Beyond the Spreadsheet: How to Grow Your Own Data Science Team

Ryan Tanner, PhD Navigant
Vergil Weatherford, PE, Navigant
Thomas Wells, Navigant

ABSTRACT

Energy efficiency consulting is becoming increasingly reliant on effective data management and cutting-edge analytics. These days, data science skills are in high demand but short supply. In this paper, the authors outline a set of guiding principles used to formalize analytics within their own data science team and provide lessons learned throughout the process.

Those principles are threefold. First, it is crucial to put aside spreadsheets and unify the team around a standard set of programming tools such as R, SQL, Python, or other data-focused languages. This means investing in internal training and cloud computing resources so that staff have the technology and skills they need to do their work.

Second, the team needs to be able to communicate effectively. The free flow of information via internal mailing lists and internal social media makes it easier for beginners to get timely help with questions about code. Regular data science team meetings provide an opportunity to show off project work internally and balance workload. The most experienced programmers must lead the way, providing trainings and developing best practices.

The third and final principle is collaboration. Mature data science teams adopt certain development best practices like code version control, code reviews, and reusable code. As the energy industry evolves into a more data-driven business, existing teams need to develop their capacity to work with that data. While no two data science teams look alike, the authors share their experience in developing a strong data science team at a top energy consulting firm.

Introduction

This summer, you have decided to give up on the weekend adventures of cutting your own lawn and trimming your hedges and have decided to hire a professional landscaper. In interviews with two potential landscapers, you notice that one has an old-fashioned push mower and hand clippers and the other has a gas-powered riding mower and gas-powered hedge trimmers. But it turns out they both charge the same rate. Which one would you choose?

Now consider being a utility collecting large amounts of data from smart meters and needing help understanding it. Do you hire the spreadsheet heroes who will try to get it done the old-fashioned way or do you choose a well-equipped data science team with cutting edge infrastructure and analytics tools? Data science used to happen on paper, written in pencil and calculated by a slide rule. Then calculators came and went, and data grew to require computers and eventually servers. Now data science happens in data centers, and data scientists have access to vast computing resources through virtual machines and cloud computing (Conry-Murray 2017). Every industry is scrambling to keep pace with the volumes of data that are only getting bigger. At the end of the day, it is the actual team of people that will make the biggest difference in staying relevant.

The Global Energy Practice at Navigant Consulting, Inc. (Navigant) has been cultivating a data science team with an intentional focus on tools, training, and culture. Although there are many legitimate approaches to modernizing an analytical team, the Navigant team has learned a few things and hopes to contribute to the conversation and help to push the energy efficiency industry forward. What follows is Navigant Energy’s take on getting a data science team to achieve its full potential.

Tooling and Resources

What does a landscaping business have in common with a data science team? In both, the tools available to the employees make a dramatic difference in their ability to get work done. A landscaping company that used only hand tools and old-fashioned push mowers would never be able to compete in today's gas-powered industry. Similarly, an analyst trying to set up a linear regression on a large dataset using only a spreadsheet is bound to get frustrated and run into barriers. The decisions that the landscaping foreman and data science team lead make in picking the tools for their team are fundamental to the success of the operation. This section details some pathways to success when equipping a modern data science team. The first step toward growing an energy data science team is to select the right kind of tools for the job.

Open versus Proprietary

In enterprise environments, proprietary software has long been the mainstay of analytics and database tools. However, the days of $10,000 per seat licenses for statistical software are numbered, and the old, expensive software systems are being supplanted by free or lower cost solutions. With the dramatic rise in popularity of open source software (e.g., R and Python), expensive closed source tools are no longer the best solution. The benefits of community-driven development and support make open source software attractive, especially to academics, small businesses, and startups that cannot always afford huge license fees. Even in larger corporations, open source is making inroads as more vendors offer enterprise-supported open source tools (Hills 2016). This represents a democratization of computing software—a graduate student working on her or his laptop can use the same software used by a Silicon Valley tech giant free of charge. The Global Energy Practice at Navigant uses a mix of proprietary and open source software, but the general trend is toward freely available software in conjunction with enterprise-level support or add-ons where warranted. This is beneficial in three ways.

First, as universities introduce more open source software in undergraduate and graduate-level technical classes, an increasing number of entry-level hires come in with backgrounds in open source programming languages and tools (Guo 2014). On-campus recruiting efforts are far more effective when there is alignment between what is being taught in class and what is needed in the workplace.

Second, the communities that form around open source software provide a rich ecosystem of forums, blogs, and knowledge-sharing sites so that finding help is only a search away. As many open source tools and packages are developed out in the open on sites like GitHub, communicating directly with the developers who are writing code is not only possible but encouraged. This breaks down the walls of typical software support mechanisms such as tickets. It is possible to track the progress of a reported bug or requested feature and see its relative priority in relation to other bugs or suggestions. And if all else fails, individuals are free to download the source code and make the patch themselves, contributing to the development of the software (Singer 2013).

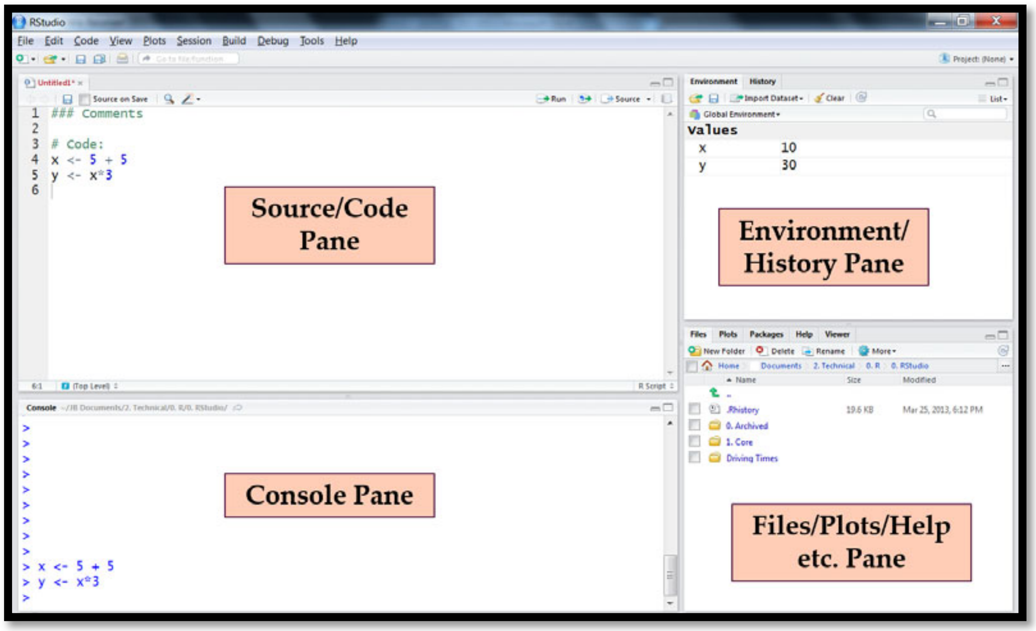
Lastly, open source tools that have flourished in the past 10 years are simply more fun to learn and use. A part of that joy comes from creating code and analyses that can be reused by others, which is easier to do with languages developed in the open. Any programmer can contribute code to the community and improve their reputation and standing in the community. An analysis team can share elegant code solutions and learn from one another to make a better end product. As discussed below, Navigant has found that making tools and collaboration fun leads to increased productivity.

Going Beyond the Spreadsheet

Project leads have a strong incentive to encourage their teams to conduct analyses and develop models using software that is already familiar to everyone on the team. The single analysis tool familiar to nearly everyone in the energy industry is the spreadsheet, so it is a natural starting point in many organizations for doing data work. Spreadsheets certainly have their place and have evolved over time to include charts and graphics, pivot tables, and even scripting/development languages (such as VBA). However, spreadsheets have a number of disadvantages when compared with the flexibility and power of a data-specific programming language. Several high-profile spreadsheet errors in recent years have brought this concern into public focus (Savier 2013).

R and Python are two of the most common programming languages used for data science. Moving from a visual point-and-click environment like a spreadsheet to a text-based integrated development environment (IDE)[1] is a big leap and is not easily made by all members of the team. Learning a programming language from scratch can be a daunting transition, but it is helpful to recognize that some analysts will not only learn, but also thrive using a mature programming language. Figure 1 shows a screenshot of the popular RStudio IDE, used by many in the data science world to develop analyses in the R language.

[1] An IDE is an application that facilitates code writing and software development, and is usually focused on a specific programming language or set of languages. Typical features include a text editor, interactive console, object browser, help documentation, compiler, version control, etc.



```
### Comments

# Code:
x <- 5 + 5
y <- x*3
```

```
Values
 x        10
 y        30
```

```
> x <- 5 + 5
> y <- x*3
>
```

```
Name          Size      Modified
.Rhistory     19.6 KB   Mar 25, 2013, 6:12 PM
0. Archived
1. Core
Driving Times
```

Figure 1 - Screenshot of RStudio Integrated Development Environment (IDE) for the R language

Computing Resources

Just like a spreadsheet has its limitations, so does a single analyst's laptop or workstation. A cloud computing environment is essential for security, scalability, and performance. Cloud environments allow for better resource management (disk space, CPUs, and RAM)[2] and enable standardization of software and files structures. Setting up a data science team to work only from a virtual machine server in the cloud will pay off dividends as dataset size grows—especially if the cloud environment allows for on-the-fly RAM and CPU additions.

Communication and Collaboration

Whether landscaping or data science, one thing arguably more important to success than choosing the right tools for the job is fostering effective and efficient collaboration among team members. In this section specific tools and platforms for communication are discussed along with their implications for a more effective and productive data science team.

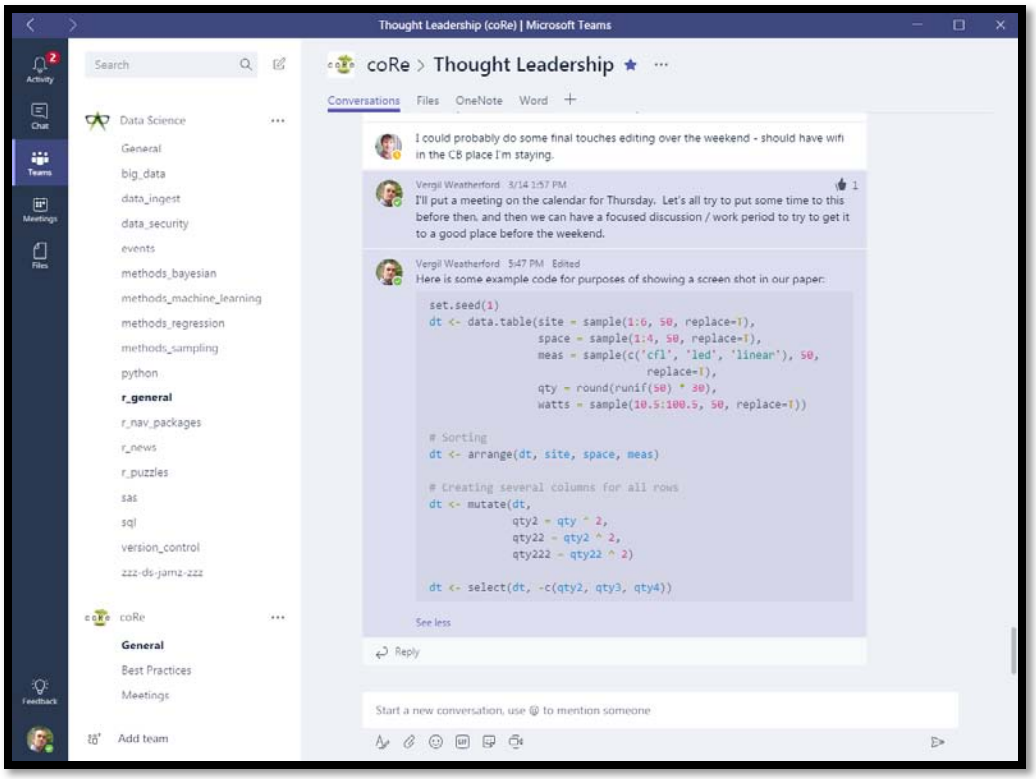
Beyond Email

Just as spreadsheets have, for good reason, long been the go-to tool for data science work, so too has email been the default method of communication. And as with spreadsheets, the Navigant team has found that alternative communication and collaboration platforms exist that better suit team needs. When a data scientist is having difficulty debugging a chunk of code or

[2] RAM stands for Random Access Memory, CPU stands for Central Processing Unit.

connecting to the analytics server, sending an email and waiting 30 minutes for a reply is not an optimal solution. In many cases, someone else on the team has already faced and solved a similar problem and can provide a quicker answer. A solution that could take the struggling analyst an hour or more to find by searching a help manual can often be provided by another team member in a couple quick sentences or a single line of code.

The data science team needs communication platforms on which it is easy to both ask and respond to questions. One requirement for such a platform involves keeping messages brief. Email makes it too easy to write long messages with detailed, multipart questions. Potential respondents to these messages often feel obligated to reply with similarly verbose answers. This slows down the pace of communication and limits the usefulness of the collaboration. By the time a response is received, the answer may no longer be relevant. Platforms like Slack or Microsoft's Teams encourage (but do not force) users to keep messages short via a comparatively small text messaging box that evokes the best parts of a 1990s-era chat room (Backaltis 2017).



I could probably do some final touches editing over the weekend - should have wifi in the CB place I'm staying.

Vergil Weatherford 3/14 1:57 PM

I'll put a meeting on the calendar for Thursday. Let's all try to put some time to this before then. and then we can have a focused discussion / work period to try to get it to a good place before the weekend.

Vergil Weatherford 5:47 PM Edited

Here is some example code for purposes of showing a screen shot in our paper:

```
set.seed(1)
dt <- data.table(site = sample(1:6, 50, replace=T),
                 space = sample(1:4, 50, replace=T),
                 meas = sample(c('cfl', 'led', 'linear'), 50,
                                 replace=T),
                 qty = round(runif(50) * 30),
                 watts = sample(10.5:100.5, 50, replace=T))

# Sorting
dt <- arrange(dt, site, space, meas)

# Creating several columns for all rows
dt <- mutate(dt,
             qty2 = qty ^ 2,
             qty22 = qty2 ^ 2,
             qty222 = qty22 ^ 2)

dt <- select(dt, -c(qty2, qty3, qty4))
```

Figure 2 - Screenshot of Microsoft Teams® chat app

Another advantage these team chat-style platforms have over traditional email is the inclusion of little features that make communication clearer and more fun. For example, Slack allows users to format a section of text as a mono-spaced, color-coded code chunk by simply placing it between leading and trailing back quotes, just as you would a normal quote. This makes it obvious that the obscure noun or verb you placed in the middle of your sentence is an extract from your code rather than a typo. While it is possible to do this in an email message, it requires several mouse clicks and thus is rarely used. Figure 2 above shows a screenshot of a workplace chat app showing teams and topic channels on the left, and code highlighting within chat messages on the right.

Code formatting and other such seemingly trivial features of a communication platform (e.g., image sharing, emojis, gifs, and the ability to like or up-vote good posts) may seem inconsequential to outsiders—such as those making software decisions at a corporate level—but they can mean the world to the day-to-day users. For instance, Navigant's data science team previously relied primarily on email for collaboration and had an internal email list that was actively used. After switching to Slack, the in-line code chunk creation was quickly adopted as standard practice. When the team experimented with a new collaboration platform that lacked the code chunk creation feature, they effectively refused to even consider the new platform until convinced that the feature would soon be available.

Maintaining Focus

One of the often-cited downsides of team chat-style platforms is the potential to overwhelm users with communication, notifications, and information. The flood of messages has the potential to be distracting. In response to this, we offer three suggestions.

First, encourage users to make use of and respect status indicators and notification settings. When in a meeting or focused on a deadline, change your status to "do not disturb." If a channel or chat thread is generating too many notifications, explore temporary or permanent notification settings adjustments for that channel or feed. If all else fails and distraction remains an issue, we encourage users to shut down collaboration applications (yes, even their email) until the need for increased concentration has passed.

Second, it is important to manage teams' expectations around response times and message visibility. It is likely that some messages will go unanswered or that messages will be buried in a thread or conversation before they are noticed by the right person. This is okay. If messages are quick and easy to create, they can easily be created or resurfaced again. Most team chat platforms have features for highlighting messages as important or tagging specific people who might have more of a need to respond than others (Markovitz 2012). In cases where it is vital that certain recipients see or respond to a message, the preferred method of communication might still be an email or a phone call. Setting expectations intentionally and explicitly for communication priority can help prevent frustration.

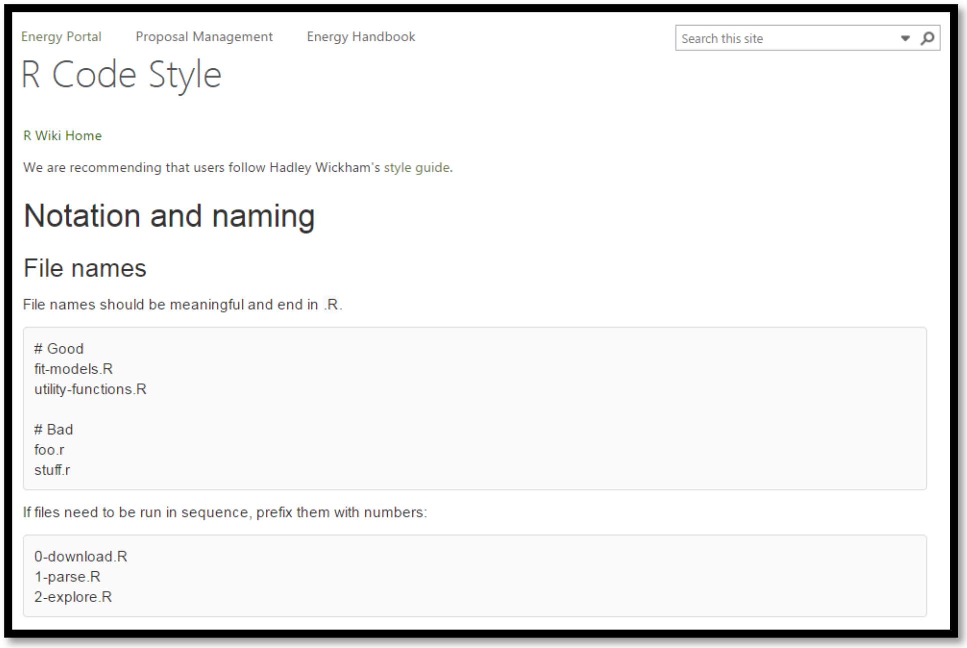
A third suggestion for mitigating the potential downside of abundant communication is to make frequent use of built-in search capabilities. If your team knows that they can easily and quickly find messages that flew across their screen during a meeting or took place while they were away on vacation, they will feel less burdened by the volume of messages. By searching based on a combination or date range, user names, and keywords, most users should be able to quickly review messages or conversations they were not able to join or respond to in real time.

Knowledge Repository

Even with a powerful search feature in our chat platform, the Navigant team has still found it necessary to maintain a knowledge repository in a more static location to supplement the live collaboration space. This is used to house frequently asked questions and answers, best practices, key knowledge shared in the real-time chat, and pointers to topical reference materials. The current platform for Navigant's knowledge repository is a SharePoint-hosted wiki, but there are likely many suitable solutions. The three most important features of this knowledge repository are:

- **Editable by users.** This allows users to update information or correct errors in real time without needing to contact an administrator. It also lessens the maintenance burden placed on the curators.
- **Searchable.** A power search engine with easy to use advanced features (e.g., Boolean operators, wildcards, etc.) needs to be available to ensure user adoption.
- **Regularly maintained.** Knowledge repositories and internal wikis are notorious for containing outdated information. Creating a small team of owner/curators and establishing an easy workflow for capturing information from team chat and emails and adding it to the knowledge repository should mitigate this.

Figure 3 shows a screenshot of a typical knowledge repository article, in this case giving recommendations on R coding style guidelines.



R Code Style

R Wiki Home

We are recommending that users follow Hadley Wickham's style guide.

Notation and naming

File names

File names should be meaningful and end in .R.

```
# Good
fit-models.R
utility-functions.R

# Bad
foo.r
stuff.r
```

If files need to be run in sequence, prefix them with numbers:

```
0-download.R
1-parse.R
2-explore.R
```

Figure 3 - Example knowledge repository article on R coding style conventions

Meetings

Despite the usefulness and many advantages of the virtual, asynchronous collaboration tools, Navigant has found there is still great value in regularly gathering teams for both formal and DIY meetings. Our data science team meets every other week to discuss client projects, resources, and workload. The team takes some time at these top-down meetings to share best practices and new directions, but the majority of innovation and internal optimization happens during monthly meetings of self-organized interest groups. These groups are typically organized around a particular tool (e.g., Microsoft's Power BI) or a specific class of problems (e.g., smart thermostats). These groups work well because of their narrow focus and ability to attract those who are passionate about the topic.

Culture and Best Practices

How often have you tried to pick up someone else's work in a spreadsheet and struggled to understand how their analysis worked or tried to reverse engineer a process that you cannot trace back through hard-coded values or complex formulas? Or have you tried to collaborate on something in a spreadsheet and tried to merge your work with someone else's? Everyone has seen work that is impossible to recreate, quality check, or easily share because it is chained to a spreadsheet file. Moving to more code-based analysis can help with most of these issues—but is subject to some of the same problems if analysts are not following a core set of best practices and embracing good habits. Establishing the right culture takes time. After sowing the seeds initially, it is necessary to continue tending to and reinforcing best practices.

Formal trainings that last 2 days or more can set a baseline for analysts, exposing them to the same tools and methods and giving everyone the same foundational knowledge of how to use the software. Today, things are changing so fast that the beyond-the-basics topics are new on a monthly basis, and teams need to be actively learning on a regular basis to stay sharp. As one person moves up the learning curve on a new function, package, or method, they can give a short training (brown bag seminar) monthly or as needed.

In addition to the knowledge of how to use software and keeping current with the latest trends, data scientists can benefit from following some conventions that the software development community has converged on to make their lives easier. Code version control, functional programming, and code reviews are all means of making code easier to use, share, and improve. These have the added benefit of accelerating knowledge propagation among their practitioners.

Version Control

Version control saves the data scientists from having duplicate files, backups, and date-stamped files. There only ever needs to be one copy of a given file, with a running log of changes to that file, accompanied by notes that indicate what the changes are. With proper version control, this file is backed up locally and remotely, and it can be reverted to any of its prior states at the click of a button. The catch is that version control comes at a price; the data scientist has to get in the habit of entering notes on what she or he has changed and committing the changes to the version control system. In other words, the data scientist is forced to do something she or he should have been doing all along: keep track of her or his work. Or if the data scientist was already keeping notes, version control formalizes the process and puts notes in a standardized system and format that others can easily digest.

Version control also enables easy collaboration. If one person is working on one part of an analysis and a partner is working on an unrelated part of the analysis, they can both work on the same file and merge their work together seamlessly. If they happen to work on the same part of the analysis, the version control system will alert them to the conflicting edits and force resolution. Note that effective communication comes back into play here; version control is never a replacement for good communication.

Think about when a smartphone needs to update an app; the app was in a working state before the update, then got some minor changes from the software developer and continues to work in its new state. This concept of having a production version and patching it or updating it is what version control systems were built to help accomplish. If an ongoing analysis task for a client needs to be tweaked, the analyst can fork[3] their code, create a development branch, add a feature, and fully test and debug it. Then, when it is fully vetted and working, the analyst can merge it into the production code without disturbing the ongoing analysis.

Reusable Code

At the single data scientist level, creating reusable code is as simple as writing functions to complete tasks that are repeated on a regular basis. A best practice to follow is to write a function for anything you have to copy and paste more than 3 times. When copying code, then pasting it and changing the inputs or the variable names over and over, it becomes likely a mistake will occur eventually.

At a data science team level, the same basic rule of thumb applies. If multiple people or teams are doing the same work over and over, they should create a tool and use it on all projects. This leads to consistency in methods and enables everyone to benefit from everyone else's improvements.

In R, formal packages from the Comprehensive R Archive Network (CRAN) are the de-facto place to find reusable code. Teams can also write their own packages, complete with documentation and user guides to make them easy for others to pick up and use. At Navigant, we have gone so far as to create internal packages with functions specifically built for the types of analysis conducted. These packages give the data science team several competitive advantages: updates to methodology are easy to distribute, collaborative improvements are encouraged, and analysis approaches are somewhat standardized.

Code Reviews and Best Practices

[3] In a distributed version control platform like github, a “fork” is a complete clone of someone else’s code which can be modified and potentially shared back to the original author.

In a spreadsheet, one person might boldface headers, while another makes them red; one person might put different chunks of analysis on different sheets, while another might put multiple tables on the same sheet. These stylistic differences can make it easier or harder for a new user to pick up an existing spreadsheet and check it for quality or carry the analysis forward. In the same way, code written according to different conventions can be easier or harder to follow. Luckily, this is another place where the software development world has established great conventions that new users can follow (Cohen 2007). It becomes unnecessary to think about how much to indent code, whether to capitalize a letter, or use an underscore in a variable name. In the same way that document templates guide how to write and style a report or presentation, new software development conventions enable workframes in which the stylistic and structural decisions have already been made. This allows data scientists to focus hard thinking on the actual work rather than on formatting.

Beyond simple formatting, the overall structure and reproducibility of a script are important. The concept of reproducible research means making an analysis script that runs from top to bottom to ensure that every step of the process can be checked and verified by someone else (Peng, 2009). Following a logical sequence of steps (load data, check data for problems, manipulate data, perform calculations, etc.) and including comments to orient a reader make it easy to follow someone else's work (or your own work, months or years later). Code reviews help to build that consistency into the work of the entire data science team. More senior data scientists or those armed with a list of best practices can easily skim through someone else's analysis script and pick out potential sources of error quickly. Navigant has adopted conventions around where to store files (data, scripts, quality control outputs, report outputs, etc.), how to name and order scripts (01-load, 02-clean, 03-plots), which functions to use, and so on. These are all things that make it possible to share work and collaborate on projects. A deeper code review (re-running the whole analysis) is much harder without a consistent and predictable roadmap to follow.

Most of the best practices and conventions outlined in this section constitute basic housekeeping and documentation—which should be a no brainer for any experienced data scientist. The challenge is getting an entire team of people on board and in sync, which requires upfront investments in new analysts through trainings, ongoing investments in documentation and knowledge-sharing, and constant communication and collaboration across the team. Using the carrot and the stick analogy, we find that the most effective method of getting buy-in from team members and management is through the carrot. At the end of the day, the benefits of having reusable, sharable code and tools outweigh the upfront costs of training and well-documented tool development. Once people can see the benefits, they embrace the newer, better tools and wonder how they ever worked in the rows and columns of spreadsheets.

Conclusion

At the end of the day, any team—be it a landscaping team or a data science team—boils down to its individual members, how well the members can work together, and how well prepared they are to meet their objective. In data science, there are some fundamental hardware and software tools that can help each team member work better; means of communication that make it easier to collaborate; and best practices passed down from the software development industry that data scientists can rally around to make their team stronger. Keeping up in a world where data-related work is not only growing but also accelerating is not an easy task, but meaningful investments upfront and on an ongoing basis can help every data science team to stay competitive.

References

Backaltis, V. 2017. "The Rapid Rise of Office Chat Apps." *New York Post*, April 30. http://nypost.com/2017/04/30/the-rapid-rise-of-office-chat-apps/

Cohen, J. 2007. "Four Ways to a Practical Code Review." *Methods and Tools.* Winter 2007. http://www.methodsandtools.com/archive/archive.php?id=66

Conway-Murray, D. 2017. "Cloud Computing By The Numbers – New Interop ITX Cloud Survey." *Packet Pushers*, January 18. http://packetpushers.net/cloud-computing-numbers-new-interop-itx-cloud-survey/

Guo, P. 2014. "Python is Now the Most Popular Introductory Teaching Language at Top U.S. Universities." *Communications of the ACM*, July 7. https://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-u-s-universities/fulltext

Hills, D. 2016. "The New World Order for Open-source and Commercial Software." *TechCrunch*, June 13. https://techcrunch.com/2016/06/13/the-new-world-order-for-open-source-and-commercial-software/

Markovitz, D. 2012. "How to Break Free from E-mail Jail." *Harvard Business Review*, August 27. https://hbr.org/2012/08/how-to-break-free-from-email-j

Peng, R. 2009. "Reproducible research and *Biostatistics*." *Biostatistics* 10 (3): 405-408.

Savier, D. 2013. "What Organizations Can Learn from Recent Spreadsheet Debacles." *Ras and Associates*, August 8. http://rasandassociates.com/ras-explains-what-we-can-learn-from-recent-spreadsheet-debacles/

Singer, L. 2013. *Improving the Adoption of Software Engineering Practices through Persuasive Interventions.* [Place of publication not identified]: Lulu Com.