, "b": 42, "c": 927}

x = 123456789.123456789e123456789

if (
    very_long_variable_name is not None
    and very_long_variable_name.field > 0
    or very_long_variable_name.is_debug
):
    z = "hello " + "world"
else:
    world = "world"
    a = "hello {}".format(world)
    f = rf"hello {world}"
if this and that:
    y = "hello " "world"  # FIXME: https://github.com/psf/black/issu


class Foo(object):
    def f(self):
        return 37 * -2

    def g(self, x, y=42):
        return y


def f(a: List[int]):
    return 37 - a[42 - u : y**3]
```

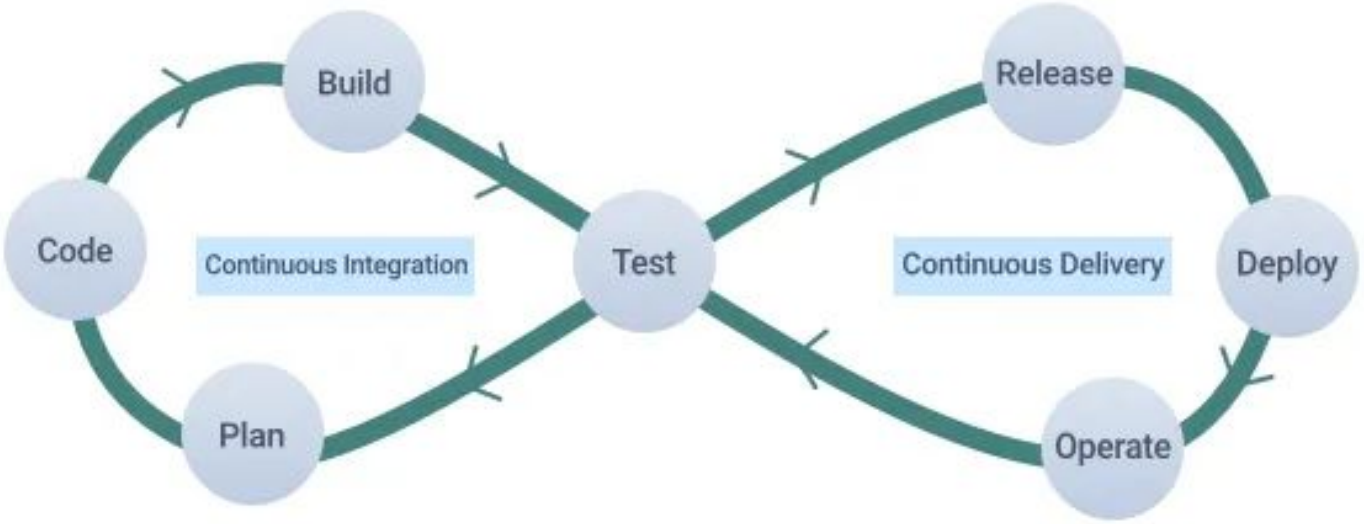
format everytime on save and ensure
consistent formatting with Python standards

Testing

Make sure code does what it is supposed to do

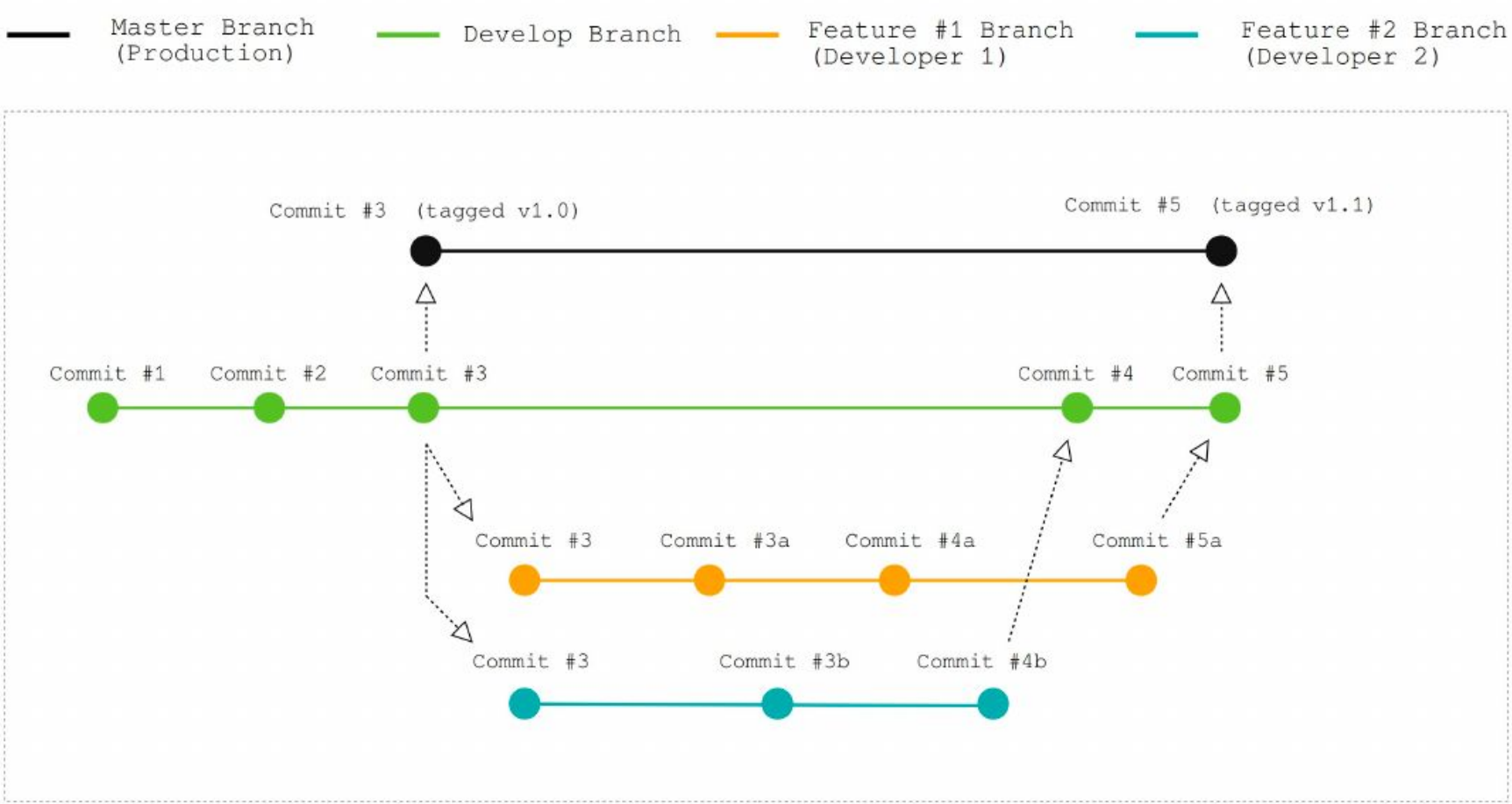
```python
# content of test_sample.py
def inc(x):
    return x + 1


def test_answer():
    assert inc(4) == 5
```

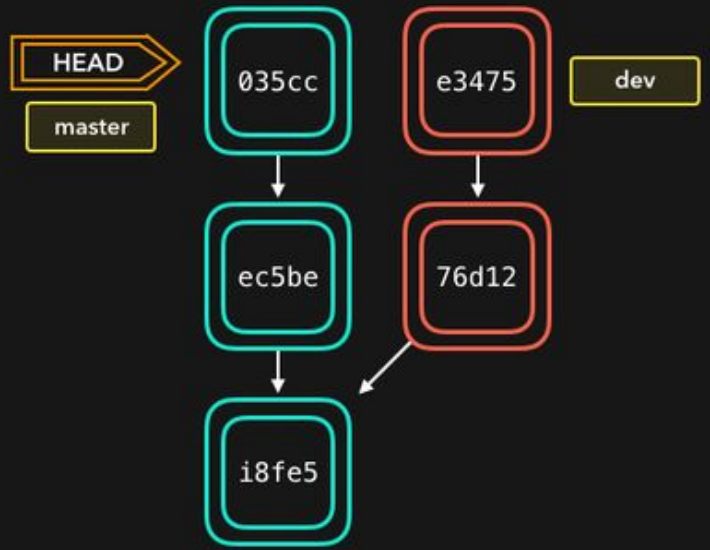
Continuous Integration(CI)

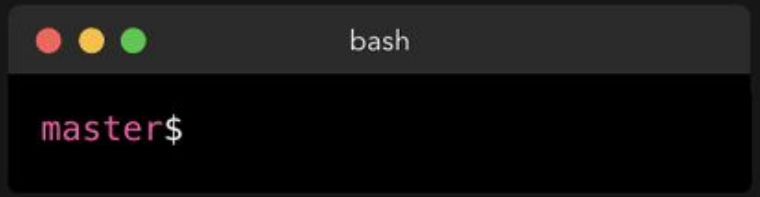


https://medium.com/digital-transformation-and-platform-engineering/a-quick-guide-to-continuous-integration-and-continuous-delivery-4df594ae281

Version control


Master Branch
(Production)
Develop Branch
Feature #1 Branch
(Developer 1)
Feature #2 Branch
(Developer 2)
Commit #3 (tagged v1.0)
Commit #5 (tagged v1.1)
Commit #1
Commit #2
Commit #3
Commit #4
Commit #5
Commit #3
Commit #3a
Commit #4a
Commit #5a
Commit #3
Commit #3b
Commit #4b



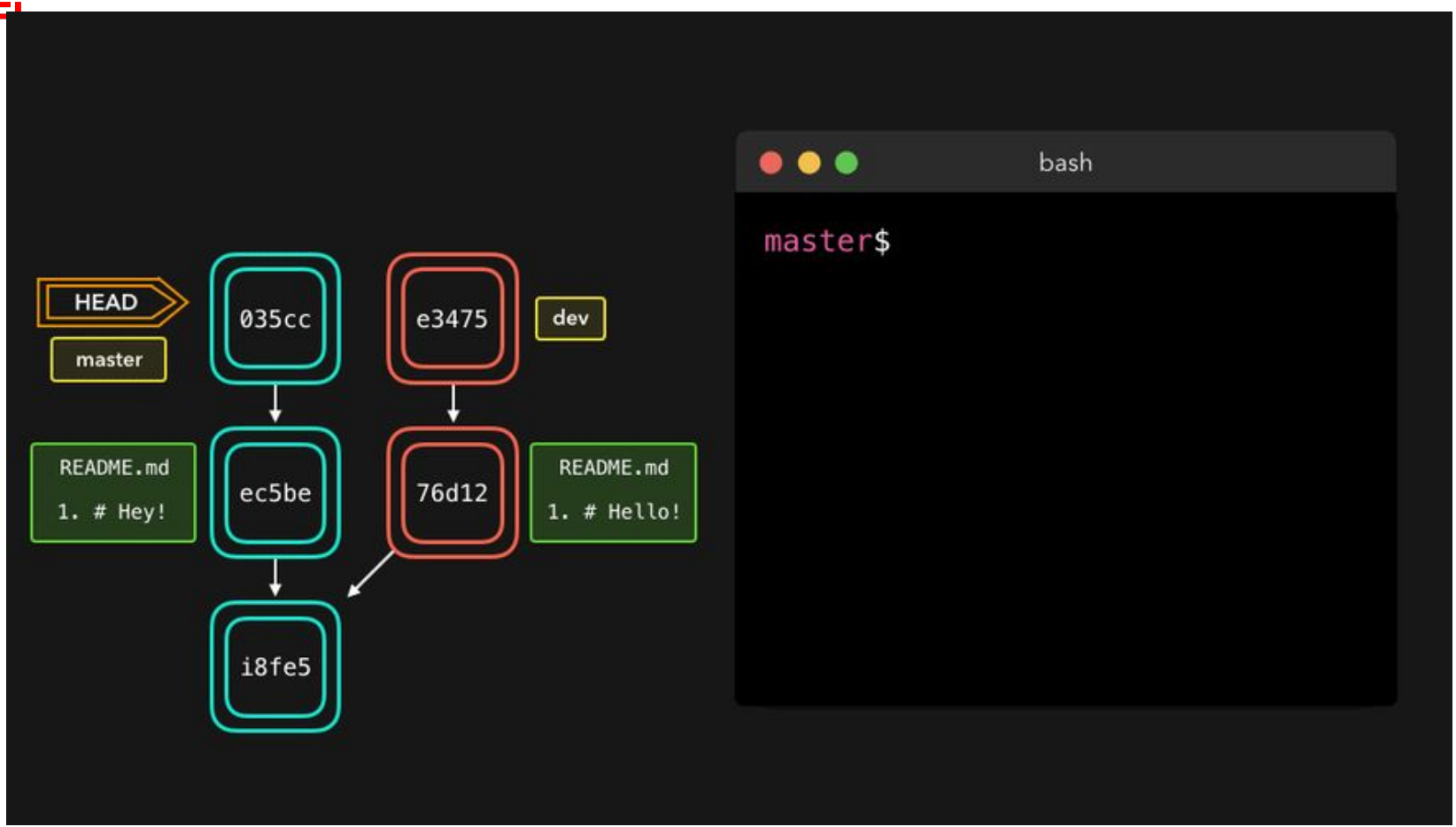


Git | Merging (no-fast-forward)

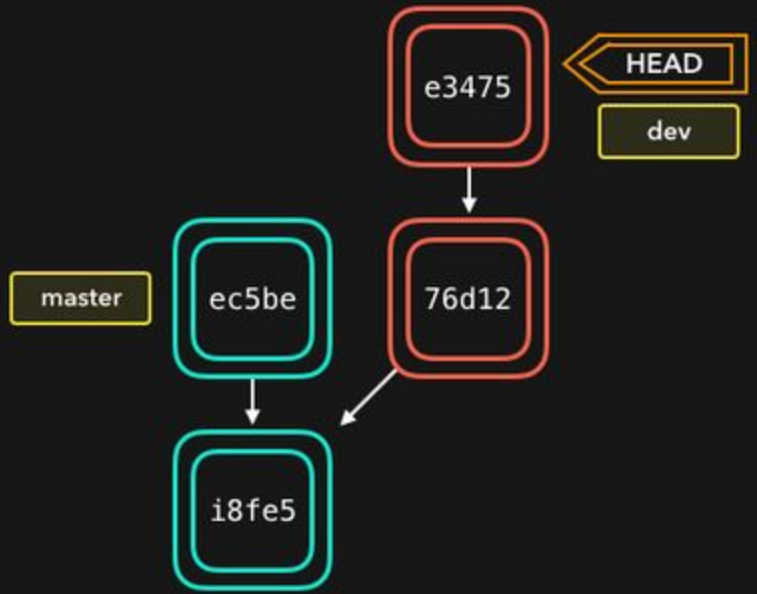
Default behavior when current branch contains commits that the merging branch doesn't have

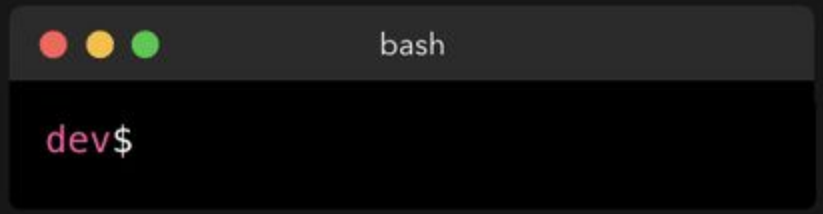
Creates a new commit which merges two branches together without modifying existing branches

https://dev.to/lydiahallie/cs-visualized-useful-git-commands-37p1



https://dev.to/lydiahallie/cs-visualized-useful-git-commands-37p1





Git | Rebasing

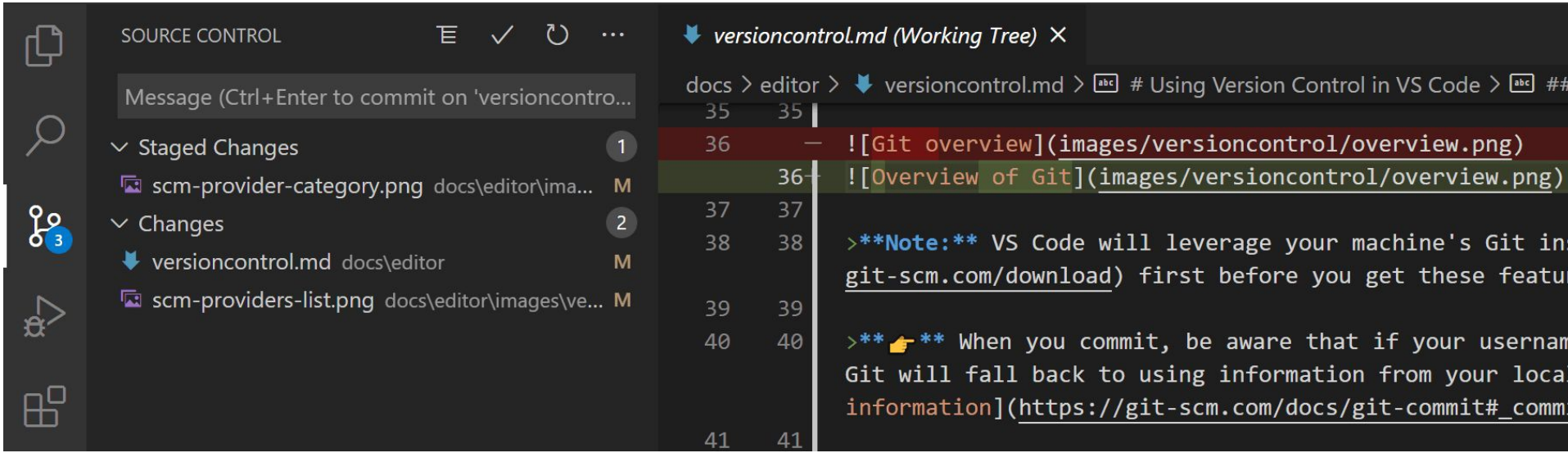
Copies commits on top of another branch
without creating a commit, which keeps a linear history

Changes the history as new hashes are created
for the copied commits

Simon Duerr

https://dev.to/lydiahallie/cs-visualized-useful-git-commands-37p1

VSCode can also help with git workflows



Documentation

```
using System;struct a{static int Main()
{object[]c={"\u0048e\x6c\x6co "+(C\u0068ar)
(86+1)+"or\x6c\x64"};typeof(Conso\u006ce).GetMet
\u0068o\u0064s()[101].Invoke(c,c);return 0;}}
```

worst

script.sh

```python
def simple_math(a, b):

    return a + b

def main():
    num1 = 10
    num2 = 5
    result = simple_math(num1, num2)

    print("Number 1:", num1)
    print("Number 2:", num2)
    print("Result:", sum_result)

if __name__ == "__main__":
    main()
```

bad

Simon Duerr

sum_numbers.sh

good

```python
def sum(a, b):
    """
    Perform summation two numbers.

    Parameters
    ----------
    a : int or float
        The first number.
    b : int or float
        The second number.

    Returns
    -------
    int or float
        The sum of the two numbers.
    """
    return a + b

def main():
    """
    Entry point of the program.
    """
    num1 = 10
    num2 = 5
    result = sum(num1, num2)

    print("Number 1:", num1)
    print("Number 2:", num2)
    print("Result:", result)

if __name__ == "__main__":
    main()
```

excellent

```python
import argparse

def sum(a, b):
    """
    Calculate the sum of two numbers.

    Parameters
    ----------
    a : int or float
        The first number.
    b : int or float
        The second number.

    Returns
    -------
    int or float
        The sum of the two numbers.
    """
    return a + b

def main():
    """
    Entry point of the program.
    """
    parser = argparse.ArgumentParser(description="Calculate the
sum of two numbers.")
    parser.add_argument("num1", type=int, help="First number")
    parser.add_argument("num2", type=int, help="Second number")
    args = parser.parse_args()
    try:
        result = sum(args.num1, args.num2)
    except Exception as e:
        print("An error occurred:", e)

    print("Number 1:", args.num1)
    print("Number 2:", args.num2)
    print("Result:", result)

if __name__ == "__main__":
    main()
```

```python
def foo(arg1, arg2):
    """
    A method's docstring with parameters and return value.

    Use all the cool Sphinx capabilities in this description, e.g. to give
    usage examples ...

    :Example:

    >>> another_class.foo('', AClass())

    :param arg1: first argument
    :type arg1: string
    :param arg2: second argument
    :type arg2: :class:`module.AClass`
    :return: something
    :rtype: string
    :raises: TypeError
    """

        return '' + 1
```

automatically generate documentation website from code

foo(*arg1*, *arg2*)

A method's docstring with parameters and return value.

Use all the cool Sphinx capabilities in this description, e.g. to give usage examples ...

Example:

```
>>> another_class.foo('', AClass())
```

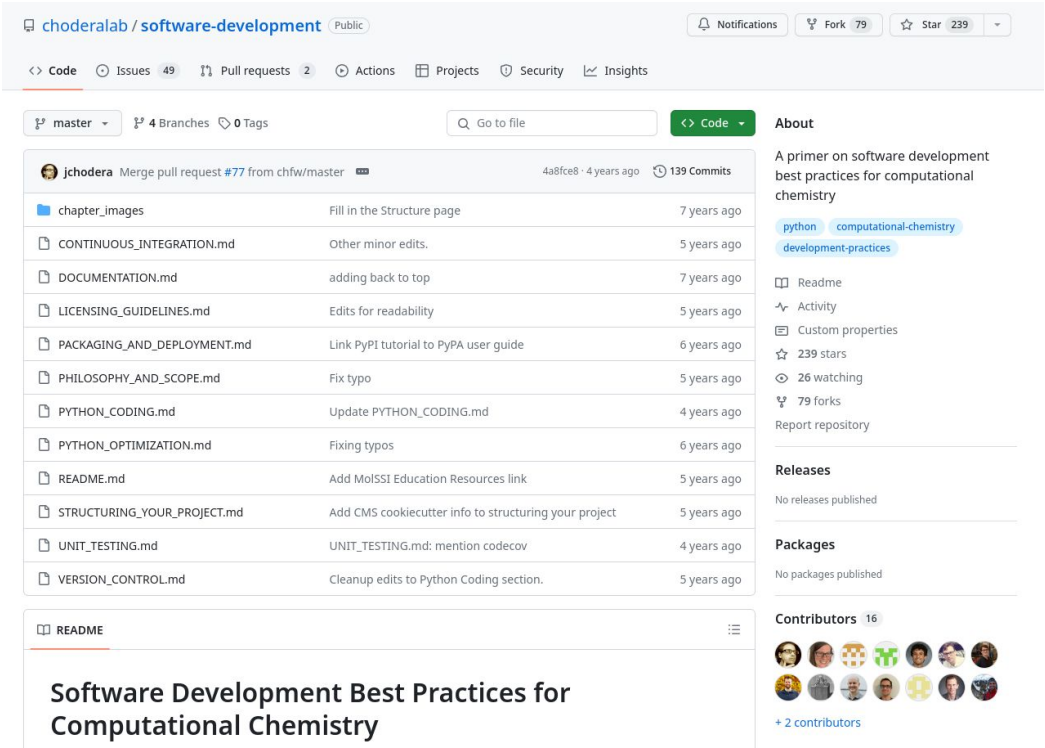
Parameters:	• **arg1** (*string*) – first argument • **arg2** (`module.AClass`) – second argument
Returns:	something
Return type:	string
Raises:	TypeError

Start a project

https://github.com/MolSSI/cookiecutter-cms

Resources

https://github.com/choderalab/software-development



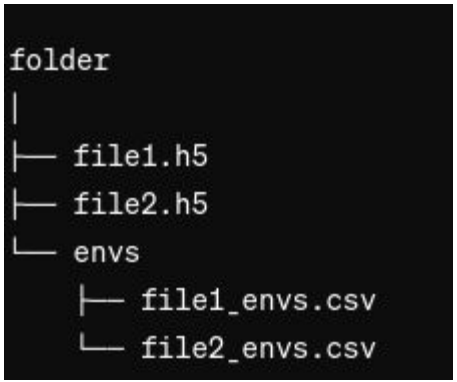
What do you really need?

	Research scripts/Notebooks	**Research code**
Unit tests	-	yes
Publish	Zenodo	Zenodo, PyPi, conda-forge, GitHub
Linting/Formatting	yes	yes
Documentation	inline/docstrings	Web documentation, tutorials (e.g using Sphinx)
Format	e.g Notebook, scripts	e.g Python modules
Environment	environment.yml	Docker container
CI/CD	-	yes
Versioning	yes, e.g git	yes, e.g git


AI

Example case:

I had a ML dataset of h5 files and corresponding csv files



I wanted to split the dataset into a smaller test set and validation set.

Use AI productively for your research data workflows

Be precise and split your commands

You

I have a folder with .h5 files with an identifier. In this folder there is a subfolder "envs" that contains corresponding csv files in the format identifier_envs.csv.

Take all of the files in the folder and move half of them to a folder test and half of them in vals

GPT-3.5 generates non working python code

You

Write a python programm that iterates over a pair of h5 files contained in a folder. Each h5 file has an associated _envs.csv file in the envs folder

generates working example that one can debug

You

Now add functionality to move half the h5/csv pairs to a new directory. Sample them pairs to choose randomly

Adds the required functionalities and results in working code

GITHUB COPILOT: CHAT

monalisa

Write unit tests for this function

GitHub Copilot

```
import unittest
import datetime

class TestParseExpenses(unittest.TestCase):
    def test_parse_expenses_with_valid_input(self):
        expenses_string = '2023-01-02 -34.01 USD\n2023-01-
        expected_output = [(datetime.datetime(2023, 1, 2)
        self.assertEqual(parse_expenses(expenses_string),

    def test_parse_expenses_with_empty_input(self):
        expenses_string = ''
        expected_output = []
        self.assertEqual(parse_expenses(expenses_string),

if __name__ == '__main__':
    unittest.main()
```

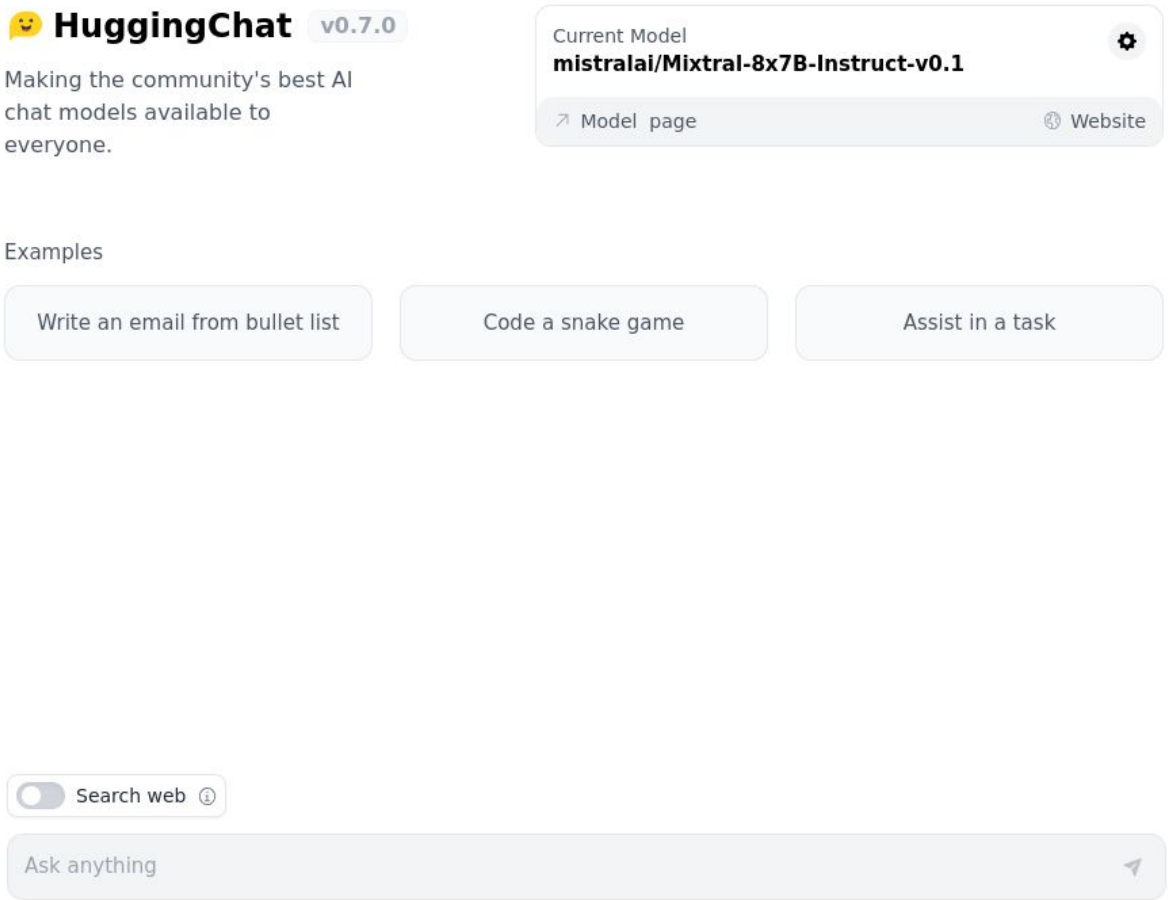
The code assumes that the datetime module is imported. The test cases cover different scenarios such as valid input, empty input, input with comments, invalid date format, and invalid value format.

Ask a question or type '/' for commands

parse_expenses.py | addresses.rb | sentiments.ts

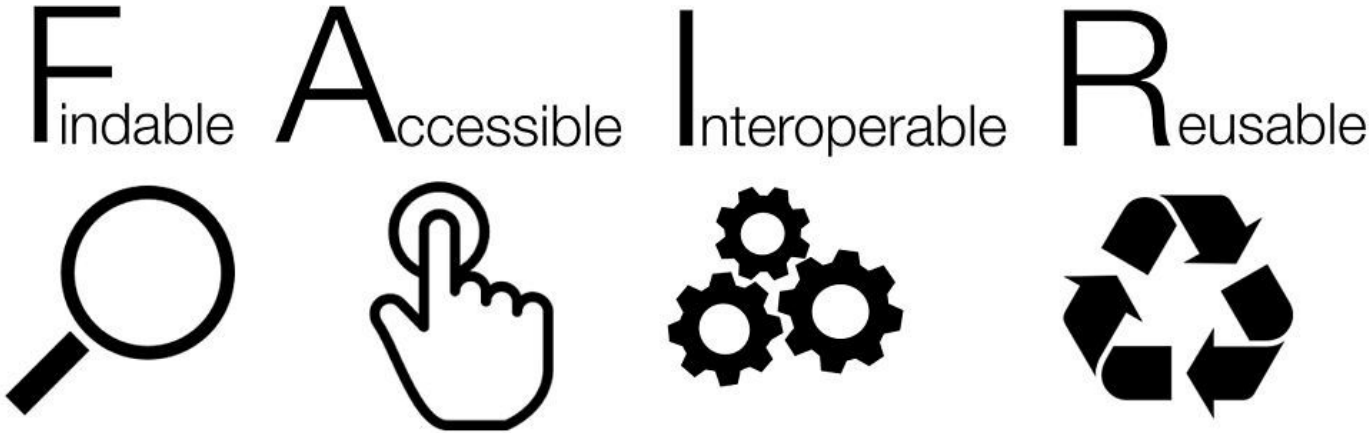
```
import datetime

def parse_expenses (expenses_string):
    """Parse the list of expenses and return the list of triples (date, amount, currency
    Ignore lines starting with #.
    Parse the date using datetime.
    Example expenses_string:
        2023-01-02 -34.01 USD
        2023-01-03 2.59 DKK
        2023-01-03 -2.72 EUR
    """
    expenses = []

    for line in expenses_string.splitlines():
        if line.startswith("#"):
            continue
        date, value, currency = line.split (" ")
        expenses.append((datetime.datetime.strptime (date, "%Y-%m-%d"),
                         float (value),
                         currency))
        return expenses

expenses_data = '''2023-01-02 -34.01 USD
                   2023-01-03 2.59 DKK
                   2023-01-03 -2.72 EUR'''
```

apart
from
chatGPT



Sharing code

We need our code to be FAIR



How to package code

Just giving someone a code file does not mean the code will run as intended.

We need:

- data
- code
- environment specification

So how can we do this?

	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
							

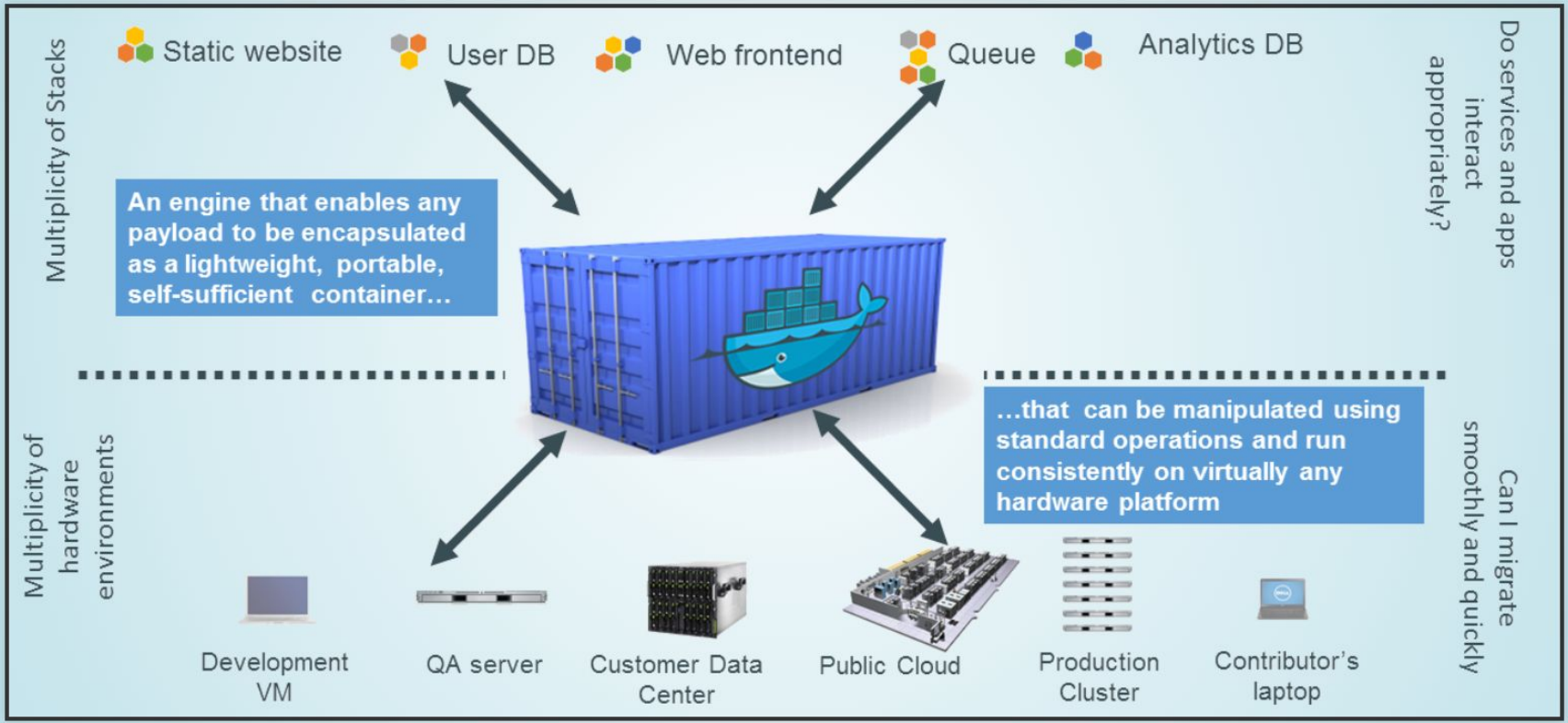
https://pointful.github.io/docker-intro/#/20

Solution: Intermodal Shipping Container

The Matrix from Hell

	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?

Docker is a Container System for Code



Build once... (finally) run *anywhere**

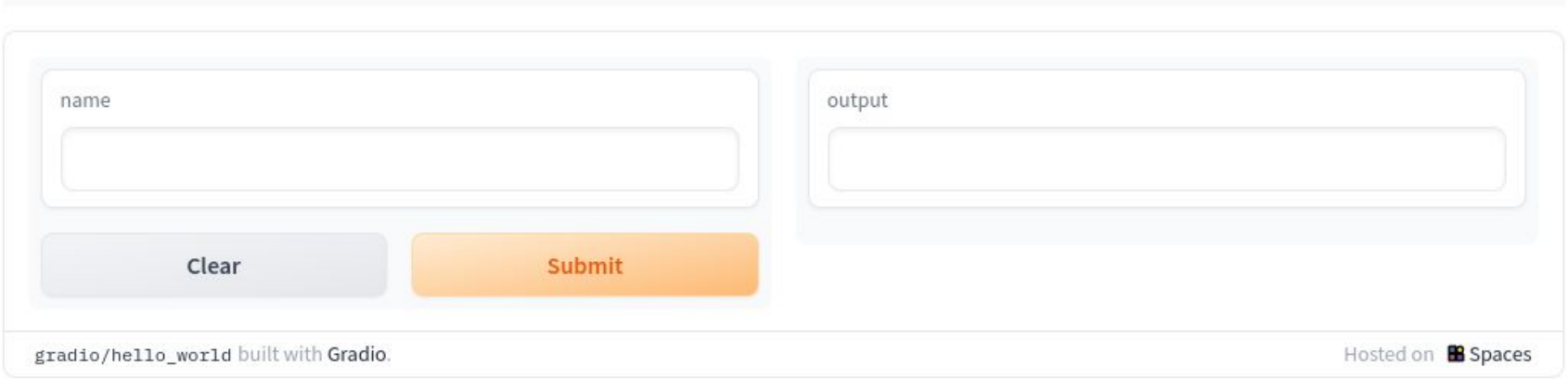
- A clean, safe, hygienic, portable runtime environment for your app.
- No worries about missing dependencies, packages and other pain points during subsequent deployments.

** Where "anywhere" means an x86 server running a modern Linux kernel (3.2+ generally or 2.6.32+ for RHEL 6.5+, Fedora, & related)*

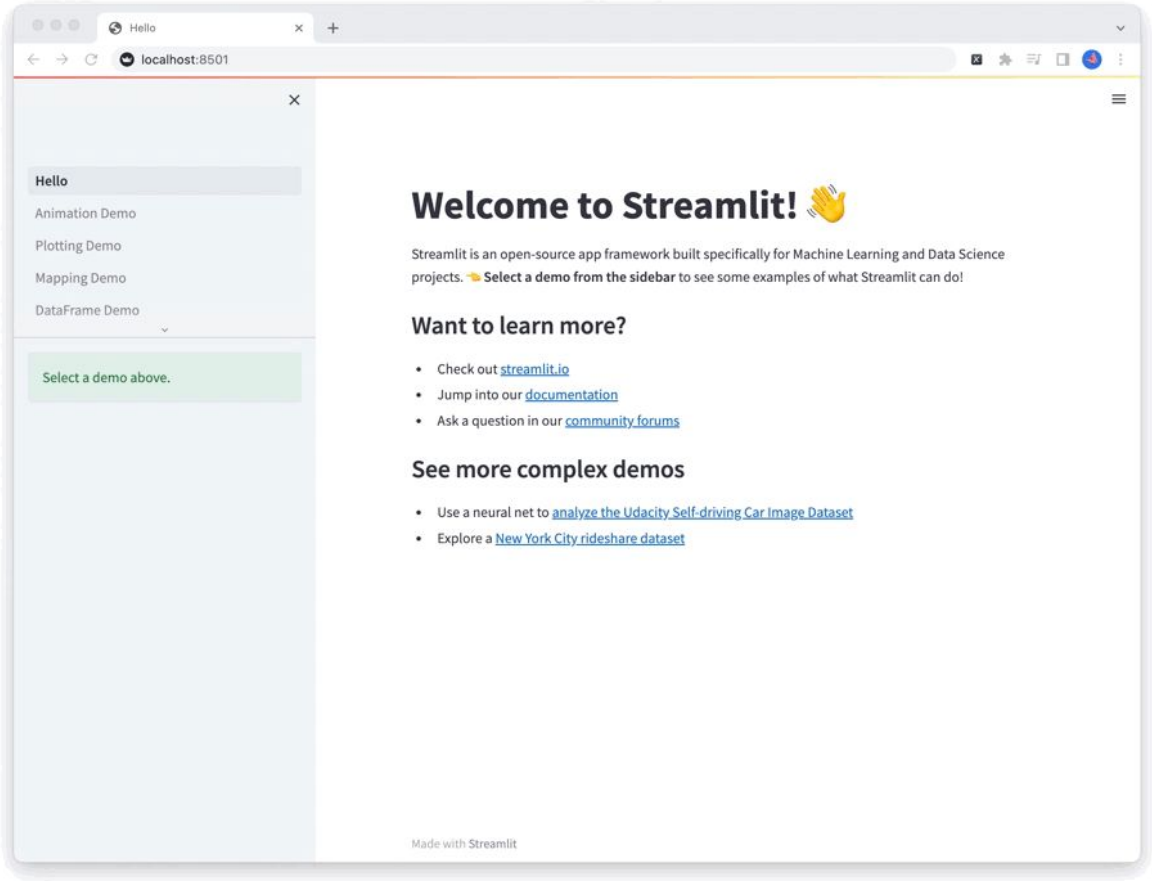
```
FROM python:3.7-slim-buster

COPY install_packages.sh .
RUN ./install_packages.sh

RUN useradd cheminfo

WORKDIR /home/cheminfo

COPY requirements.txt .

COPY dimensionality_reduction ./dimensionality_reduction

COPY README.md .

RUN pip install --no-cache-dir -r requirements.txt
```

Webapps

Gradio

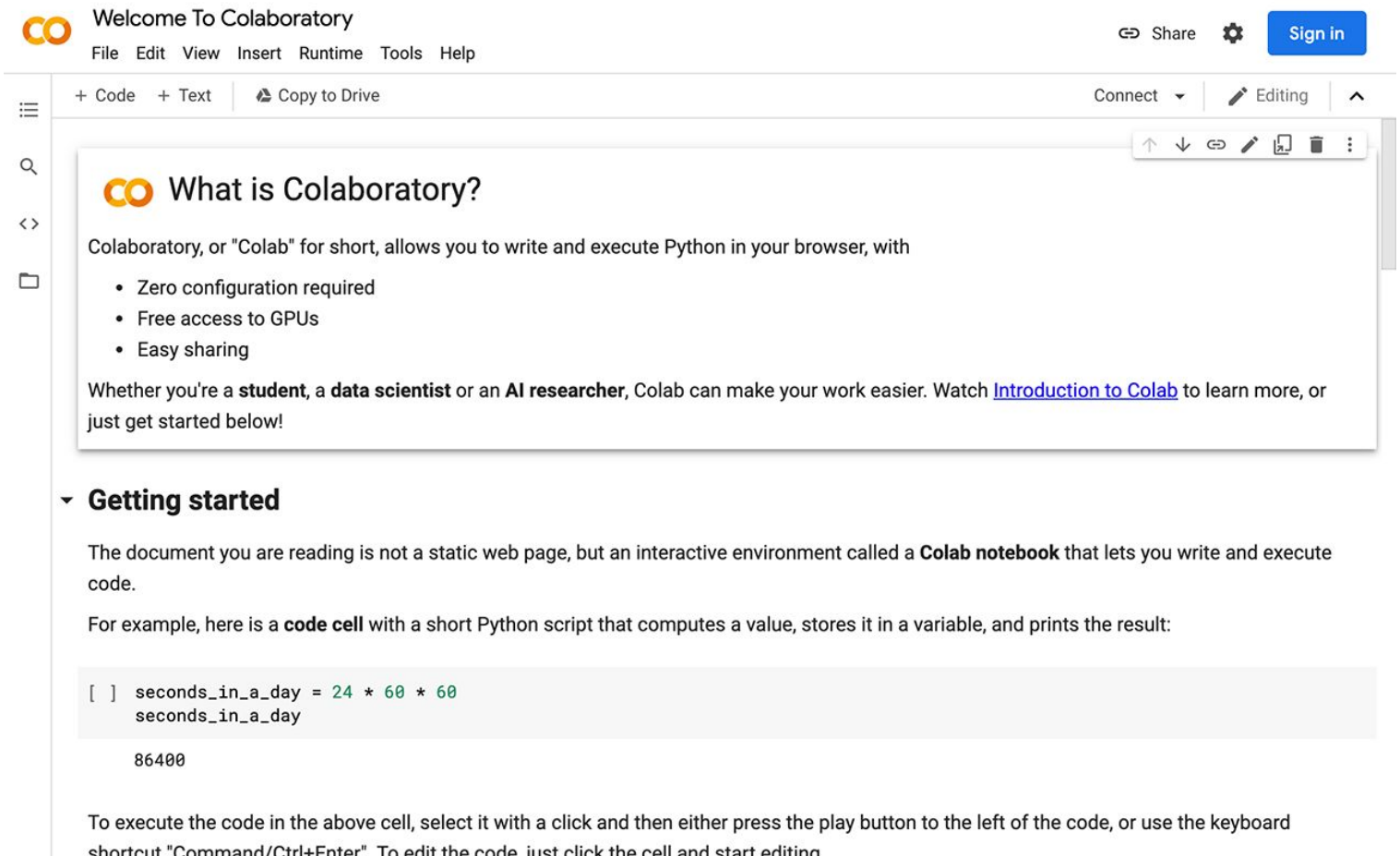


https://www.gradio.app/guides/quickstart



https://docs.streamlit.io/streamlit-community-cloud/get-started/quickstart

Google Colab



AlphaFold2.ipynb

File Edit View Insert Runtime Tools Help

+ Code + Text Copy to Drive

ColabFold v1.5.5: AlphaFold2 using MMseqs2

Easy to use protein structure and complex prediction using AlphaFold2 and Alphafold2-multimer. Sequence alignments/templates are generated through MMseqs2 and HHsearch. For more details, see bottom of the notebook, checkout the ColabFold GitHub and read our manuscript. Old versions: v1.4, v1.5.1, v1.5.2, v1.5.3-patch

Mirdita M, Schütze K, Moriwaki Y, Heo L, Ovchinnikov S, Steinegger M. ColabFold: Making protein folding accessible to all. *Nature Methods*, 2022

Input protein sequence(s), then hit `Runtime` -> `Run all`

query_sequence: PIAQIHILEGRSDEQKETLIREVSEAISRSLDAPLTSVRVIITEMAKGHFGIGGELASK

- Use `:` to specify inter-protein chainbreaks for **modeling complexes** (supports homo- and hetro-oligomers). For example **PI...SK:PI...SK** for a homodimer

jobname: test

num_relax: 0

- specify how many of the top ranked structures to relax using amber

template_mode: none

- `none` = no template information is used. `pdb100` = detect templates in pdb100 (see notes). `custom` - upload and search own templates (PDB or mmCIF format, see notes)

Show code

Install dependencies

Show code

Simon Duerr

colab frequently updates the environment

not reliable enough for sharing research code

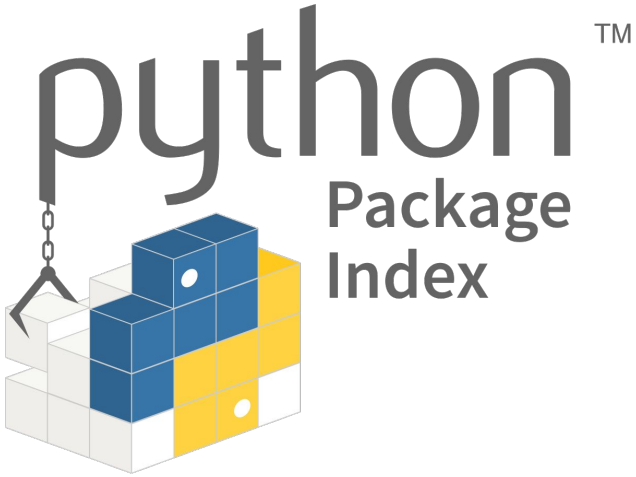
You will need to constantly maintain it

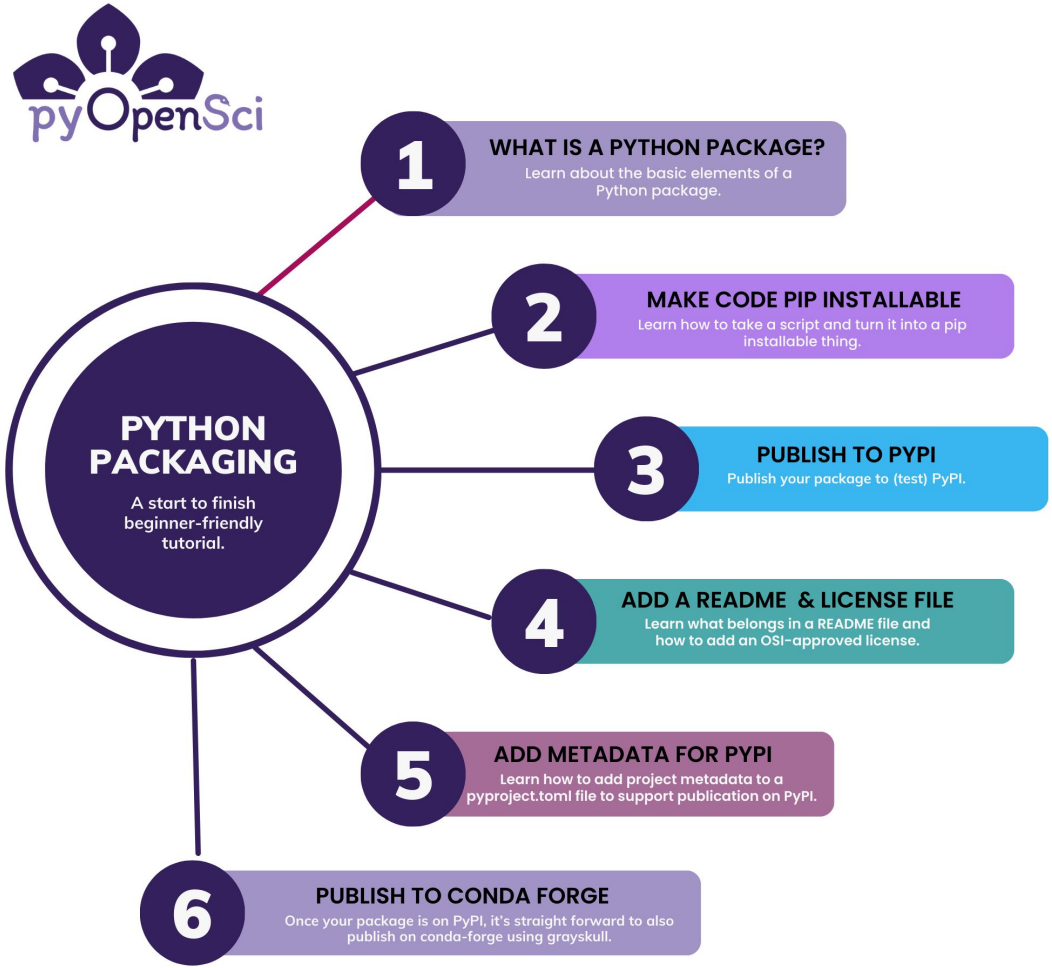
2023-12-18

- Expanded access to AI coding has arrived in Colab across 175 locales for all tiers of Colab users
- Improvements to display of ML-based inline completions (for eligible Pro/Pro+ users)
- Started a series of notebooks highlighting Gemini API capabilities
- Enable ⌘/Ctrl+L to select the full line in an editor
- Fixed bug where we weren't correctly formatting output from multiple execution results
- Python package upgrades
  - CUDA 11.8 to CUDA 12.2
  - tensorflow 2.14.0 -> 2.15.0
  - tensorboard 2.14.0 -> 2.15.0
  - keras 2.14.0 -> 2.15.0
  - Nvidia drivers 525.105.17 -> 535.104.05
  - tensorflow-gcs-config 2.14.0 -> 2.15.0
  - bigframes 0.13.0 -> 0.17.0
  - geemap 0.28.2 -> 0.29.6
  - pyarrow 9.0.0 -> 10.0.1
  - google-generativeai 0.2.2 -> 0.3.1
  - jax 0.4.20 —> 0.4.23
  - jaxlib 0.4.20 —> 0.4.23
- Python package inclusions
  - kagglehub 0.1.4
  - google-cloud-aiplatform 1.38.1

https://colab.research.google.com/notebooks/relnotes.ipynb

Packaging





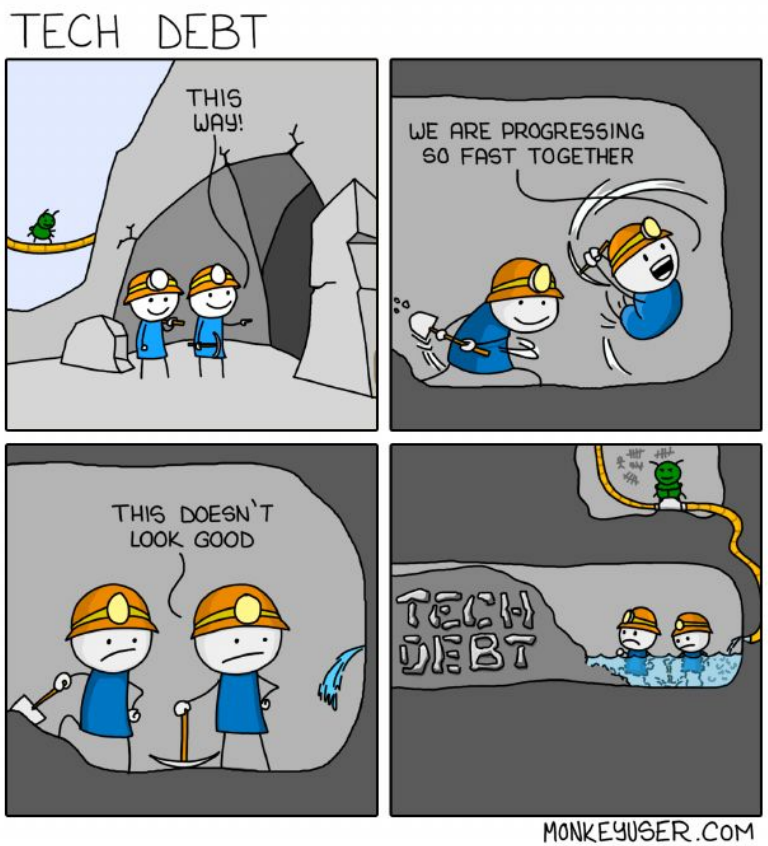
https://www.pyopensci.org/python-package-guide/tutorials/intro.html

Zenodo

Archive your code as is

https://www.youtube.com/watch?v=gp3D4mf6MHQ

When should you start doing this?



Simon Duerr