Teaching and Learning with Jupyter

Lorena A. Barba et al.

2018-11-30

Contents

Chapter 1

Prerequisites

This is a *sample* book written in **Markdown**. You can use anything that Pandoc's Markdown supports, e.g., a math equation $a^2 + b^2 = c^2$.

The **bookdown** package can be installed from CRAN or Github:

```
install.packages("bookdown")
# or the development version
# devtools::install_github("rstudio/bookdown")
```

Remember each Rmd file contains one and only one chapter, and a chapter is defined by the first-level heading `#`.

To compile this example to PDF, you need XeLaTeX. You are recommended to install TinyTeX (which includes XeLaTeX): https://yihui.name/tinytex/.

Chapter 2

Introduction

Project Jupyter is a broad collaboration that develops open-source tools for interactive and exploratory computing. The tools include: IPython, the Jupyter Notebook, Jupyter Hub, and an ecosystem of extensions contributed by a large community. Jupyter Notebook exploded in popularity since late 2014, fueled by its adoption as the favorite environment for doing data science. It has also grown as a platform to use in the classroom, to develop teaching materials, to share lessons and tutorials, and more. Notebooks are documents containing text narratives, combined with executable code (many languages are supported) and the output of that code. This marriage of content and code makes for a powerful new form of data-based communication. Educators everywhere are adopting Jupyter for teaching.

This handbook is for any educator teaching a topic that includes data analysis or computation to support the learning—not just courses in engineering or science, but also data journalism, business and quantitative economics, data-based decision sciences and policy, quantitative health sciences, and others. It aims to give an entry point, and a broad overview of Jupyter in education. Whether you are already using Jupyter to teach, you have found learning materials built on Jupyter that piqued your curiosity, or have never heard of Jupyter, the material in this open book can help you empower your teaching with this new technology.

Educators newly adopting Jupyter can be overwhelmed by having to navigate the ecosystem of tools and content. They could study many examples, or consume myriad blog posts and talk videos to distill the patterns of good practices and technical solutions to best serve their students. Several early adopters, having much experience to share, decided to begin collecting this know-how, and sharing open documentation about using Jupyter for teaching and learning.

The Jupyter Community Workshop in DC (November 2018) began that process, with a book sprint aimed at producing the first version of this handbook. The collaboratively written book consolidates explanations and examples covering key topics, including: what is Jupyter, how to try Jupyter, sharing notebooks with students, locally installing Jupyter, cloud offerings, finding example notebooks, writing lessons in Jupyter, making collections for a course, exporting to other formats with nbconvert, writing textbooks with Jupyter, using Binder, JupyterHub, making assignments and auto-grading, making online courses, teaching with Jupyter in the classroom, active learning and flipped learning pedagogies with Jupyter, guiding learners to create their own content in Jupyter, and more. This open handbook will grow to encompass all you need to know about Jupyter in Teaching and Learning.

If you find these materials helpful or inspiring, give us a shout-out on Twitter using #Jupyter4Edu. We hope you do!

2.1 Acknowledgements

The book sprint was held at the George Washington University in Washington, DC, on 28–30 November 2018, and organized by Lorena A. Barba. Funding to support the logistics and travel of all participants was possible thanks to a grant from Bloomberg to Project Jupyter, and managed by NumFOCUS. The group was fêted at a reception sponsored by Leidos. Participants traveled from all over the country and volunteered their precious time and hard work to give this work to the Jupyter community, with a heartfelt sense of gratitude to all the contributors to the software projects we love and depend on. Thank you!

GitHub repository for this book: https://github.com/jupyter4edu/jupyter-edu-book

Chapter 3

Why we use Jupyter Notebooks

3.1 Introduction

In Chapter 2, you will be introduced to why and how educators are using Jupyter Notebooks. We will highlight examples illustrating how notebooks are being used to increase student engagement, participation, understanding, and performance. Notebooks can also have benefits for students that extend beyond your course, while also offering substantial benefits to you, the teacher, over other tools.

3.2 Why do we as educators use Jupyter?

As teachers we are responsible for a vast array of activities, including creating lessons, lectures, courses, assignments, and supportive environments; encouraging engagement and performance in the classroom; helping students learn to think critically so they can become lifelong learners and problem solvers; making material relevant and meaningful to students' diverse interests and backgrounds; assessing student learning (including grading and evaluation); encouraging students to persist with emotional labor (feedback, communication, etc.); and trying out teaching and learning practices that improve our ability to do all of these things.

In short, teachers design learning environments and experiences. We use Jupyter Notebooks to design learning environments to help support these activities. The goal of this handbook is to provide you with ideas to help you address your own instructional and pedagogical goals. We believe that incorporating Jupyter Notebooks in our teaching has allowed us to help improve students' understanding of course content, help increase student engagement and participation in class, and help make concepts more meaningful and relevant to students' diverse interests. We have found that this can be achieved in a variety disciplines and across many diverse types of instructional goals.

Through a series of anecdotes we will illustrate how you, as an educator, can help increase your students' 1) engagement, 2) participation, 3) understanding, 4) performance, 5) preparation for their career, using Jupyter notebooks. These are starting places and we are confident that you will also take these examples in new and exciting directions.

3.3 But first, what is a Jupyter Notebook?

Project Jupyter is actually three things: a collection of standards, a community, and a set of software tools. A Jupyter Notebook is one part of Jupyter; it is a document that supports mixing executable code, equations, visualizations, and narrative text. Specifically, Jupyter Notebooks are a tool that allows the user to bring together data, code, and prose, to tell an interactive, computational story. Whether analyzing a corpus of

American Literature, creating music and art, or illustrating the engineering concepts behind Digital Signal Processing, the notebooks can combine explanations traditionally found in textbooks with *the interactivity of an application.*

Figure

Visual of a notebook - Side Box with an anatomy of a notebook - descriptions with arrows to particular cells. Link to Binder: Try it now.

Title Cell: Point out prose (markdown)

Introduction Cell: Links to other content

Engaging visualization (a few lines of code 5 or less)

Make the example interactive with widgets

Add a cell with a problem to solve

Solution cell

It would be nice also to have a callout box here that illustrates a relatively simple use of JN for say, one learning objective – or maybe one of each of the following: domain knowledge, programming, and other use.

Jupyter is a free, open source platform that is an excellent learning environment for students. For teachers, it increases our efficiency and decreases cognitive load so we can engage students. Notebooks can be useful for achieving your goals as a teacher in numerous environments from STEM labs or humanities narratives, to podium lectures or flipped classrooms. We use Jupyter Notebooks in small classes and for classes that have hundreds of students. Jupyter Notebooks can be used for teaching part of one lecture or can be used to teach a whole course. Jupyter Notebooks enable us and our students to have a conversation with a problem and link to resources, like audio, video, images, visualizations–and even allow students to mix and remix these. And yet students need to install nothing beyond a modern web browser to use this free software.

Jupyter Notebooks can be used to organize classroom materials and objects, store and provide access to reading materials for students, present and share lecture materials, perform live coding, explore and interact with materials, support self-paced learning, grade students' homework, solve homework problems, or make materials reusable to others (see Chapters 3 and 4).

Read on to find out how we have used Jupyter Notebooks for teaching and learning to benefit both our students and ourselves.** **Jupyter Notebooks support a wide range of learning goals, including learning to program, learning domain knowledge, and practicing communication skills like storytelling. The authors of this book have used Jupyter Notebooks to teach:

- Sciences
  - Physics and astronomy
  - Geoscience
  - Biology
  - Cognitive Science
  - Computer science
  - Data science
  - Statistics
  - Social sciences
- Writing
  - Writing Seminar
  - Writing and technical communication
- Digital Humanities
  - Music
  - Text analysis
  - Metadata processing
- Engineering

    - Chemical engineering (kinetics and reactor design)
    - Mechanical engineering
    - Aerospace engineering
- Introduction to Programming
    - High school
    - College and university-level courses (CS0 and CS1)

Our other use of notebooks for education include:

- Building models/simulations (with and without programming)
- Using widgets to demonstrate and interact with simulations
- Visualizations of process and data

3.4 Course Benefits & Anecdotes

3.4.1 1) Engagement

As teachers we routinely struggle to engage our students, especially when we are constrained by the format of the course (e.g., online, 50-minute lecture), available technologies, students distractions, and/or other factors. Nevertheless, it is substantially our responsibility to create environments and experiences within these limits that engage students in our courses. This is where notebooks can give you another tool to break out of the mundane, and get students engaged in their learning.

3.4.1.1 Conversations with Data

The creators of Jupyter describe it as a set of open-source tools for interactive and exploratory computing, and a platform for creating computational narratives. Jupyter allows us, as educators, to narrate a "conversation between the student and data". Consider this example, using the data of life expectancy of many countries over the years:

```
_I use a short bit of code to make a graph showing the time evolution, in what is called a "spaghetti pl
```

Jupyter notebooks are essential tools of connection – tools that engage learners in transitions in their thinking. The opportunity of intermingling computation into a narrative, creating a conversation with data is a powerful and effective form of communication. With Jupyter, you now have a new form of content to create and share with learners: *computable content*. In a world where every subject matter can have a data-supported treatment, where computational devices are omnipresent and pervasive, the union of natural language and computation creates compelling communication and learning opportunities.

3.4.2 2) Participation

Engaging students in your courses requires their participation and interaction with you, their peers, and/or the content [Michael Moore, 1989]. How, when, and why you use student participation in yours will, of course, depend on your goals, the specific objectives for teaching the content within your course, your students, and other factors. Using notebooks, however, encourages participation and gives you more tools for promoting participation. Notebooks can connect students to authentic external audiences as well. Students can, for example, consume notebooks from other classes, and publish notebooks where others can read them.

3.4.2.1 Real World Experience – bringing concepts to life

Notebooks are living documents, meaning they can be edited to respond to questions or input from students and used a conversation piece during a lecture or presentation.

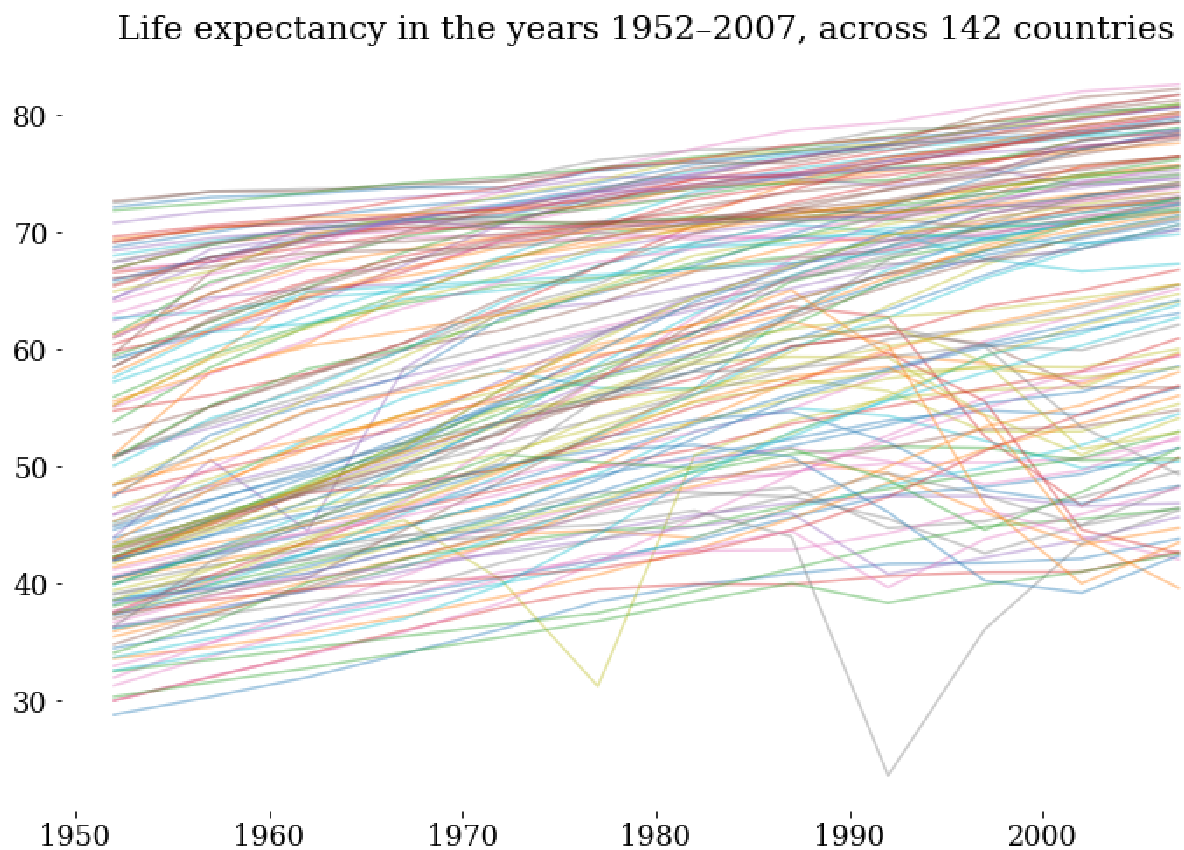

Figure 3.1: From: http://go.gwu.edu/engcomp2lesson4

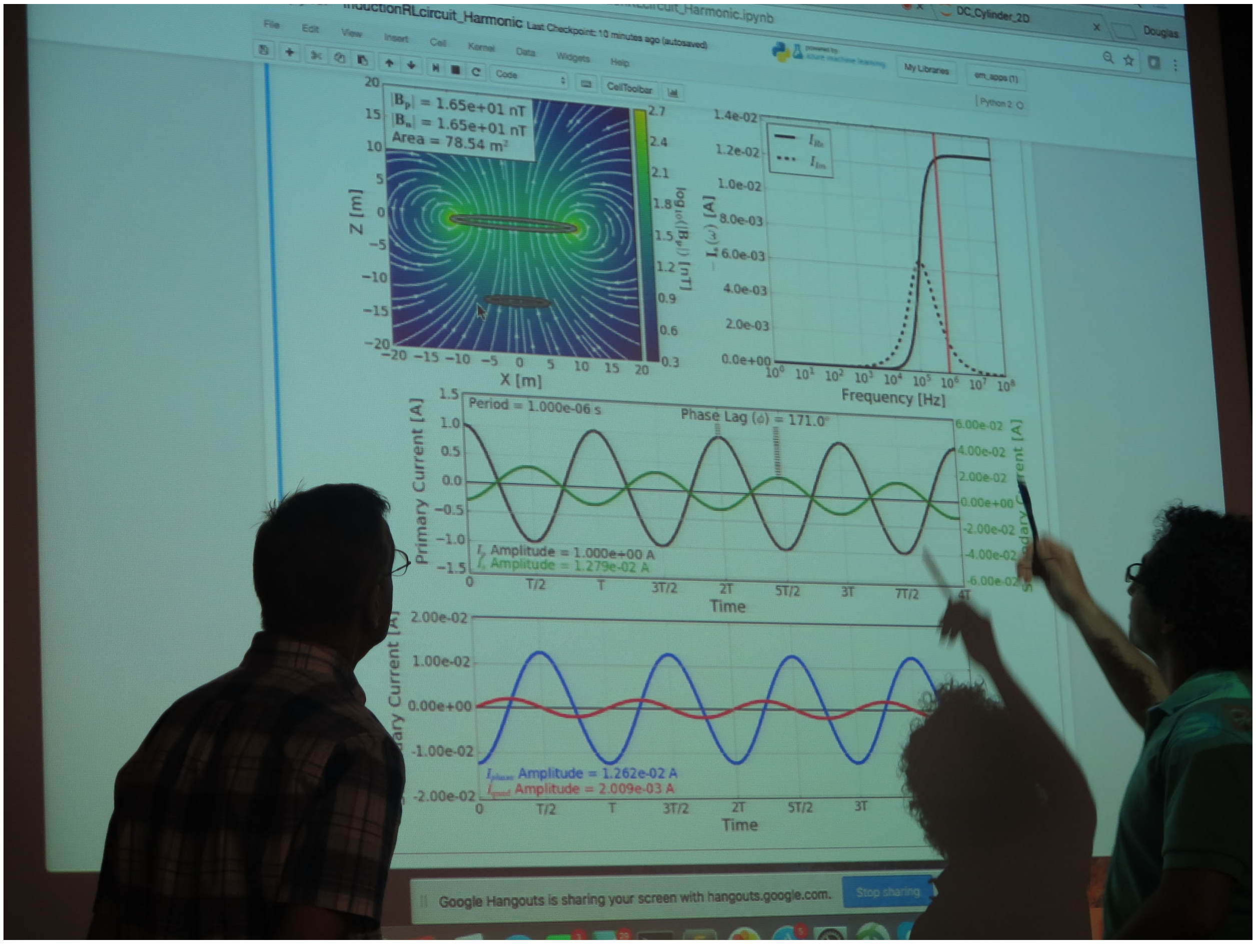

Figure 3.2: Dr. Douglas Oldenburg (left) engaging with a student during a short course on geophysical electromagnetics (https://geosci.xyz). Photo credit: Seogi Kang

```
_Our group uses Jupyter notebooks as "apps" to demonstrate concepts in geophysics. These notebook-apps
```

3.4.2.2 Real World Experience – Ticket to leave

Another example of generating participation in the classroom with Jupyter notebooks is the Activity magic, available as an extension. It creates what has been called a "ticket to leave" (or "exit ticket") via the notebook. The idea of a "ticket to leave" is an excellent way to end a class or lab. Briefly, it is just a survey that you give the students (see figure). Often, these surveys are given via a Personal Response System (also known as "clickers" or PRS) or cell phones. There are a few uses of such surveys:

1. Give the instructor some feedback on the students' understanding, as a whole
2. Provide time and opportunity for students to review and synthesize today's materials
3. Allow the students to apply their recent knowledge to a novel problem
4. An additional instance to learn the materials

These questions do not typically require much time to answer, but are meant to capture the essence of the conversation of the class. After a minute or so to contemplate the question, the students select their answer (by clicking one of the buttons), and instructor shows the gestalt results (see figure).

Good "exit ticket" questions can be domain specific questions, but can also be metacognitive questions (about one's learning style, for example), or high-level organizational questions (e.g., "what was the goal of

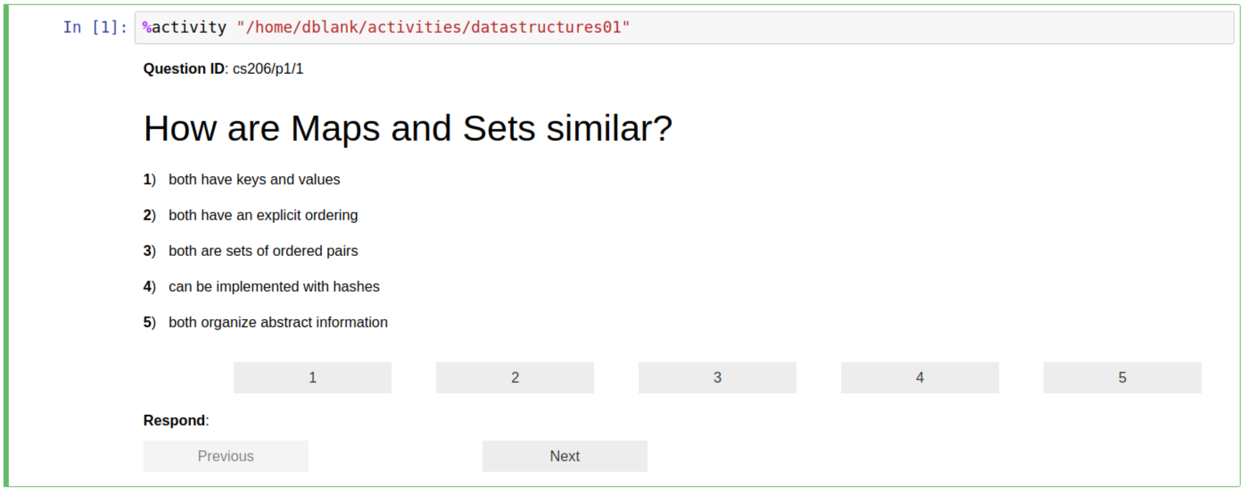

Figure 3.3: Example of the Activity magic seen from the students view. A question, with multiple choice answers is shown, with buttons for their input.

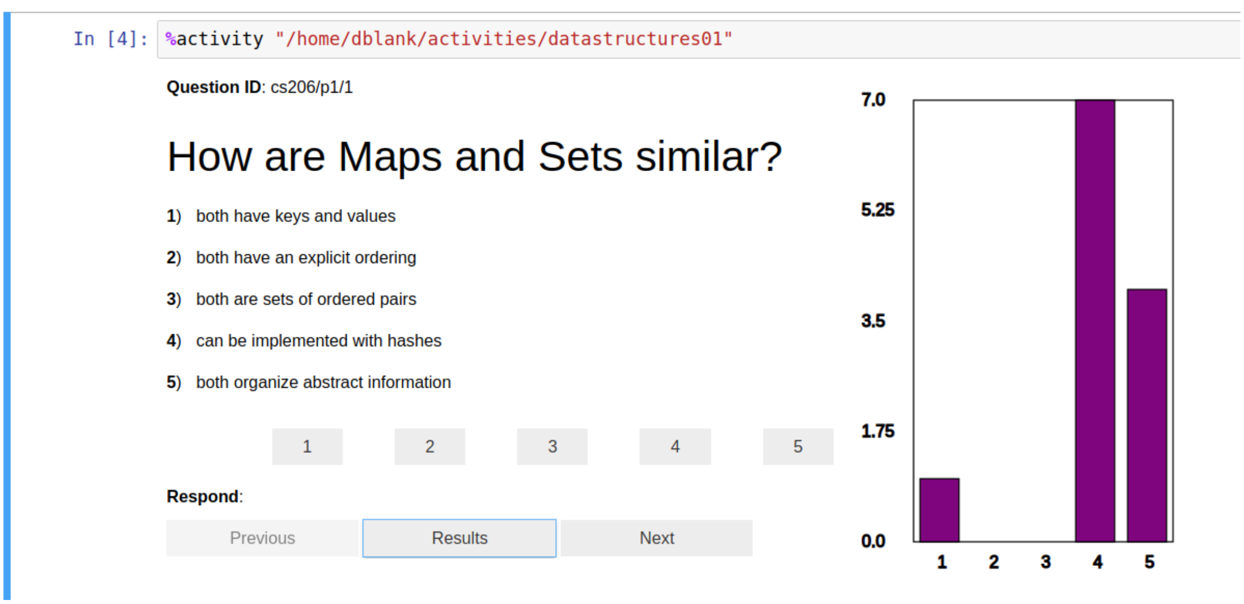

Figure 3.4: The Activity magic, from the the instructor's perspective. The barchart is shown on the project once all of the students have had a chance to respond.

today's discussion?"). We recommend leaving enough time at the end of class (perhaps 10 minutes) to have a full and complete wrap-up discussion. After the discussion, you may wish to adjust the following class meeting if you feel that not enough students had the insight you were aiming for. For more information on "tickets to leave" see https://www.brown.edu/sheridan/teaching-learning-resources/teaching-resources/course-design/classroom-assessment/entrance-and-exit/sample

3.4.3 3) Increasing Understanding

Within any course you will typically try to achieve a diverse set of objectives. Benjamin Bloom (https://en.wikipedia.org/wiki/Bloom%27s_taxonomy) provided a framework for the detailed objectives we want to achieve, ranging from basic knowledge (such as, terminology, specific facts, trends and sequences, classifications and categories, etc.) all the way to ability to evaluate and create (such as, abstract relationships, judgments but based criteria, original works). Achieving the former (i.e., basic knowledge and comprehension) is far easier to achieve that understanding (i.e., evaluation and creation); yet, most often we, as educators, are striving for increasing the complex understanding of our students on the topics we are teaching. The good news is that notebooks offer a valuable tool for teaching toward understanding – moving students, for example, from passively viewing course content to exploring, analyzing, synthesizing, and evaluating the content in active ways.

3.4.3.1 Real World Experience – Guiding Learners at their Own Pace

The fundamental theory behind Computational Fluid Dynamics (CFD) used in Aerospace Engineering is based on understanding the Navier-Stokes equations. "CFD Python" is a collection of Jupyter Notebooks based on a practical module that I began using in class in my Computational Fluid Dynamics (CFD) course at Boston University in 2009. The 5-week module develops worked examples that build on each other to incrementally guide the learner to create a program to solve the Navier-Stokes equations of fluid mechanics, in 12 steps.

```
_In 2013, I was invited to teach a 2 day mini-course in the Latin-American School in High-Performance C
```

Based on the experience developing the "CFD Python" learning module [*Barba et. Al 2018* https://doi.org/10.21105/jose.0002], this basic design pattern was adopted for creating lessons using computable content:

1. Break it down into small steps
2. Chunk small steps into bigger steps
3. Add narrative and connect
4. Link out to documentation
5. Interleave easy exercises
6. Spice with challenge questions/tasks
7. Publish openly online

This was particularly helpful for student understanding.

3.4.4 4) Increasing Student's Performance

The goal of learning is often actualized through the performance of students. This is routinely most visible by what we attempt to assess during and at the end of instruction. Using notebooks we can create a variety of a performance opportunities for students, thereby giving them more opportunities for practice and feedback, as well as more opportunities for us, as instructors, to assessment their ability to perform.

3.4.4.1 Real World Experience – The Worked-example effect

The worked-example effect is the best known and most widely studied of the cognitive load effects [Sweller, J. (2006)]. It refers to providing full guidance on how to solve a problem, resulting in better student performance than problem-solving conditions with no guidance. For complex tasks, inexperienced or beginner learners benefit the most from the worked-examples procedure. One study (Chen et al., 2015) concludes that: "worked example effect occurs for complex, high-element interactivity materials that impose a heavy working memory load" and "when dealing with complex material that learners may have difficulty understanding, high levels of guidance are likely to result in enhanced performance over lower levels of guidance." This research-based guidance seems especially relevant for teaching novice programmers to use computation in the context of their subject matter (science, engineering, or other).

3.4.5 5) Increasing Students' Preparation for Their Career

In preparing students to apply what they have learned, striving to align what happens in the course with what they will experience in their career is important. From using parallel software to mirroring workflows, we want our students to experience and be prepared for the workplace. Recognizing, of course, that workplaces are not static and the skills required for a career are always emerging, using notebooks provides a flexible platform to build skills and build portfolios of what students can do.

3.4.5.1 Real World Experience – Publishing a data narrative as a demonstration of industry ability

For Data Science careers, a publicly shared narrative about a data analytics project goes a long way at demonstrating the student's potential at an interview. Elizabeth Wicks has her students develop a Jupyter notebook that tells the story of a data munging and analysis project done in the class. The students then publish this notebook to their Github profile pages. Being that Jupyter is one of the most popular ways in industry to communicate data science results, the students have a very valuable key to a potential career.

TODO: Add quote from Elizabeth

3.5 Student benefits

Creating opportunities for students to develop as learners stretch beyond the boundaries of any specific course where you may use notebooks. By enriching their learning experience in your course, you will help them develop valuable skill-sets and mind-sets that they will take with them into other courses and into their career.

3.5.0.1 Computational Thinking

Jupyter Notebooks support a wide range of learning goals. Its interactivity enables building intuitive understanding of domain knowledge, such as the understanding of a mechanical response of a system while varying parameters or understanding how an algorithm behaves. Notebooks can also help teach effective communication skills, combining prose with graphics into a strong narrative. Finally, notebooks can support teaching or strengthening programming skills, by combining code with text descriptions and visualizations. Even if a notebook is designed to be consumed passively, the exposure to code helps show students how to do something—and that they can do it themselves. This also helps demystify coding for students who do not view themselves as traditional "computer science" types.

Using notebooks, you can create rich learning experiences that link together the core foundations of computational thinking:

- *Decomposition*: Breaking down data, processes, or problems into smaller, manageable parts
- *Pattern Recognition*: Observing patterns, trends, and regularities in data
- *Abstraction*: Identifying the general principles that generate these patterns
- *Algorithm Design*: Developing the step by step instructions for solving this and similar problems (see https://usr55.dayforcehcm.com/CandidatePortal/en-US/myeyedr/Posting/View/8612?fbclid=IwAR0BVRfn38L7PMftCSbYY_n7IZDhMba0HA7Mmn78ASu5rRIivvPtcAYqxWs)

3.5.0.2 Open-source

Integrating notebooks into classes also exposes students to a large and growing ecosystem of open-source tools. This supports their education, but also provides experience in the same environment of tools used in industries in high demand for trained employees, such as data science and machine learning. The open-source nature of these tools also ensures that course content remains accessible and affordable to all students—including those outside the traditional university environment.

Unlike proprietary notebook technologies such as Mathematica, or specific programming languages/environments such as Matlab or C++, the barriers to entry for students learning with Jupyter notebooks can be extremely low. At a minimum, during a lecture, students can simply watch/read an interactive demo using a notebook, to replace slides or lecture notes. On their own, using a cloud service such as Binder or JupyterHub, students can open any modern web browser to some address and interact with a notebook (an example of this technology can be found at https://jupyter.org/try) , without needing any installation or configuration. In the most complicated case, students can install Anaconda and follow simple instructions to install the Jupyter Notebook, which works and looks the same on all platforms—and is free and open source.

3.5.0.3 Active learning

Thanks to their interactivity, notebooks enable a spectrum of active learning methods, which have been shown to increase performance in science, engineering, and mathematics [Freeman et al. 2018]. To start, students can consume notebook content by reading and running notebooks, then move to editing or completing notebooks as assignments. This allows students to focus on the content and concepts, rather than just note-taking.

At the top of Bloom's Taxonomy is pure creation, where students can, for example, author complete computational essays. In both cases, notebooks support courses where students have a wide range of experience and ability: students who need help can rely on the scaffolding of prose explanations and existing code, while also providing room to stretch and explore for more-experienced students. The additional annotation and prose that accompanies code also helps support non-traditional learners and students from underrepresented groups who may have less initial experience/comfort with programming.

Instilling the habits of active learning, through the use of notebooks, will also provide benefits beyond the boundaries of your course. Interactivity drives engagement, interest, and exploration of concepts. Engaged students in your course, are more likely to be engaged learner in other courses and beyond.

3.6 Instructor benefits

Notebooks can be adopted at a variety of levels and formats, offering flexibility based on the needs of a course and comfort/interest level of the instructor: in-class demos, interactive labs, auxiliary material (e.g., book replacements, lecture note supplements), assignments, or full course content in a flipped learning environment. Notebooks offer a route to active learning methods for instructors without experience of them, but do not force a particular teaching style.

At a minimum, notebooks can be used to make publishable and interactive lecture notes that blend narrative text, images, videos with image and results to present the concepts. Furthermore, these course materials can be developed gradually, starting with a low-effort draft to a more-polished, publishable document that can be easily extended over time—and adopted by others. The growth of open-source communities around software tools and educational resources creates more opportunities for the re-use and adaptation of existing resources.

While many notebook authors do use Python, the Jupyter Notebook supports many languages, so students (and instructors) are not tied to one specific language. Indeed, the name Jupyter comes from three languages: Julia, Python, and R. Furthermore, these (free) tools have minimal barriers to entry—using a cloud infrastructure means students and instructors do not have to install anything, while in the "worst" case installations require a few command-line excursions, but these are free, openly available, and cross-platform.

3.7 Conclusions

We hope that this chapter has illustrated that teaching with Jupyter notebooks can be valuable for you and your student. We have notebooks to be a tool that can increase student engagement, participation, understanding, performance, and preparation for their careers. These are substantial accomplished that can be achieved in a variety of disciplines and content areas. Using several real world examples, we attempted to illustrate the numerous ways teachers are using notebooks. Hopefully these, when combined with the chapters that follow, will guide you in 1) supporting your students' learning, 2) giving you confidence that you can use notebooks, 3) help you understand the necessary logistics, and 4) help give you clear expectations of what can be accomplished with Jupyter notebooks.

Chapter 4

Jupyter ecosystem: notebooks

The Jupyter system supports over 100 programming languages (called "kernels" in the Jupyter ecosystem) including Python, Java, R, Julia, Matlab, Octave, Scheme, Processing, Scala, and many more. Out of the box Jupyter will only run the IPython kernel, but additional kernels may be installed. Additional language support continues to be added by the open source community and https://github.com/jupyter/jupyter/wiki/Jupyter-kernels is the best source for an up-to-date list. Note that these projects are developed and maintained by the open source community and exist in various levels of support. For example, some kernels may be supported by a vast number of active (and even paid) developers, while others may be a single person's pet project. When trying out a new kernel, we suggest exploring a kernel's community of users and documentation to see if it has an appropriate level of support for your (and your students') use.

Jupyter's kernel flexibility allows instructors to pick the right language for a particular context. For example instructors may use Python to teach programming, while switching to R to teach statistics, and then perhaps Scala to teach big data processing. Regardless of the language chosen, the Jupyter interface remains the same. Thus, some cognitive load can be lessened when using multiple languages in, or across courses (e.g., the user interface doesn't change at all between the student's Digital Humanities and Biology courses). Of course, if you can remain in one language in a course, that is often easier for the student.

4.1 Resources for authoring Jupyter notebooks

Before embarking on writing notebooks for your course, we recommend that you look around on the internet for related courses. There may be a similar course for which an instructor has already generated notebooks that you can use or adapt for your course. In many cases, instructors are very happy to have collaborate on open source educational resources and have the resources be used by other instructors. The following sections will be oriented towards Python simply because it is currently the language with the largest Jupyter feature support.

4.1.1 Accessing documentation in the notebook

One of the best features of quality libraries is their documentation, which students and other users will likely consult regularly. From a notebook cell, the TAB key autocompletes (or gives completion tips) and SHIFT-TAB brings up full documentation. Similarly, using a question-mark after a method or function will bring up the documentation after the cell is run, as shown in Figure XX.

Using this feature in class during live coding and modification or while just explaining how code works helps increase student comfort and enable them to work effectively with libraries.

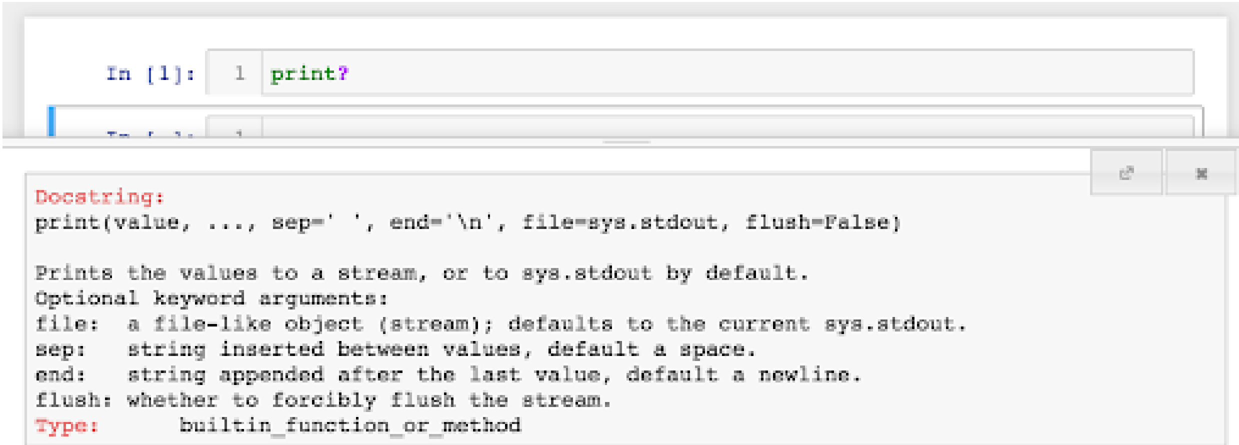

Figure 4.1: A question mark can be used after a method or function and the cell run in order to bring up the documentation.

4.1.2 Widgets

Widgets provide the opportunity a for learners and instructors to interact with code outputs, such as charts and tables, interactively. Widgets are simple and quickly generated “mini” Graphical User Interfaces (GUI) that give the notebook user access to slide bars, toggle buttons, and text-boxes. They can be used in conjunction with code, allowing a change of mindset from programming as a primary goal to exploring a model or computation as the primary goal. Alternatively, the code can be hidden and the widgets used to create a notebook “app” that might connect input parameters with a simulation and a plot.

Currently only a small subset of kernels have widget functionality. For example, the reference implementation of widgets are the Jupyter-Python widgets (https://ipywidgets.rtfd.io). It includes widget components to generate and display sliders, progress bars, text boxes, check boxes, toggle buttons, etc. Many popular visualization tools, such as matplotlib, Plotly, leaflet.js, three.js, have Jupyter-Python widget implementations. The documentation contains an up-to-date list of all of the widgets and their variations. Additionally, the `interact` method wraps allows you to wrap a function, which might be a simple computation or a complex simulation that produces a plot, and provides widgets for each of the inputs to the function. Figure XX shows a simple example of a sinusoid plot whose frequency is controlled by a slide-bar. Another kernel that has some widget functionality is C++ (https://github.com/QUantStack/xwidgets).

In addition to the IPywidgets library, the ipyleaflet library (https://ipyleaflet.rtfd.io) enables an interactive map to be displayed in a notebook.

4.1.2.1 Example

```
from ipyleaflet import Map
Map(center=[34.6252978589571, -77.34580993652344], zoom=10)
```

For the ambitions reader, there are resources available for you to write your your own custom widgets. The widget cookie cutter project (https://ipywidgets.rtfd.io) is a good place to start.

4.1.3 Magics

Magics are meta-commands that only function within Jupyter and allow a user to access features that are language/kernel specific. For instance, the IPython kernel provides a number of magics that can be useful while developing Jupyter notebooks that uses Python as the primary language. These are documented and

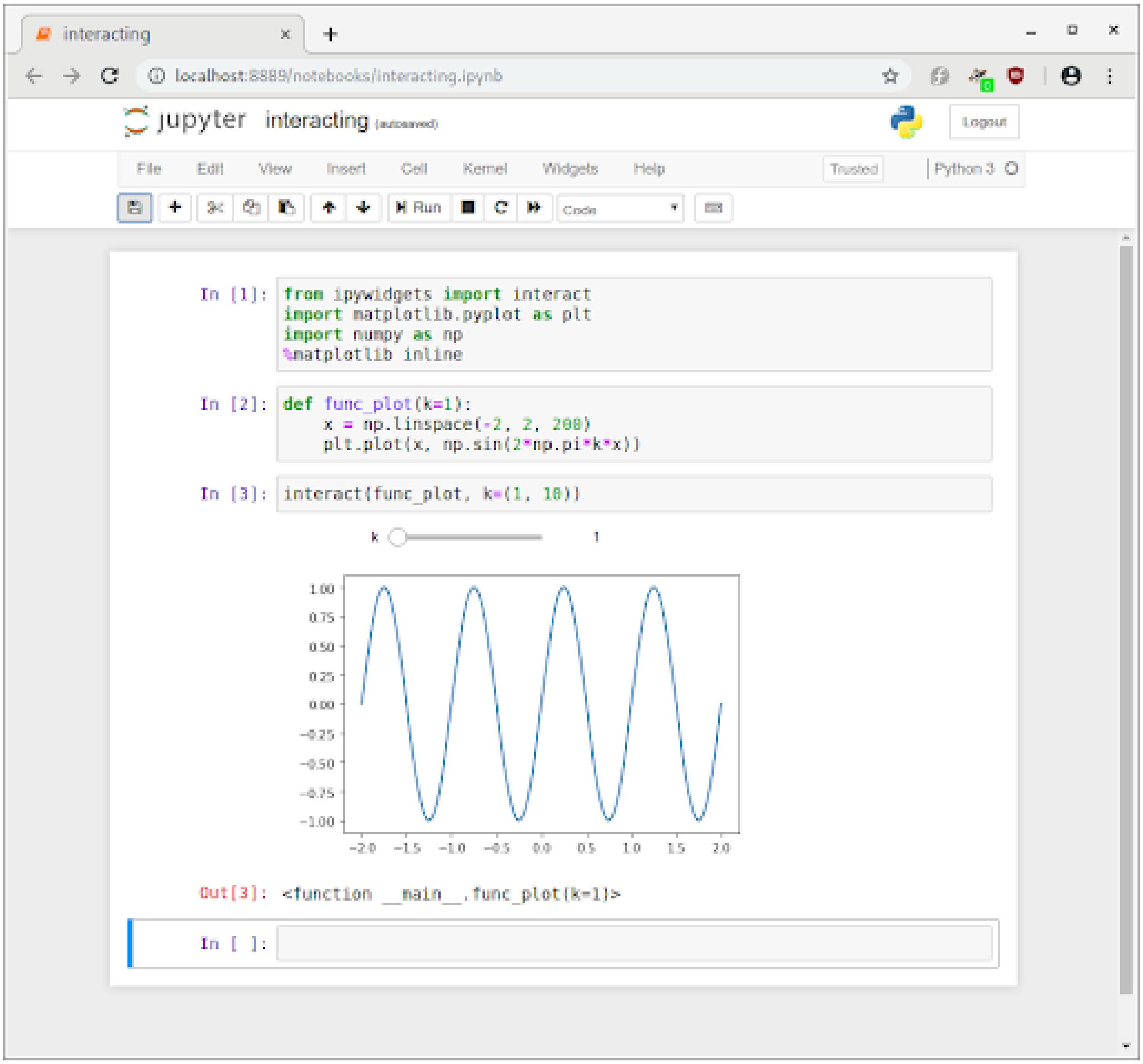

Figure 4.2: Here, a simple slider is used to interactively change the variable k in our function as we plot it.

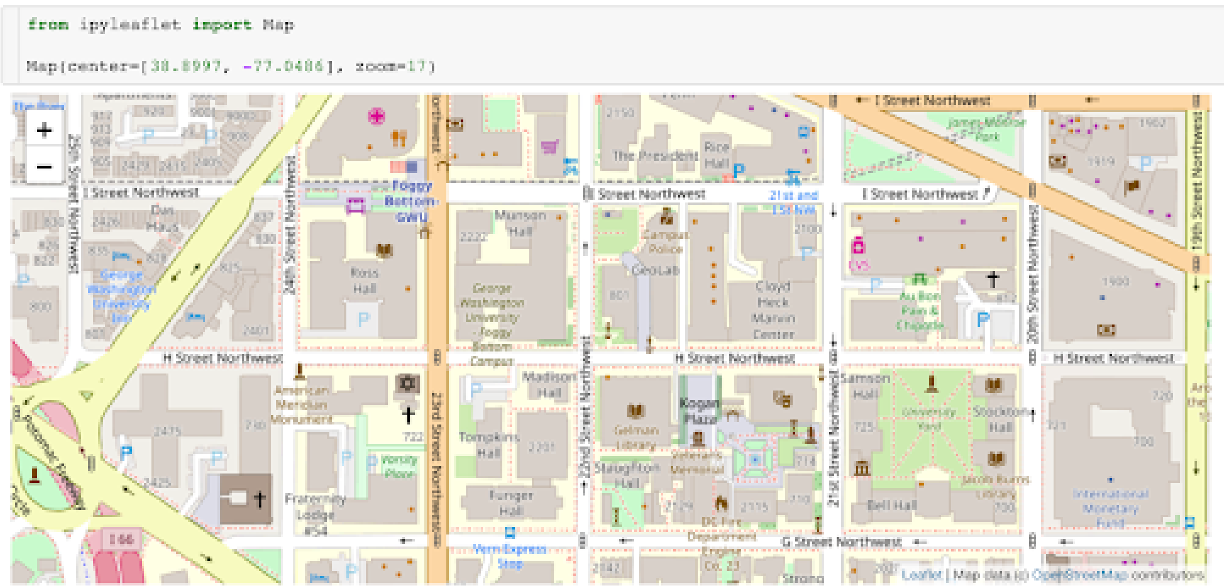

Figure 4.3: Interactive map widget with `ipyleafletalt_text`.

we will only call out a few of these here. There are many other magics available for different kernels but the downfall of these are that they are specific to Jupyter so does not translate to scripts and in a teaching environment we generally recommend to be sparing in their use so as to not obfuscate what is happening. Magics always begin with either a single `%` for single line commands or double `%%` for applying a command to an entire cell. Many magics can be used with single or double percents, but some do not.

- `%run` allows for running external scripts, capturing output and displaying it in the notebook, e.g. `%run my_script.py`.
- It is common to use matplotlib for visualization. In Jupyter, the magic `%matplotlib` allows the matplotlibs resulting figures to be displayed in the notebook. `%matplotlib inline` and causes static images to be embedded in the notebook and `%matplotlib notebook` causes interactive images (zooming, panning, etc) images to be embedded.
- `%time` times the execution of the Python expression following the command, e.g. `%time sum(range(1000))`.
- `%timeit` is is similar to `%time` with the difference that it runs the expression multiple times and reports the average execution time.
- `%reset` removes all user defined variables along with input and output. Magics often have "flags", following the Unix command pattern. For example, `%reset -s` is a soft reset and only removed user defined variables. These commands can be useful to help to overcome out-of-order execution problems.

A good example of a magic operating on the entire contents of cell is the `%%HTML` magic that will force the cell to be interpreted as HTML and render it appropriately. You can also use magics to call other languages while running the IPython kernel. For example, you can run R code from within an IPython notebook by using the `%%R` magic.

4.1.4 Notebooks Under Version Control

Keeping notebooks under version control is a great way to not only keep track of changes to your created content but can also allow for sharable content. In a course where multiple people are contributing to the development of notebooks for the course, using version control in conjunction with a platform like GitHub, allows authorship to be tracked and provides communication tools for reviewing new contributions or outlining requested development for a new assignment, activity, etc. Another advantage of using version control is that there are services that will provide rendered views of your notebooks that are publicly accessible. GitHub for instance will show a rendered version of the notebook rather than the direct text that a notebook is comprised of. There are a few pitfalls with this with LaTeX rendering as platforms do not always render the notebooks the same as they would appear in an active Jupyter interface.

There are a few caveats to keeping notebooks under version control that should be kept in mind. First of all the output will be stored in the repository, including images, if clear output is not used before committing changes. This can make detecting changes difficult as changes in output will be detected when nothing has actually changed content wise. The tracked notebooks also can become large if output is tracked. Even with clearing the output reading through changes can be difficult due to the format of the notebook, Notebooks are plain-text files and the file format is represented as JSON. This can be an issue if you are storing notebooks in a version control system, like git, and you wish to see differences between versions. For more information, see: https://github.com/jupyter/nbdime and https://github.com/mwouts/jupytext.

4.1.5 Testing Notebooks

Before distributing notebooks, and in particular if you are working with multiple contributors to the course material, testing the notebooks before they are distributed to students or used in a live demo can help mitigate unexpected bugs. At a minimum, you can test that the notebook executes cleanly from top to bottom by restarting the kernel and running all cells from top to bottom. This can be done from the menu (Restart + Run all).

Though it requires a bit more setup, tests can be run automatically using a continuous integration service, such as TravisCI (https://travis-ci.org). This will require executing the entire notebook via the command line, for example `jupyter nbconvert --ExecutePreprocessor.enabled=True --to=html my_notebook.ipynb` will execute the notebook (same as pressing "Restart + Run All") and then convert it to HTML. These services can be connected to GitHub so that any time that the notebooks are changed, the tests are run automatically. Simplifying this process is an area that is under development in the open source community. The package https://github.com/opengeophysics/testipynb provides an easy way to test notebooks.

4.1.6 Essential Python Libraries

The purpose of this section is to introduce some of the most widely used packages within the Python ecosystem. Python is not the only language that can be used in the notebook; over 90 different kernels enable different programming languages be used. We discuss the use of other languages in the section: Using other languages. Python has a large open-source community which develops and maintains an ecosystem of over 150,000 software packages, making it a common language of choice in many disciplines.

The core Python library (https://docs.python.org) contains data types such as lists and dictionaries, as well as core functionality such as arithmetic operators and simple file parsers. Most tasks can be achieved with core python, however, they are often made easier with higher-level libraries. These libraries are particularly useful for scientific computing with Python.

Among the vast number of packages in the Python ecosystem, Numpy, Scipy, Matplotlib and Pandas are among the most commonly used. Numpy (http://www.numpy.org/) is a fundamental library for numerical and scientific computing with Python. It contains data structures for numerical arrays, tools for linear algebra, and random number capabilities. Scipy (https://docs.scipy.org/) contains a variety of functionality for common scientific computing tasks, such as optimization, interpolation, statistics and signal processing. It also includes fundamental constants from many disciplines such as the speed of light as well as data structures for sparse matrices. Matplotlib (https://matplotlib.org/) is the core plotting library for python and can be used inline in the notebook with the `%matplotlib notebook` or `%matplotlib inline` cell magic.

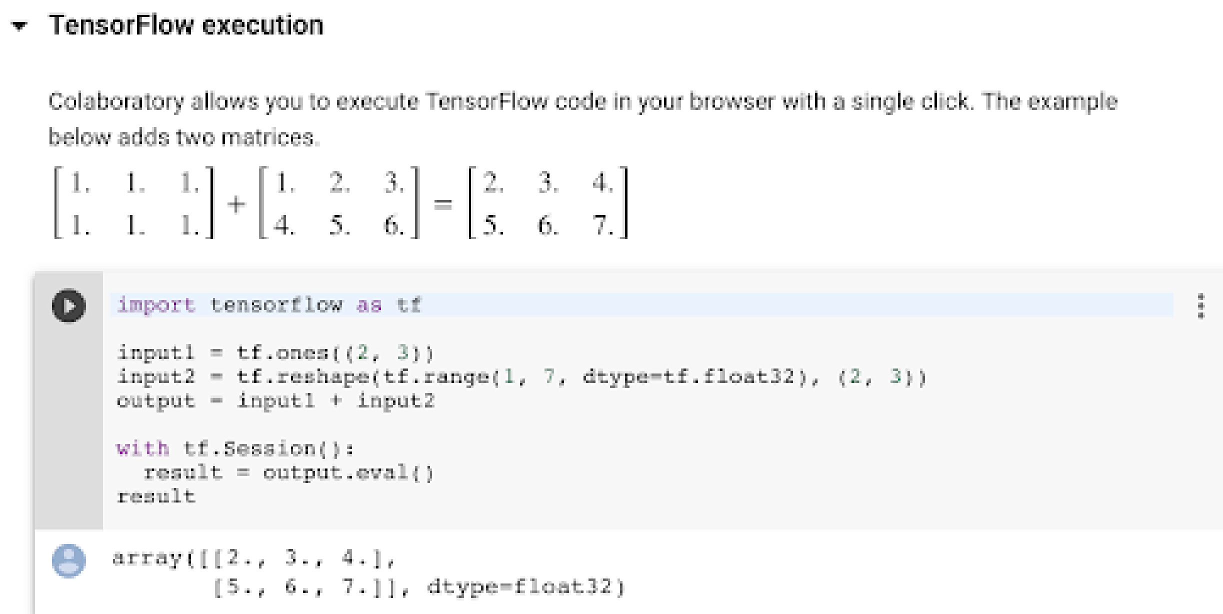

Figure 4.4: Google Collaboratory uses Jupyter extensions to customize Jupyter for their users. The run/play icon to the left of the code cell is created using extensions. This is not present in the standard Jupyter software. TensorFlow is a library for creating Machine Learning experiments in Python.

Pandas (https://pandas.pydata.org/) provides resources for data analysis and more flexible data structures than core python. A good resource for getting familiar with these libraries is the Scipy Lecture Notes (https://www.scipy-lectures.org).

4.1.7 Advanced topic: extensions

There are many community contributed extensions that add functionality to Jupyter notebooks. Extensions vary from displaying an automated table of contents for a notebook, or prettify code, or hiding/showing solution cells. Here is the link for how to install and enable extensions: https://jupyter-contrib-nbextensions.readthedocs.io/en/latest/install.html

Here is a list of a collection of extensions that are bundled together: https://jupyter-contrib-nbextensions.readthedocs.io/en/latest/nbextensions.html

Creating custom extensions is a way to extend or customize Jupyter to add a capability that is not currently available with current extensions or out of the box. These extensions may be targeted for a specific kernel. Here are instructions for how to create and install custom extensions: https://jupyter-notebook.readthedocs.io/en/stable/extending/frontend_extensions.html

Figure X shows shows how Google Collaboratory, one of many tools to interact with Jupyter notebooks, leverages the power of Jupyter extensions for custom interaction and presentation.

4.2 Tips and tricks

4.2.1 Reminders

If you are using a single notebook as a standalone exercise in a traditional class (i.e., this is the only computational component of your class), then it is helpful to have a few cells at the top of that notebook that reviews how to navigate through the notebook and how to insert cells, etc.

4.2.2 Feedback

How do we get feedback from students in an interactive session to see if students have completed an exercise?

A low tech solution is to give students sticky notes of different colors, one meaning "finished" and one meaning "need help", that they can stick on the back of their computers. The instructor can then quickly look up to take a survey of the state of the class and decide how to proceed.

Projecting Slack or a similar chat group on a screen and having student copy-paste solutions (provided they are short functions) is a nice way to let everyone in the class see one another's solutions. A positive aspect of having multiple student solutions projected is that it can show the variety of ways to solve a problem. This gives an opportunity to talk about the readability of solutions and their efficiency. A downside is that in a large class, the shear volume of posts can make it overwhelming. Instead polling can be used to aggregate student answers and provide some form of feedback to the instructor. Nbgrader or travis-CI can also be options here, requiring students to submit completed code where it is assessed automatically. These will however require more setup and can take some time to complete.

4.2.3 Explaining each cell

A good rule of thumb is to always include a markdown cell above every code cell.

4.2.4 Custom Styling

New notebook creators often try to centrally manage the formatting of headings, equations, and other textual items. For example, rather than using a standard markdown heading, a creator may over-design the headings by using HTML styles. This may create two problems:

1. The rendering of the notebook markdown may change and your formatted HTML header may not maintain the same look over time.

2. Headers created using markdown can be used by notebook tools, such as automatically creating a Table of Contents.

Our recommendation is to resist the desire to customize the styling and simply use the default representations. If you want to do customization (for example if you want to color certain cells) you can use CSS.

4.2.5 Length of Notebooks

Notebook authors sometimes make the notebooks very long with many topics and sections. Notebook sections and cells are currently not easily reused in a copy/paste sense for mixing intra-notebook content. Until this functionality is available, we recommend that authors make short, self-contained notebooks around short topics. This allows other notebooks authors to mix and match notebooks to create curriculum.

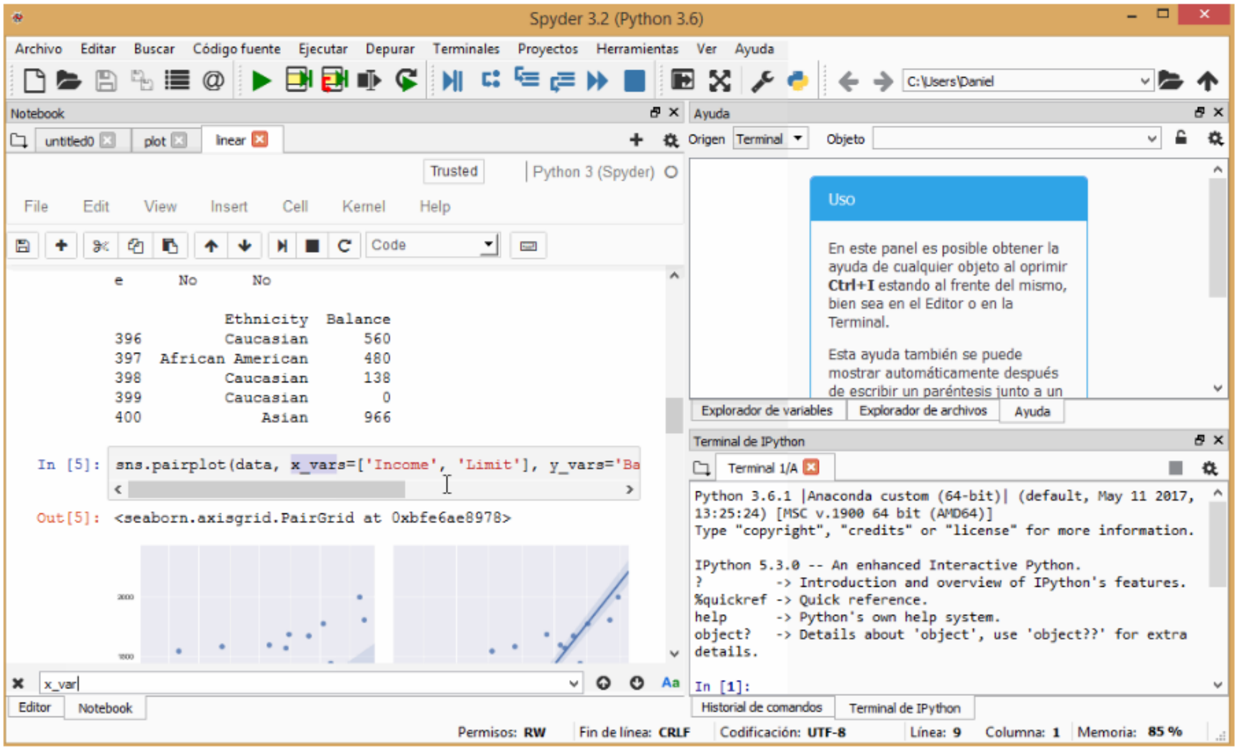

Figure 4.5: alt_text

4.3 Gotchas

4.3.1 Programming Language Jupyter

Teaching a class entirely with Jupyter can give the sense to students that this is the way all computational exploration is done. In particular, students can be confused into thinking that programming requires the notebook, instead of understanding that a notebook is just one way to interact with a particular language. This point should be made clear periodically. A good way to reinforce this is to show how to take a function that has been developed and debugged in a notebook and cut-paste it into a script (such as a file ending in .py for Python) and then import it into the notebook to regain that functionality. Also, the Integrated Development Environment (IDE), Spyder, has a plugin (https://github.com/spyder-ide/spyder-notebook) that allows notebooks to be displayed alongside Python scripts and a python terminal which can be useful for showing this dichotomy.

Caption: Jupyter notebook displayed in a window pane inside Spyder.

4.3.2 Restart, restart, restart...

Often, students may need to stop a computation, and this can be accomplished by pressing the “Interrupt” button in the toolbar. However, students should also be made aware of how to restart the kernel in a notebook, and what this means. There are several instances when students might need to do this. Sometimes students write code that can go into an infinite loop. The visual cues that notebooks give in this case are subtle, and students may not realize this and don’t understand why the notebook is non-responsive. In live-coding situations, it can be useful to demonstrate this to students and show them how to restart the kernel and carry on.

A second instance of where restarting a kernel might be needed is due to how the notebook stores the state of the computation. We like to think that, since the notebook is laid out in a linear fashion, that the state will always reflect what would happen if the notebook was run from the start up to that point. However, it is common to work in a notebook out of order, for instance if students ask a question about some previous example. If the variable has been changed in subsequent cells, then its value might not reflect what you expect when you rerun a cell earlier in the notebook. Restarting the kernel is sometimes the only solution.

4.3.3 Notebook hygiene

Many gotchas can be mitigated by developing notebooks that will be robust to incremental and non-linear execution. The main principle is to minimize side-effects of executing a cell and manifests itself somewhat differently in different languages; our suggestions here will be relevant to Python and may need to be adapted for other languages. Notebooks should generally be able to execute sequentially, such as via "restart kernel and run all cells". (An exception is when a notebook is intentionally incomplete for the purpose of live coding or student exercises, see nbgrader or the exercise estnations for more elegant ways to handle this.) Variable mutation is the most common way in which a notebook may malfunction when executing cells in a non-linear way (e.g., in response to student questions or when comparing and contrasting different methodologies). Sometimes this mutation is incidental, through dummy variables that were not meant to have significance outside the scope of the cell in which they are used. Their scope can be limited by placing them in a function, even if that function is only called once. Redefinition of functions can often be avoided by parameterizing the desired functionality as would typically be done if designing a library (though this may be a distracting software design for novice programmers). Function definitions should have little or no dependency on variables from their enclosing scope. When modifying cells for demos and formative assessments during class, it is useful to either copy the cell or modify/execute such that a conforming implementation remains present when moving on to other cells where it may be used. Additionally, you can minimize these issues by grouping code in a single cell that should always be executed sequentially, because code within a cell will always be sequential.

Chapter 5

About the Authors

5.1 Project Lead

5.1.1 Lorena A. Barba

- George Washington University
- labarba@email.gwu.edu
- @LorenaABarba

Lorena A. Barba is Associate Professor of Mechanical and Aerospace Engineering at the George Washington University. She adopted Jupyter in 2013 and since then used it in every course she teaches. Her open course materials are well known and used by thousands of learners: CFD Python and Numerical MOOC are the best examples.

5.2 Authors at the sprint

5.2.1 Lecia J. Barker

- University of Colorado Boulder
- lecia.barker@colorado.edu
- @leciab

Lecia Barker is an Associate Professor and Associate Chair of Undergraduate Studies in the Department of Information Science at the University of Colorado Boulder. She is also a Senior Research Scientist for the National Center for Women & IT. Her research group is studying the diffusion and adoption of teaching practices in undergraduate computer science. Lecia holds a Ph.D. in Communication from CU Boulder and an MBA in Marketing from San Diego State University.

5.2.2 Douglas Blank

- Bryn Mawr College
- dblank@brynmawr.edu
- @dougblank

Douglas Blank is Associate Professor in the Department of Computer Science at Bryn Mawr College, a small, all-women's college outside of Philadelphia, PA, USA. He has a joint Ph.D. in Cognitive Science and

Computer Science from Indiana University, Bloomington. For over 20 years, Douglas has taught all levels of Computer Science. For the last 4 years, he has used Jupyter notebooks exclusively in the classroom. Douglas has published in the areas of Computer Science Education, Robotics, Artificial Intelligence, and Deep Learning. He is on the advisory board of Engage-CSEdu.org, a joint project between Google and the National Center for Women and Information Technology (NCWIT). Douglas also writes text and code at his website douglasblank.com.

5.2.3 Jed Brown

- University of Colorado Boulder
- jed@jedbrown.org
- @five9a2

Jed Brown is an Assistant Professor of Computer Science at the University of Colorado Boulder. He has been teaching numerical and scientific computing courses using Jupyter Notebook and nbgrader for three years, and leads a research group that develops computational methods and community software for computational science.

5.2.4 Allen Downey

- Olin College
- downey@allendowney.com
- @AllenDowney

Allen Downey is a professor of Computer Science at Olin College and the author of a series of open-source textbooks related to software and data science, including *Think Python*, *Think Bayes*, and *Think Complexity*, published by O'Reilly Media. These books, and the classes based on them, use Jupyter notebooks extensively. Prof Downey holds a Ph.D. in computer science from U.C. Berkeley, and M.S. and B.S. degrees from MIT.

5.2.5 Tim George

- Project Jupyter
- tgeorgeux@gmail.com

Timothy George is the Lead UI/UX Designer for Project Jupyter, focusing primarily on JupyterLab. In addition to his formal duties, Tim is also in working with Jupyter on design strategy, future products, governance, diversity and inclusion. He studied HCI at UC Irvine's Donald Bren School of Informatics and Computer Science where he received a Master's Degree.

5.2.6 Lindsey Heagy

- University of California Berkeley
- lindseyheagy@gmail.com
- @lindsey_jh

Lindsey Heagy is a Postdoctoral Researcher at the University of California Berkeley working on Project Jupyter and Jupyter in the geosciences. She recently completed her PhD at the University of British Columbia in geophysics. She is a project leader of GeoSci.xyz, an effort to build collaborative, interactive, web-based textbooks in the geosciences, and a core contributor to SimPEG, an open source framework for geophysical simulation and inversions. The GeoSci.xyz project relies heavily on Jupyter for making the content come to life.

5.2.7 Kyle Mandli

- Columbia University
- kyle.mandli@columbia.edu
- @KyleMandli

Kyle Mandli is an Assistant Professor in the Department of Applied Physics and Applied Mathematics at Columbia University. He has developed a set of openly available course notes centered around Jupyter notebooks and uses Jupyter for homework in conjunction with nbgrader. His other research interests include development of computational methods for coastal hazards such as storm surge and tsunamis.

5.2.8 Jason K. Moore

- University of California, Davis
- jkm@ucdavis.edu
- @moorepants

Jason K. Moore is an Assistant Teaching Professor of Mechanical and Aerospace Engineering at the University of California, Davis. He currently teaches dynamics and mechanical design related courses. He utilizes Jupyter notebooks to teach modeling and simulation and is working on a textbook about Mechanical Vibrations. He is responsible for the Jupyter related features in the LibreTexts project and is also a core developer of the SymPy and PyDy projects which utilizes Jupyter for training workshops, e.g. PyDy Tutorial and SymPy Code Generation Tutorial. Jason has PhD, MSc, and BSc degrees in mechanical engineering from UC Davis and Old Dominion University.

5.2.9 David Lippert

- Leidos

David Lippert is a software engineer at Leidos in Arlington, Virginia. He utilizes Jupyter notebooks primarily for exploratory data analysis and for training and evaluating machine learning algorithms. He has written Jupyter notebooks to create new Dr. Seuss sonnets and to evaluate if the Rotten Tomatoes Tomatometer can be trusted. He has a BA in computer science from Middlebury College.

5.2.10 Kyle Niemeyer

- Oregon State University
- kyle.niemeyer@oregonstate.edu
- @kyleniemeyer

Kyle Niemeyer is an Assistant Professor of Mechanical Engineering in the School of Mechanical, Industrial, and Manufacturing Engineering at Oregon State University. He teaches courses in numerical and analytical methods for solving differential equations as well as gas dynamics, and recently developed a graduate course on software development for engineering research. His research group develops and applies methods for modeling combustion and chemically reacting fluid flows. He is also on the steering committee of the Cantera open-source project for chemical kinetics, thermodynamics, and transport processes.

5.2.11 Ryan Watkins

- George Washington University
- rwatkins@gwu.edu
- @parsingscience

Ryan Watkins is a Professor of Educational Technology at George Washington University in Washington DC. He leads the Human-Technology Collaboration (HTC) PhD program area, and he teaches courses in needs assessment, instructional design, and research methods. Ryan's research focuses on how people and organizations define and assess needs. He is co-host of Parsing Science, a podcast where researchers share the stories behind their science. He also developed the We Share Science platform for sharing video abstracts of research.

5.2.12 Richard West

- Northeastern University
- R.West@northeastern.edu
- @richardhwest

Richard West is Associate Professor of Chemical Engineering at Northeastern University in Boston. He leads a research group in computational modeling for complex reacting systems like combustion or catalysis. He is a core member of the Cantera open-source project. As well as in an elective on "computational modeling in chemical engineering", he has integrated Python and Jupyter into core classes on chemical kinetics and reactor design, at both the undergraduate and graduate levels. As part of his NSF CAREER award, he is developing modules to teach students to use Python and SciPy to solve chemical engineering problems.

5.2.13 Elizabeth Wickes

- University of Illinois at Urbana-Champaign
- wickes1@illinois.edu
- @elliewix

Elizabeth Wickes is a Lecturer at the School of Information Sciences at the University of Illinois at Urbana-Champaign. She teaches foundational programming from an information and data sciences perspective, as well as other coursework on open data and reproducibility. Her programming course lectures are written in Jupyter Notebooks and the class is taught via live coding.

5.2.14 Carol Willing

- Cal Poly San Luis Obispo
- willingc@gmail.com
- @WillingCarol

Carol Willing is a Research Software Engineer at Cal Poly San Luis Obispo working full-time on Project Jupyter. She is a Python Software Foundation Fellow and former Director; a Project Jupyter Steering Council member; and a core developer on CPython and Jupyter. Carol has an M.S. in Management from MIT and a B.S.E. in Electrical Engineering from Duke.

5.2.15 Michael Zingale

- Stony Brook University
- Michael.Zingale@stonybrook.edu
- @Michael_Zingale

Michael Zingale is an Associate Professor and computational astrophysicist at Stony Brook University. He has a PhD from University of Chicago (2000). He frequently teaches numerical methods and Python for scientific computing graduate courses, relying on Jupyter notebooks and python for much of the presentation. He is an advocate for open educational resources, as a founder of the Open Astrophysics Bookshelf project where he hosts his *Introduction to Computational Astrophysical Hydrodynamics* text.

Chapter 6

Glossary

Anaconda: a free, open-source package manager, environment manager, Python distribution, and collection of over 1,500+ open source packages including and also Jupyter. https://www.anaconda.com/what-is-anaconda/

API (Application Programming Interface): the exact details of interacting with software, usually used by other software.

Binder: a hosted service that allows anyone to launch their own sandboxed notebook environment from a Git repository. https://mybinder.org

cell: the area in a Jupyter Notebook where you can enter markdown, or computer code.

cloud, in the: used to describe software or documents hosted on a remote computer accessed over the internet.

CSV (Comma Separated Values): referring to a comma-separated value file. A plain-text file format such that each line is a list of data separated by commas.

execute: technical term for having the computer perform the instructions of your program. Alias for "run it."

extension, Jupyter: in this instance, it is not a request for more time. Rather, a Jupyter extension is a bit of code, often developed by a third-party, that adds additional functionality to Jupyter. For example, a popular extension is a Table of Contents creator.

flipped classroom: a teaching style where students work on their own outside of class to learn new material (sometimes by watching recorded lectures or reading descriptive/interactive notebooks) and the come together in the classroom to practice what they've learned through exercises or experiments.

Git: a popular version control system (VCS) used for keeping track of changes of files over time.

IDE (Integrated Development Environment): software that assists in the development of additional software.

Jupyter: The term "Jupyter" may refer to one of a couple of different things: a community of users and developers focused on the open source software; the collection of tools and standards that, together, allow projects like the Jupyter Notebook to operate. The name refers to the three core programming languages supported: Julia, Python, and R.

JupyterHub: a cloud service that can provide access to Jupyter notebooks and environments to multiple users via a modern web browser. http://jupyter.org/hub

kernel: In Jupyter, a kernel is the packaging up of a language, and related programs needed to run it. For example, Python2 and Python3 are separate kernels.

LMS (Learning Management System): a cloud service that helps instructors manage aspects of classrooms.

load: how many students can a computer support?

Markdown: a text format that allows for basic formatting (headers, text styles, links) mixed inline with the text. Markdown files usually have the extension `.md` and can be rendered natively by GitHub and other tools.

magic: a meta-command typically starting with one or two percent signs. Changes the meaning of the contents of a line (one percent sign, `%`) or the cell (two percent signs, `%%`) from code to a particular meta-instruction. For example, `%%R` indicates that the cell contents will be interpreted as commands to the R language. Magics are kernel-specific (e.g., vary with the kernel in use).

nbgrader: a tool for creating, handling, and automatically grading assignments based on Jupyter notebooks. https://nbgrader.readthedocs.io

nbviewer: a web application for rendering Jupyter notebooks as static web pages, providing a URL to share and view them with a modern web browser. https://nbviewer.jupyter.org

nbconvert: a tool for converting Jupyter notebooks into other formats such as PDF, HTML, LaTeX, Markdown, reStructuredText, and others. https://nbconvert.readthedocs.io

notebook hidden state: a technical term referring to the value of variables that may have surprising results due to cells having been executed in a non-sequential order.

open source: software and documents that are created in a manner that give you rights to be able to use, and reproduce.

pattern: A "pattern" is a technical term referring to an abstract description of a labeled process. For example, "wash, rinse, repeat" is a common pattern for cleaning various objects.

service, JupyterHub: JupyterHub can take advantage of additional separate, but integrated, software extensions. These are called "services."

script: a colloquial term for a computer program.

unit test: a technical term for a "test" for checking to see if software is operating correctly.

URL (Universal Resource Locator): the address of a resource (e.g., webpage) on the internet.

widget: a user interface (such as buttons, sliders, and checkboxes) that allow the easy control of hidden computer code.