**To Code or Not to Code: Wrong Question! Bringing Jupyter to Non-Coding Analysts, Starting with Myself**

*Amy Gurtler, April 2019*

This is not the inspiring story of a liberal arts major who knew no code, totally taught herself, and is now amazing and being sought out by all the tech startups. My “coding” pretty much consists of some strategic copy/pasting.

Instead, this is the story of a middle ground. It’s not simply “to code or not to code.” I wanted to share this perspective because I know that some analysts—whether intentionally or unintentionally—have gotten the message that they need to learn to code to use Jupyter and/or that they need to learn to code as a general job skill.

But you can use Jupyter to tackle some key workflow challenges without knowing any code. You can also just dabble in code and find a ton of value there.

In this post, I’ll discuss 2 things:

1. Our “Jupyter Tuesday for Analysts” approach which brings relevant Jupyter notebooks to analysts of all skill levels and backgrounds.
2. Some of the specific value I’ve found on the middle ground, as a slightly more savvy copy/paster.

To paint the picture of where I’m coming from and where I know a lot of other analysts are coming from, I’ll tell you about the first time I saw Jupyter. My friend Ben walked me through how to set up a virtual machine, how to go to the notebook gallery, find a notebook, send it to my VM, then run these massive blocks of code. All this mixed in with a “don’t worry, nothing bad will happen.”

I actually said to him, “Ben, no analysts are ever gonna use this!”

There was too much setup, I didn’t understand what I was looking at, where my input should be, where my output would go, and whether I was breaking something in the process. The code just hit me in the face.

Looking back, 3 things helped me break through this:

1. Seeing notebooks that solved major challenges I was having;
2. Recognizing that—not only did these notebooks solve challenges—but they did so in ways that no other tools could do; and
3. My friend, Ben, patiently walking me through this process and giving me some easy and trustworthy notebooks to run.

It was just these simple principles that ultimately inspired me to start holding a monthly event called Jupyter Tuesday for Analysts. I’ve been holding Jupyter Tuesdays once a month for over a year now, so I’ll share the framework I’ve used and continued to refine over time. This is definitely not intended to be a one-size-fits-all approach—every office has different needs—but I hope this framework will spark some concrete ideas for you.

At the end of the day Jupyter Tuesdays are really just brownbag training sessions.

But we’ve probably all been to some less useful or less effective brownbags, and with Jupyter in particular, it can definitely be a challenge to hold a brownbag where you likely have a wide variety of skill sets. You might get the spectrum—from data scientists that are actually writing notebooks themselves to liberal arts majors like myself that never even heard of Jupyter till they showed up at the session.

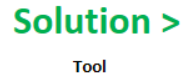
And with Jupyter—probably more so than any other “tool” you could brownbag—there are a TON of different tangents you can go off on, a huge variety of expectations people might have, and a bunch of different outcomes people hope to achieve. So although yes, it’s a brownbag, it can definitely be a challenge to make these sessions as meaningful as possible for a broad audience.

But before even diving into this framework, what problem are we trying to solve or at least mitigate with training sessions in general, and Jupyter Tuesday specifically?

A lot of times we analysts feel like this guy who’s running on a treadmill with his friend dumping legos on it. Our organization tends to be very tool-focused. “Hey, learn about this new tool! It’s so amazing and will solve all your problems!”

But I’m busy. So if you are going to throw a tool at me, tell me:

* What problem does this tool solve? Or what questions does it answer?
* And, how is this tool better than what I’m currently using?

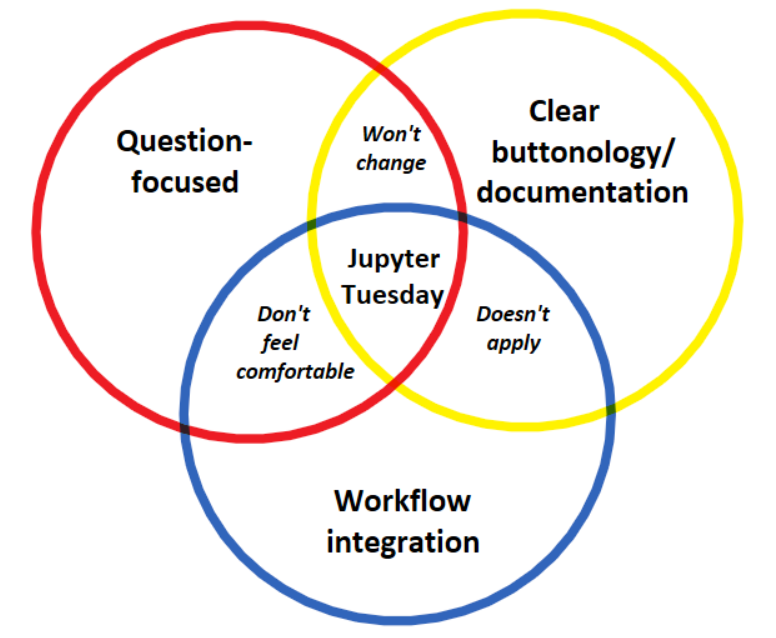


Because even if a tool answers a question I have, I won’t invest time in changing my workflow unless you can tell me how it’s different and the “different” has to be significant.

If you don’t answer these questions, I may not have time to even think about learning some new buttonology, much less integrate it into my workflow.

A lot of discussions on “how to train analysts to use Jupyter” have focused on the code or virtual machine setup as an intimidation factor. That’s where I personally started with my own experience in this post and it’s an important discussion to have. But plenty of analysts are NOT intimidated by some code, or at least are very willing to overcome that barrier when these 2 questions are answered.

So here’s our framework.



**Question-focused**—tell me what questions a notebook answers or what problems it solves.

**Clear buttonology/documentation**—yes, the intimidation can be real. So when I see a notebook that answers questions I have and fits into my workflow, please make it as easy as possible for me to integrate it! I’m not going to remember all the clicks you showed me during a session. So have something quick and easy to refer back to.

**Workflow integration**—tell me how a notebook fits into my workflow—is it better than what I’m currently doing and if so, how?

How are we trying to hit all 3 of these points?

1. **Question-focused**. When creating and advertising Jupyter Tuesday sessions, we start with the question the notebook answers. The session is not “COOLNOTEBOOK demo.” Instead, it’s “How can I do X?” And because we know our follow-on documentation is solid, we try to spend a good portion of the session focusing on the problem we’re solving, rather than the buttonology.
2. **Clear buttonology/documentation**. Here I’m referring a little to what we show during the session, but mostly I’m referring to what we make available (and easy to find!) after the session. In general, I’m not a fan of having a lot of buttonology during sessions because most of us aren’t going to remember all the clicks we saw when we get back to our desks anyways. A little buttonology is good—we want to become somewhat familiar with what we’re about to try ourselves. But the main time I need buttonology is when I’m back at my desk. So, we provide an easy-to-navigate FAQ-style article so people can quickly get set up when they get back to their desks.

In addition to guides on virtual machine setup and cell execution, a key part of what I mean by documentation includes curation of notebooks, including some that are especially basic and user friendly.

We’ve also set up an email help alias. This is used both to answer questions when people get stuck with the buttonology and to solicit recommendations for notebooks. As the notebook gallery continues to grow, curation of mission-relevant and functioning notebooks becomes even more important.

1. **Finally, and perhaps most importantly: workflow integration**. Where Jupyter Tuesday has succeeded, it has largely been because we hit all 3 points in our framework: we gave analysts a solution to actual problems they were having; we gave them clear instructions and easy-to-find/easy-to-follow documentation; and we showed them how a notebook was superior to whatever they were previously doing, thus making it easier to integrate Jupyter into their workflow.

Where Jupyter Tuesday has failed, it’s most often been because we missed this last point—we have not given analysts something that makes sense for their workflow, even if the desire is there, as with this quote from a very forward-leaning analyst:

*“For the most part, I* want *to be able to use Jupyter more frequently, but I often find that I’m not always sure how to manipulate the resulting data, or apply it in my workflow, and/or report based on it.”*

Another tech savvy analyst also touched on this theme:

*“[Failure to adopt Jupyter] is often attributed to an ‘intimidation’ factor…. I personally don’t think this is the case. I believe that, and this is the case with tons of tools/capabilities, that human nature tends to be one of habit.”*

I shared my story at the beginning where I was initially super intimidated. But lots of analysts *aren’t* intimidated and we shouldn’t assume they will be. We need to take the time to demonstrate how a notebook may answer their questions in a way that’s superior to their current processes.

So I’ve walked through the 3 key points for our Jupyter Tuesday overall framework. Now I’ll share 2 more pieces that have to do with the format of the session to make it as broadly user friendly as possible.

**Format: Reverse agenda**. The goal of the reverse agenda is to deliver sessions that are equally relevant to all Jupyter skill levels: from those that regularly use Jupyter to those that have never seen it. We do this by spending the first 30 minutes or so answering the day’s question. What challenge does the notebook-of-the-day solve? What does the output look like and what do I do with it? What is my input?

We then take a quick 30-second break so experienced Jupyter users can duck out.

We spend the remaining 30 minutes or so holding an “Intro to Jupyter.” We discuss what Jupyter is, how to set up a virtual machine, and how to run a notebook. We also discuss a wide variety of notebooks and the various problems they solve. There are 2 overall goals for this session:

1. Help people feel comfortable and ready to go back and try Jupyter
2. Spark some ideas about how Jupyter might help them

**Format: No code discussions allowed**. As I mentioned earlier, one of the challenges with holding a Jupyter session is that there are so many potential tangents. I try to nip this in the bud by stating a blanket rule upfront—no code discussion allowed during the session. But of course, if people do have code questions, we welcome those questions after the session.

What happened when I made this rule was that a few people said, “wait, but I wanna talk about code.” And that’s when Jupyter Thursday for Programmers was born, which I’ll touch on in a bit.

**Has Jupyter Tuesday Been A Success?**

What does it mean to have a successful session? I thought about putting up a nice little chart showing increased Jupyter usage in our organization over time. We love our metrics—that’s how we define success, right?

But I didn’t do that for a couple reasons. 1) Jupyter usage *has* increased, but I’m not confident I could say Jupyter Tuesdays were the reason for that; and 2) Frankly, I don’t care much about tool metrics as an analyst. Sure, they’re part of the story, but they’re far from everything. I’m more interested in qualitative outcomes. Qualitative outcomes are usually harder to define, so I’ll do my best.

Many training sessions or any sort of talks in general are often not as successful as they could be because not enough thought is given to the audience. While preparing, the speaker often focuses on what to cover, rather than what the audience should be able to do. “I need to cover this, I need to make sure to address that….” Especially with Jupyter, there’s a lot to “cover”: programming languages available, all the buttons on the dashboard and their various functionalities, everything that can possibly go wrong, etc.

*But when we step back and think about who might be in our audience and what we hope for them to* do*, it changes everything.* We don’t need to cover so many scenarios or buttons or features.

With this in mind, our overall session goals are not to cover as much as we can about Jupyter. Instead, as I mentioned earlier, we aim to help people feel comfortable trying Jupyter out and to spark ideas about how Jupyter might help them. So if someone comes back later and asks, “hey, is there a notebook that…?”, to me, that’s a success. It means someone sees the value of Jupyter and wants to apply it to his own challenges. I think that *these follow-on informal “notebook consultations” and the creation of a community of people willing to help may be the most meaningful outcome of Jupyter Tuesday.*

I know several analysts that only regularly run 1 or 2 Jupyter notebooks. But those 1 or 2 notebooks have a massive impact on their discovery processes and/or workflow. This is hard to quantify for metrics, for leadership, for whatever. But it matters!

I believe our organization often places too much emphasis on huge, sexy success stories. A lot of the real magic happens in how we tackle our day-to-day challenges. So much of our analyst work can be a straight-up slog, often with no amazing payout—at least not immediately. One analyst noted that she didn’t have any particularly amazing Jupyter success stories to highlight yet, but had this to say about Jupyter Tuesday:

*“It helps highlight the easier, user friendly notebooks that are available with a step-by-step explanation, as well as provides an avenue for asking questions about notebooks that we might be trying to use on our own but are having some issues with. In short, [Jupyter Tuesday] creates a very accessible and enlightening environment.”*

Cool success stories are nice, of course. But I think if we’re helping people optimize their workflows and giving them reference points and a community for their follow-on questions, that’s a success!

Every Jupyter Tuesday session we’ve held has had a mix of both experienced and new Jupyter users. Of course we can never confidently say that we always hit the mark for everyone. But I think our consistent and varied attendance at least somewhat demonstrates the relevance we’re aiming for and the validity of our reverse agenda model.

Other departments throughout our organization have started their own Jupyter Tuesdays. Of course every office has different needs. For example, some of these Jupyter Tuesdays focus more on teaching coding basics. But I think a key outcome of any Jupyter Tuesday should be the creation of a community where people can hopefully find the support they need—whether they just want to run a notebook, or actually learn some code themselves.

**Jupyter Thursday for Programmers**

I mentioned briefly that some people weren’t too happy with my “no coding discussions” rule for Jupyter Tuesday for Analysts, so we ended up creating a separate event—Jupyter Thursday for Programmers. Every third Thursday we get together for either a talk on a programming topic or to hold a code review. This is not as structured as it arguably could be. It’s entirely driven by what the community members want to present. But our community continues to grow and hopefully provides programmers with additional resources and networking opportunities.

**Learning Some Code**

Personally, getting from “I like using Jupyter” to “I want to learn Python” has taken me quite a bit longer. As I started using more Jupyter notebooks (written by others) and chatting with notebook authors, I loved how relatively quickly we could outline an analytic or automation challenge and turn it into a tangible and effective solution. As I started to understand which parts of my workflow could be better programmed or even automated, I began to realize and become impatient with how much of my day was spent doing menial tasks—writing queries, bouncing different tools off each other, and manipulating my data in various ways. Of course not everything could be automated, but many pieces could. I started to recognize that automating some of those chunks would free up time to do more meaningful analysis.

So instead of thinking about my workflow largely in terms of tools, I started thinking about it more in terms of questions and data. What specific data do I need, where is that data, and how do I want to view that data so I can make better judgments?



I was still hesitant to learn any code for a long time, though, because I sort of thought it was an all-or-nothing proposition. Either I drop everything I’m working on to get good enough at Python to actually write useful Jupyter notebooks. Or I continue developing my other work skills and just keep using the cool notebooks that others have written. I knew that I did NOT have the time it would require to get super proficient.

But then someone set up an option to telework just twice a week for a couple hours to do some online code courses, so I figured I’d have nothing to lose. I was able to learn some basics at home, but there are only so many Codecademy for loop exercises you can do before you start to stagnate. What has helped me make more progress than anything else is to just dive into a work project and start asking questions when I get stuck.

Because I had already been using Jupyter for a long time and—as I mentioned—had started thinking about my workflow more “programmatically,” coming up with coding projects has never been an issue. The biggest challenge is to make the time to work on them!

So yeah, ultimately I was wrong—learning to code is not all-or-nothing. I’m now starting to find a valuable middle ground between becoming an expert Python programmer (which will likely never be my goal) and remaining code illiterate.

Now I sort of think of myself as a coding toddler. So, what can a toddler do?

If you were to search on my name in the notebook gallery, you’d see that most of my notebooks are private. This is both because they are fairly basic data manipulation solutions that are somewhat specific to me and also because my code is scary!

Of my public notebooks, you’d see that many are collaborations I’ve done with coders that actually know what they’re doing.

Even with my toddler skills, I’m already finding that coding is:

1. Saving me a ton of time
2. Helping me share tradecraft more effectively
3. Plugging me into a community of smart and innovative problem solvers
4. **Saving time**. I often need to do some specific filtering on both my query inputs and outputs. For example, I want to grab all inputs of a particular type from multiple sources for use in a query. When I get my result set—which is often large—I want to filter it in a few different ways. This is all easy enough stuff to do manually, but starts to eat away quite a bit of time when I have to do it often. With code, I can quickly spell out all these specifics upfront, push a button, and come back to the exact result set I need.
5. **Using and sharing tradecraft more effectively**. Quick story. Last year I’d spent some time refining and sharing tradecraft on how to take advantage of an opportunity that had led me to some very lucrative data. I’d worked hard to try to optimize and better promulgate the tradecraft because despite how lucrative the opportunity can be, I found that so many teams and individuals were not yet aware of it.

After developing what I believed to be the best tradecraft, I began working with other programmers to automate it in Jupyter. For a couple of weeks, we offered to run the automation for teams and deliver them the opportunities and associated tradecraft. During those 2 weeks, we received almost 100 requests from throughout our organization.

Without exception, every request had multiple untapped opportunities. Of course we don’t know how all those opportunities ultimately turned out, but if even just a small percentage turned into a windfall like I’d experienced, I’d say it was a very worthwhile endeavor. From those teams that did follow up, we learned that many opportunities did indeed turn into successes.

This analytic has been run about 2,000 times by hundreds of analysts throughout the organization and has led to several key discoveries. Coding is what enabled me to quickly find and deliver actionable results on a much larger and impactful scale than I ever could’ve done with an article or brownbag training session.

On a much smaller scale, I’ve recently been writing some simple Jupyter notebooks to help with our new Just-in-Time Tradecraft initiative which delivers actionable training and information to analysts within their workflow—in the place and moment they need it. These notebooks help me quickly update and maintain our database of tradecraft recommendations.

1. **Plugging me into a community of smart and innovative problem-solvers**. There are a lot of really great people in our organization that enjoy geeking out over this stuff and sharing innovative ideas. I know—innovation has sort of become a meaningless buzzword these days. Kinda like we learned from the Incredibles, if everything is innovative, nothing is innovative.

But dabbling in code has introduced me to such a wide variety of people with diverse skill sets, working disparate missions through the enterprise. In some cases, it seems that the only thing this community has in common is the desire to make things better—improving our workflows by getting better data, making quicker sense of it, and automating the boring stuff. These exchanges tend to be far more truly innovative than what’s intended by the now-often-meaningless buzzword.

Learning or dabbling in code is not a solo adventure. Coding knowledge is a community. And the better you know some of the common vocabulary, the more closely you can collaborate.

In sum, we analysts can find a tremendous amount of value in Jupyter without knowing any code. By focusing on questions, workflow integration, and clear buttonology/documentation, we can significantly lower barriers-to-entry. In addition, learning code does not have to be all-or-nothing. We can dabble in some of the basics and find a lot of value there.

While I hope to keep expanding my novice coding skills, I realize that becoming a truly awesome programmer is probably not in the cards. I enjoy so many other aspects of being an analyst—using language skills, writing, making judgments from data, and developing and sharing tradecraft. For me, learning some code is not a career shift, but a means to better think about and attack my workflow.