CS 8803-MDS
Human-in-the-loop Data Analytics

Lecture 14

10/10/22

Logistics

Part I survey:

https://forms.gle/B8zFUZN148YADBGh7

Vote on your favorite/least favorite papers!

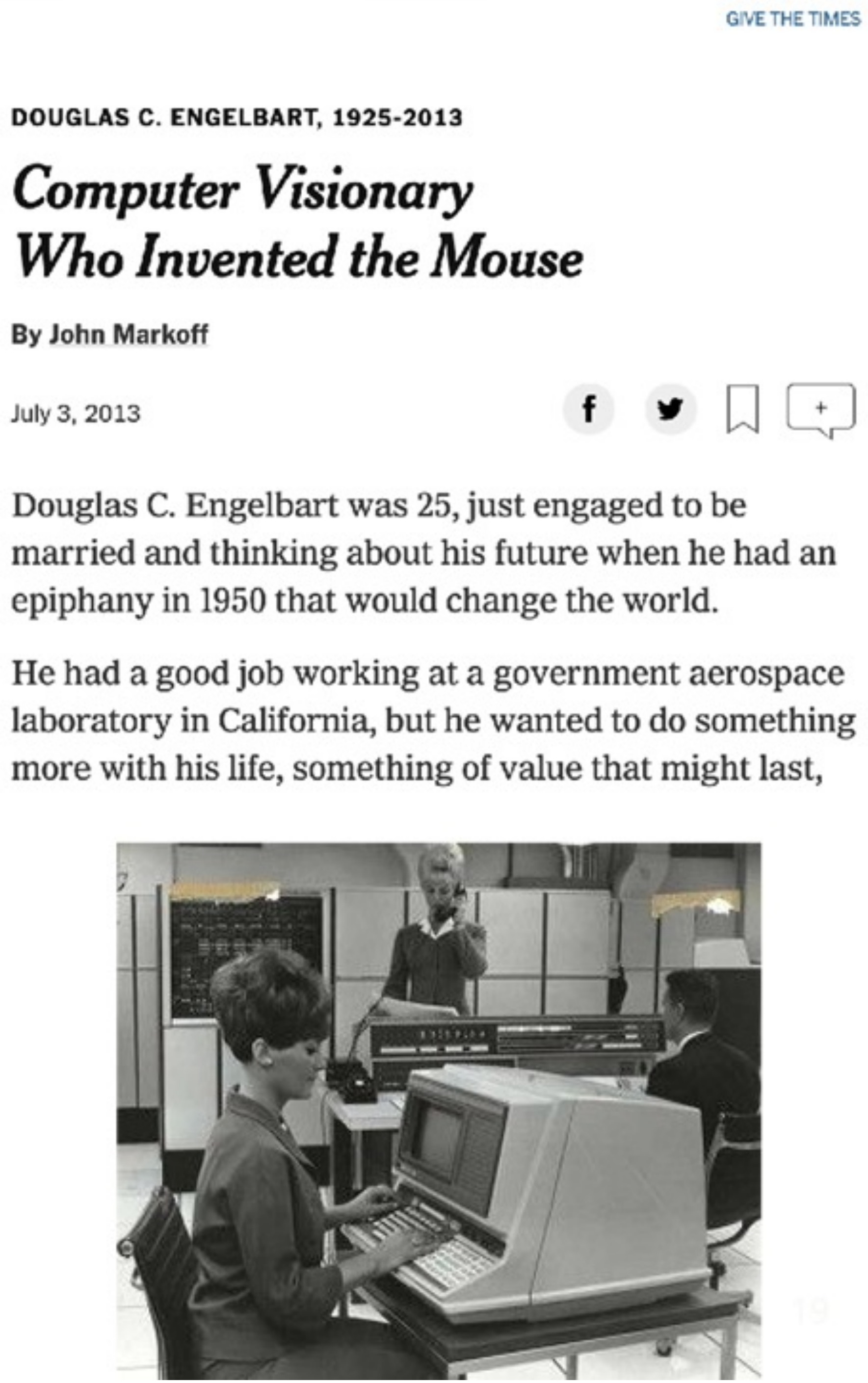
Project proposal grade to be released

Interactive computing

Examples:

- word processors
- spreadsheets
- Jupyter notebook

“An environment in which users execute code, see what happens, modify and repeat in a kind of iterative conversation between researcher and data.”



DOUGLAS C. ENGELBART, 1925-2013

Computer Visionary Who Invented the Mouse

By John Markoff

July 3, 2013

Douglas C. Engelbart was 25, just engaged to be married and thinking about his future when he had an epiphany in 1950 that would change the world.

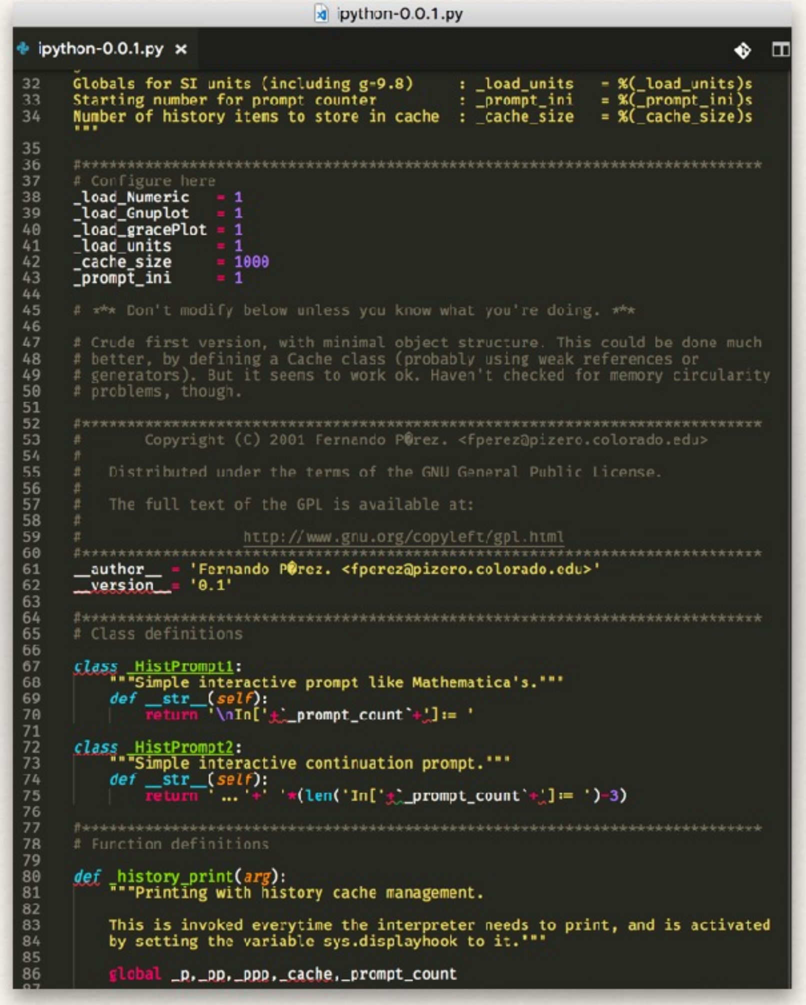
He had a good job working at a government aerospace laboratory in California, but he wanted to do something more with his life, something of value that might last, https://bids.berkeley.edu/events/project-jupyter-architecture-and-evolution-open-platform-modern-data-science

IPython: Interactive Python, 2001

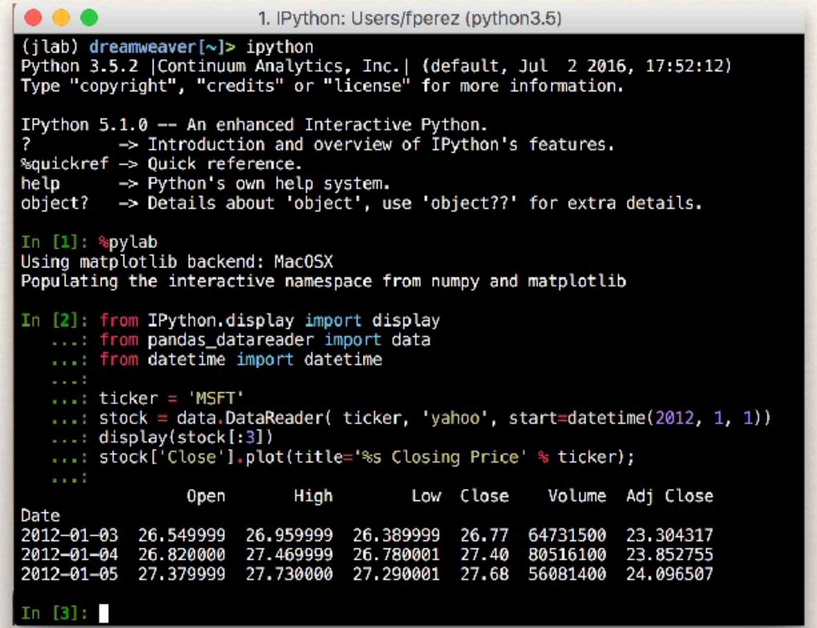
A humble start:

IPython 0.0.1, 259 LOC

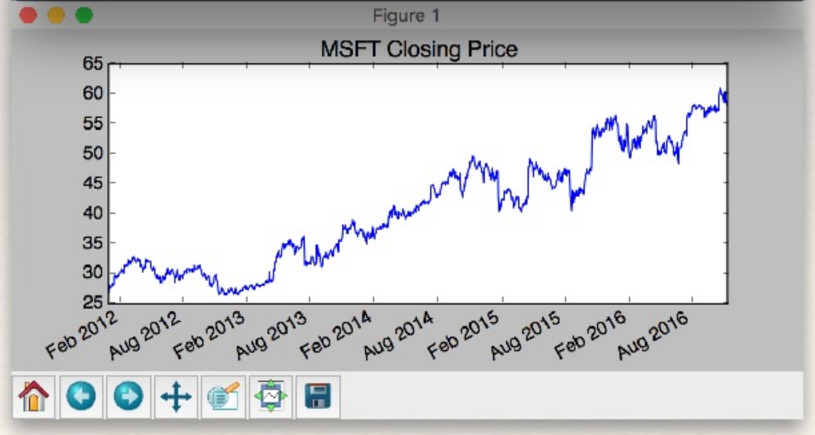
“Just an afternoon hack”



```
Globals for SI units (including g-9.8)      : _load_units   = %(_load_units)s
Starting number for prompt counter          : _prompt_ini   = %(_prompt_ini)s
Number of history items to store in cache   : _cache_size   = %(_cache_size)s
"""

#*****************************************************************************
# Configure here
_load_Numeric   = 1
_load_Gnuplot   = 1
_load_gracePlot = 1
_load_units     = 1
_cache_size     = 1000
_prompt_ini     = 1

# *** Don't modify below unless you know what you're doing. ***

# Crude first version, with minimal object structure. This could be done much
# better, by defining a Cache class (probably using weak references or
# generators). But it seems to work ok. Haven't checked for memory circularity
# problems, though.

#*****************************************************************************
#       Copyright (C) 2001 Fernando P@rez. <fperez@pizero.colorado.edu>
#
#   Distributed under the terms of the GNU General Public License.
#
#   The full text of the GPL is available at:
#
#                   http://www.gnu.org/copyleft/gpl.html
#*****************************************************************************
__author__ = 'Fernando P@rez. <fperez@pizero.colorado.edu>'
__version__= '0.1'

#*****************************************************************************
# Class definitions

class _HistPrompt1:
    """Simple interactive prompt like Mathematica's."""
    def __str__(self):
        return '\nIn['+`_prompt_count`+']:= '

class _HistPrompt2:
    """Simple interactive continuation prompt."""
    def __str__(self):
        return ' ...'+' '*(len('In['+`_prompt_count`+']:= ')-3)

#*****************************************************************************
# Function definitions

def _history_print(arg):
    """Printing with history cache management.

    This is invoked everytime the interpreter needs to print, and is activated
    by setting the variable sys.displayhook to it."""

    global _p,_pp,_ppp,_cache,_prompt_count
```

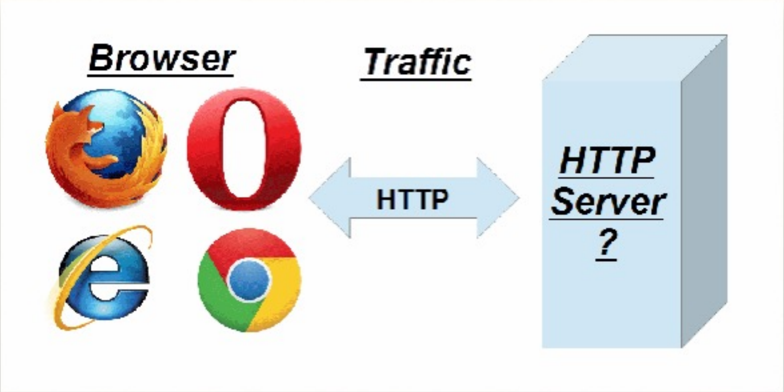


```
(jlab) dreamweaver[~]> ipython
Python 3.5.2 |Continuum Analytics, Inc.| (default, Jul  2 2016, 17:52:12)
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: %pylab
Using matplotlib backend: MacOSX
Populating the interactive namespace from numpy and matplotlib

In [2]: from IPython.display import display
   ...: from pandas_datareader import data
   ...: from datetime import datetime
   ...:
   ...: ticker = 'MSFT'
   ...: stock = data.DataReader( ticker, 'yahoo', start=datetime(2012, 1, 1))
   ...: display(stock[:3])
   ...: stock['Close'].plot(title='%s Closing Price' % ticker);
   ...:
                 Open       High        Low  Close    Volume  Adj Close
Date
2012-01-03  26.549999  26.959999  26.389999  26.77  64731500  23.304317
2012-01-04  26.820000  27.469999  26.780001  27.40  80516100  23.852755
2012-01-05  27.379999  27.730000  27.290001  27.68  56081400  24.096507

In [3]: 
```



https://gist.github.com/fperez/1579699

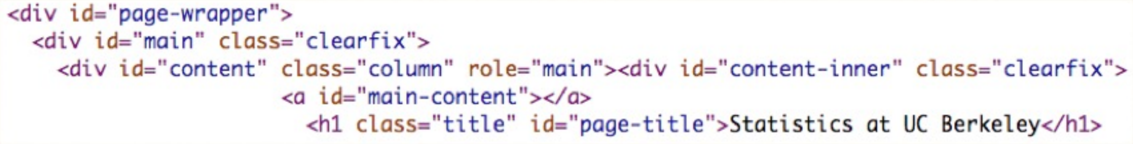
Core ideas of the web: HTTP & HTML



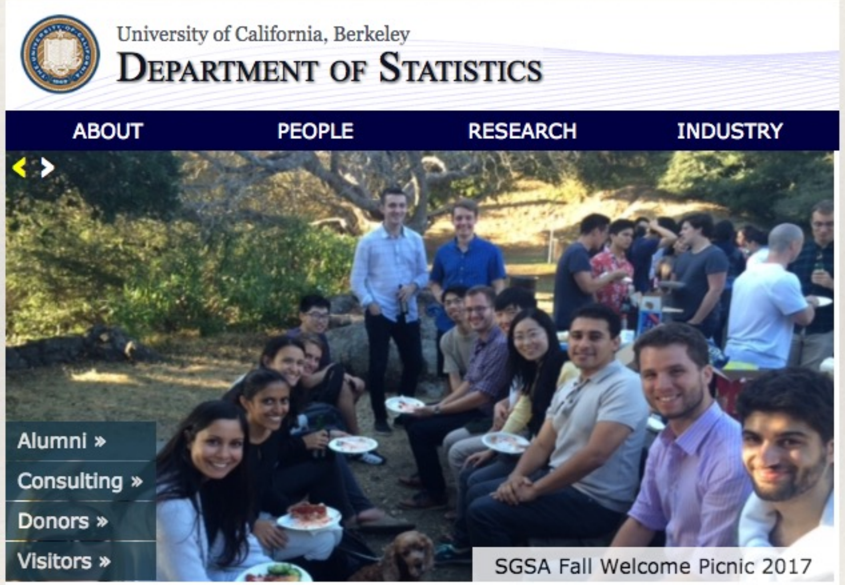
HTTP: protocol to connect clients and servers

HyperText Transport Protocol

Image credit: eviltester.com



```
<div id="page-wrapper">
  <div id="main" class="clearfix">
    <div id="content" class="column" role="main"><div id="content-inner" class="clearfix">
                  <a id="main-content"></a>
                    <h1 class="title" id="page-title">Statistics at UC Berkeley</h1>
```



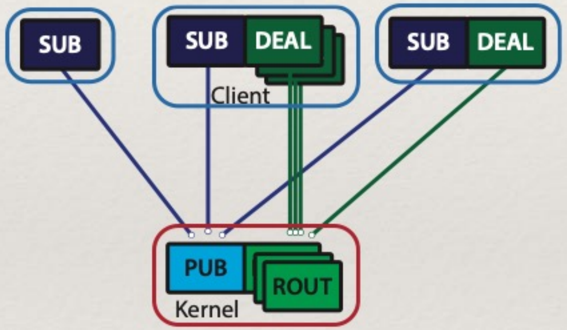
HTML: format to represent content

HyperText Markup Language

https://bids.berkeley.edu/events/project-jupyter-architecture-and-evolution-open-platform-modern-data-science

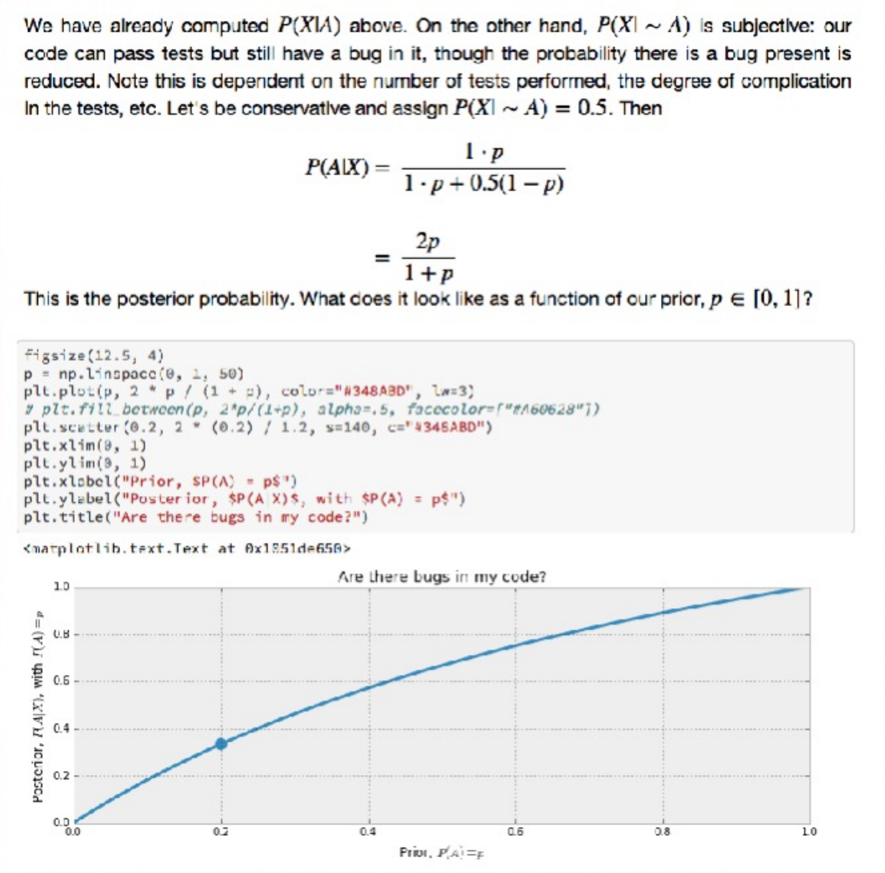
Core ideas of Jupyter

Interactive Computing Protocol



ØMQ + JSON

Document Format



We have already computed $P(X|A)$ above. On the other hand, $P(X|\sim A)$ is subjective: our code can pass tests but still have a bug in it, though the probability there is a bug present is reduced. Note this is dependent on the number of tests performed, the degree of complication in the tests, etc. Let's be conservative and assign $P(X|\sim A) = 0.5$. Then

$$P(A|X) = \frac{1 \cdot p}{1 \cdot p + 0.5(1-p)}$$

$$= \frac{2p}{1+p}$$

This is the posterior probability. What does it look like as a function of our prior, $p \in [0, 1]$?

```
figsize(12.5, 4)
p = np.linspace(0, 1, 50)
plt.plot(p, 2 * p / (1 + p), color="#348ABD", lw=3)
# plt.fill_between(p, 2*p/(1+p), alpha=.5, facecolor=["#A60628"])
plt.scatter(0.2, 2 * (0.2) / 1.2, s=140, c="#348ABD")
plt.xlim(0, 1)
plt.ylim(0, 1)
plt.xlabel("Prior, $P(A) = p$")
plt.ylabel("Posterior, $P(A|X)$, with $P(A) = p$")
plt.title("Are there bugs in my code?")
```

```
<matplotlib.text.Text at 0x1851de650>
```

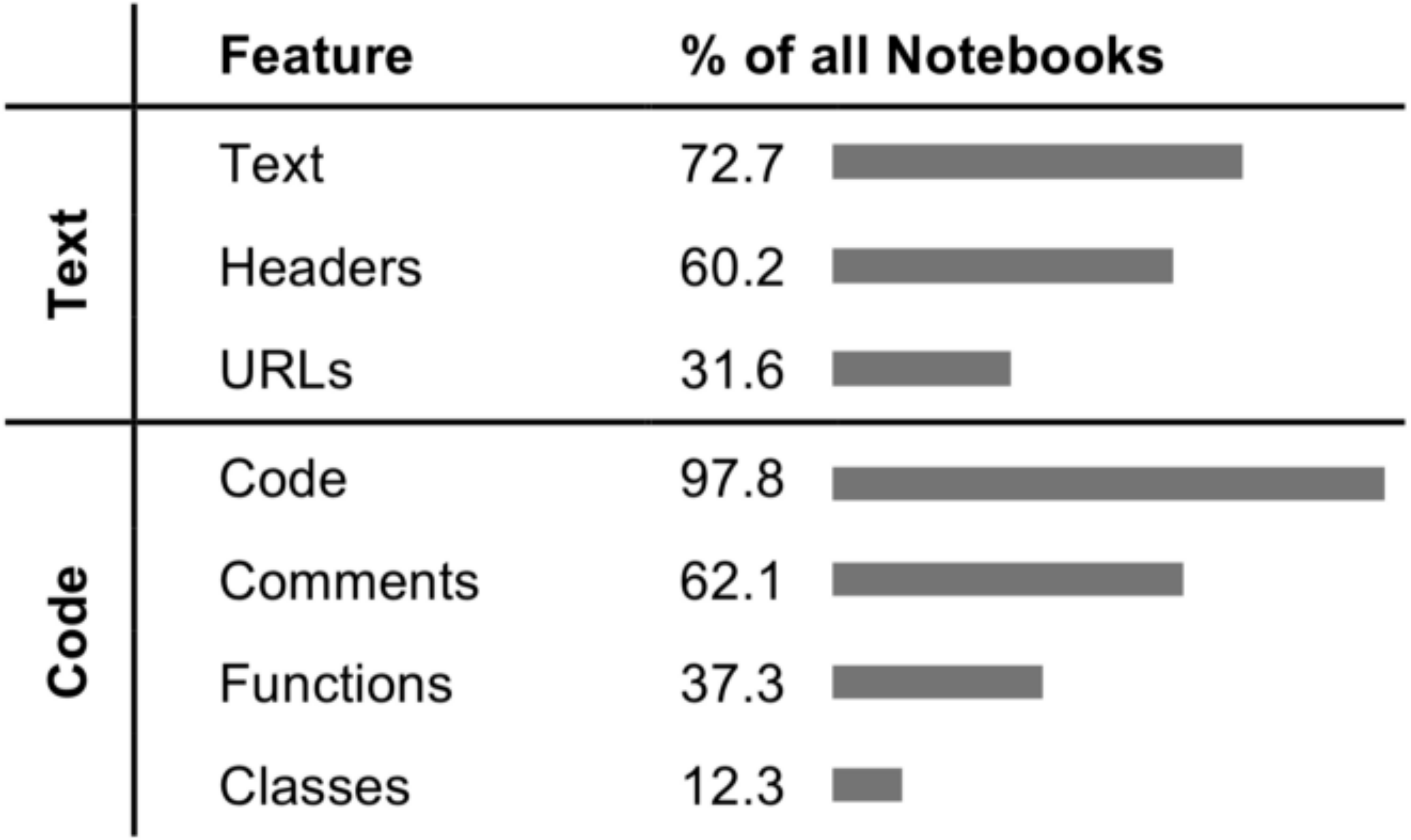
https://github.com/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers

Jupyter Protocol is language agnostic

Scala
perl
R
jupyter
julia
ERLANG
Spark
python™
IP[y]:
IPython

~100 different kernels: https://github.com/jupyter/jupyter/wiki/Jupyter-kernels

How people use notebooks



	Feature	% of all Notebooks
Text	Text	72.7
	Headers	60.2
	URLs	31.6
Code	Code	97.8
	Comments	62.1
	Functions	37.3
	Classes	12.3

Adam Rule, Aurélien Tabard, and James D. Hollan. 2018. Exploration and Explanation in Computational Notebooks. CHI ‘18.

Reproducibility in notebooks

A 2019 study [1] found that just 24% of 863,878 publicly available Jupyter notebooks on GitHub could be successfully re-executed, and only 4% produced the same results.

“Notebooks are messy. You write stuff, you keep old crusty code behind, and it’s hard to kind of figure out which cells to execute in which order, because you were trying different things.”

[1] J. F. Pimentel et al. A Large-Scale Study About Quality and Reproducibility of Jupyter Notebooks

Today’s class

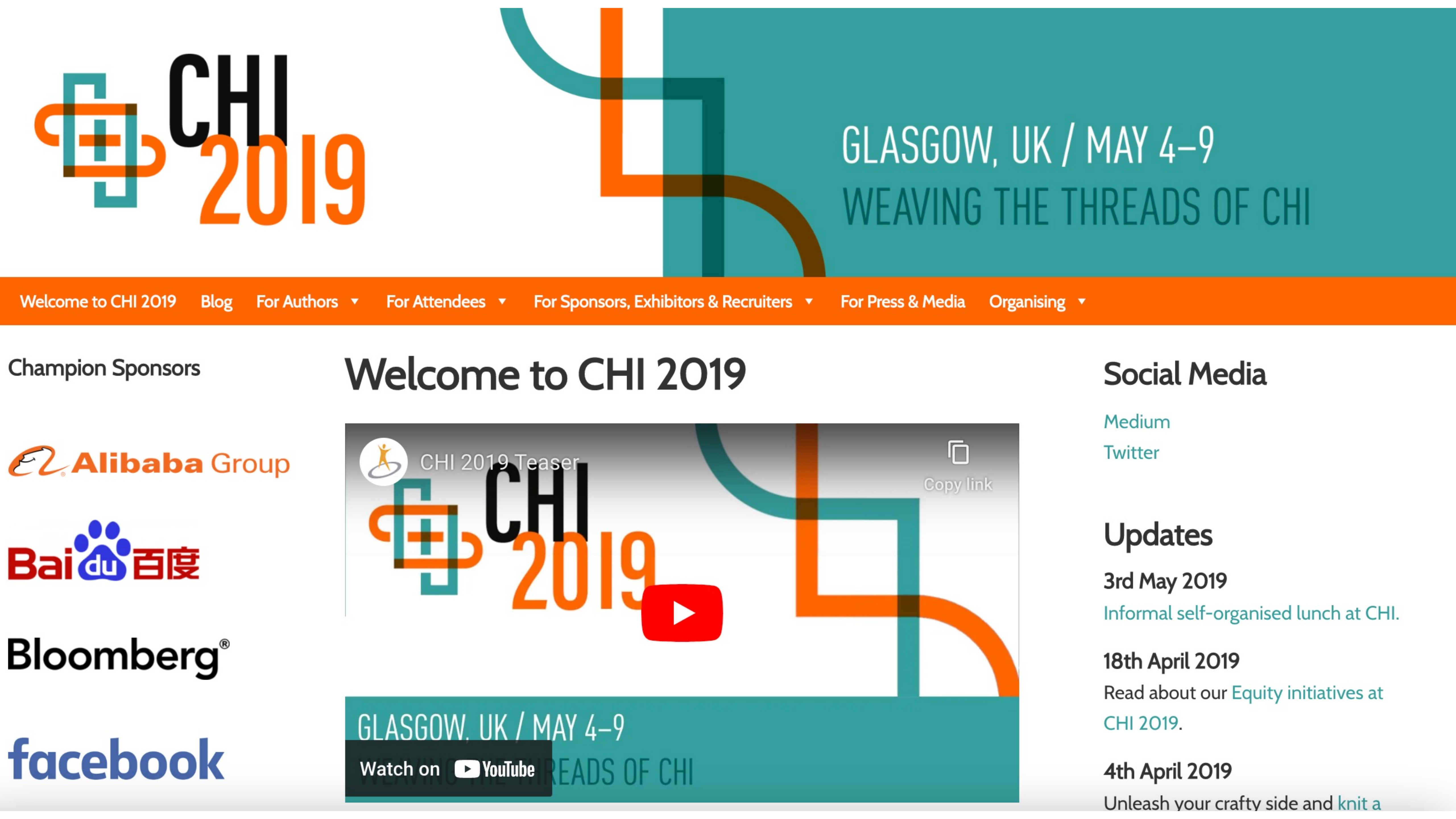
Towards Effective Foraging by Data Scientists to Find Past Analysis Choices

Author: Myna

Reviewer: Tanya, Siddhi

Archaeologist: Sahil

Practioner: Cangdi



Champion Sponsors

Welcome to CHI 2019

Social Media

Medium

Twitter

Updates

3rd May 2019

Informal self-organised lunch at CHI.

18th April 2019

Read about our Equity initiatives at CHI 2019.

4th April 2019

Unleash your crafty side and knit a

Towards Effective Foraging by Data Scientists to Find Past Analysis Choices

Mary Beth Kery, Bonnie E. John, Patrick O’Flaherty, Amber Horvath, Brad A. Myers

Presented by Myna Prasanna Kalluraya

Georgia Tech

Why is access to history important?

- Working with data is ubiquitous in all fields of science and industry.

- Numerous iterations behind every data-driven decision.

- It is necessary to understand the reasoning behind every decision made during the process.
  - “Why did I discard this data feature from my model?”

Verdant

- Tool to aid data scientists in examining the history of their work. The focus of this system is to work with computational notebooks.
- Verdant is a JupyterLab extension that automatically records history of all experiments that are run in a Jupyter notebook.
- We investigate support for the specific challenges that data scientists face around question-answering from history.

Background

- Foraging in source code:
  - IFT is based on the analogy of an animal deciding what to eat, where it can be found and the best way to obtain it.
  - Predators, prey, patches.
  - Design uses IFT to provide foraging cues like date, version previews, and diff highlighting.

- Version Control:
  - Git
  - Google’s Colaboratory project.
  - We focus on foraging and finding.

Design Use Case

Three use cases identified:

- Data scientist using their code history to find intermediary results.
- Data analyst using history to justify the model to a colleague.
- Process transparency - for instance, a professor understanding the approach to the solution by a student.

Design For Versioning Artifacts

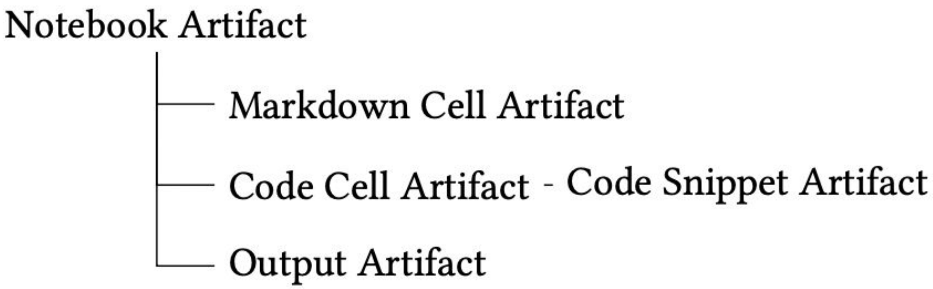
- Version model “lilGit” based on Git and older version of Verdant.

- In Git, developer can only access history in 2 levels of granularity:
  - list of commits of the entire project.
  - versions specific to when a particular file was changed.

- Git blame, grep and log have drawbacks.

lilGit

- A computational notebook is broken down into:



Notebook Artifact

Markdown Cell Artifact

Code Cell Artifact - Code Snippet Artifact

Output Artifact

Figure 1: Artifacts types in lilGit.

- This tree history structure is is saved in a single JSON .ipyhistory file, next to the .ipynb file.
- This makes it easily portable with or without the history file.

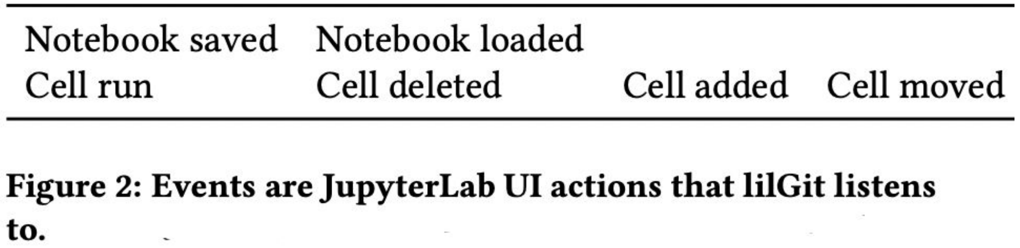
lilGit

- Versioning Procedure:

  Step 1. Notebook is loaded

  Step 2. User makes an edit

  Step 3. Notebook-level event



Notebook saved	Notebook loaded		
Cell run	Cell deleted	Cell added	Cell moved

Figure 2: Events are JupyterLab UI actions that lilGit listens to.

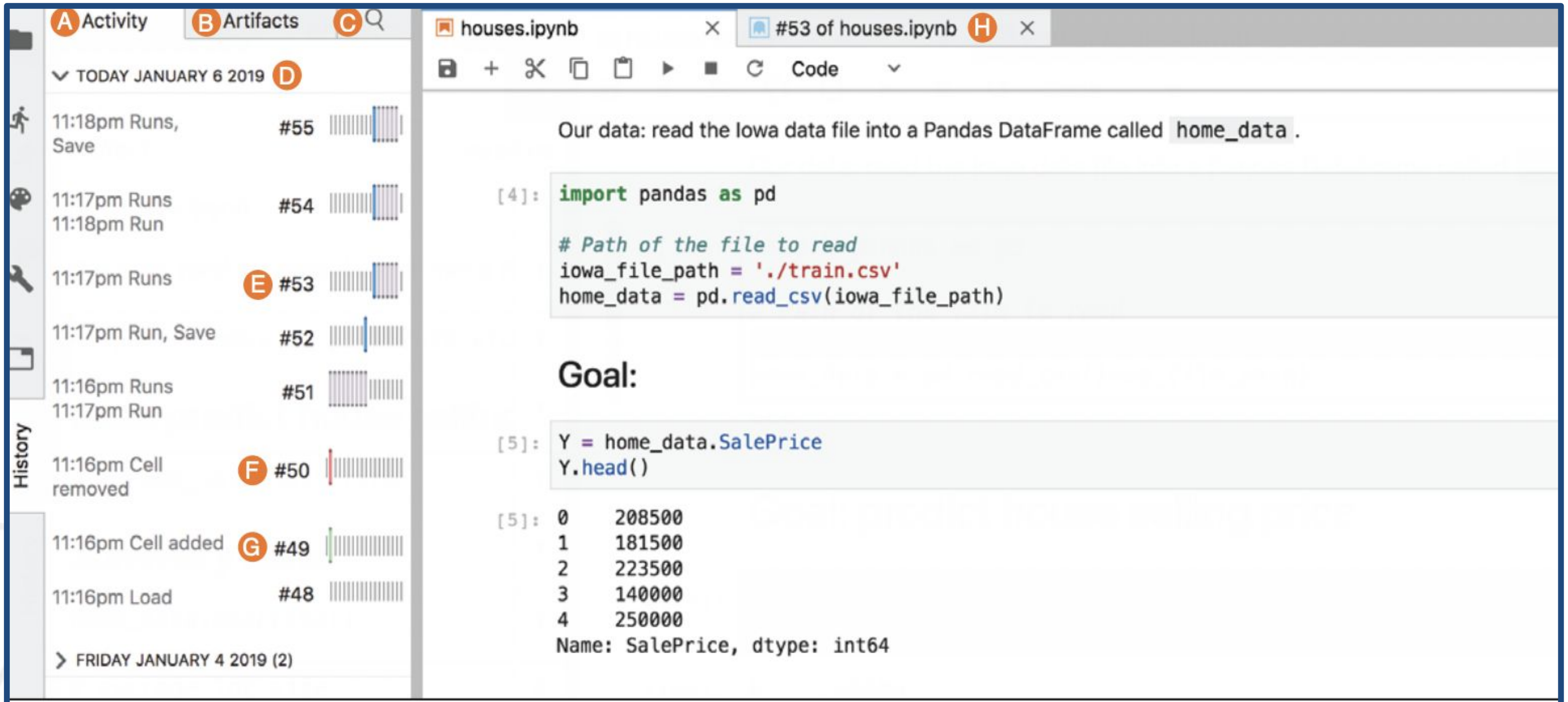
  Step 4. Resolve - Generate or Match

  Step 5. Commit

  Step 6. Save to file

Design for Improved Foraging

- Three tabs: Activity, Artifacts, Search
    - Activity Tab



11:18pm Runs, Save #55

11:17pm Runs 11:18pm Run #54

11:17pm Runs #53

11:17pm Run, Save #52

11:16pm Runs 11:17pm Run #51

11:16pm Cell removed #50

11:16pm Cell added #49

11:16pm Load #48

Our data: read the Iowa data file into a Pandas DataFrame called `home_data`.

```
[4]: import pandas as pd

     # Path of the file to read
     iowa_file_path = './train.csv'
     home_data = pd.read_csv(iowa_file_path)
```

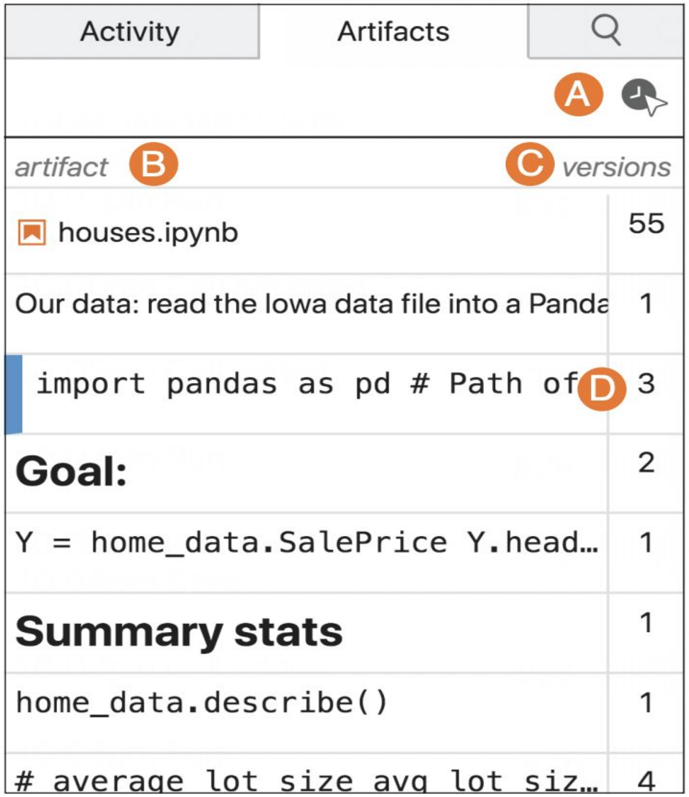
Goal:

```
[5]: Y = home_data.SalePrice
     Y.head()

[5]: 0    208500
     1    181500
     2    223500
     3    140000
     4    250000
     Name: SalePrice, dtype: int64
```

igure 3: The history tab opens the sidebar for Verdant containing three tabs.

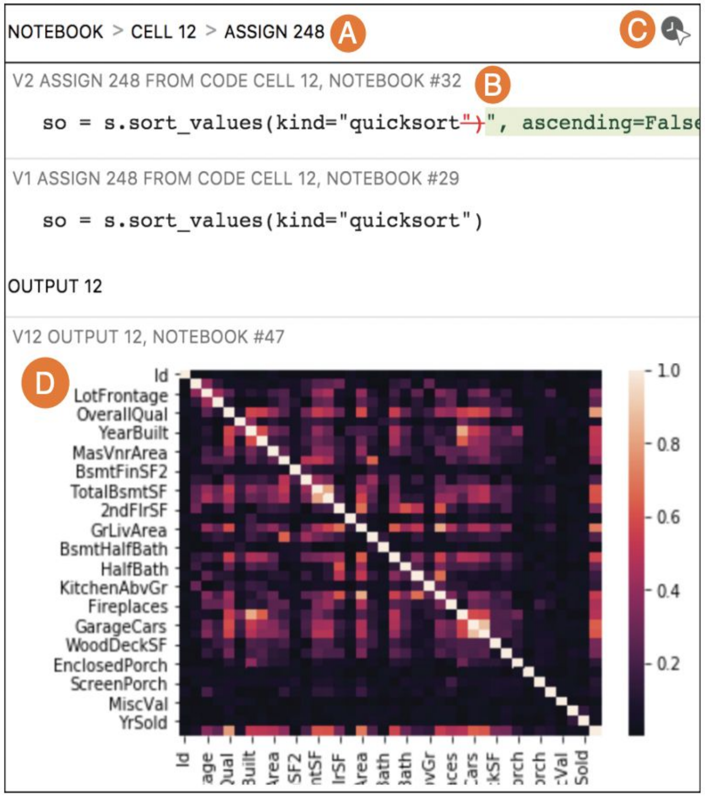
Design for Improved Foraging

- Artifacts Tab



artifact	versions
houses.ipynb	55
Our data: read the Iowa data file into a Panda	1
`import pandas as pd # Path of`	3
Goal:	2
`Y = home_data.SalePrice Y.head…`	1
Summary stats	1
`home_data.describe()`	1
`# average lot size avg lot siz…`	4

Figure 5: The Artifacts tab's table of contents view



NOTEBOOK > CELL 12 > ASSIGN 248

V2 ASSIGN 248 FROM CODE CELL 12, NOTEBOOK #32

so = s.sort_values(kind="quicksort~~)~~", ascending=False

V1 ASSIGN 248 FROM CODE CELL 12, NOTEBOOK #29

```
so = s.sort_values(kind="quicksort")
```

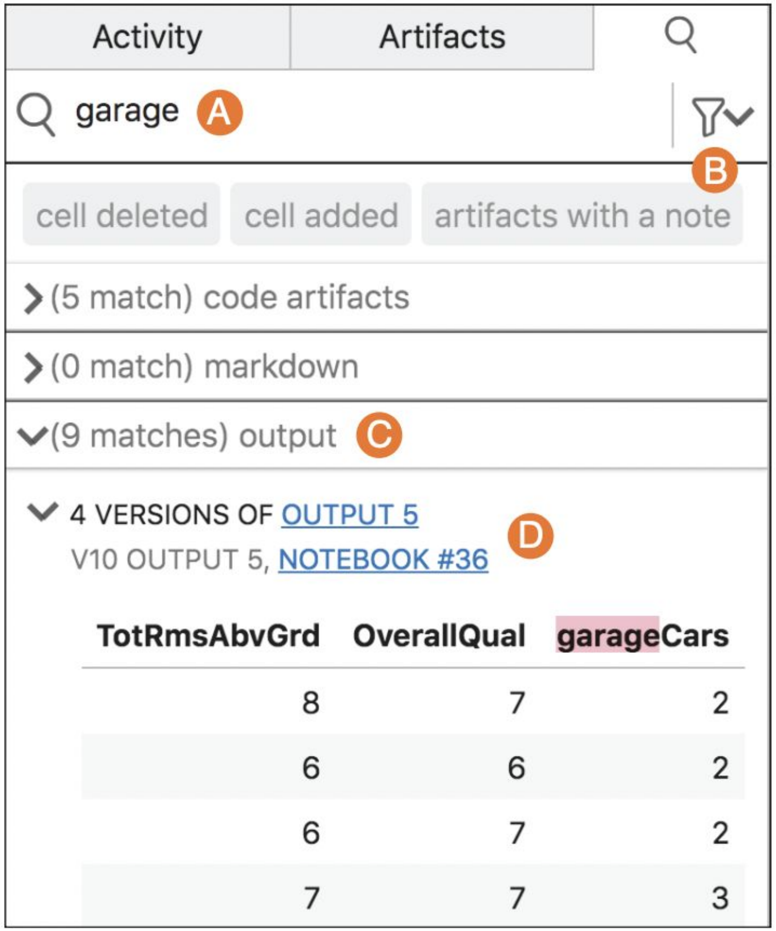
OUTPUT 12

V12 OUTPUT 12, NOTEBOOK #47

Figure 6: The Artifacts tab showing versions of an assign statement within a code cell (A).

Design for Improved Foraging

- Search Tab



(5 match) code artifacts

(0 match) markdown

(9 matches) output

4 VERSIONS OF OUTPUT 5

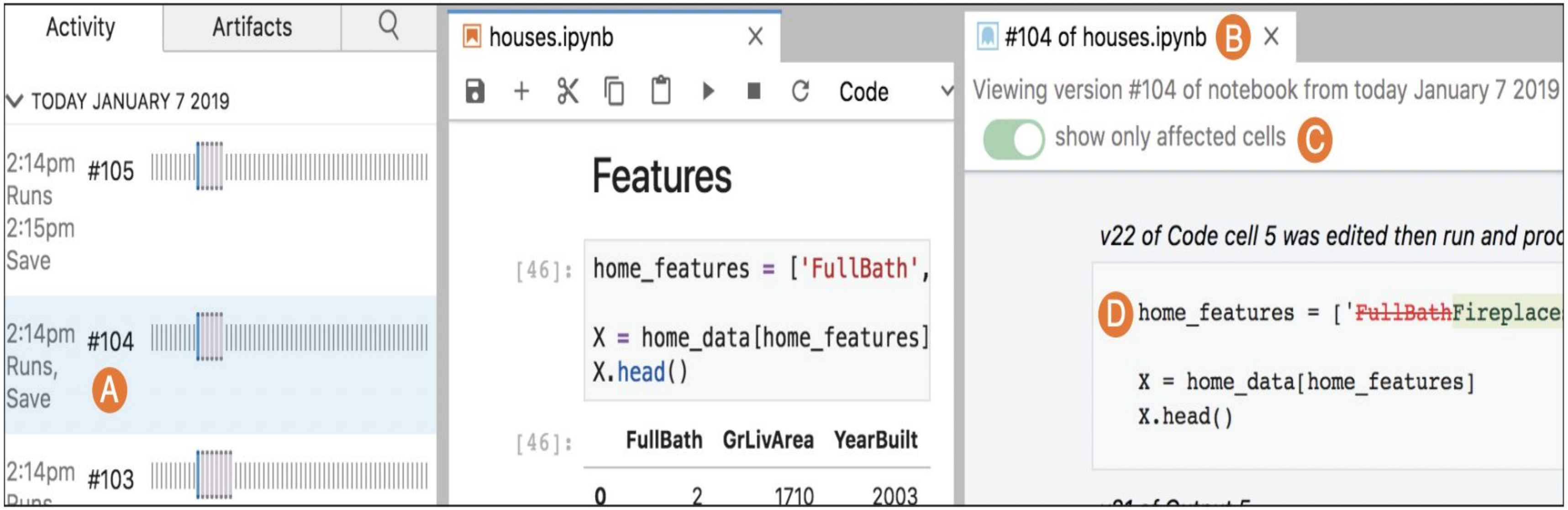
V10 OUTPUT 5, NOTEBOOK #36

TotRmsAbvGrd	OverallQual	garageCars
8	7	2
6	6	2
6	7	2
7	7	3

Figure 7: Searching for “garage” (A).

Design for Improved Foraging

- Ghost notebook



Features

```
[46]: home_features = ['FullBath',

      X = home_data[home_features]
      X.head()
```

[46]:	FullBath	GrLivArea	YearBuilt
0	2	1710	2003

Viewing version #104 of notebook from today January 7 2019

v22 of Code cell 5 was edited then run and prod

```
home_features = ['~~FullBath~~Fireplace

X = home_data[home_features]
X.head()
```

Evaluation of Verdant

- Primary goal: to gather data about how the features of Verdant assist or hinder data scientists.

- Study conducted at the JupyterCon2018 conference.

- Materials Evaluated:
    - Verdant JupyterLab Extension
    - Notebook
    - Tour
    - Tasks

Evaluation of Verdant

- 16 experienced data scientists completed 15 tasks across 4 task categories.

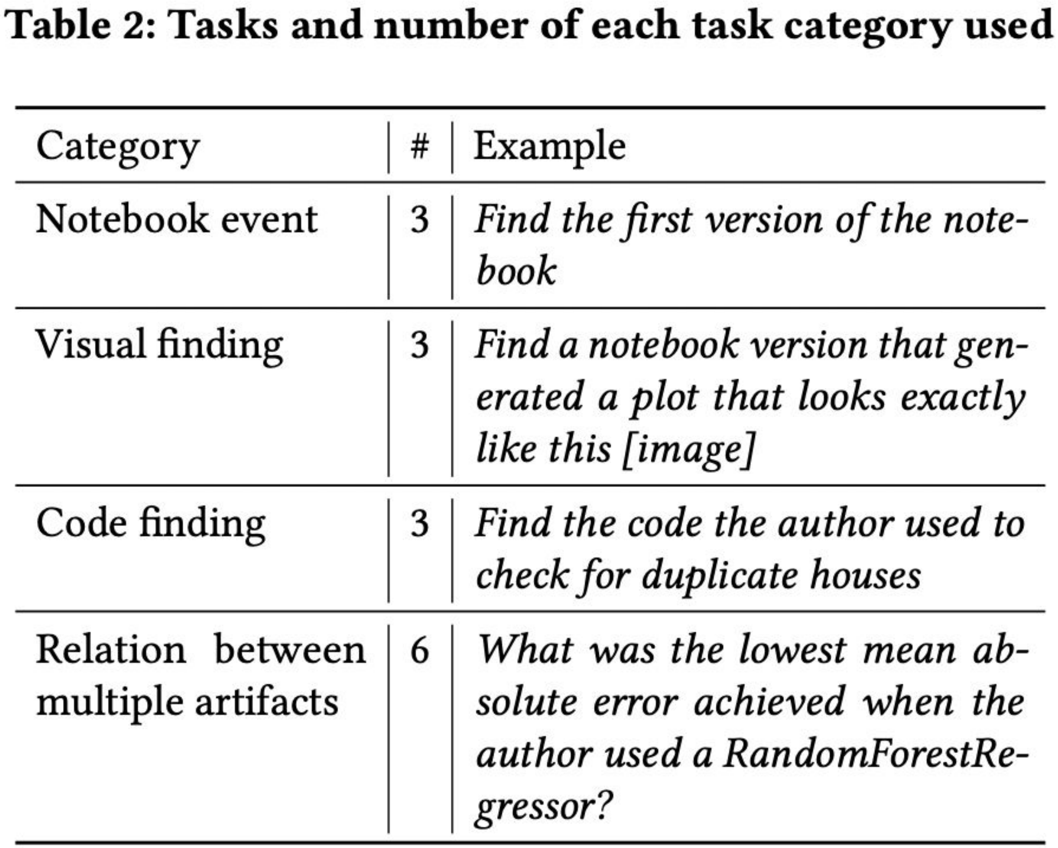


Table 2: Tasks and number of each task category used

Category	#	Example
Notebook event	3	*Find the first version of the notebook*
Visual finding	3	*Find a notebook version that generated a plot that looks exactly like this [image]*
Code finding	3	*Find the code the author used to check for duplicate houses*
Relation between multiple artifacts	6	*What was the lowest mean absolute error achieved when the author used a RandomForestRegressor?*

Quantitative Analysis

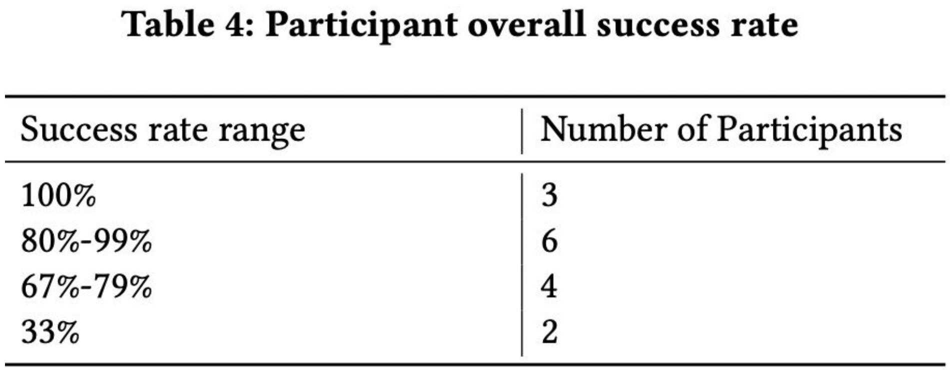


Table 4: Participant overall success rate

Success rate range	Number of Participants
100%	3
80%-99%	6
67%-79%	4
33%	2

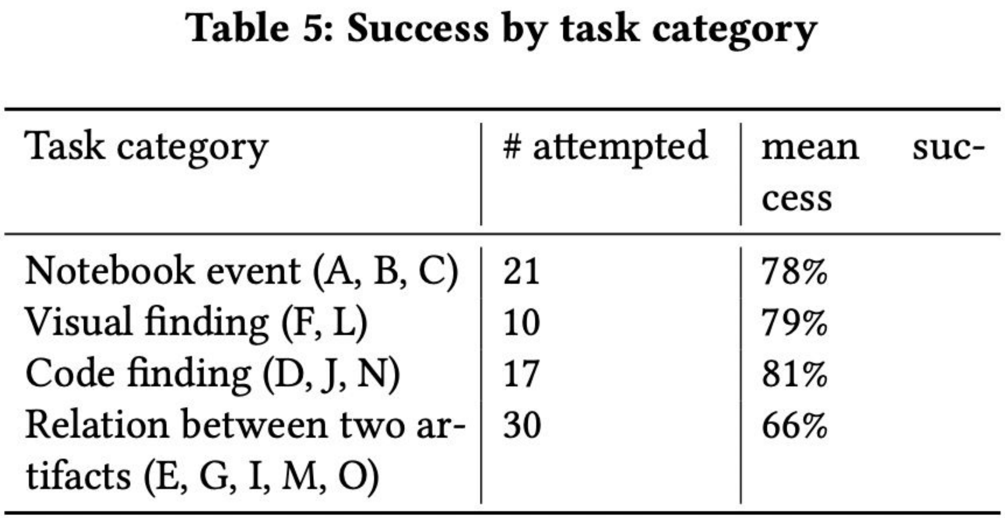


Table 5: Success by task category

Task category	# attempted	mean success
Notebook event (A, B, C)	21	78%
Visual finding (F, L)	10	79%
Code finding (D, J, N)	17	81%
Relation between two artifacts (E, G, I, M, O)	30	66%

Qualitative Analysis

- Usability analysis exploring the different features of the Verdant UI.

- Based on analysis, the paper focuses on fixing 3 areas:
  - Confusion about how to navigate within Verdant.
  - The need for excessive scrolling.
  - Participants resorting to brute-force looking through ghost books.

Future Work

- Future work includes:
    - Bug fixes.
    - Tweaks in UI elements.
    - More areas of redesign in the UI.
    - Smoothen the transition through search and filter strategies.

- Long term studies are to be conducted to fully evaluate the benefits of Verdant in a realistic scenario.

Thank you.

Archeologist Presentation

Excavated by: Sahil Ranadive

Summary

- The paper presents an effective tool to forage through the work done by a data scientist in the past
- It adds 3 key features to the existing tool (Verdant):
  - Activity tab
  - Artifacts tab
  - Search
- It also provides versioning through lilGit

Past Work? Theories Used!

- One of the core additions that the authors make to Verdant as described in this paper (in addition to better versioning via lilGit) are adding better visualizations/methods for “Foraging” through past work.
- Ideas for this are taken from Information Foraging Theory (book by Peter Pirolli) - rooted in biology!
- What is Information Foraging Theory?
  - “Explain and predict how people will best shape themselves for their information environment

    and more importantly

  - *How information environments can be best shaped for people*”

Information Foraging Theory

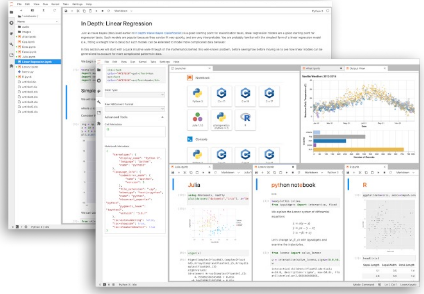
Hunts for

in

Predator

Prey

Environment



Putting IFT in context

- The theory explains and predicts how people navigate in response to the information in their environment (in this case, how a user navigates through the enormous code history generated while using Jupyter notebooks).
- IFT has proven particularly helpful in explaining what makes for an effective Web design (i.e., such that users can find their desired content easily and efficiently) and has a track record of successfully predicting how people will seek information on the Web. ex: the paper introduces "cues" such as dates, previews etc in each of the tabs to enable users to forage effectively.

And the paper does much more…

- In the activity tab:
    - Events are displayed in the Activity tab as a chronologically ordered stream so that the user can visually scan down to the rough date and time that constitutes “earlier today”
    - All events that share the same notebook version are chunked into the same row. If each event were to have its own row in the stream, the user would need to scroll a long way to get a notion of what had occurred within just a few minutes.
- In the artifacts tab:
    - Summarizes each cell artifact of the notebook using a single line, along with the number of versions it has had, for a quick way to see the cell histories, much like a table of contents
    - Mimic “style inspector” in web design to create a “history inspector”

Looking Ahead… What can be improved?

- Verdant enables “Search” to find code snippets that may have been deleted by going through artifacts.
- But, it does not list them all(more importantly doesn’t rank) since

  “*We explored showing all results from all artifact types sorted chronologically, but this led to a glut of information for the user to scroll through, and did not perform well in the evaluation. Thus , the Search results are now chunked by artifact type and by artifact ID (Fig. 7) to lower the amount of reading and scrolling required.*”

- Can the search itself be made stronger?

NBSearch

- NBSearch powers semantic code search in notebook collections and interactive visual exploration of search results.
    - [*What is semantic code search?*

      retrieving relevant code snippet given a natural language query*. Different from typical information retrieval tasks, code search requires to bridge the semantic gap between the programming language and natural language*]
- NBSearch utilizes the language model to process the query and retrieve search results (i.e., a collection of relevant cells from multiple notebooks) by searching in the semantic space of descriptors based on similarity and then retrieving associated code segments based on the code-descriptor pairs.
- The paper also presents a UI tool for interactive visualization, NBLines, which reveals both intra- and inter-notebook relationships among cells

Thank You

Towards Effective Foraging by Data Scientists to Find Past Analysis Choices

Industry Practitioner Review: Cangdi Li

Georgia Tech

Background

-- Jupyter is a free, open-source, interactive web tool known as a computational notebook that most data scientists use.

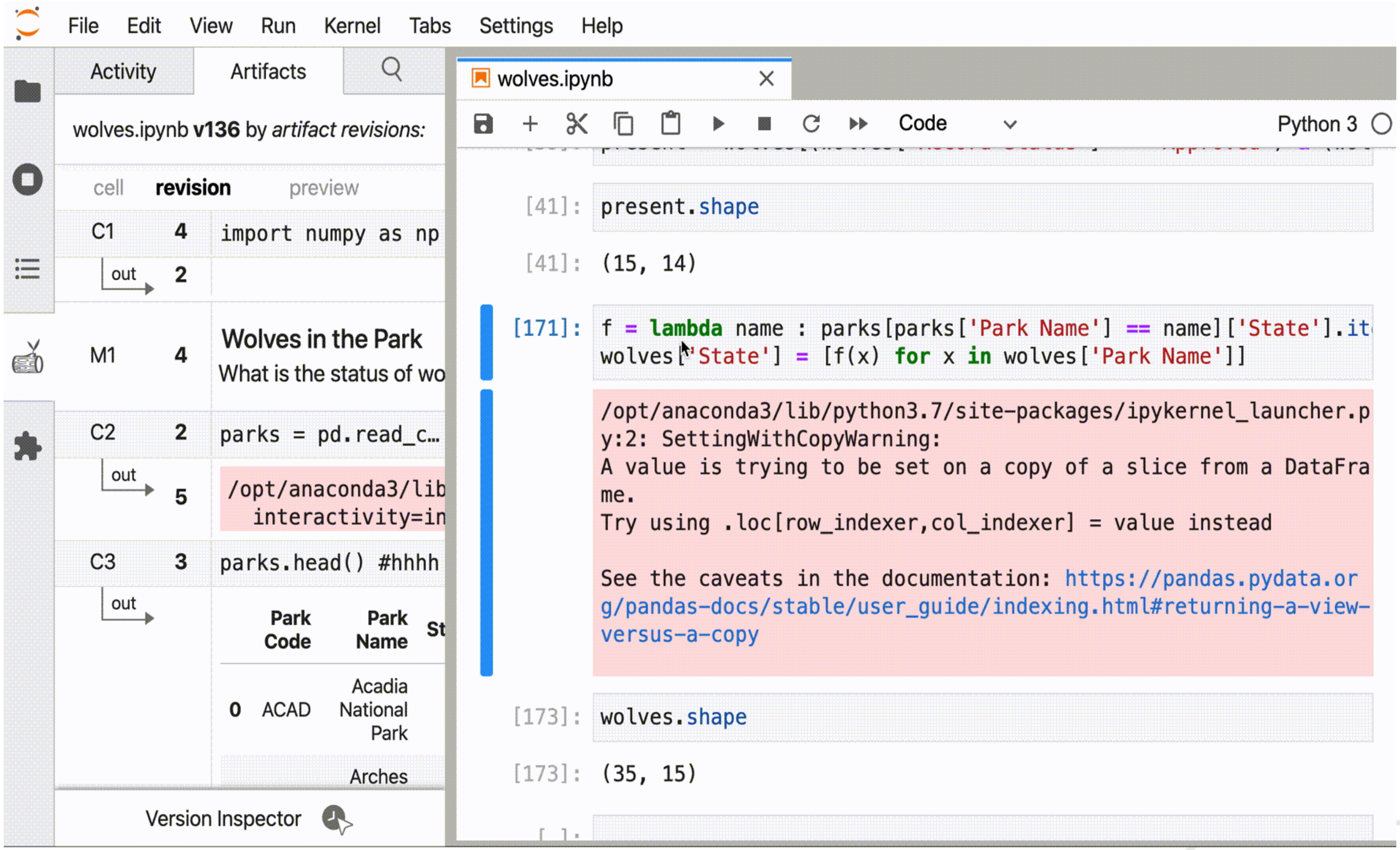
-- A “data scientist” can be anyone, from an engineer to a chemist to a financial analyst, to a student.

-- Jupyterlab provides a list of extensions and tools that help user with better jupyter notebook experience.

-- This paper provides a novel extension to the current version control tool---Verdant, and re-implements it to be an extension in Jupyterlab, it helps data scientist examine the history of their work more efficiently by automatically saving all important user events and results. Verdant also enables visualization and version comparison that provides better user experience.

Quick demo from author

Using version inspector of Verdant to check the history of a specific code block, including revision details, results, and so on.



wolves.ipynb **v136** by *artifact revisions:*

cell	revision	preview
C1	4	`import numpy as np`
out	2	
M1	4	**Wolves in the Park** What is the status of wo
C2	2	`parks = pd.read_c…`

out	5	`/opt/anaconda3/lib`
		`  interactivity=in`

C3	3	`parks.head() #hhhh`
out		

	Park Code	Park Name	St
0	ACAD	Acadia National Park	
		Arches	

```
[41]: present.shape

[41]: (15, 14)

[171]: f = lambda name : parks[parks['Park Name'] == name]['State'].it
       wolves['State'] = [f(x) for x in wolves['Park Name']]

       /opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.p
       y:2: SettingWithCopyWarning:
       A value is trying to be set on a copy of a slice from a DataFra
       me.
       Try using .loc[row_indexer,col_indexer] = value instead

       See the caveats in the documentation: https://pandas.pydata.or
       g/pandas-docs/stable/user_guide/indexing.html#returning-a-view-
       versus-a-copy

[173]: wolves.shape

[173]: (35, 15)

[ ]:
```

Practical use of Verdant

- Firstly, this paper is providing an extension of a current tool library in Jupyter, it’s definitely of practical value to apply.

- We will evaluate it in the following perspective:

- How effective is this tool? Is it easy to setup and use?
- Compare with similar tools
- Who should use it?

How effective is this tool?
Is it easy to setup and use?

- Verdant is supported in Jupyterlab, a well-known extension of Jupyter notebook, thus easy to install and setup.

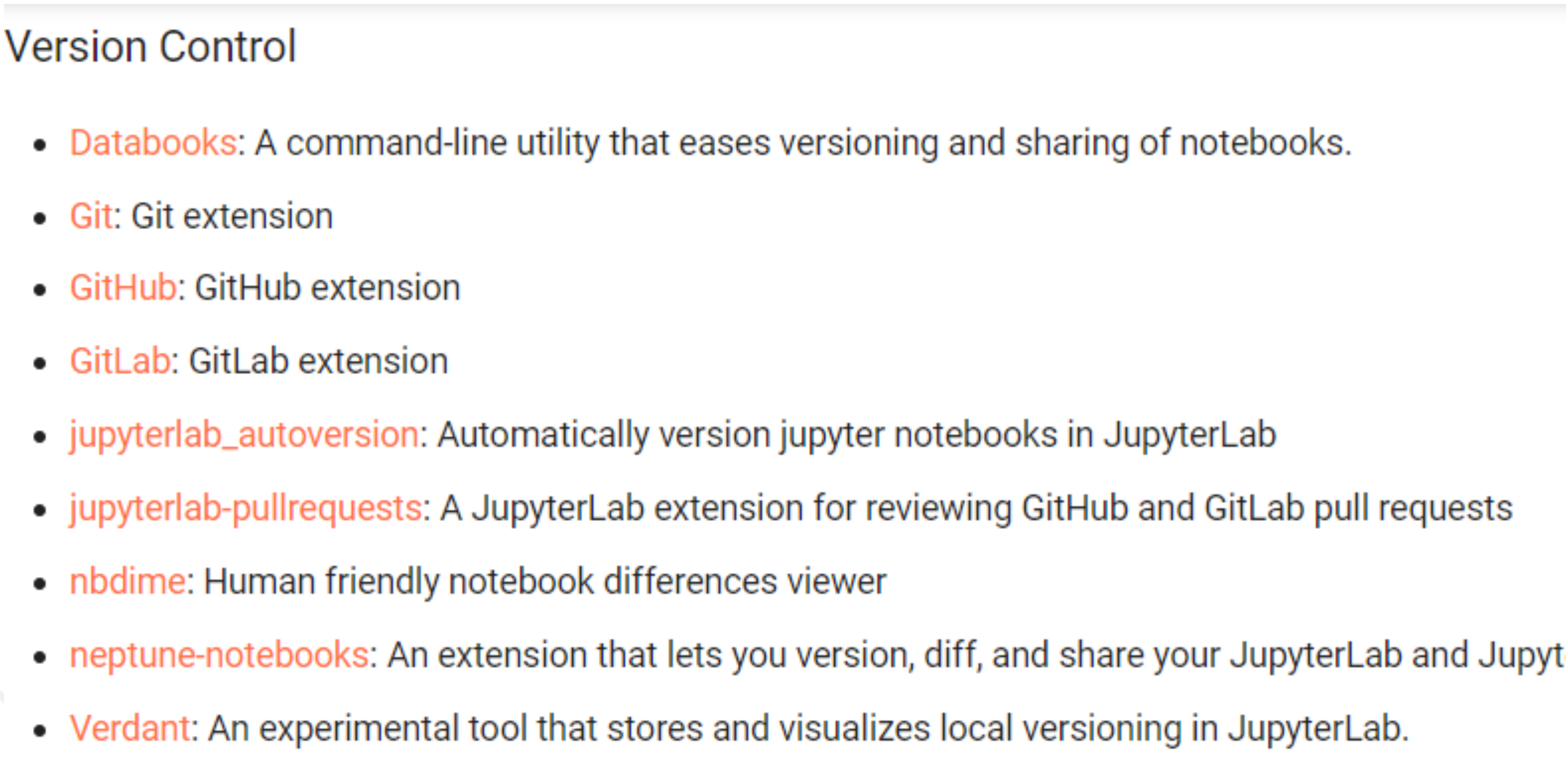
- Rather than a basic version auto-save functionality, the activity/artifacts-inspection option in the tool could significantly help user to focused more on approaching final result, with no fear of losing track of intermediate progress and result, because user can always perform search/backtrace to reasoning their path along the way of getting the solution.

Continue..

- However, there’s some drawbacks of Verdant:

- It is a bit complex to use and people needs tie to learn.
- There’s some bugs in the tool.
- Since the history information stored in Verdant could be huge in some extreme case, it is not sure if Jupyter can handle this much data and still be stable. User might want to drop some detail information in this case.

Compare with similar version tools in Jupyterlab



Version Control

- Databooks: A command-line utility that eases versioning and sharing of notebooks.
- Git: Git extension
- GitHub: GitHub extension
- GitLab: GitLab extension
- jupyterlab_autoversion: Automatically version jupyter notebooks in JupyterLab
- jupyterlab-pullrequests: A JupyterLab extension for reviewing GitHub and GitLab pull requests
- nbdime: Human friendly notebook differences viewer
- neptune-notebooks: An extension that lets you version, diff, and share your JupyterLab and Jupyt
- Verdant: An experimental tool that stores and visualizes local versioning in JupyterLab.

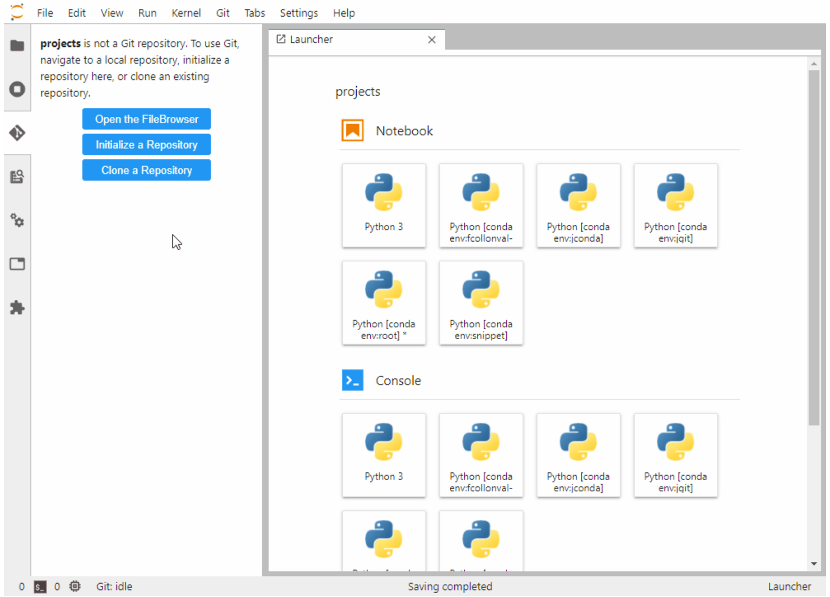
Compare with other version tools in Jupyterlab

-----Git related extensions:

Git extension, GitHub extension and GitLab:

Pros: It's just like git, easy to understand and use.

Cons: It does not auto-save all the history, only works like a manual commit PR tool, thus not friendly if you want to backtrace any intermediate results that you tried but didn't save.



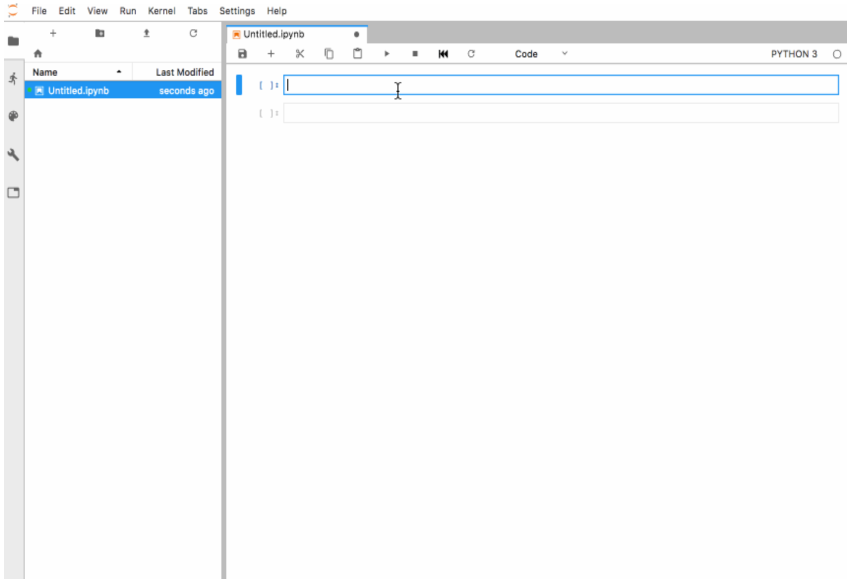
projects is not a Git repository. To use Git, navigate to a local repository, initialize a repository here, or clone an existing repository.

Saving completed

Compare with other version tools in Jupyterlab -----jupyterlab_autoversion

Pros: Enhanced checkpoints, versioned and persistent between restarts on every save.

Cons: It only provides a basic history auto-save function for each block, no search or visualization enabled, no results saved.

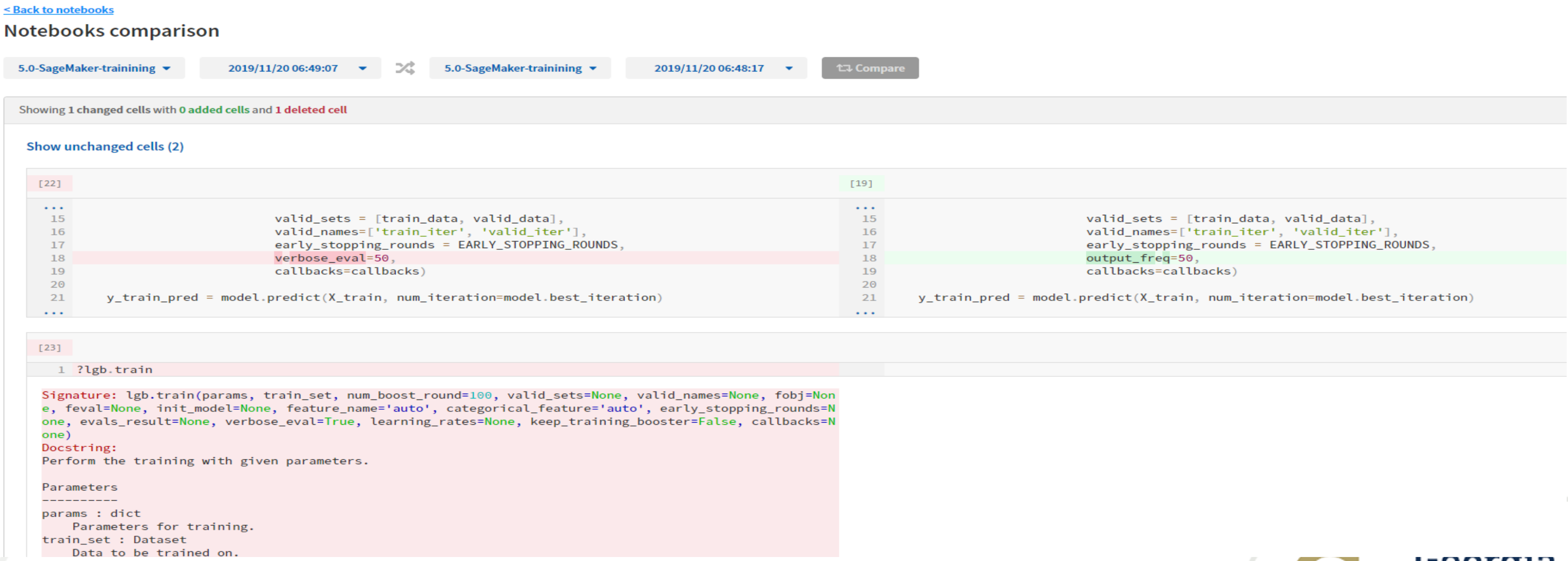


Georgia Tech

Compare with other version tools in Jupyterlab -----neptune-notebooks

Pros: Enables versioning of each notebook and comparison between different versions

Cons: It does not provdes a way to auto trace all actions and store the result, it does not provide searching options.



< Back to notebooks

Notebooks comparison

Showing 1 changed cells with 0 added cells and 1 deleted cell

Show unchanged cells (2)

[22]

```
...
                            valid_sets = [train_data, valid_data],
                            valid_names=['train_iter', 'valid_iter'],
                            early_stopping_rounds = EARLY_STOPPING_ROUNDS,
                            verbose_eval=50,
                            callbacks=callbacks)

    y_train_pred = model.predict(X_train, num_iteration=model.best_iteration)
...
```

[19]

```
...
                            valid_sets = [train_data, valid_data],
                            valid_names=['train_iter', 'valid_iter'],
                            early_stopping_rounds = EARLY_STOPPING_ROUNDS,
                            output_freq=50,
                            callbacks=callbacks)

    y_train_pred = model.predict(X_train, num_iteration=model.best_iteration)
...
```

[23]

```
?lgb.train
```

```
Signature: lgb.train(params, train_set, num_boost_round=100, valid_sets=None, valid_names=None, fobj=Non
e, feval=None, init_model=None, feature_name='auto', categorical_feature='auto', early_stopping_rounds=N
one, evals_result=None, verbose_eval=True, learning_rates=None, keep_training_booster=False, callbacks=N
one)
Docstring:
Perform the training with given parameters.

Parameters
----------
params : dict
    Parameters for training.
train_set : Dataset
    Data to be trained on.
```

Conclusion

We look at some similar version tools in the market, and it seems that Verdant is the only one of its kind, which provides a bunch of techniques such as versioning, foraging, and searching with visualization. Although it has some bugs, it does not affect common use of the tool, and it's still under continue developing.

Another concern is that Verdant is considered to be a complex tool and user need to spend time understanding it, if the user only wants to do basic versioning and comparison, Neptune is a great choice, speaking of auto-saving the checkpoints, jupyterlab_autoversion is also a simple tool to use.

That being said, we believe Verdant is more for professional data scientist or researchers who works on big projects and need a way to track and autosave all intermediate info for them to look back, reasoning and making decisions.

Towards Effective Foraging by Data Scientists to Find Past Analysis Choices

Peer Review

Reviewer: Tanya Garg

Summary

- ➔ A new release of the Verdant tool for effective data foraging by using algorithmic and visualization techniques for notebook code environments.

- ➔ Develops a new version control tool, lilGit, and addresses three artifacts namely: Markdown Cell Artifact, Code Cell Artifact and Output Artifact.

- ➔ Offers users three ways to look into their logs : Activity tab (time-based change log), Artifact tab (location based change log) and Search Tab (keyword based filtering)

Strong Points

- ➔ Add to Git by allowing data scientists answer important heuristic questions such as why was a feature discarded by providing exploratory features for non-code artifacts.
- ➔ Focuses on foraging and finding rather than just exploring through the history.
- ➔ Provides immense flexibility to the user by providing three different types of data foraging strategy.
- ➔ Gives a ghost notebook feature with prominently marked changed cells to look at older versions of the full notebook.
- ➔ Built as an extension for an open-source environment.

Weak Points

- ➔ Did not use a strong evaluation method: Small sample space and done in a hurried manner.
- ➔ Number of bugs and confusing UI for new users.
- ➔ Some users conveyed that version control is not recommended in their company for data analytics tasks.
- ➔ No evaluation done with trained data scientists with considerable amount of support and time to explore the tool.
- ➔ No evaluation done on the improved version of Verdant after addressing the above problems.

Weak Accept!

Review:

Towards Effective Foraging by Data Scientists to Find Past Analysis Choices

Siddhi Pandare

Summary - Contributions

1. The paper proposes **Verdant**, an open-source extension to JupyterLab to help with logging, navigation and search within previous versions of the notebooks.
2. Verdant **automatically records the history of cells and outputs** in an .ipyhistory file alongside the notebook.
3. The **activity pane** shows a stream of live updates and edits by date and time.
4. A **ghost book** feature: a full previous version notebook with diff highlighting to show what content was changed in that version.

Strong Points

1. It utilises data science notebook-specific artifacts, each saving its own history consisting of both **visual and text data** in a hierarchy preventing duplicates.
2. Verdant provides compact representation using well-researched designs for the cells edited.
3. Verdant provides 3 different ways for searching requirements: **Activity tab** (when and where), **Artifacts tab** (what and how), and **search tab**.
4. Verdant is iteratively developed.

Improvements

1. There were only 16 participants in the evaluation study
   a. Some had no experience with Python or Jupyter notebooks
   b. No experience with Verdant
2. Decluttering the number of versions and number of ghost notebooks.
3. Flexibility in logging for privacy issues.
4. Multi-user version control: Verdant for Google Collab.

ACCEPT

Discussion

- Is it better to do frequent background logging, or to require users explicitly make commits/checkpoints?
- How does storing intermediate values help data scientists track their work?
- Is the tool applicable to collaborative settings such as Google collab?

Next class

How to make progress in research

- Vectoring
- Velocity

Part II topics overview