Code Duplication and Reuse in Jupyter Notebooks

Andreas P. Koenzen, Neil A. Ernst, Margaret-Anne D. Storey
akoenzen@uvic.ca, nernst@uvic.ca, mstorey@uvic.ca
University of Victoria
Victoria, Canada

Pre-print: https://arxiv.org/abs/2005.13709

Presenter: Andreas P. Koenzen

VL/HCC 2020 (August 2020)

Studies Conducted

Study #1: Quantified self-duplicated code snippets in Jupyter notebooks at the repository level **(Artifacts)**

Study #2: Observed participants solving tasks using Jupyter notebooks **(Behaviour)**

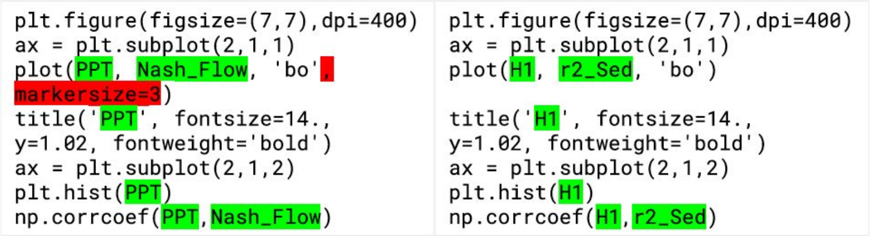
Why..?

Does it happen..? + What method..? + From where..? = Understanding code reuse can lead to tools that expedite data exploration using Jupyter notebooks

Code Duplication (Artifact)

RQ1: How much cell code duplication occurs in Jupyter Notebooks?

Artifacts



```
plt.figure(figsize=(7,7),dpi=400)
ax = plt.subplot(2,1,1)
plot(PPT, Nash_Flow, 'bo',
markersize=3)
title('PPT', fontsize=14.,
y=1.02, fontweight='bold')
ax = plt.subplot(2,1,2)
plt.hist(PPT)
np.corrcoef(PPT,Nash_Flow)
```

```
plt.figure(figsize=(7,7),dpi=400)
ax = plt.subplot(2,1,1)
plot(H1, r2_Sed, 'bo')

title('H1', fontsize=14.,
y=1.02, fontweight='bold')
ax = plt.subplot(2,1,2)
plt.hist(H1)
np.corrcoef(H1,r2_Sed)
```

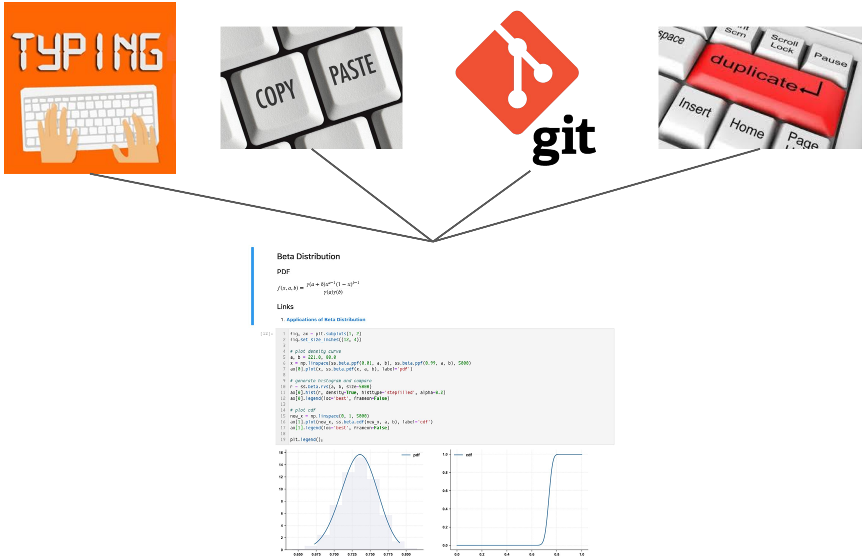
Fig. 4: Example of a Type-2 duplicate detected by our algorithm with Levenshtein distance of 42 and Duplicate Ratio of 0.27. Coded as *Visualization*.

Code Reuse (Behaviour)

RQ2: How does cell code reuse happen in Jupyter notebooks?

How these artifacts are introduced into notebooks

Method



Beta Distribution

PDF

$$f(x, a, b) = \frac{\gamma(a+b)x^{a-1}(1-x)^{b-1}}{\gamma(a)\gamma(b)}$$

Links

1. Applications of Beta Distribution

[12]:

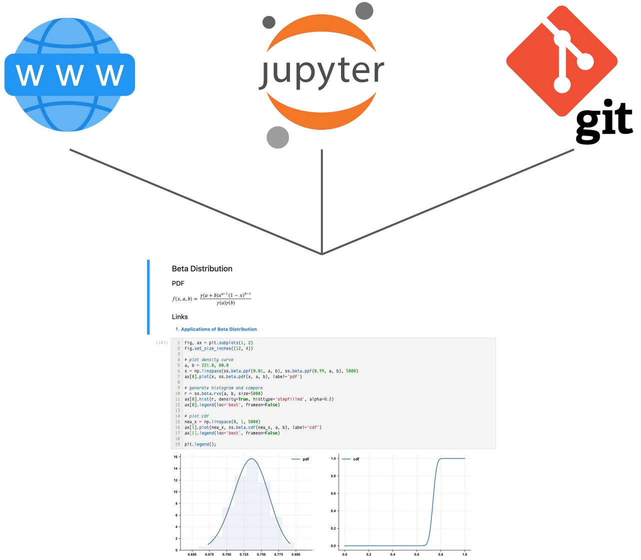
```
fig, ax = plt.subplots(1, 2)
fig.set_size_inches((12, 4))

# plot density curve
a, b = 221.0, 80.0
x = np.linspace(ss.beta.ppf(0.01, a, b), ss.beta.ppf(0.99, a, b), 5000)
ax[0].plot(x, ss.beta.pdf(x, a, b), label='pdf')

# generate histogram and compare
r = ss.beta.rvs(a, b, size=5000)
ax[0].hist(r, density=True, histtype='stepfilled', alpha=0.2)
ax[0].legend(loc='best', frameon=False)

# plot cdf
new_x = np.linspace(0, 1, 5000)
ax[1].plot(new_x, ss.beta.cdf(new_x, a, b), label='cdf')
ax[1].legend(loc='best', frameon=False)

plt.legend();
```

Code Reuse (Behaviour)

RQ3: What are the preferred sources for code reuse in Jupyter notebooks?

From where these artifacts are introduced into notebooks

Source



Beta Distribution

PDF

$$f(x,a,b) = \frac{\gamma(a+b)x^{a-1}(1-x)^{b-1}}{\gamma(a)\gamma(b)}$$

Links

1. Applications of Beta Distribution

[12]:

```
fig, ax = plt.subplots(1, 2)
fig.set_size_inches((12, 4))

# plot density curve
a, b = 221.0, 80.0
x = np.linspace(ss.beta.ppf(0.01, a, b), ss.beta.ppf(0.99, a, b), 5000)
ax[0].plot(x, ss.beta.pdf(x, a, b), label='pdf')

# generate histogram and compare
r = ss.beta.rvs(a, b, size=5000)
ax[0].hist(r, density=True, histtype='stepfilled', alpha=0.2)
ax[0].legend(loc='best', frameon=False)

# plot cdf
new_x = np.linspace(0, 1, 5000)
ax[1].plot(new_x, ss.beta.cdf(new_x, a, b), label='cdf')
ax[1].legend(loc='best', frameon=False)

plt.legend();
```

Image source: https://www.vecteezy.com/free-vector/www-icon

Study #1

- Analyzed GitHub repositories that contained at least 1 Jupyter notebook and computing clones within these repositories

- Thematically coded clones detected before, to assign a computational purpose

Study #1 / Methodology

- Randomly sampled 1,000 repositories from GitHub
- Computed duplicates (Type-1, Type-2 and Type-3) with custom function
- Randomly sampled 500 detected clones from previous step and thematically coded them based on the programming task these clones performed e.g. data visualization

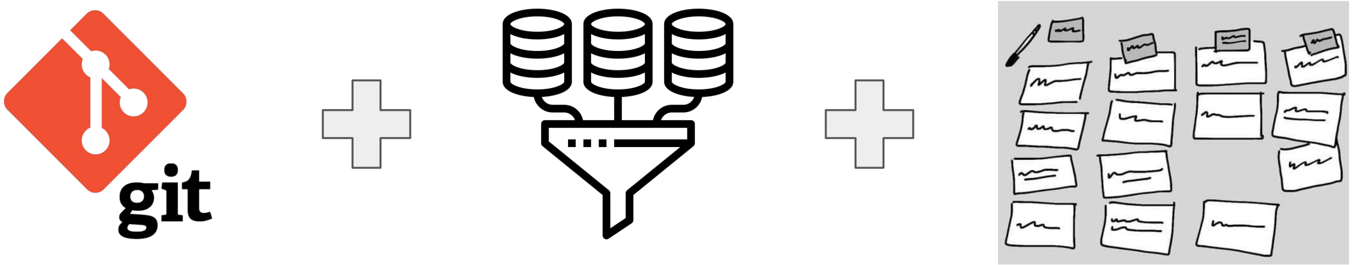
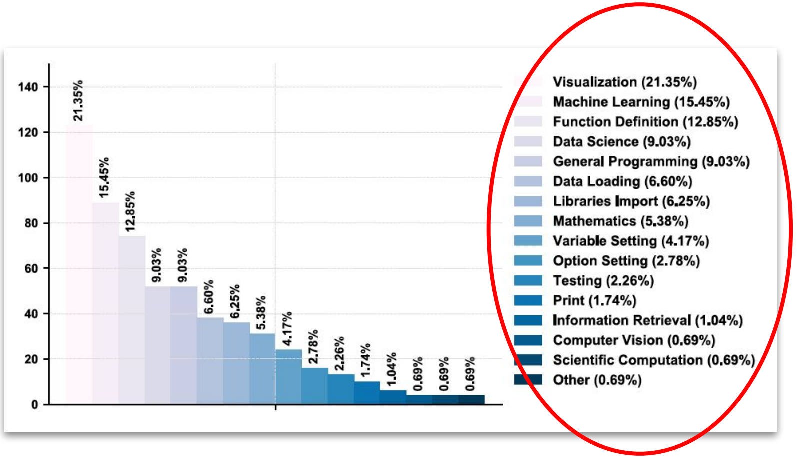


Image source: https://www.flaticon.com/authors/becris

Study #1 / Results

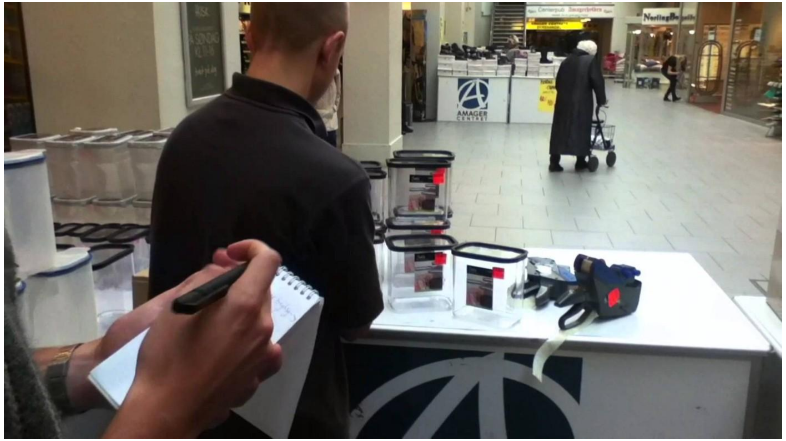
- 1 in 13 code cells are clones
- Visualization routines are duplicated the most



Study #2

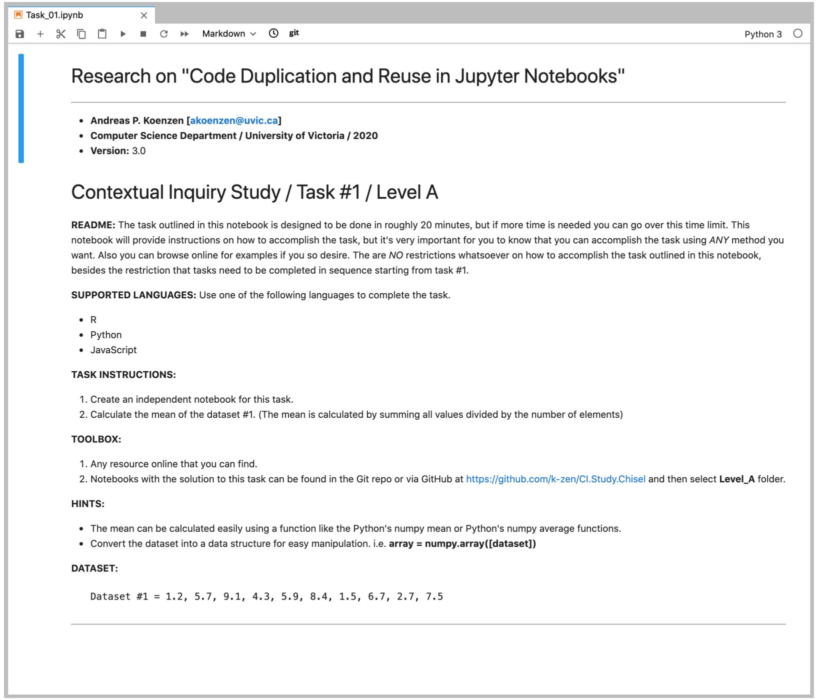
- Observed users perform previously designed specific tasks using Jupyter notebooks

Conducted at the CHISEL lab in the University of Victoria with 8 participants (All students)



Study #2 / Methodology

- Observational study in a lab setting
- Participants had to solve 3 tasks using Jupyter notebooks according to self-reported expertise
- Minimal limitations (Extra time was given if necessary)
- Solutions to tasks were available in the study's git repository



Research on "Code Duplication and Reuse in Jupyter Notebooks"

- **Andreas P. Koenzen [akoenzen@uvic.ca]**
- **Computer Science Department / University of Victoria / 2020**
- **Version:** 3.0

Contextual Inquiry Study / Task #1 / Level A

README: The task outlined in this notebook is designed to be done in roughly 20 minutes, but if more time is needed you can go over this time limit. This notebook will provide instructions on how to accomplish the task, but it's very important for you to know that you can accomplish the task using *ANY* method you want. Also you can browse online for examples if you so desire. The are *NO* restrictions whatsoever on how to accomplish the task outlined in this notebook, besides the restriction that tasks need to be completed in sequence starting from task #1.

SUPPORTED LANGUAGES: Use one of the following languages to complete the task.

- R
- Python
- JavaScript

TASK INSTRUCTIONS:

1. Create an independent notebook for this task.
2. Calculate the mean of the dataset #1. (The mean is calculated by summing all values divided by the number of elements)

TOOLBOX:

1. Any resource online that you can find.
2. Notebooks with the solution to this task can be found in the Git repo or via GitHub at https://github.com/k-zen/CI.Study.Chisel and then select **Level_A** folder.

HINTS:

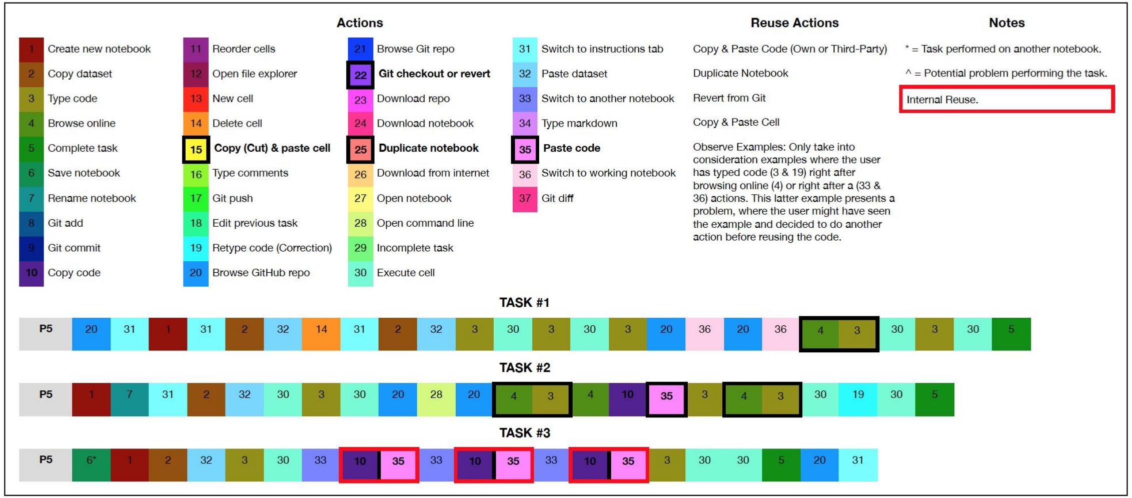
- The mean can be calculated easily using a function like the Python's numpy mean or Python's numpy average functions.
- Convert the dataset into a data structure for easy manipulation. i.e. **array = numpy.array([dataset])**

DATASET:

```
Dataset #1 = 1.2, 5.7, 9.1, 4.3, 5.9, 8.4, 1.5, 6.7, 2.7, 7.5
```

Study #2 / Methodology

- Video coding
- Audio transcriptions
- Notes
- Questions based on observations. i.e. why did you used that particular method
- Short interview and questionnaire



Actions

1	Create new notebook	11	Reorder cells	21	Browse Git repo	31	Switch to instructions tab
2	Copy dataset	12	Open file explorer	22	**Git checkout or revert**	32	Paste dataset
3	Type code	13	New cell	23	Download repo	33	Switch to another notebook
4	Browse online	14	Delete cell	24	Download notebook	34	Type markdown
5	Complete task	15	**Copy (Cut) & paste cell**	25	**Duplicate notebook**	35	**Paste code**
6	Save notebook	16	Type comments	26	Download from internet	36	Switch to working notebook
7	Rename notebook	17	Git push	27	Open notebook	37	Git diff
8	Git add	18	Edit previous task	28	Open command line		
9	Git commit	19	Retype code (Correction)	29	Incomplete task		
10	Copy code	20	Browse GitHub repo	30	Execute cell		

Reuse Actions

Copy & Paste Code (Own or Third-Party)

Duplicate Notebook

Revert from Git

Copy & Paste Cell

Observe Examples: Only take into consideration examples where the user has typed code (3 & 19) right after browsing online (4) or right after a (33 & 36) actions. This latter example presents a problem, where the user might have seen the example and decided to do another action before reusing the code.

Notes

* = Task performed on another notebook.

^ = Potential problem performing the task.

Internal Reuse.

TASK #1

P5	20	31	1	31	2	32	14	31	2	32	3	30	3	30	3	20	36	20	36	4	3	30	3	30	5

TASK #2

P5	1	7	31	2	32	30	3	30	20	28	20	4	3	4	10	35	3	4	3	30	19	30	5

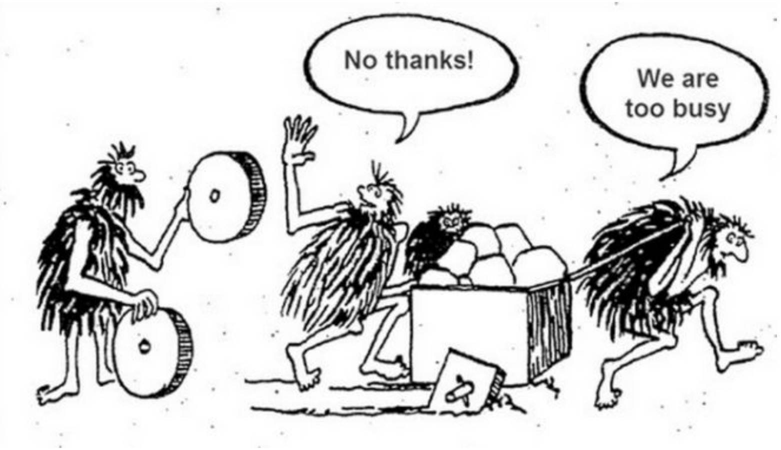
TASK #3

P5	6*	1	2	32	3	30	33	10	35	33	10	35	33	10	35	3	30	30	5	20	31

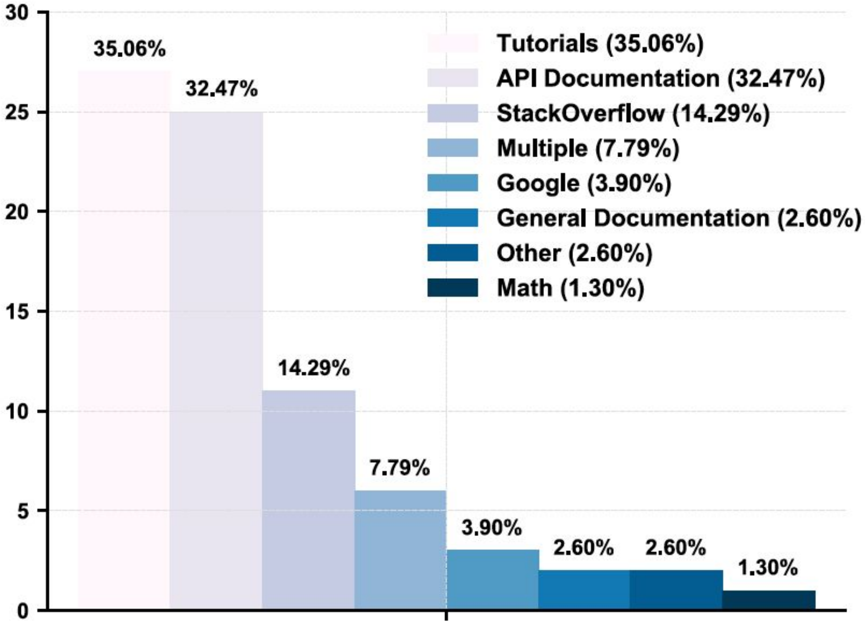
Fig. 2: Example coding of steps one of our participants (P5) made during the observational study, based on video and audio recordings.

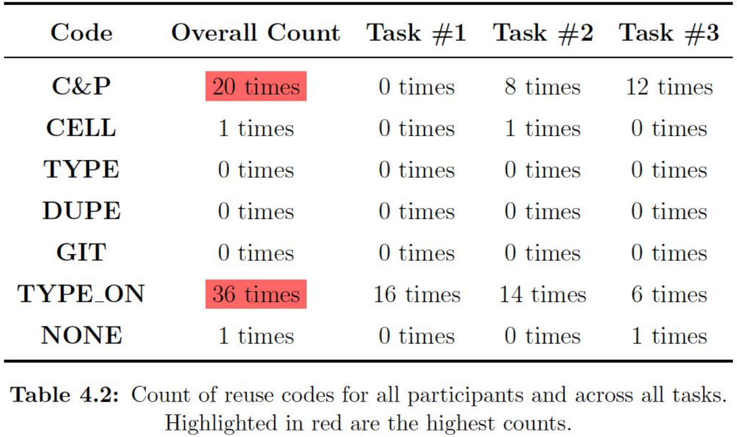
Study #2 / Results

- Participants reused extensively
- External libraries like “numpy” were used extensively
- Most common source of reuse: online sources (18% of total time)
- Least common source of reuse: VCS (0 participants)
- Reutilization was done through copying and pasting, copying by typing of code
- Least common method of reuse: duplicating a notebook



Study #2 / Results





Code	Overall Count	Task #1	Task #2	Task #3
C&P	20 times	0 times	8 times	12 times
CELL	1 times	0 times	1 times	0 times
TYPE	0 times	0 times	0 times	0 times
DUPE	0 times	0 times	0 times	0 times
GIT	0 times	0 times	0 times	0 times
TYPE_ON	36 times	16 times	14 times	6 times
NONE	1 times	0 times	0 times	1 times

Table 4.2: Count of reuse codes for all participants and across all tasks. Highlighted in red are the highest counts.

Limitations

Construct Validity

Function parameters were found empirically (Cut-off value, λ) => Grid search using an oracled data set

Participants were constrained to use lab equipment => Instead of using their own

Internal Validity

Self-assessed level of proficiency => Might introduce Observer-Expectancy bias

Tasks might have been too simple => But we asked proficiency before 👆

External Validity

Notebooks were sampled from GitHub, which may differ from corporate settings

Students differ from industry practitioners

Our findings should be seen as restricted to the sample we used

Future Work

- Observational studies in real settings where participants used their own tools to solve real problems (Advanced students, industry practitioners)

- Harder and more complex problems might shed light into reutilization of complex routines via methods not observed during this study

Summary

- 1 in 13 code cells are clones
- Visualization routines are duplicated the most
- Users reuse extensively when using Jupyter notebooks
- The most common source of reuse is the web
- git only for storage
- Less reuse from own code (Reinvent the wheel attitude)

So what...

- Code reuse support via tools like **Google Colab's Code Snippets** can be beneficial and might speed up the development process
- Extensive codebase should be put into modules (JupyterLab's "**autoreload**" magic)
- **Light version control** with a simple interface is more suitable for Jupyter notebooks

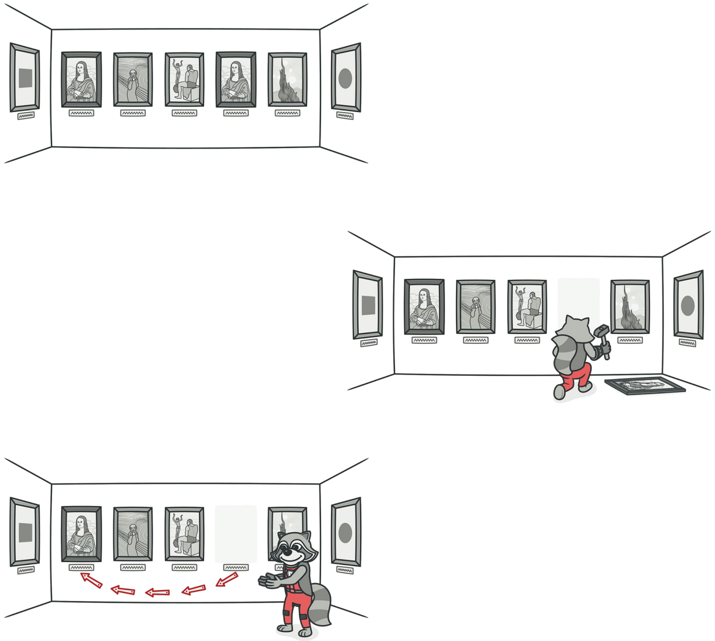
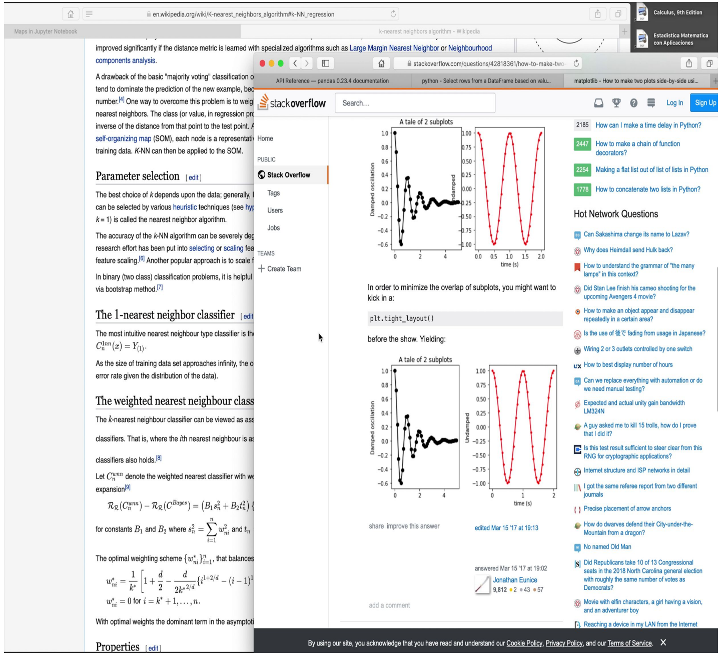


Image source: https://refactoring.guru/smells/duplicate-code

My own personal experience…



en.wikipedia.org/wiki/K-nearest_neighbors_algorithm#k-NN_regression

Maps in Jupyter Notebook | k-nearest neighbors algorithm - Wikipedia

Calculus, 9th Edition

Estadística Matemática con Aplicaciones

improved significantly if the distance metric is learned with specialized algorithms such as Large Margin Nearest Neighbor or Neighbourhood components analysis.

A drawback of the basic "majority voting" classification o… tend to dominate the prediction of the new example, bec… number.[4] One way to overcome this problem is to weig… nearest neighbors. The class (or value, in regression pro… inverse of the distance from that point to the test point. A… self-organizing map (SOM), each node is a representativ… training data. K-NN can then be applied to the SOM.

Parameter selection [edit]

The best choice of k depends upon the data; generally, l… can be selected by various heuristic techniques (see hyp… k = 1) is called the nearest neighbor algorithm.

The accuracy of the k-NN algorithm can be severely deg… research effort has been put into selecting or scaling fea… feature scaling.[6] Another popular approach is to scale f…

In binary (two class) classification problems, it is helpful… via bootstrap method.[7]

The 1-nearest neighbor classifier [edit]

The most intuitive nearest neighbour type classifier is th… $C_n^{1nn}(x) = Y_{(1)}$.

As the size of training data set approaches infinity, the o… error rate given the distribution of the data).

The weighted nearest neighbour class…

The k-nearest neighbour classifier can be viewed as ass… classifiers. That is, where the ith nearest neighbour is as… classifiers also holds.[8]

Let C_n^{wnn} denote the weighted nearest classifier with we… expansion[9]

$$\mathcal{R}_\mathcal{R}(C_n^{wnn}) - \mathcal{R}_\mathcal{R}(C^{Bayes}) = (B_1 s_n^2 + B_2 t_n^2)\{$$

for constants B_1 and B_2 where $s_n^2 = \sum_{i=1}^{n} w_{ni}^2$ and t_n

The optimal weighting scheme $\{w_{ni}^*\}_{i=1}^n$, that balances…

$$w_{ni}^* = \frac{1}{k^*}\left[1 + \frac{d}{2} - \frac{d}{2k^{*2/d}}\{i^{1+2/d} - (i-1)^{1\dots}$$

$w_{ni}^* = 0$ for $i = k^* + 1, \dots, n$.

With optimal weights the dominant term in the asymptoti…

Properties [edit]

stackoverflow.com/questions/42818361/how-to-make-two…

API Reference — pandas 0.23.4 documentation | python - Select rows from a DataFrame based on valu… | matplotlib - How to make two plots side-by-side usi…

stackoverflow | Search… | Log In | Sign Up

Home

PUBLIC

Stack Overflow

Tags

Users

Jobs

TEAMS

+ Create Team

In order to minimize the overlap of subplots, you might want to kick in a:

```
plt.tight_layout()
```

before the show. Yielding:

share improve this answer | edited Mar 15 '17 at 19:13

answered Mar 15 '17 at 19:02

Jonathan Eunice

9,812 ●2 ●43 ●57

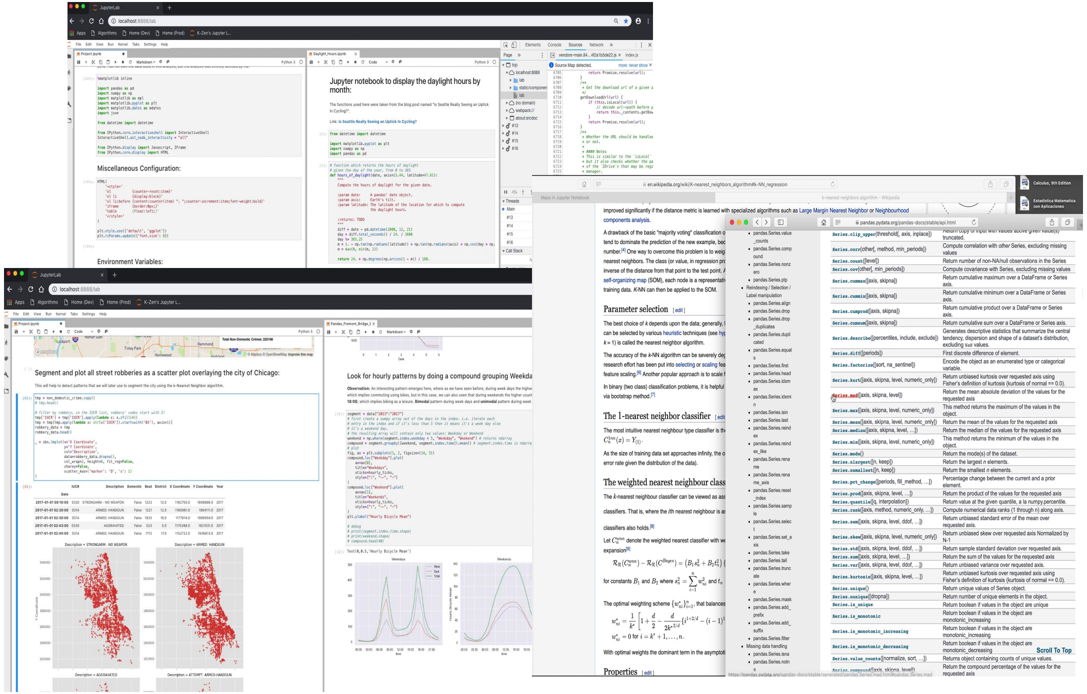
add a comment

2185	How can I make a time delay in Python?
2447	How to make a chain of function decorators?
2254	Making a flat list out of list of lists in Python
1778	How to concatenate two lists in Python?

Hot Network Questions

- Can Sakashima change its name to Lazav?
- Why does Heimdall send Hulk back?
- How to understand the grammar of "the many lamps" in this context?
- Did Stan Lee finish his cameo shooting for the upcoming Avengers 4 movie?
- How to make an object appear and disappear repeatedly in a certain area?
- Is the use of 後で fading from usage in Japanese?
- Wiring 2 or 3 outlets controlled by one switch
- How to best display number of hours
- Can we replace everything with automation or do we need manual testing?
- Expected and actual unity gain bandwidth LM324N
- A guy asked me to kill 15 trolls, how do I prove that I did it?
- Is this test result sufficient to steer clear from this RNG for cryptographic applications?
- Internet structure and ISP networks in detail
- I got the same referee report from two different journals
- Precise placement of arrow anchors
- How do dwarves defend their City-under-the-Mountain from a dragon?
- No named Old Man
- Did Republicans take 10 of 13 Congressional seats in the 2018 North Carolina general election with roughly the same number of votes as Democrats?
- Movie with elfin characters, a girl having a vision, and an adventurer boy
- Reaching a device in my LAN from the Internet

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service. ✕



JupyterLab | localhost:8888/lab

Apps | Algorithms | Home (Dev) | Home (Prod) | K-Zen's Jupyter L…

File Edit View Run Kernel Tabs Settings Help

Project.ipynb | Daylight_Hours.ipynb

Jupyter notebook to display the daylight hours by month:

Miscellaneous Configuration:

Environment Variables:

Segment and plot all street robberies as a scatter plot overlaying the city of Chicago:

Look for hourly patterns by doing a compound grouping Weekda…

Elements | Console | Sources | Network

en.wikipedia.org/wiki/K-nearest_neighbors_algorithm#k-NN_regression

pandas.pydata.org/pandas-docs/stable/api.html

Series.clip_upper([threshold, axis, inplace])

Series.corr(other[, method, min_periods]) — Compute correlation with other Series, excluding missing values

Series.count([level]) — Return number of non-NA/null observations in the Series

Series.cov(other[, min_periods]) — Compute covariance with Series, excluding missing values

Series.cummax([axis, skipna]) — Return cumulative maximum over a DataFrame or Series axis.

Series.cummin([axis, skipna]) — Return cumulative minimum over a DataFrame or Series axis.

Series.cumprod([axis, skipna]) — Return cumulative product over a DataFrame or Series axis.

Series.cumsum([axis, skipna]) — Return cumulative sum over a DataFrame or Series axis.

Series.describe([percentiles, include, exclude]) — Generates descriptive statistics that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values.

Series.diff([periods]) — First discrete difference of element.

Series.factorize([sort, na_sentinel]) — Encode the object as an enumerated type or categorical variable.

Series.kurt([axis, skipna, level, numeric_only]) — Return unbiased kurtosis over requested axis using Fisher's definition of kurtosis (kurtosis of normal == 0.0).

Series.mad([axis, skipna, level]) — Return the mean absolute deviation of the values for the requested axis

Series.max([axis, skipna, level, numeric_only]) — This method returns the maximum of the values in the object.

Series.mean([axis, skipna, level, numeric_only]) — Return the mean of the values for the requested axis

Series.median([axis, skipna, level, …]) — Return the median of the values for the requested axis

Series.min([axis, skipna, level, numeric_only]) — This method returns the minimum of the values in the object.

Series.mode() — Return the mode(s) of the dataset.

Series.nlargest([n, keep]) — Return the largest n elements.

Series.nsmallest([n, keep]) — Return the smallest n elements.

Series.pct_change([periods, fill_method, …]) — Percentage change between the current and a prior element.

Series.prod([axis, skipna, level, …]) — Return the product of the values for the requested axis

Series.quantile([q, interpolation]) — Return value at the given quantile, a la numpy.percentile.

Series.rank([axis, method, numeric_only, …]) — Compute numerical data ranks (1 through n) along axis.

Series.sem([axis, skipna, level, ddof, …]) — Return unbiased standard error of the mean over requested axis.

Series.skew([axis, skipna, level, numeric_only]) — Return unbiased skew over requested axis Normalized by N-1

Series.std([axis, skipna, level, ddof, …]) — Return sample standard deviation over requested axis.

Series.sum([axis, skipna, level, …]) — Return the sum of the values for the requested axis

Series.var([axis, skipna, level, ddof, …]) — Return unbiased variance over requested axis.

Series.kurtosis([axis, skipna, level, …]) — Return unbiased kurtosis over requested axis using Fisher's definition of kurtosis (kurtosis of normal == 0.0).

Series.unique() — Return unique values of Series object.

Series.nunique([dropna]) — Return number of unique elements in the object.

Series.is_unique — Return boolean if values in the object are unique

Series.is_monotonic — Return boolean if values in the object are monotonic_increasing

Series.is_monotonic_increasing — Return boolean if values in the object are monotonic_increasing

Series.is_monotonic_decreasing — Return boolean if values in the object are monotonic_decreasing

Series.value_counts([normalize, sort, …]) — Returns object containing counts of unique values.

Series.compound([axis, skipna, level]) — Return the compound percentage of the values for the requested axis

Scroll To Top

✓ Motivation ✓ Study #1 ✓ Study #2 ✓ Limitations ✓ Future Work ✓ Conclusion

Q&A

Reminder:

- 1 in 13 code cells are clones
- Visualization routines are duplicated the most
- Users reuse extensively when using Jupyter notebooks
- The most common source of reuse is the web
- git only for storage
- Less reuse from own code (Reinvent the wheel attitude)

Andreas P. Koenzen, Neil A. Ernst, Margaret-Anne D. Storey
akoenzen@uvic.ca, nernst@uvic.ca, mstorey@uvic.ca
University of Victoria
Victoria, Canada

Pre-print: https://arxiv.org/abs/2005.13709