EPISODE 303

[INTRODUCTION]

[0:00:00.3] JM: Data science is typically done by engineers writing code in Python, R, or another scripting language. Lots of engineers know these languages, and their ecosystems have great library support. But these languages have some issues around deployment, reproducibility, and other areas that we will get into in this episode.

The programming language Golang presents an appealing alternative for data scientist. Daniel Whitenack transitioned from doing most of his data science work in Python to writing code in Go. In this episode, Daniel explains the workflow of a data scientists and discusses why Go is useful. We also talked about the blurry line between data science and data engineering and how Pachyderm is useful for versioning and reproducibility.

Daniel works at Pachyderm, and listeners who are more curious about it can check out the episode I did with Pachyderm founder, Joe Doliner, which is in the show notes for this episode. I really enjoyed speaking with Daniel Whitenack, and I hope you enjoy this episode too.

[SPONSOR MESSAGE]

[0:01:011.3] JM: Dice helps you easily manage your tech career by offering a wide range of job opportunities and tools to help you advance your career. Visit Dice at Support Software Engineering Daily at dice.com/sedaily and check out the new Dice Careers mobile app. This user-friendly app gives you access to new opportunities in new ways. Not only can you browse thousands of tech jobs, but you can now discover what your skills are worth with the Dice Careers Market Value Calculator.

If you're wondering what's next, Dice's brand new career pathing tool helps you understand which roles you can transition to based on your job title, location, and skill set. Dice even identifies gaps in your experience and suggests the skills that you'll need to make a switch. Don't just look for a job, manage your tech career with Dice. Visit the app store and download the Dice Careers app on Android or IOS.

To learn more and support Software Engineering Daily, go to dice.com/sedaily. Thanks to Dice for being a sponsor of Software Engineering daily. We really appreciate it.

[INTERVIEW]

[0:02:25.3] JM: Daniel Whitenack is a data scientist with Pachyderm. Daniel, welcome to Software Engineering Daily.

[0:02:31.3] DW: Thank you, it's great to be here. Thanks for inviting me.

[0:02:34.5] JM: Data science is a term that means different things to different people.

[0:02:40.9] DW: Indeed.

[0:02:42.7] JM: Yeah. We should define a little bit of what we're talking about before we go into Go, and Python and some other languages around data science. In the modern context, in 2017, what does it mean to be a data scientist?

[0:02:56.7] DW: Yeah, you're exactly right that this term gets applied in so many different ways and so many different places. I kind of like to frame this in a couple of different ways. There's the hashtag data science that you hear about on Twitter and other places like machines, playing board games, like Go or maybe self-driving cars and that sort of thing. Then, there's what I would consider practical data science of day-to-day data science, which is really what a lot of people are doing in an industry that are called data scientists.

In a lot of those scenarios, they're not attempting to play board games or make cars drive by them self. A lot of times what they're trying to do is just figure out how to make various processes within a business more data driven.

For example, that could be anything from on the very back end sort of side of things. You might be analyzing loglines to try to predict or give some insight into your back end processes to improve uptime. Or it could be all the way on the other side of the spectrum, like sales, trying to analyze your various channels that you're pouring money into, like your social media, and your website, and blogs, or whatever, to figure out where your customers are coming from, what to pour money into.

Really, what it comes down to is gathering data, doing some sort of analysis on that, that eventually ends up helping people make decisions and helping people make decisions that have some value within a company.

[0:04:36.7] JM: How much of it involves just crunching data in a place that's separate from the actual application, and how much of it is building models that are going into production and processing user requests on the fly?

[0:04:52.7] DW: Yeah. Actually, I would say that even though a lot of visibility is given to building models and kind of predicting things, a lot of the time that a data scientist spends day-to-day, and this is proven out in various polls by Forbes and other people, is actually spent in gathering data, organizing it, parsing it and really preparing your data to be used in a useful way.

That might be aggregating data from a bunch of different sources into a single dataset, or it might be cleaning up your data, formatting it, or filling in missing values or that sort of thing. Really, a lot of time is spent in that organizing phase. Then, outside of that organizing, then you kind of build up some of these other, maybe more sophisticated things including models.

Sometimes it might just even be like calculating a maximum value or account of how many users were on your website. A lot of times, these more simple aggregations and statistics are kind of a — Well, they really should be a precursor to more sophisticated things, and they also provide a lot of value within a company. Remember, we're trying to utilize our data to provide value to the company. I would say that that's kind of the split that I see.

[0:06:18.1] JM: Yeah, I like that you are tempering your presentation of what data science is with, basically, the idea that 90% of what a data scientist does is this data cleaning, this data organization process, and this is a lot of the motivation for why you advocate the use of Go as language for data science.

Before we get into Go, let's talk a little bit about Python, because the most popular language for data science today is Python. At one point in your career you were doing most of your work in Python.

[0:06:52.1] DW: Yup.

[0:06:52.2] JM: What were the ways in which Python was making your life more difficult than you thought necessary as a data scientist?

[0:07:00.2] DW: Well, to start out this discussion, I definitely will say that Python has some really great strengths in this area and that's why a lot of people are using it. I don't want to necessarily bash Python, let's say. Python is really great because it provides like this really convenient way to explore your data interactively, because it is dynamically typed and you can kind of pull things in and it's very easy with goals like Pandas and others to do transformations and interactively explore your data, maybe, in IPython or in a Jupyter Notebook, or whatever it is.

The problem is kind of twofold. First of all, this sort of workflow that's kind of natural in working with that sort of language doesn't necessarily produce — Or it doesn't necessarily lend itself to producing really robust code that has a lot of integrity and maintains itself well overtime and is easy to maintain. Two; it kind of creates a lot of these scenarios where things work well while you're working locally on your local machine with the data that you're familiar with. When you see kind of different looking data, maybe things behave kind of strangely and it's not really that easy to figure out why things are behaving strangely.

When it comes down to being a data scientist who's trying to help people make decisions, if people see that your code is kind of — You don't necessarily know why it's behaving the way it is and don't have an explanation for it, then people lose trust right away with your model. Even if you fix it eventually, you'll likely never regain that trust. It's definitely a really dangerous thing to get into those sorts of situations as data scientist.

[0:08:54.9] JM: We'll talk about the integrity of data science models, machine learning models a little bit later in this conversation, because I know you've written about these things.

[0:09:04.7] DW: Sure.

[0:09:06.0] JM: You mentioned, basically, this "it works on my machine" phenomenon that can happen within Python. This propagates to the deployment process, because you have problems deploying your model. It's going to create frictions between the data scientists and the ops, or the data engineering teams. Describe this phenomenon in more detail. How do the discrepancies between "it works on my machine" and "it works in production", how do those occur and what types of problems do those lead to?

[0:09:45.2] DW: Yeah. I can kind of share at least from my experience what I've noticed. Again, this might be different for different people, but there's kind of two groups of people that I found myself kind of struggling to connect with as a data scientist within a company. The first group is, like you mentioned, the DevOps side of things. With these people, really, the friction comes about that, like, I have my nice environment setup on my local machine with all this kind of data science ecosystem of Python stuff. When it comes to deploying it, first of all, all of that stuff is super heavyweight, and even if I'm putting all of that inside of a Docker container for pushing things onto some machine somewhere, it's still very heavyweight. It's not uncommon to see Docker images for data science stuff be like over two gigs, which kind of ruins the whole portability part of it.

I kind of found myself getting into friction with those guys and then them always saying, like, "Oh, Daniel's got another data thing to deploy. This is going to be fun," sort of experience. Then, the second group, or what I would call like data engineers. These people being the people that are, most of the time, in charge of scalable data pipelines within a company, maybe they're doing some stuff with Kafka and Spark, or something, to really efficiently process events from somewhere in your company. You're basically the consumer of that data and supposed to do something sophisticated with it.

I found myself running into issues with them, because things would be good all on my local machine, and this is really having more to do with scaling, maybe. With Python, things kind of go really well serially, and then when I had to start thinking about processing things on a larger scale, one of two things happen; either it was too complicated for me and I couldn't figure it out and I couldn't get them to figure it out for me, or I figured it out and ended up with a piece of code that was almost entirely unreadable and didn't resemble anything that I saw when I was testing locally, which also introduces more maintainability problems and that sort of thing.

[0:12:05.1] JM: As we are talking about the difference between Python and Go, as we discussed, there are these Python issues where maybe the outcome on the production machine is not the same as your own machine. There's also, perhaps, these problems where you might — If you have a large dataset and maybe there's some missing fields in different places in the dataset, Python may not catch those errors or those missing fields as consistently as something like Go might. Could you explain how the data cleaning process proceeds a little bit more in Python relative to Go and why you find Go to be an appealing language for that data cleaning process?

[0:12:56.0] DW: Sure. A lot of times, in Python, people like to use libraries like Pandas and other things to do their transformations and data cleaning and that sort of thing. For sure, Pandas is an amazingly powerful tool. Again, I'm not casting any shame on it or anything like that, but the experience that I had would be that Pandas is very — It's very smart and that it takes care of a lot of things for you. For example, like you were talking about, taking care of missing fields and inserting ways or maybe doing a join and inferring certain things about how you want to do that.

Even when I was still doing data science in Python and I still write, occasionally, some things in Python. Even when I would do that, I got out of the habit of using Pandas and things like that in production just because those conveniences that you really latch onto when doing exploratory analysis can really bite you when you go into production and you don't necessarily get errors out of a process, but it's not producing the numeral results that you expect. It's really hard to debug that sort of situation where you're not returning anything, but you know something is off or you at least suspect it. That's the scenario that I kind of try to avoid there.

Like you were saying, the scenario is definitely different in Go. For one; Go is statically typed, and because of that, it kind of forces you to deal with some of these conversion issues or whatever it is. Then, also, there's kind of this culture in Go, very much part of the language is handling errors gracefully, and so there's great best practices around that and there's also kind of the language itself kind of forces you to deal with these errors in some way.

You can still choose to ignore them, that's your choice, but it at least forces you to make some decision around that. As a mentor of mine told me, "Error handling code is the main part of the code. It shouldn't be kind of this thing that we brush to the side." I think that continues into the data science field very relevantly.

[0:15:10.5] JM: Yeah. Somebody on Twitter yesterday, just yesterday, said that I should do a show on embracing errors and exceptions, versus handling errors and exceptions. I'm not sure if I understood what the distinction between those two things is, but perhaps it's what you just articulated.

[0:15:32.0] DW: Yeah. I mean, really — And this gets back to, again, putting that in the data science context. When you deploy something and you're depending on someone to make — Or people are depending on what you're producing to make decisions, sometimes, especially when these decisions have very either interesting or dangerous consequences even, right? If you're driving a self-driving car and something goes wrong, there could be very dangerous consequences with that.

When you're doing something that is data driven in that way and people are making decisions based on it, really, this sort of thing should not be an afterthought. You should be able to understand how your data is transformed and the various EduCases that you might experience as you're transforming your data and be able to handle those errors as part of the code. In fact, the main part of the code that will allow you to maintain integrity. Yeah, interesting way to put it.

[SPONSOR BREAK]

[0:16:34.7] JM: You are building a data-intensive application. Maybe it involves data visualization, a recommendation engine, or multiple data sources. These applications often require data warehousing, glue code, lots of iteration, and lots of frustration.

The Exaptive Studio is a rapid application development studio optimized for data projects. It minimizes the code required to build data-rich web applications and maximizes your time spent on your expertise. Go to exaptive.com/sedaily to get a free account today. That's exaptive.com/sedaily.

The Exaptive Studio provides a visual environment for using back end algorithmic and front-end component. Use the open source technologies you already use, but without having to modify the code, unless you want to, of course. Access a k-means clustering algorithm without knowing R, or use complex visualizations even if you don't know D3.

Spend your energy on the part that you know well and less time on the other stuff. Build faster and create better. Go to exaptive.com/sedaily for a free account. Thanks to Exaptive for being a new sponsor of Software Engineering Daily. It's a pleasure to have you onboard as a new sponsor.

[INTERVIEW CONTINUED]

[0:18:06.7] JM: The 90% of the work that is a data scientist's job is, as you've said, this data cleaning, and Go is very useful for this. For this 10%, the Python libraries make it, perhaps, quite easy. You've said that the library support is actually pretty good with Go. It's, maybe, a little more well-developed in the Python ecosystem. Can you talk a little bit more about the experience of actually building models and doing data processing in Go relative to Python and how the library situation looks and go? Perhaps, for those who are less familiar with data science, explain the importance of libraries and what those libraries are accomplishing for a data scientist.

[0:18:57.0] DW: Sure. Yeah, that's some great questions. As you mentioned, this 90%, that's gathering, and organizing, and parsing data. Actually, Go has been doing that well for quite some time and there's great connections to all the major databases, those great matrix libraries and text parsing and all of that sort of thing. Go has been doing this well for a long time along with time series sort of things.

As a data scientist, a lot of times when we're trying to actually crunch some numbers and convert the data that we gathered and organized and cleaned into some sort of insights, we need to be able to do a few more things. First of all, we need to be able to do some statistics. Calculate some statistical measures. Maybe do some histogramming and other things like that. This really gives us an idea about what our data looks like, how it's distributed. It gives us some intuition about what we should expect, what ranges, values are in? What models are valid to use based on their assumptions, and other things like that.

Then, we might want to actually make some predictions using what's often called machine learning. Machine learning is another one of these words that has a lot of baggage and is used differently in different places. For our sake here, let's just consider machine learning a way to fit some sort of model and use that fitted model in memory to make a prediction on something.

For example, we might be getting, like — We have counts of users that are using our website every month, and that might be correlated with our total sales for the month. Maybe we fit a model that compute sales or predict sales for next month based on usage in the previous month, or something like this.

This sort of modeling — Like there's a whole zoo models that are used for different things as far as predicting yes or no, fraud or no fraud, or predicting continuous values, like the sales. There's regression, classification clustering. There's really — This is a whole world.

As you mentioned, Python has a lot of pretty much anything you would want to view. There's a package for it in Python and there's certain accumulations of these functionalities, like scikit-learn that have a whole bunch all in one kind of ecosystem.

Go is making some really great strides in this area, and I'd really encourage people to go take a look at — On GitHub — If you go to github.com/gofords, as in go for data science, /resources, there's a whole listing of tooling. Basically, you can do — It might not have as much as Python, but there's not much you can't do as far as — You can do networks, you can do clustering, you can do regression, time series analysis and a lot of other things. You can even — There's a Go API for TensorFlow and other things.

There's really not a lot you can't do, and when you find something you can't do or maybe it's time to reach out to something, one of these larger frameworks, like H2O or something like that that you can utilize for some sophisticated modeling.

[0:22:25.1] JM: You're talking here about very tangible aspects of Go. The less tangible aspects that make it perhaps harder to be a data scientist using Go that I've seen you comment on is the fact that the community and the centralization of knowledge and the training around common data science tasks in Go is somewhat lacking. This is relative to Python, which has had many years to mature and build a network of people who are very familiar with how data science in Python works. What are the ways in which these hurdles to being a data scientist working with Go manifest in your day-to-day work?

[0:23:13.3] DW: Yeah. I think the story is very much different, let's say, even just a year ago from now. When I was coming in and starting to try a few things in Go, there were a handful of different articles that pointed to some packages. Basically, it was case by case. Every day, I would come in and say, "Okay. I'm doing this today. I've enjoyed using Go for other things. Can I do this in go?" Then I'd go look around and try to find some way to do it. Eventually, maybe figure out a way to do it or maybe hit a wall.

Today, I would say that there's still some of that for people coming in. it might still be a little bit hard to find things. As a community, we're really making an effort to mitigate this in a few different ways. Now, if you search for Go data science, that list of resources should come up, and that's on GitHub, which should provide kind of this picture of the ecosystem.

Also, the Gopher community has a public Slack channel, which is Gopher Slack. You can join it, and there's a data science channel on there where things are constantly being discussed and you can ask, "Hey, can I do this? Can I do that? What's the best way to do this, or that?" There's even a mailing list now.

There's definitely an effort to mitigate this. Yeah, it's still a work in progress. For example, there's no data conference, like there's PIE Data where everybody comes together and discusses things, but maybe someday. Let's hope.

[0:24:46.0] JM: Yeah. Are there benefits from having to define your own workflow, because if Go is a less defined and common path for a data scientist, are there things that you have to figure out for yourself that end up benefitting you?

[0:25:03.1] DW: Yeah, I definitely think there are. I think some of this gets back to — I recently read a set of a document that was basically rules for doing machine learning at Google, which someone at Google had put together. They basically had this statement, “Do machine learning as the engineer you are, not the academic expert that you aren’t.”

I think part of doing data science in Go is realizing that the engineering side of it is really important coming back to the integrity, things that we mentioned. Sometimes, it is well worth the effort to write a simple function to do something small. Let’s say calculate a high squared statistic. That’s not that hard to write that small function. It might be worth you just writing that in your own code rather than pulling in this massive package that does one thousand other things that you aren’t going to use just to remove a dependency and maintain that integrity of your code.

This is definitely a philosophy that is permeating the whole Go community, not just the data science side of things. I think that has a really great benefit for data scientists, that they can learn about some of these things around design philosophy in your code and setting things up in a very readable, maintainable way. Those are things that I really appreciated and learn about interacting with other people that were programming Go.

[0:26:34.4] JM: That quote about you do data science as the engineer that you are, rather than the academic that you want to be. I think that’s really interesting and it gets us towards the conversation around data engineering, perhaps, because I think of data engineering, as it’s defined in some places, is the team that’s setting up the infrastructure that allows the data scientists to work productively. I think that definition might vary.

I think it’s interesting, because if you push responsibilities for creating good code and good engineering practices to the data scientist, perhaps it pushes some of the responsibilities of the data engineer towards the data scientist. Maybe you agree with that or disagree with that or have a comment on that. Main case, I'm curious, where you see the dividing line between data science and data engineering, at least, today in 2017?

[0:27:34.4] DW: Yeah, there's a variety of views on this out there and what follows is definitely my opinion. I tend to agree with your comments about that kind of learning line. I tend to agree with people like Magnusson at Stich Fix who wrote a really great article. I forgot the topic off the top of my mind, but it was something to the effect of data science and ETL, or data scientists should or shouldn't do ETL.

Basically, the thought is that if data scientists have end to end ownership of their project from developing the code to implementing that in some sort of pipeline in a company, into a data pipeline to where it actually fits in that company. Really, that might impose some burden on the data scientist to learn some engineering type things. At the end of the day, it creates a data science team that has a lot more value, because they're producing things that are actually getting deployed and actually producing value within a company, rather than a data science team that's producing a lot of really cool stuff locally and then pushing it into some queue for other engineers to implement in some robust way.

As far as the kind of this line between data engineering and data science, I'm definitely a proponent of kind of mixing the two in the ways that I've just mentioned. You'll see different manifestations of this across industry. At smaller startup sort of companies, the line is already very blurred because you have to wear a lot of different hats and you might be setting up a database one day and writing a predictive model the next day an tying those things together

At much larger companies that have team of maybe 150 data scientists, that can be sort of a more, like I mentioned afterwords, the data scientists are producing this kind of queue of interesting things that are pushed through Java, Scala people to actually implement and run on some data infrastructure. Those people may be are the data engineers. There is kind of this line and it's definitely not a strict line, and I think proponents like me and Jeff Magnusson and other people would kind of want to see the two more and more merged.

[0:29:59.3] JM: Because it sounds like there are — The problems of have this experimental data science team working on building a prototype for a model and then having this other team put it into production, perhaps, in a different language. That sounds just like what we're trying to get rid of in the DevOps movement where develop and writes code, the operations team has to deploy and manage it, and you get the same misaligned incentives for the two parties, and it creates frictions, and it's really not a productive place to be, but sometimes that's the only place you can be if the specializations impose that.

It is interesting to see, perhaps, Docker bridging that gap in both of these, both of these worlds, and both the DevOps world and, perhaps, the data science versus data engineering, or data ops world.

[0:30:52.4] DW: Yeah. Yeah, I agree with all of that.

[0:30:57.0] JM: Since we're getting into talking about Docker — You work at Pachyderm. We did a show a while ago about Pachyderm. This is one of my favorite shows, and the shirt that Joe Doliner gave me after I did that show with him is actually my favorite t-shirt.

[0:31:14.9] DW: Nice. It is a good shirt. I have one too.

[0:31:17.1] JM: You have the maroon one?

[0:31:18.8] DW: I have the green one.

[0:31:20.2] JM: Okay. I have a maroon one. I really like it. It's a great design. Explain what Pachyderm does.

[0:31:25.8] DW: Yeah. We've been talking about how data, in a lot of places, this kind of stuffed from. Data scientists doing things locally to production is really kind of hard. That's exacerbated even further in scenarios where maybe you're running analysis on a big Hadoop cluster and maybe you have 30 people that are managing your Hadoop cluster and your data infrastructure. Really, it's just a completely foreign concept for data scientists, and it's really also hard to maintain even for those people. There are teams where you have 30 people running their Hadoop infrastructure and it still goes down every two weeks.

Really, Pachyderm came out of this sort of struggle on both ends. The goal of Pachyderm is to create a distributive processing framework that is kind of a modern reimagining of Hadoop running on Docker and Kubernetes, but then also kind of gives power over to the data engineers, the data scientists who focus on their analysis and do that in any language, any framework they like and be able to deploy that to this distributed system that is Pachyderm and have that run in a distributed way.

That's kind of like the backstory of it. Like I said, it's a distributed data processing framework that's built on top of Kubernetes. We kind of leveraged this great orchestration logic of Kubernetes to distribute data processing over a cluster.

[SPONSOR BREAK]

[0:33:16.2] JM: Couchbase is a NoSQL database that powers digital businesses. Developers around the world choose Couchbase for its advantages in data model flexibility, elastic scalability, performance, and 24 by 365 availability to develop enterprise web, mobile and IoT applications. The Couchbase platform includes Couchbase, Couchbase Lite, which is the first mobile NoSQL database, and Couchbase Sync Gateway.

Couchbase is designed for global deployments with configurable cross data center replication to increase data locality and availability. Running Couchbase in containers on Docker, Kubernetes, Mesos, or Red Hat OpenShift is easy. At developer.couchbase.com, you could find tutorials on how to build out your Couchbase deployment. All Couchbase products are open source projects.

Couchbase customers include industry leaders, like AOL, Amadeus, AT&T, Cisco, Comcast, Concur, Disney, Dixons, eBay, General Electric, Marriott, Neiman Marcus, PayPal, Ryanair, Rakuten/Viber, Tesco, Verizon, Wells Fargo, as well as hundreds of other household names.

Thanks to Couchbase for being a new sponsor of Software Engineering Daily. We really appreciate it.

[INTERVIEW CONTINUED]

[0:34:43.2] JM: Since we're talking about Docker containers, you wrote a two-part article on containerized data science. I'm sure that could be an entire show in and of itself. Give an overview of why containers are useful to data science? Perhaps, maybe you could touch on how containers are solving problems that are unique to data science. Also, how they are solving those problems that we touched on earlier that seem to mirror the types of problems that DevOps movement is solving with containers?

[0:35:16.4] DW: Yeah. I have some seen containers be like a super powerful tool in data science both in the context of Pachyderm and outside context of Pachyderm. Again, when we're talking about maintain integrity in our data science applications and being able to give power over to data scientists to get them kind of more end to end ownership of their projects, I think Docker plays a huge role in that, because with Docker, basically, you're saying, "Okay. I'm developing this analysis, and now I'm going to pair that with a Docker file which I have ownership of, and I'm saying, "This is how I expect the environment that I expect my application to run in. These are the needs I have," and it really gives that ownership over to them and really helps to make things more affordable."

Again, there's various issues with that in the data science side as far as Docker images being very, very heavy in a Python ecosystem, but there's also ways to deal with that, and that's actually blog article that I have in the works about dealing with that problem. Yeah, I've seen that very powerful in the data science context.

The other side of that reproducibility. There's the deployment story, but there's also the fact that if we're going to call data science a science, it should be kind of reproducible, right? If you're my colleague data scientist, I should be able to say, "Hey, I did this thing. Now, you do it and see if you get the same results and what your thoughts are on it, and let's have that interaction." That's a whole lot easier with Docker. As well with tagging Docker images, you're able to actually gain some historical record of the exact thing that was run overtime in the hopes that you're also able to reproduce some things.

Pachyderm, then, brings the other side of that into the picture, which is data versioning. You kind of need both of those pieces for reproducibility, the reproducibility on code side and the reproducibility on the data side.

[0:37:33.2] JM: The versioning — Why is that important? Why is it so useful to have versioned data sets?

[0:37:39.7] DW: Yes. The simple answer to that is that if you have — Let's say you have the exact code that you ran a week ago to produce a certain result, you have that code and you might know the result that was produced. If you don't know exactly what data was input to that analysis to produce that result, then you don't always have a shot at exactly reproducing that and figuring out what happened, or how to improve it, or incrementally improving your analysis with these sorts of things.

The idea with Pachyderm and data versioning is that we're going to create these complicated analyses. Let's say that our analysis has 15 different transformation, or predictions, or whatever it is, and if we now data versioned the input and output of all of those stages, we basically create this time machine for our analysis that we can have exact reproducibility over what happened, which is great both for debugging and for incrementally improving your analyses.

The other side of that is it actually does have some benefits as far as efficiency and processing. If you compare that to something like Airflow or Luigi that are often used for kind of these pipelining things, those don't necessarily keep you analyses in sync with your data, but because Pachyderm is aware of what data is new and what data has already been there, we can actually update our analyses and keep them in sync with the latest data. Instead of relying on humans to kind of update things as things are changing and whatever, we can have this system that is kind of diff aware and keeps things in sync.

[0:39:31.7] JM: Is the functionality, in terms of using it, is it similar to — We did a show about Airflow. I don't think we've done a show about Luigi, but these are systems for orchestrating data engineering pipelines. It could give a multistage series of things that you want to run on incoming data. You can use Airflow to schedule those different steps against a compute cluster.

Is Pachyderm accomplishing a similar role, and what you're saying is it does that with a versioning component?

[0:40:07.3] DW: Yeah. It definitely provides the pipelining orchestration piece that is, in some ways, solved by these other frameworks.

[0:40:16.3] JM: Kubernetes.

[0:40:17.4] DW: Yeah. It uses Kubernetes to basically perform this orchestration of data pipelining. That is also attacked by Airflow, and Luigi, and these other things. By Pairing that with the data versioning element, there's this idea of keeping your analyses in sync with your data. In addition to that, because it's containerized, as supposed to something like Luigi that might be really tied to Python, you could have data pipelines in Pachyderm that have a stage written by a data engineer who like to work in Java, Skala and wrote some really efficient thing, and that's one stage. Then you could have another stage that was written by a data scientist who wrote it in R and did some really cool statistics, and then you could have another stage that's just like doing some bash commands to organize something.

It's really like this containerized data pipelining allows you to combine — It gives the power to the engineers and the data scientist to say, "What is the best tool that you want to do for these different stages of your pipeline?" Whatever those tools are, distributing that over the cluster.

[0:41:35.2] JM: When you look at Pachyderm, you're a big proponent of it. What does that make you think about how the data science and the data engineering are blurring into one another? The way you're talking about it, it sounds like data versioning is something that I would consider in the per-view of a data scientist, because if you're working on a very specific problem, you're the data scientist. You're responsible for the integrity of the solution that comes out of your model. If debugging that integrity requires you to dive into the versions, which means diving into your Pachyderm containerized solutions, that sounds like more of a data engineering task.

When you are looking at the data science versus data engineering dichotomy from the angle of somebody working closer to Pachyderm, how do you see the roles evolving?

[0:42:32.9] DW: Yeah, it's a really interesting question, and I don't know if I've fully reached an answer to that yet. I think we're still definitely learning some things about how that is becoming blurred by Pachyderm. I would say that some of the philosophies that were kind of C developed naturally is really this philosophy of data scientists having ownership over what they're creating and then being able to create something that is deployed in a distributed way.

It almost kind of partly eases this engineering burden on the data scientist, because, for example, let's say that I'm doing a join in Pandas, in Python Pandas. Normally, if I did that on my laptop and I created two data frames and I was joining those two datasets. That pulls everything into memory, right? Then, I might create a third data frame that's the joined one, which pulls everything into memory. Deploying that and pushing that out into production is going to cost problems at production scale, because you might run out of memory and it might not also be efficient on larger datasets.

With Pachyderm, some of that cognitive overhead of thinking about distributing that sort of processing is taken care of for you, because I could write that Pandas join and then push that up to Pachyderm and then tell Pachyderm to say, "I want to distribute this processing over my datasets A and B, and Pachyderm is smart enough to split those datasets up and provide them to my very naïve Pandas join such that I don't ran out of memory anymore and I'm doing the processing in a distributed manner." Part of the goal here is to allow data scientists to focus more on the analysis, and although they should be aware of engineering things, and I'm always a proponent of that, it's also helping to ease effort.

[0:44:45.1] JM: Absolutely. I want to talk a little bit about managed services, because we've done a lot of shows recently where the conversation on managed services has really came up, and the way that cloud computing manifests right now is mostly we are managing servers, and whether we're looking at those servers from the point of view of a Docker container, or a VM, or our local machine, but it seems like more and more computing is moving into these managed services. From the database point of view, we've got things like Fire-based. From the machine learning point of view, we've got this Google managed machine learning that is apparently really good.

I'm wondering, what are the managed services that a data scientist can leverage today? Where do you see this going? How rich of an operation will a data scientist be able to lean on a managed service to be able to accomplish? How much of the data science of the future will still be rolling our own data science pipeline?

[0:45:53.9] DW: Yeah, it's a really interesting question. The short answer is I think there will always be both pieces in place. One; because we'll always be wanting to do custom things. Two; because a lot of people are cheap, and a lot of companies would rather, for example, maybe deploy Spark in their own infrastructure, rather than using an enterprise solution.

That being said, there's definitely a lot out there that data scientists can leverage, and I often point on newer data scientists to some of those services, because, again data scientist are wanting to produce value within a company. If there's a way — For example, you mentioned Google's services. There's no need for me to write another sentiment analysis algorithm if Google's service for this sentiment analysis, for me, just fine, if my budget allows it and I can implement very quickly. I think that's a great solution.

In other cases, maybe companies don't have the — Smaller startups don't have the budget for funding some of these things. Maybe they would rather bring up a Dockerized version of TenzorFlow, let's say, and get someone's training model and run it in TenzorFlow in their own infrastructure that they have control over and manage.

I think it will always be mixed, but there are great things on either side. I think, now, what Docker — It's also very easy for you to try out all sorts of things locally. I can pull down Dockerized Spark, I can pull down Docker as TenzorFlow and other things and evaluate them. That kind of eases some of the burden that would motivate people to sign up for a managed service to try out certain things without exerting the overhead of deploying it themselves. There's also great utility in that.

[0:47:58.8] JM: There was a show I did a while ago with Chris Fregly who has a company called PipelineIO, and the lesson that I learned from that episode was there are a lot of challenges to getting better online support of our models and online — In this context, means I guess you can stream individual training examples to the models, rather than batch large amounts of training examples. This is pretty important, because you would much rather have a model that can update throughout the day, or throughout the minute, rather than running batch updates. What are the challenges to getting this to work properly? Does what I just articulated reflect your own experience?

[0:48:52.0] DW: Yeah, I think the challenge which was brought up is very valid. I think part of the challenges culturally, or — I don't know — Philosophy-wise. As data scientists, we're very used to — From the various backgrounds we come from, it seems like we very much like pulling down sample data and figuring out what to run on that sample data and then deploying that.

Working with these online models, it's a very different scenario than that, because it's a little bit hard to figure those things out just logistically. Also, there's challenges, potentially, with depending on what algorithm you're using. Then, when you actually deploy things, you also have to deploy this in sort of a streaming manner.

There's definitely a variety of challenges there. What I would probably say in this regard is that regardless of what you're building — I mean, this goes for developers along with data scientists, is that we're developing for production. What we're prototyping, we're prototyping for production. You should have in your mind where this is eventually going to go. In the case of online models, you should have a way to kind of create this sort of environment and be able to test that in a controlled way.

There's an increasing number of frameworks and tooling to deal with these sorts of things, whether that'd be any of the number of streaming frameworks that's part-streaming and other things. Of course, things like Pachyderm, which allow both batch and stream processing. I think part of it is just that kind of mindset that we've developed of doing things the other way, which is kind of preventing us from thinking outside of the box sometimes.

[0:51:01.4] JM: Yeah. Yeah, that makes sense. Really, to something you said a moment ago and just throughout this conversation, we've touched on this question of the machine learning model accountability, and you've written about this in more detail. The current way that we think about machine learning is that these models that we build, they're somewhat imprecise, and if they make an error now and then, it’s not a big deal. Why are you calling for an increase in accountability?

[0:51:31.9] DW: Yeah. There’s two aspects to this as well, or maybe two or more. The ones I like to think about have to do with both the development cycle and then the second one would be related to the people you’re impacting. Regarding the development cycle, there’s this weird sort of scenario in data science where for years and years people have used unit testing and code versioning to kind of wrangle the development cycle in software engineering.

As I’ve mentioned multiple times in our conversation today, often times, that’s not enough for data science, at least in my view, because your code can run as you expect it to run and it can be versioned and it can pass your unit test, but it doesn’t necessarily mean that your analyses are doing better than previous analyses. It doesn’t mean that your analyses are producing relevant results. It doesn’t mean that your model is better than the model before you improved it, that sort of thing.

If you don’t have a way of understanding the relevance of what you’re producing and being able to reproduce that, being able to hold it accountable, as you’re saying, then it really hampers your ability to improve upon what you’re doing, because you have no solid basis to compare to.

Then, outside of the development cycle, now we’re experiencing kind of this influx of machine learning that is actually impacting users in a very drastic way. Examples like self-driving cars, or examples like, “Oh! I got turned down for this insurance policy, because some model told me that I couldn’t.” Those sorts of things are really — They make a huge impact on a user’s life potentially.

Because of that, we’ve started to, and we’re going to see more and more regulation around this user impacting algorithmic decisions. You can see recently regulations put forth by the EU in this regard, and it’s not clear yet how those will be enforced or what their implication are, but it’s definitely a sign that there are going to be at least many industries and many scenarios in which you’re going to have to answer for what your algorithms came up with, and you’re going to have to provide some explanation or some reproducibility or understand what happened in a certain situation, especially if that makes one of these drastic user impacts.

Being able to have that reproducibility and understand what happened is going to be more and more critical as the time goes on, which is, again, one of the reasons why I'm kind of pushing for this and other people are and we're building things like Pachyderm, and we're using Docker, and we're versioning this and that and trying to come up with these design philosophies around maintaining integrity.

[0:54:42.7] JM: I love how you're pulling on your academic background, you have a Ph.D. If people don't know, in academia right now, there's what's called the reproducibility crisis, which is basically the idea that we can't reproduce our scientific papers.

[0:55:00.0] DW: Yeah.

[0:55:00.3] JM: If somebody comes out with a scientific paper then we try to reproduce it and it doesn't happen. This is terrible. This really totally undermines what we consider to be science. I had a long conversation with — Oh gosh! What was the name of that company? Anyways, this robotic cloud lab, if people want to look up the episode. Basically, this company that has a shipping container — Well, not a shipping container size. It's like a small little box that contains a robotic arm and centrifuge and some other things that basically allow you to robotically have experiments done, like wet lab experiments.

This is a type of thing that improves reproducibility in the world of bits, which is very promising. Especially, if you've just seen these types of studies that — Like cancer studies, for example, that can't be reproduced. It's really unnerving, and I don't think we can really even fathom how much this undermines our understanding of the world, when there seems to be problematic, statistical bias, and just whatever you want to account for the fact that we can reproduce anything. That's why I really love your emphasis on the accountability, the reproducibility. I actually love that it is a bleeding of the lines between the academic mindset and the industrial mindset.

In some sense, I kind of like the fact that, in an industry, we don't care what paper you came out with. What we care about are results. That, to me, is what is more inspiring about industry than academia. All credit to academics, and I'm sure there are plenty of academics out there who bulk to what I just said and say that they're going for results as well. In an industry, there's a bottom line to building products that adhere to those results, such as self-driving cars.

[0:57:01.2] DW: Yeah. I definitely like — I appreciate your agreement there, and along with that, I just emphasized that, definitely, we're an industry, we're held accountable within our company for the value that we're producing and hopefully held accountable with our fellow engineers and scientists as far as the quality of what we're producing.

It is hard — I think, maybe, part of the conflict with this in data science is you do have a lot of people coming from academia into data science and from different science background. I, as having a background in science, love to read about the latest and greatest cool stuff, like Google training this algorithm to play the board game of Go. This is super complicated and it's amazing. It really is super amazing.

Then, what I always encourage people and data scientists who are working is we get excited about these things and it's easy to say, "Oh, here's the next problem coming up in my company. Google did this thing where they trained this algorithm to play a board game. That's really powerful. I'm going to use that technology." Then, what I would say, "Are you playing a board game? This is not relevant to your scenario."

Often time, it's kind of — I don't know if I call it sad, but it's maybe not as appealing to say, "Okay, let's calculate a maximum value first," right? Maybe that gives you enough information to make all the decisions in this one area in your company. Guess what? That's super maintainable and runs really efficiently and is easy to deploy.

I always kind of have to give myself that check too because I'm very quick to say, "Oh, this is a really cool thing. I want to use it," but it's not necessarily always fitting for the situation.

[0:59:11.8] JM: All right, Daniel. That sounds like a wrap up. People who are curious — If you're a data scientist out there and you're curious about Go and you're thinking about looking for a language to get started in data science, or to switch to data science if you're already at, perhaps, a Python data scientist, I recommend checking out Daniel's writing on the subject.

There are some really great articles, and we'll link the stuff in the show notes about what you've written about Go and data science.

There's also — I think you were on Go Time, another podcast, and I listen to that one as we're preparing for this. Daniel, thanks for coming on Software Engineering Daily.

[0:59:46.6] DW: Thank you so much, Jeff. It was a great time.

[0:59:49.0] JM: Yes. Great talking to you.

[END OF INTERVIEW]

[0:59:52.0] JM: Listeners have asked how they can support Software Engineering Daily. Please write us a review on iTunes, or share your favorite episodes on Twitter and Facebook. Follow us on Twitter @software_daily, or on our Facebook group, called Software Engineering Daily.

Please join our Slack channel and subscribe to our newsletter at softwareengineeringdaily.com, and you can always e-mail me, jeff@softwareengineeringdaily.com if you're a listener and you want to give your feedback, or your ideas for shows, or your criticism. I'd love to hear from you.

Of course, if you're interested in sponsoring a Software Engineering Daily, please reach out. You can e-mail me at jeff@softwareengineeringdaily.com. Thanks again for listening to this show.

[END]