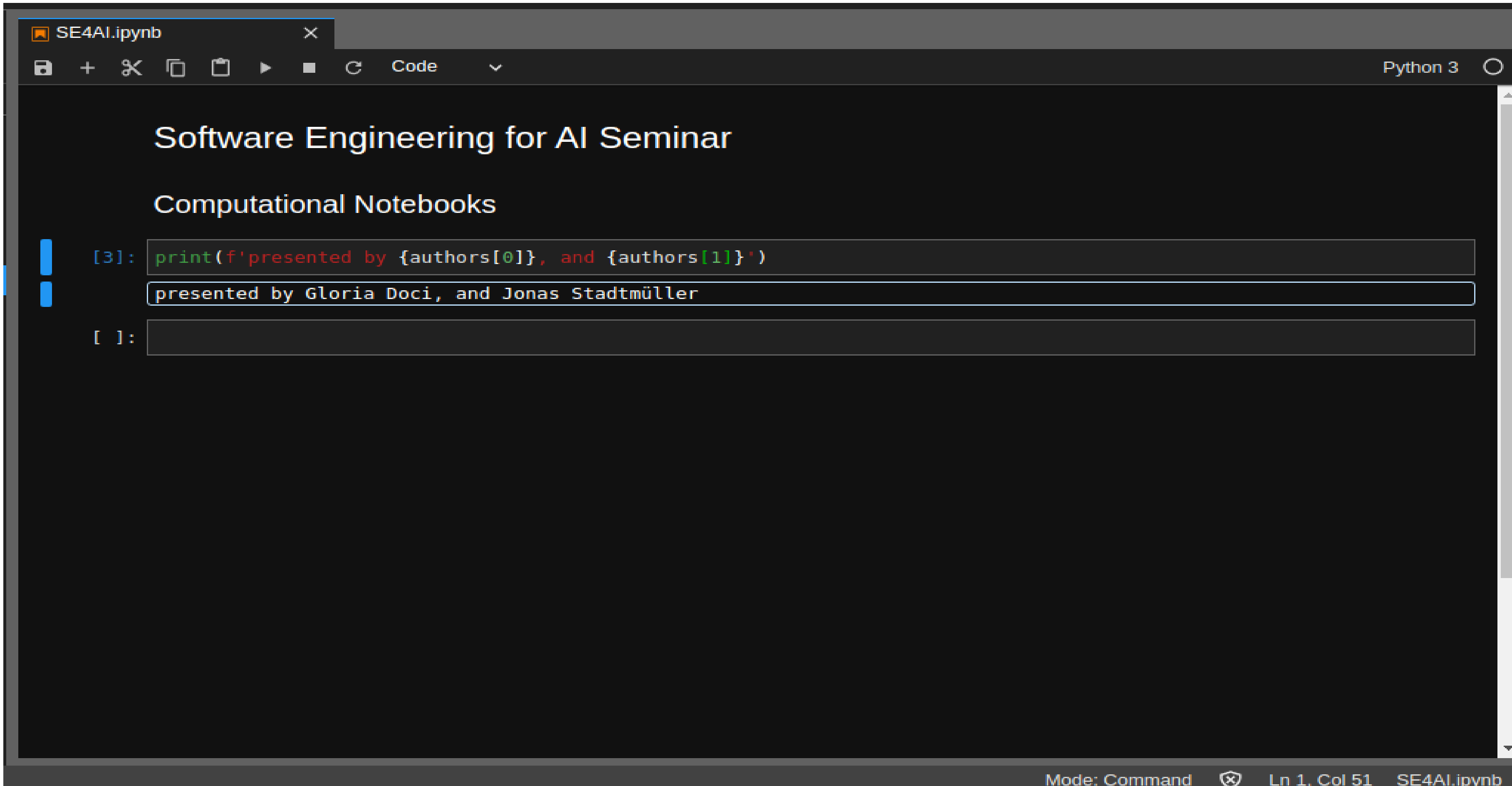
Computational Notebooks

30.06.2020

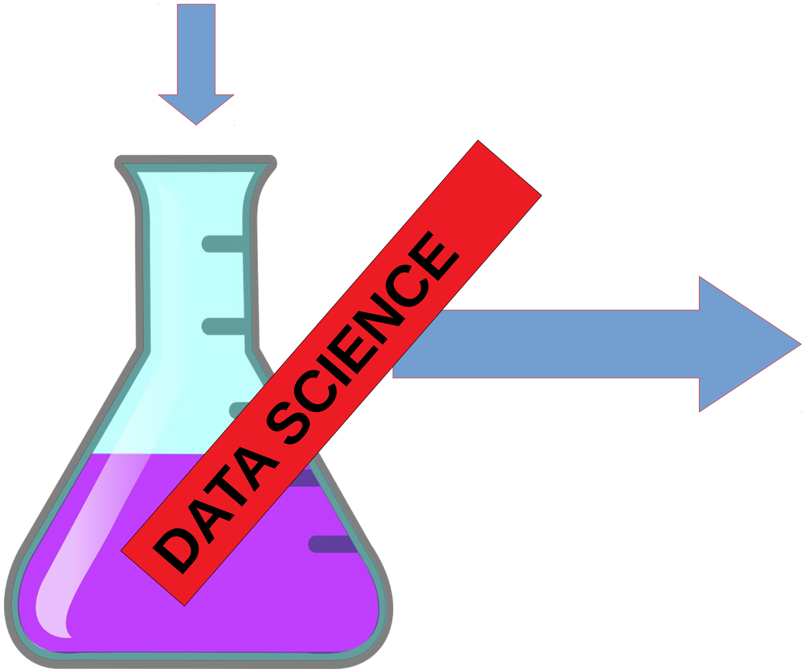


Outline

- Motivation
- Strong points
- Pain points & messiness
- Existing approaches and solutions
- Conclusion & Outlook

Motivation

- Big data explosion
- Advancements in computing hardware(GPU, TPU)
- Advancements in ML



Gain insights over data for better decision making, innovations and improvements

Foundation of Notebooks

- Data science is open-ended, highly interactive, exploratory and iterative
- Wide range of contexts and audiences → narrative is central [1]
- Literate programming paradigm (1984) by Donald Knuth [2] combines code snippets and macros to make the program more understandable to humans (WEB = Pascal + TeX)
- Computational notebooks are tools for interactive and exploratory computing to support scientific computing and data science

Computational Notebooks

- Traditionally used in labs to document research computations and findings
- Computational notebooks make possible to include code, data analysis and visualizations into a single document

Mathematica
1988

- Focus today is on open access and reproducibility of data analyses

Computational Notebooks

- The code executes in a kernel, but the interface is easy to use
- In data science mostly used for visualization, statistical analysis, classical ML and DNN [3]

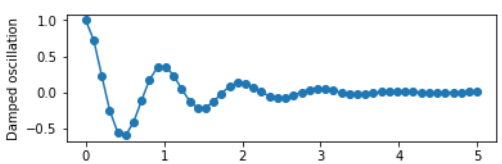
Example using matplotlib

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

[6]:
```
# Here we can write code eg.to read data [from .csv files]
# And then analyze them
```

[10]:
```
x = np.linspace(0.0, 5.0)
y = np.cos(2 * np.pi * x) * np.exp(-x)
plt.subplot(2, 1, 1)
plt.plot(x, y, 'o-')
plt.ylabel('Damped oscillation')

plt.show()
```



[11]:
```
print('Another code cell')
```

```
Another code cell
```

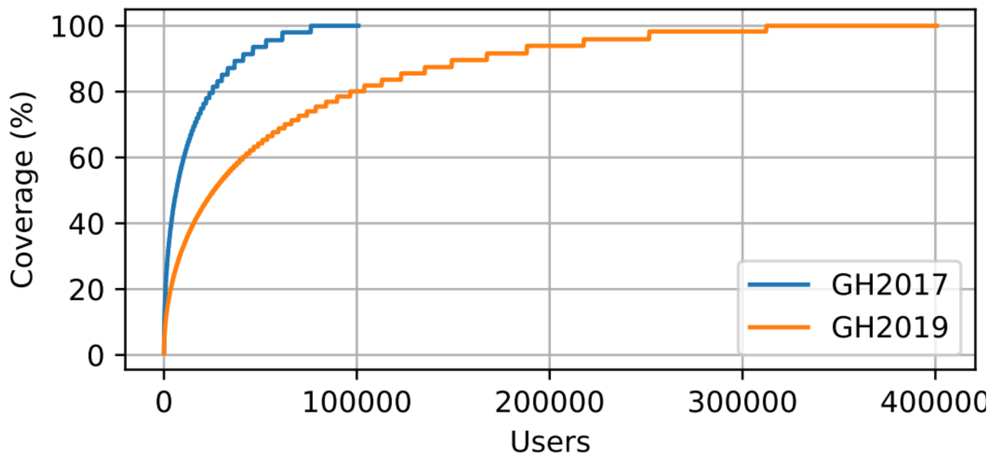
input cells

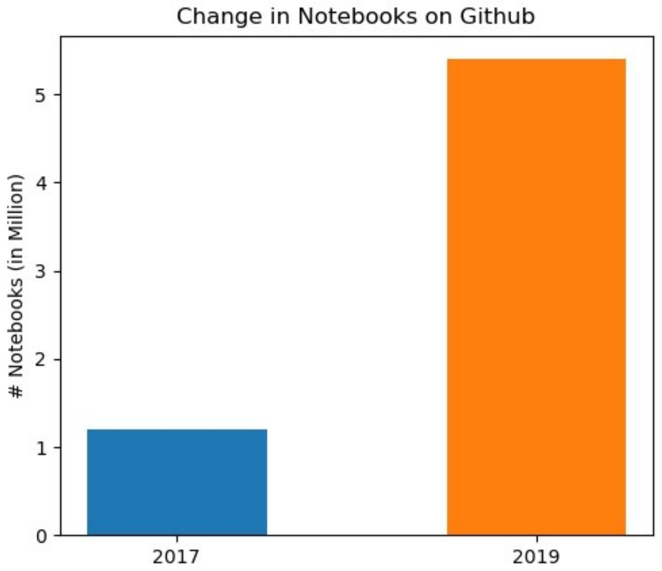
output cells

Can be interleaved

Popularity of Notebooks

- Survey on public public Jupyter notebooks on Github [3]
- Notebooks gain more popularity
- More people are using notebooks





Strong Points

- Advantages of notebooks, that are essential for a data scientist
  - Support for data exploration and visualization
  - Fast for prototyping
  - Easy-to-use also for non-programmers (besides hidden state)
  - Supplementary text cells help with collaboration

- -> Notebooks are suitable tool for data scientists to write and refine code in order to understand unfamiliar data, test hypotheses and build models to solve ill-defined problems
- However, their flexibility does come with a cost...

Example: Code with Explanation

```python
import matplotlib.pyplot as plt
%matplotlib inline
import keras
```

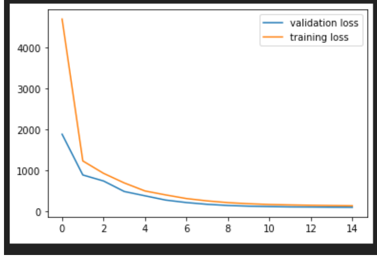
We perform a regression on the boston housing dataset. We want to predict the median house price.
It consists of 13 numeric and categorical features, like for example property tax, mean rooms per dwelling and a town's crime rate.
As a regressor we use a simple MLP with 2 hidden layers. As an optimizer we choose Adam (Kingma, Ba, 2014) and used the Mean Squared Error as our loss function.

```python
(x_train, y_train), (x_test, y_test) = keras.datasets.boston_housing.load_data(path='b_housing', test_split=0.1)

model = keras.Sequential([keras.layers.Dense(10, activation='relu')])
model.add(keras.layers.Dense(5, activation='relu'))
model.add(keras.layers.Dense(1))

model.compile(optimizer='adam', loss='mse')
model.fit(x=x_train, y=y_train, batch_size=32, epochs=15, verbose=0, validation_split=0.2)
hist = model.history.history
plt.plot(hist['val_loss'], label='validation loss')
plt.plot(hist['loss'], label='training loss')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x1c4140ade88>
```



- Initial Text cell describes dataset and it’s features
- Description of employed ML-model and architecture
- Reference theoretical paper on optimizer
- Inline plotting enables easy inspection of learning curve

Question

From those of you who have used computational notebooks, what didn‘t you like about them or while using them?

Pain Points

- Study on general hardships in notebooks:
  - Setup and Reliability
    - Loading data is tedious
    - Limited processing power inhibits scalability
  - Exploratory nature leads to messy code [Disorder, Deletion, Dispersal]
    - Cells are copied for different hyperparameters
    - Out-of-order execution can create hidden states
  - Data security
    - Access management lacks granularity

Example: Out of order Execution

[33]
```python
import numpy as np
```

[42]
```python
w = np.random.normal(scale=2., size=(5, 1)) #  See if double std changes result
# This is just for testing. Remove later.
```

[35]
```python
w = np.random.normal(size=(5, 1))
```

[43]
```python
x = np.random.uniform(size=(3, 5))
print(x @ w)
```

```
[[ 0.19564825]
 [-0.18246406]
 [ 1.36893631]]
```

- Second block has been executed for a quick check
- Kernel still holds in w the value with std = 2

Difficult Tasks

- Survey on critical activities in notebooks:
  - Deploy in production
    - Data science languages differ from production environment
    - DevOps usually not a data scientists expertise
  - Explore version history
    - Out of order cell execution may aggravate reproducibility
    - Long running tasks
    - Computation inhibits interactivity
  - Missing coding assistance
    - autocompletion, refactoring tools often deficient, live templates

Why not use IDEs instead of Notebooks?

- Why not use well-established and modern IDEs (Integrated Development Environment) instead (e.g. Spyder, PyCharm)?
  - Auto-completion
  - Help with method parameters
  - Go to definition
  - Syntax highlighting
  - Code Refactoring possibilities
  - Version control system supports
- But main activity/goal is to develop generally useful and reusable products

  -> Not exactly what the goal of data scientists is

  -> So the way to go is to provide better support for notebooks, and not to replace them

Possible Solutions: Extensions

- To better work with notebooks extensions have been proposed that solve certain problems
- Nbgather [11]:
  - Logs every cell execution to enable:
    - Version history for every cell
    - Code gathering: for a chosen output, find minimal cells needed to produce it

Extensions II

- Commuter:
  - Provides notebook storage and access control
- Papermill:
  - Parameterizes notebooks to allow running different versions of the notebook
  - Saves the results to an output notebook, with the specific parameters used
- Further nteract Libraries:
  - Scrapbook: Save results of notebook drafts
  - Bookstore: Enables versioning and storage

Conclusion & Outlook

- Computational **Notebooks**
  - dual heritage in software and **science**
  - Trade-off/need for balance between exploration and software engineering
- Notebooks are a popular and inherent tool in Data Science
- Vital part in development of Machine Learning Applications
- Shortcomings of notebooks make the effective use challenging
- People in Data Science need to employ the right workflows and extensions to use notebooks as powerful tools for developing machine learning products

- In a relatively early stage and can be further leveraged and improved

References

[1] https://blog.jupyter.org/project-jupyter-computational-narratives-as-the-engine-of-collaborative-data-science-2b5fb94c3c58 (Retrieved 06.2020)

[2] http://www.literateprogramming.com/knuthweb.pdf

[3] Psallidas et al. Data Science Through The Looking Glass And What We Found There [https://arxiv.org/pdf/1912.09536.pdf]

[4] Chattopadhyay et al. What's Wrong With Computational Notebooks? Pain Points, Needs and Design Opportunities [https://web.eecs.utk.edu/~azh/pubs/Chattopadhyay2020CHI_NotebookPainpoints.pdf]

[5] https://yihui.org/en/2018/09/notebook-war/

[6] https://www.neilernst.net/matrix-blog.html

[7] https://ljvmiranda921.github.io/notebook/2020/03/16/jupyter-notebooks-in-2020-part-2/

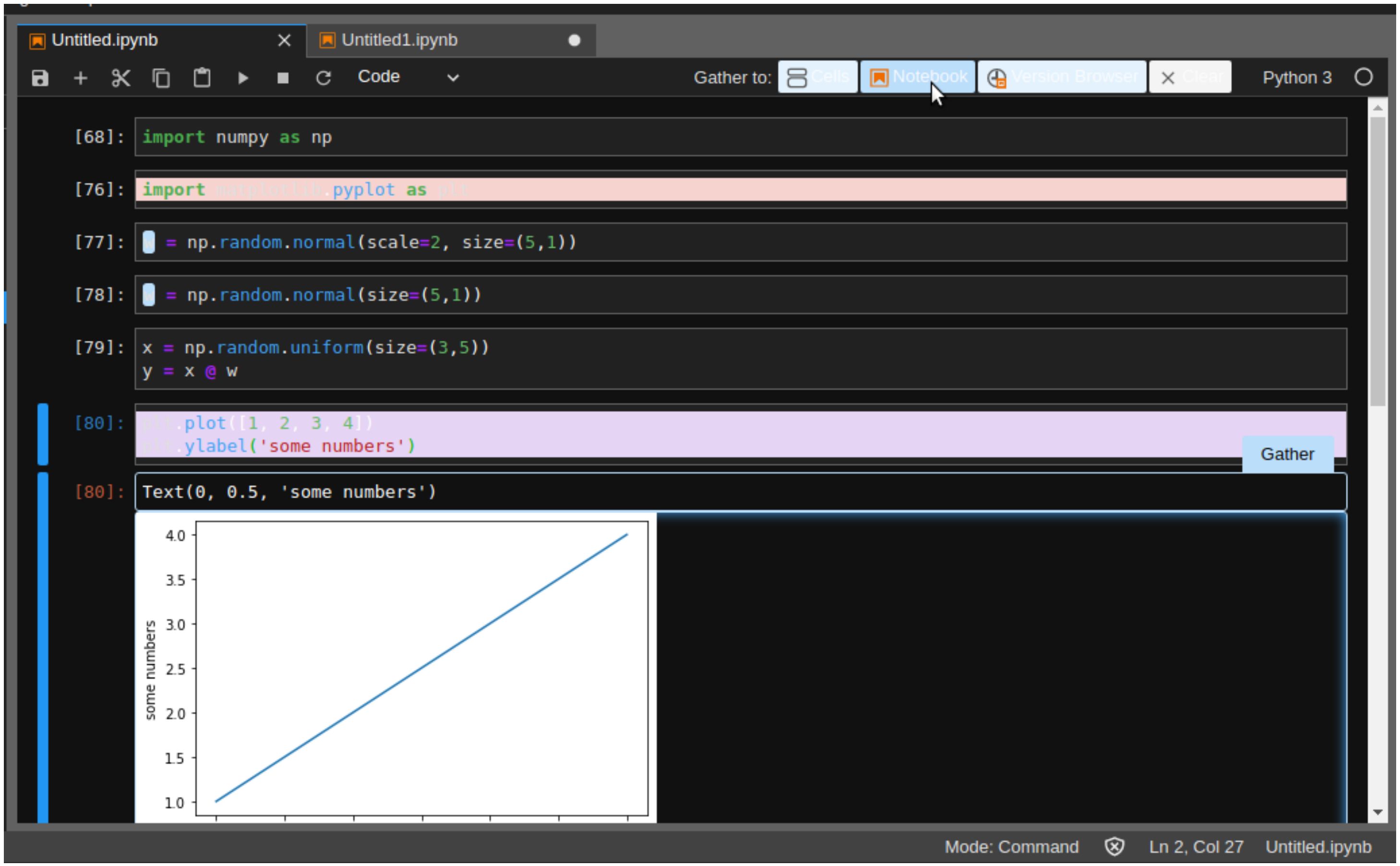
[8] https://jupyter4edu.github.io/jupyter-edu-book/jupyter.html

[9] https://netflixtechblog.com/notebook-innovation-591ee3221233 Notebook infrastructure

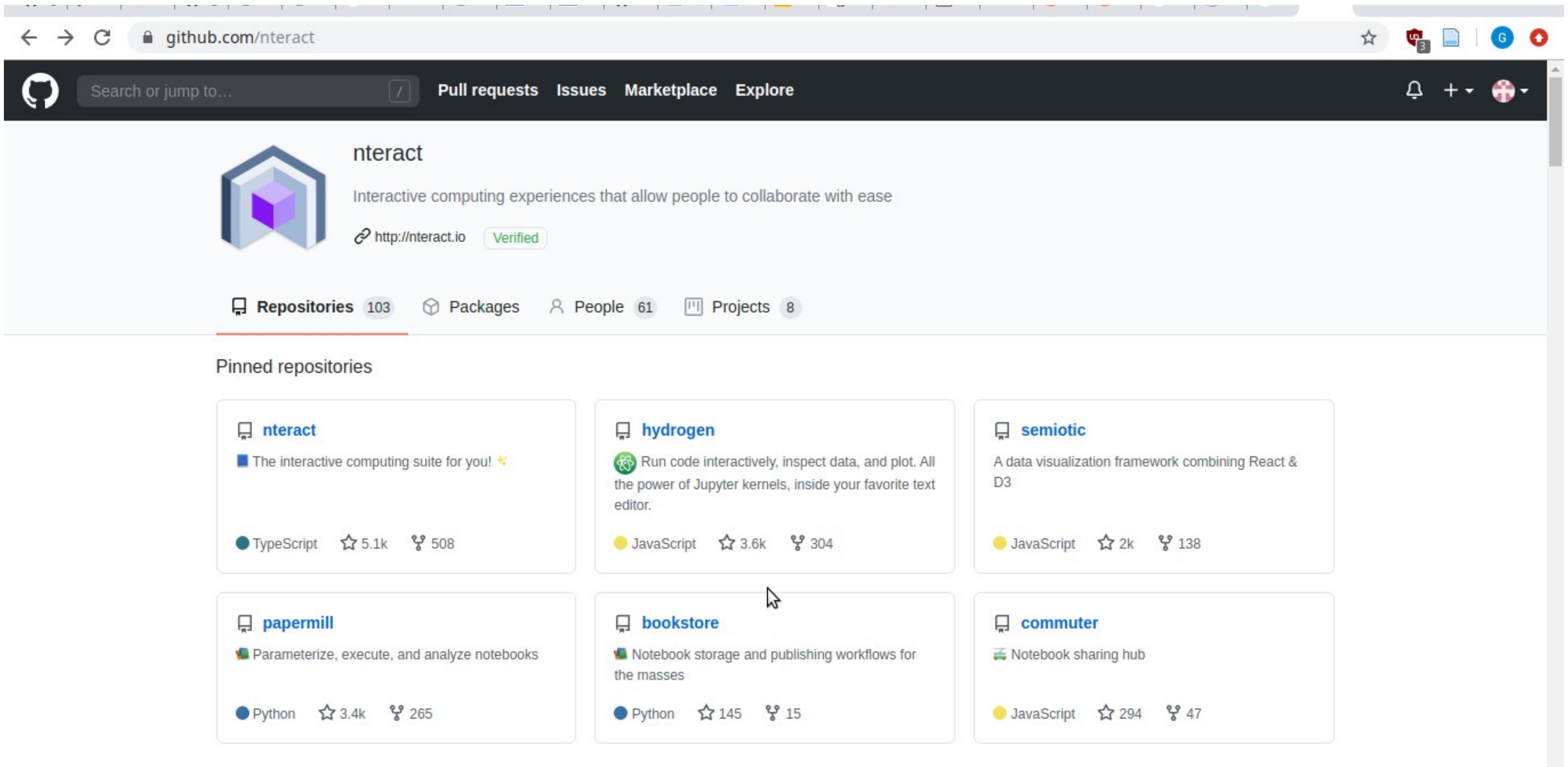
[10] https://dl.acm.org/doi/pdf/10.1145/3173574.3173606

[11] Head et al. Managing Messes in Computational Notebooks [https://dl.acm.org/doi/pdf/10.1145/3290605.3300500]

Tools: nbgather



Other Tools: From nteract



https://github.com/nteract

Acknowledgments & License

- Material Design Icons, by Google under Apache-2.0
- Other images are either by the authors of these slides, attributed where they are used, or licensed under Pixabay or Pexels
-

Extras

https://github.com/jupyter/design/wiki/Jupyter-Logo#where-does-the-jupyter-name-come-from

Jupyter naming reasons:

- Planet jupiter = science
- Core supported languages Julia, Python, R
- Galileo was the first to discover the moons of jupiter. He included the underlying data in the publication. -> leads to reproducibility in science, which is one of the focuses of Jupyter project