Technical White Paper

Data Science and the Balanced Team

By Amanda White, VMware Tanzu Labs

vmware®

Table of contents

Author's note

Hunting the bear

I was on a team building a data science model for a mobile product. While there were several directions to pursue, my team was 100 percent focused on improving the accuracy of our predictions based on our goals. We ran many different experiments, analyzed the outputs, agonized over which change should make it into our production code and which should be dropped. Every story we picked up in our backlog, we said, "Maybe this is it. Maybe this is the change that will get us to the results we want." It usually wasn't, but it gave us new ideas of what else to try.

I would go to meetings with the other teams in the portfolio—the ones building the mobile apps that our model would be incorporated into. Their status updates would be, "We just delivered this feature, we are currently working on this other set of features that will help us achieve this outcome, we found a couple bugs that we fixed, and our analytics show that this integration isn't getting much use so we want to reconsider maintaining it."

Mine would be, "We're still working on the same problem." Yes, I could tell them why, what we were doing to solve it, and what approaches we had tried, but from the stakeholder perspective, it seemed like we were spinning our wheels.

After several weeks of this, I felt the need to explain why my status updates were different from those of my colleagues. "The difference," I said, "is that your teams are hiking up a mountain, and my team is hunting a bear."

Please note that I am a vegetarian and an animal lover and would never actually hunt a bear. But for some reason, that's the image that popped into my mind.

"You are looking up at your goal, and there are different ways of getting there and obstacles on the way. But it's a known entity. It has an altitude, a latitude and a longitude. You have to figure out what approach to take—the short, strenuous one; the long, relaxing one; the one with the best views. Now, there might be obstacles—muddy paths, fallen trees, flooded rivers. Or there might be weather conditions you have to factor in. But you have a loose sense of what it's going to take to get up that mountain.

"I'm hunting a bear. I'm on the same mountain as you, but with a different goal. The difference is that I don't know where mine is. I'm looking for it. Yesterday, we saw what might have been bear tracks, and my team went to look around the area, but they couldn't find the bear. Hopefully, if we follow those tracks, we will be led to it, but we may not. In fact, there might not even be a bear. It might have wandered off to another mountain. I think, based on my experience and with the evidence we've collected so far, that there is one, and we should keep looking. However, there's no guarantee of that like there is with the mountaintop.

"Mind you, I have a very clear goal. My equivalent of a bearskin rug—an accurate data science model. I know exactly what I want to do once I find the bear. I just don't know when or where I'm going to find it, or if it's even possible. But my team and I, we're following the clues. We think we're getting close."

Everybody cracked up, picturing me as Elmer Fudd with a double-barreled rifle putting out traps. But I think they got the idea, and at least my status update was more interesting that week.

To the software engineers, designers, product managers, delivery leads, and fellow nature lovers on Software Mountain, this guide will help you step off the trail and start hunting. Our goal is to walk you through the differences between working on a traditional software product and one that incorporates data science, show you how your existing skillsets apply to this type of undertaking, and provide you with some high-level information and data points to get you started. Whether or not you end up finding what you're looking for, this guide will equip you for a safe, productive and fun journey through the woods.

Get ready, my friends. It's data season.

Introduction

This guide is primarily for agile software practitioners (engineers, product managers, UX designers, etc.) who want to learn how to work with data scientists. The goal is to walk you through the differences between working on a traditional software product and one that incorporates data science, show how your existing skillsets apply to this type of undertaking, and provide you with some high-level information and data points to get you started.

This guide keeps in mind the methodologies used at VMware Tanzu Labs™ (i.e., extreme programming [XP], lean product management, user-centered design, and a balanced team) and assumes you have some level of proficiency with these concepts. The focus is on how these practices translate into the data science domains.

Where to start

This guide is written in such a way that you can skip around, picking and choosing the content most relevant to you. The guide uses a through-running example product of a widget that tells a user on a travel site if they should purchase their flight now or wait for prices to go down.

If you're looking for a place to start, the following sections might be of most interest based on your role:

- UX designer – The data scientist persona, User testing for data science
- Software engineer – Testing strategies for data science, Versioning strategies, Pairing in data science
- Product manager – Data science backlog, Story writing for data science, Vision, strategy, goals and anti-goals
- Department lead – Possible product risks, Roles on a data science project, The data science workflow (and where you can help)

A note on terminology

Data science is a broad field, and the term can be used to encompass everything from analytics to artificial intelligence. Throughout this guide, we are using data science as the default term, data scientist as the default role, and data science model as the product built by data scientists. In your specific scenario, you might work with machine learning or artificial intelligence, but for simplicity's sake, we used a broader term.

Data science overview

Data science vs. AI vs. machine learning

You might see terms such as data science, machine learning (ML), and AI used interchangeably. There is a lot of overlap among these three fields, and for the purpose of this guide, the same principles apply more or less equally and are extensible to the others. For the purpose of this guide, data science is used as the default term.

Data science is an umbrella term for making practical use of large data sets, irrespective of whether that use involves ML. It is a broad field that can include everything from analyzing big data sets to data-driven AI.

AI is training software in human-like behavior, such as independent decision-making. Beyond a set of programming techniques, the idea of intelligence represents a visionary ideal of a program that can create and problem-solve as well as or better than a human.

Machine learning is a subset of AI focusing on leveraging data as lessons to train an algorithmic machine (e.g., a machine is learning from lessons) for use in decision support.

Note: Not all AI systems are based on ML or even data science because some AI models, such as rules-based systems derived from domain knowledge, don't involve the analysis of data.

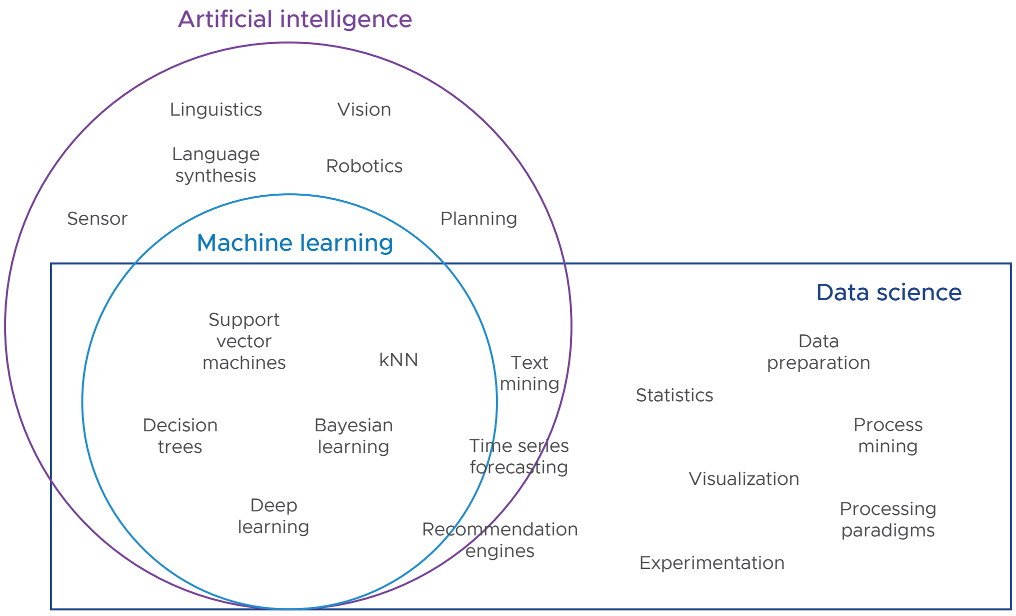


Figure 1: How *Data Science* describes the overlap among AI, machine learning, and data science.[1]

The data science workflow (and where the balanced team can help)

Data science projects follow a typical lifecycle from inception to delivery. It's valuable to understand what this lifecycle is and the value behind each step, especially for trying to help your data science team become more lean and agile. While a time-consuming process can appear like a waterfall to a practiced agilist, understanding the why behind each phase can help you understand where there are opportunities to bring in an agile mindset. **Note:** These steps are iterative, and each can be revisited depending on product needs.

1. Morgan Kaufmann Publishers. *Data Science*. Vijay Kotu, Bala Deshpande. 2019.

Problem formulation

First, you must understand the business/user problem and ways in which it might be approached from a data science or machine learning perspective.

To get involved in problem formulation, anyone on the balanced team can ask:

- Is the proposed problem solving a real business and/or user need? What kind of research can we do to validate?
- What enablement goals does the team have? How do those goals compare to our product goals?
  - For example, are we seemingly committed to a particular technical approach early in the development of our product for reasons that go beyond the product's validated needs?
  - If so, who benefits from trying this approach, and how do we measure success?
- Are we sure data science is the best way to start solving this problem? Is there something cheaper and faster we can do to validate while the data scientists work on the next steps?
- Can we narrow the scope of the problem to an initial minimal viable product (MVP) that we can start learning from?

Data availability

You can't have data science without data, and getting it in the quality and quantity you need can be a major roadblock. It's not only a matter of getting user permissions—the data has to exist.

The balanced team can get involved in the following ways:

- **De-risk by engaging** – Because access to data is a major product risk, get involved in mitigating early.
- **Define the scope** – If you can't access everything you need right away, see if you can narrow the scope of the problem to something you can solve with what you have, rather than get blocked.

Data preparation

Once you have the data, it needs to be transformed into a format you can use. This can involve a lot of cleaning of inconsistent and ill-formatted data, and, depending on the quality of what you collected, can take a long time.

The balanced team can get involved in the following ways:

- **De-risk the product** – Because this can be a drawn-out process, the team should use this time to de-risk other areas of the product.
- **Scope out an MVP** – You can reach a point where the data isn't clean enough to get the quality of results you need for production, but it might be enough to build a proof of concept or MVP for user testing. Work with your data science team to find opportunities to build out a lean slice of the product you can learn from.

If this work starts to overwhelm your team and bottleneck your product development, it might be time for the organization to dedicate resources to data engineering, possibly hiring a data engineer (or training one internally) to turn their full-time attention to the preparation of data.

Exploratory data analysis

Once the data is in a usable state, the data scientists will perform an initial analysis on it, specifically looking for trends that can inform how they can build the predictive model.

The balanced team can get involved in the following ways:

- **Assess results** – Seeing the initial results, do you still feel that this product will achieve your business goals? Will these results make sense to users?
- **Decide whether or not to productize** – Now that you are getting a better idea of what the model will look like, think of what it will mean to productize it. Will there be an API? What will it return? Do you need to productize it at all, or can you use the results of the analysis?

Model training and tuning

The data scientists will now build out the model, train it, and make changes to increase its accuracy and usefulness.

The balanced team can get involved in the following ways:

- **Establish priorities** – Help prioritize what changes are made and what experiments are run. Use a backlog to get alignment on what tasks will best meet the goals of the product.
- **Create action items** – Ensure the analysis the data scientists produce results in clear actions and decisions.

Data evaluation

At each stage of training and tuning, the team needs to evaluate the outputs, glean any insights from analysis, and decide how to proceed.

The balanced team can get involved in the following way:

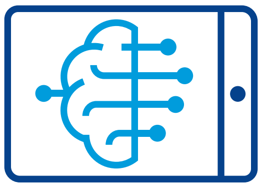
- **Discuss with your team** – Have regular conversations about when to go to production and help your team fight the urge to perfect and gold plate. Keep everyone focused on your goals and anti-goals.

Model deployment

Getting your model into production is an afterthought to many teams, but it can be a complex process that requires forethought.

The balanced team can get involved in the following ways:

- **Assign responsibility of the model** – If you don't have dedicated data engineers or ML operations specialists, the software engineers on the balanced team can be responsible for productionalizing the model or at least pairing with the data scientists on this.
- **Oversee implementation** – Ensure the data science team incorporates practices such as continuous integration and continuous delivery (CI/CD) and versioning.



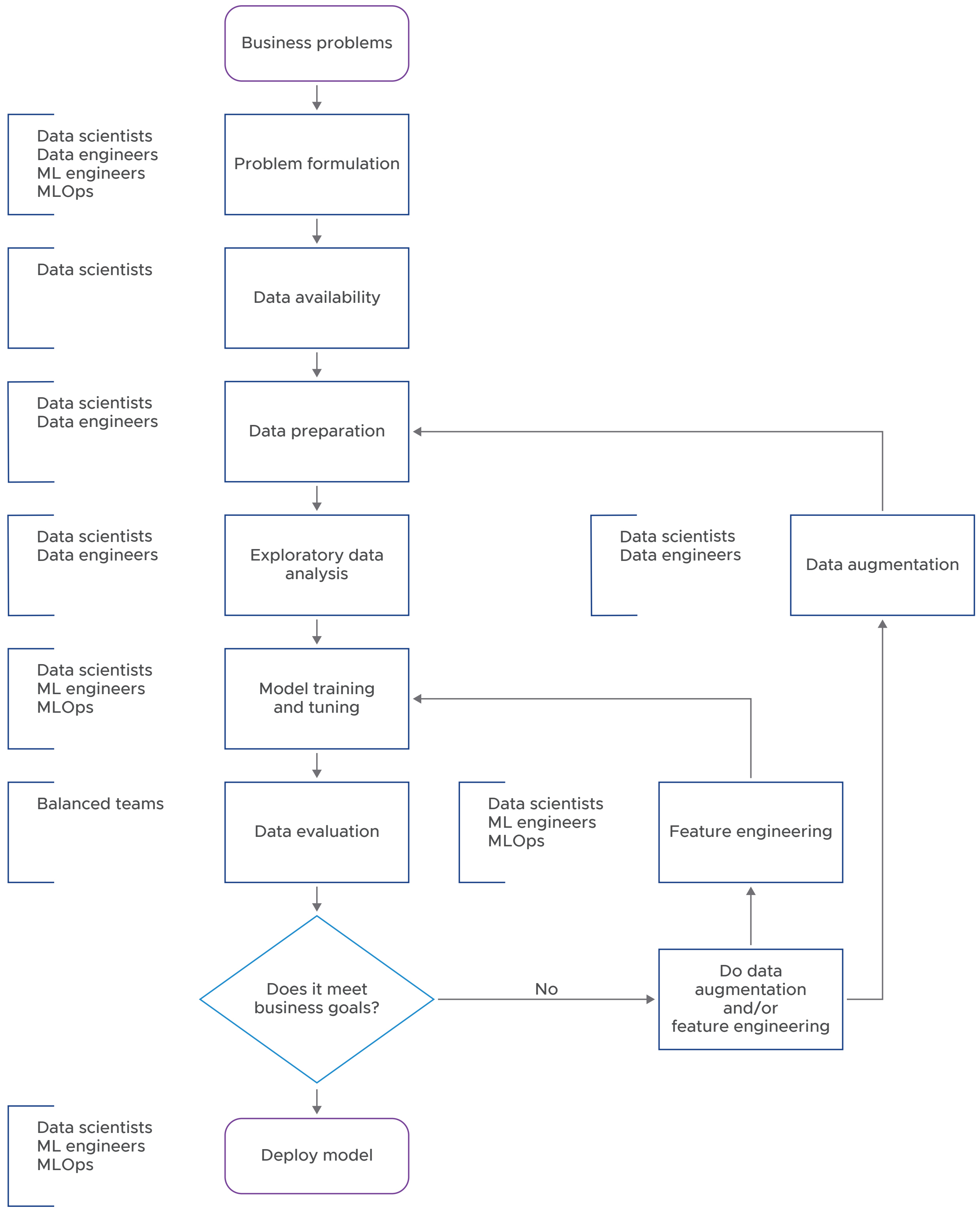


Figure 2: Typical machine learning pipeline.

The data scientist persona

When a data scientist joins your product team, what background and working styles will they bring to the mix? While individual experiences, skills and backgrounds vary, there are some norms you might observe.

More than other roles on a software team, data scientists are likely to be in a second career. They might have a Ph.D. and a background in academics and be drawn to the data science field because of a love of creative problem-solving (and a more stable income than academia often provides). They might even be new to the field. Honestly, you are more likely to find data scientists with less than five years of work experience, although that will surely change over the coming decade as the field grows in popularity.

Data science is a field that requires attention to detail, and practitioners might enjoy working heads-down on difficult problems (pairing is rare). Data scientists often begin a new project by reading and studying to ensure they understand the problem space and potential solutions.

Pain points

Because of the norms previously mentioned, data scientists might feel like they don't fit into the balanced team (or any cross-functional team) at first, especially if the rest of the team doesn't understand the differences in their workflows. This can even mean pushback against agile, lean and XP overall unless a mutual understanding of each other's processes and the reasons behind them is established.

Outside of the immediate team, pain points bubble up to the organizational level where most companies don't have the infrastructure to support their work (e.g., not having the computing resources to run the models or the engineering resources to productionalize them). Their leaders hire them to build moonshots, but resources and attention get diverted to more immediate problems with more obvious solutions, leaving the data science department unsupported.

Needs

One thing to be aware of is that some data scientists, depending on their specialization, are accustomed to a different type of deliverable. They might be used to delivering the analysis that is the output of their model rather than handing off the model itself to be productionalized. This is a shift in perspective as well as process (e.g., they might not be used to checking in and versioning their product code). This doesn't apply to all data scientists, though; an ML engineer, for example, is more accustomed to delivering their model.

Because the work of data scientists is more experimental than that of typical software projects, the timeline of delivery is difficult to project. This means it can be difficult for a data science team to work with a software project team where expectations on delivery timelines are more fixed. The data scientists on your team will need the understanding that their process is different and less predictable. They also might not be used to delivering incomplete results.

What does this mean to your team

Due to the detail-oriented work and a lack of established collaborative practices for data scientists, practitioners can default to working in a silo and deep-diving solutions. However, our experience has shown it's worth finding a middle ground due to the results that come from balanced team collaboration. To reach this compromise, show the value of incremental results and early feedback, and understand the different processes and expectations of the data scientists. Balanced teams have a history of integrating all sorts of individuals into the fold so a project can benefit from multiple perspectives, and working alongside a data scientist shouldn't be any different. It's up to the team to keep an open mind and support their data scientist as they do for anyone on the team.

Balanced team practices for data science products

Roles on a data science project

There are two fundamental ways to staff a data science product. One is to have a **full product team** that builds out an entire user-facing product—both the model and the software it integrates with. The other is to have a **data science model team** that only focuses on the data science model, and that will collaborate with the team building the rest of the software product.

Neither of these approaches is inherently better, they are merely different ways of dividing work (similar to deciding if you want a dedicated team to work on an API or if you want the same product team working on the API and the software that will use it). This will probably depend on the scope of the overall user-facing product.

For example, if you have a large, multifaceted product that has a single data science widget within it, it can make sense to break out a data science model team so the rest of the team can focus on other areas of the product. On the other hand, if your entire product is a front end to use the data science model, it might make more sense to have a full product team.

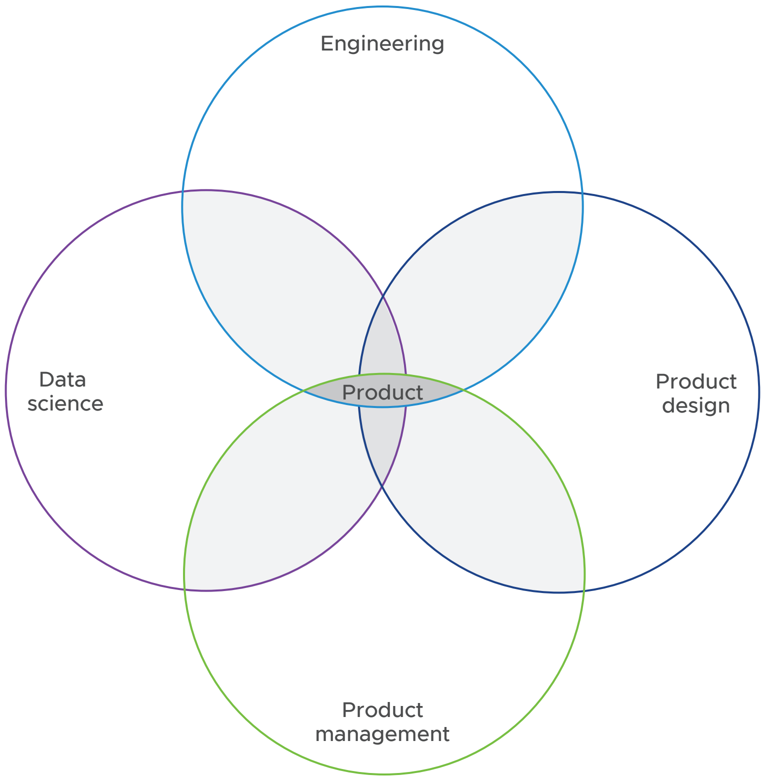


Figure 3: A full product team.

Data scientist

Data science is a different discipline than software engineering. Work on a data science model should be staffed by data scientists, not by software engineers (unless they are also well-versed in the discipline of data science).

Product manager

Having someone dedicated to performing the product manager role on a data science team is critical. Without someone to keep the team focused on product goals, the team is likely to focus on optimizing the model in a vacuum without realizing when it's good enough to start using and getting feedback on. Data science models can also have different goals and anti-goals, and a product manager can help the team stay focused on those and decide what trade-offs should be made.

Designer

Whether you have a full product team or a data science model team, designers should be involved in research and decision-making for your data product. Your product should be based on user needs just like any other or you risk spending months building something that's ultimately not useful or usable.

If you have a data science model team, you will need to negotiate who is responsible for the user-facing designs for the features that use the model. For example, if you are building a tool that predicts flight prices and tells the user whether to buy now or to wait, that widget will be incorporated into a larger page with many other components. Someone will need to decide which team's designers will design the UI for that widget: the team that owns the rest of the page (they have to make sure the widget works within the rest of the page) or the data science model team (they have more context on how users will interact with the feature)? Either way, collaboration is key. Alternatively, you can have a single set of designers working across both teams. In this case, they should make sure they devote sufficient energy to testing the model itself and not only the UI and surrounding features.

Software engineer

If you have a full product team, you will be working with software engineers as you would on any other software product. If you have a data science model team, you might not have fully dedicated software engineers and will possibly need some support to productionalize the model. This can be a software engineer or someone else (see the **Other possible roles** section). It's worth having engineers collaborate with data scientists early on to ensure the outputs of the model are easily consumable. Waiting until you think you are ready to release to production before working with engineering adds technical risk.

Other possible roles

There are some more specific roles you can come across on your project. Depending on your organization, these roles might all be played by the same person or by different individuals. For a smaller organization, there might not be dedicated people for any of this work, and the data scientists and software engineers will have to cover for it.

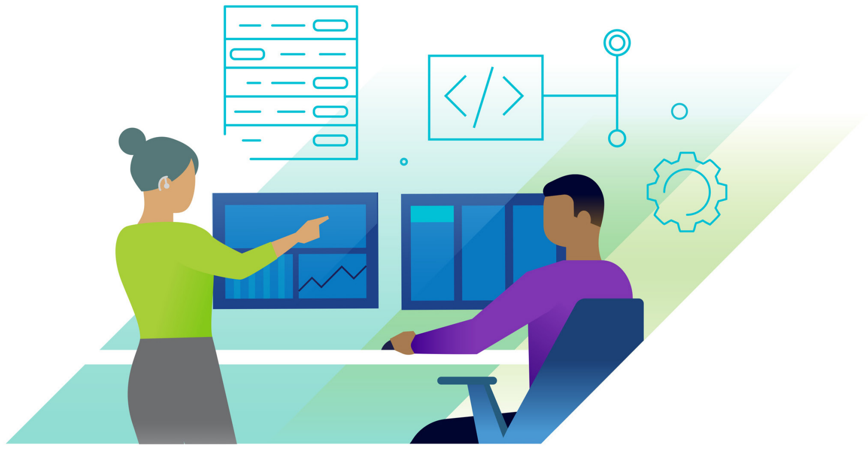
Some organizations have a separate **data engineer** role. This person focuses on the architecture surrounding the model and is indispensable to help get your product into production. If you don't have a data engineer, this type of work can be done by a software engineer or a data scientist, depending on who on your team has the skillset. Data engineers are usually skilled in ETL, Hadoop, and Apache Spark.

Some organizations distinguish between a data scientist who queries and analyzes big data, and a **machine learning engineer** who actually designs ML models. Machine learning engineers are usually skilled in Python and familiar with different libraries for working with supervised and unsupervised models.

ML operations (MLOps) is a machine learning variation on DevOps. As such, an **MLOps engineer** has similar responsibilities to a DevOps engineer but with an emphasis on more data science-specific areas, such as continuous testing (continually retraining the model) and experiment management.

Pairing in data science

Do data scientists pair? Do we pair with them? Do they pair with each other? These are some of the first questions XP practitioners have when joining their first data science project.



Should software engineers pair with data scientists?

Data science and software engineering are distinct disciplines with different skillsets. For that reason, the default pairing configuration is for software engineers to pair with each other rather than with data scientists. There are situations where it makes sense for software engineers to pair with data scientists, but this would mostly be considered cross-discipline pairing, such as a product manager pairing with a product designer or a designer pairing with a software engineer.

That said, there are some areas on the boundaries between data science and software engineering where it can be beneficial to cross-pair, especially if you don't have dedicated data engineers or MLOps engineers to assist with these tasks:

- Working on an API by which the software product can access the data model
- Reimplementing a proof of concept or prototype model as a core product
- Helping with data aggregation or ETL

Areas more aligned with the core skillsets of each discipline, such as data scientists designing a data science model and software engineers working on web development, are less likely to benefit from cross-discipline pairing.

Should data scientists pair with each other?

While data science, like most fields, doesn't have a strong tradition of pair programming, VMware strongly recommends data scientists pairing with each other. In addition to the typical benefits of pair programming—such as better code quality, shared ownership and accountability, and upskilling through knowledge sharing—the context sharing aspect of pair programming is arguably more important than on a typical software project. Algorithms on a data science model can be extremely complex and exceedingly difficult to comprehend without having full context of what was coded and why. Having data scientists pair on this work ensures that more than one person understands each part of the model, which will help your team maintain, iterate on, and troubleshoot it over the lifecycle of the product.

Vision, strategy, goals and anti-goals

Vision and strategy

Vision and strategy on a data science product are no different than they would be on a traditional product (see the Product Manager Playbook for more information). If anything, consciously referring back to your goals can be even more important on a data science project when you are deciding what to implement. Data products can do real harm to people if they are inaccurate or if they are not properly aligned to your goals.

Goals and anti-goals

Because goal clarity is so important, it can be useful to list a few goals and anti-goals. Consider a scenario in which you are building a flight price predictor for an online travel site. The function of your predictor is to let your user (Tina the Traveler) know if they should buy a flight now or wait for the price to go down.

Discussing with your stakeholders, you capture the following goals:

- Increase loyalty to our site (Tina comes to us first because she finds this feature helpful).
- Save Tina money.

You also capture some anti-goals:

- We lose money because Tina doesn't book as many flights (she sees the suggestion to wait and then never comes back and books).
- Tina misses important life events because she waited too long to book.

You can see how some of these points are in tension with each other. You want the prediction model to be accurate enough that Tina trusts it, or else you won't win her loyalty. You also want it to be accurate because your goal is to save her money. But you risk her not booking if the predictor tells her to wait, especially if she looks at a competitor's website and it tells her to buy now. There are different ways of mitigating this, some using data science and some using other approaches.

In the data science model itself, you might play around with the threshold at which it tells Tina to wait. For example, instead of telling her to wait if you calculate a 51 percent confidence rate that the price will go down, you might raise that to a 70 percent confidence rate. Or you might put in a price differential threshold, so you will only tell her to wait if the predicted price decrease is greater than USD $100 or greater than 10 percent of the current price.

For each of these ideas, you need to refer back to the goals and anti-goals. If you raise the bar at which you tell Tina to wait, you risk not meeting the goal of saving her money. If Tina buys a ticket and later the price goes down, you might damage her loyalty. Or perhaps making the sale and having her go on her dream vacation is more valuable to everyone involved than saving her 1 percent on her ticket price.

You can also try resolving these issues outside of the data science model. For example, if the result is "Wait," you can add a link that will search nearby or similar destinations—perhaps Tina doesn't need to go to the Bahamas and would be just as happy with Aruba. Or you can show total seat availability along with the price predictor (e.g., "We expect the price to go down, but there are only 10 seats left that meet your search criteria"). Some analytics and A/B testing can give you a sense of whether or not these features are effective.

The point of this isn't to simply express that product management is hard because these trade-offs need to be decided, but that these conversations might be glossed over entirely if you don't define your goals and anti-goals. Without that, you might spend years making your prediction model as accurate as possible, and once you release, it causes sales to plummet, the feature to be removed, and your team to get laid off.

Another way of framing this scenario: Knowing that your results will never be 100 percent accurate, would you prefer to have **false positives** or **false negatives**? This is a common trade-off in data science models that should be decided in consideration with the product's goals and anti-goals. In this example, a false positive would be telling Tina to buy when the price would have actually gone down if she had waited, and a false negative would be telling Tina to wait when the price was already at its lowest.

Being an ecommerce site that needs sales to make money, you might decide to favor false positives in this case. That would help avoid the anti-goal of "We lose money because Tina doesn't book as many flights." But if you over-rotate to false positives, you will miss the goal of "Save Tina money." On the other hand, if you are working on a model that determines whether or not a convicted criminal should get the death penalty, you will probably have a lower tolerance for false positives and want to err on the side of false negatives.

Roadmaps and risks

Data science roadmaps

Your product roadmap will, for the most part, be similar to other outcome-oriented roadmaps you create for any software project (see Product Manager Playbook for more information). But when first working on a new data science model, there can be a long lead time of preparation and exploration before your product is ready to be launched, or even before it is in the experimental phase. So how do you help your team prioritize what to work on in these early stages? A valuable approach can be to focus on your biggest risks, prioritizing and methodically de-risking them as early as possible to increase your chances of success.

What those risks are will depend on the specifics of your product and the organization you're working in. But there are some common risks for data science projects to factor into your planning.

Possible product risks

Can you get access to the data you need?

The data might exist in silos and have rigid access restrictions. At best, this leads the team spending cycles waiting for access. At worst, it leads to infeasibility of the project. Ensure as early as possible that you can get access to production data.

Do you have the tools and/or skillset to transform the data?

If you get the data and find it unusable, it might need to be transformed. However, you might not have access to the data processing tools and computational power needed, or the skillset to carry out the transformation. Additionally, if this transformation needs to be performed regularly, it will benefit from automation (e.g., as part of a pipeline), which will require additional expertise.

Do you have enough data to get meaningful results from the model?

In most cases, the volume of data available for analysis limits the accuracy of results. If your data set is small, you won't be able to generate useful results, potentially leading to infeasibility of the project. Work with your data scientist teammates to determine your minimum sample size and ensure you can get access to the needed data.

Can you provide enough transparency to your stakeholders so they understand any high-stakes decisions you make?

While all projects carry some assumptions that should be de-risked early, data science models can have high-risk impacts to users and businesses. And, unlike in some other types of projects, your stakeholders and business partners might not have the knowledge about data science models to understand the trade-offs made. It's your responsibility to communicate the decisions your team makes when working on your model to ensure your stakeholders are bought in and not caught by surprise by decisions that might have repercussions.

How will you determine the accuracy of your outputs?

Imagine you're building a model that will predict the effects of climate change hundreds of years in the future. You might produce some interesting results, but how will you verify that your approach is effective? While it's not possible to travel into the future to validate your model, your team should have some sort of plan to determine if you are on the right track or not. This might include comparison with a status quo benchmark or direct feedback from users.

Does your team have the skillset to productionalize your model?

If the code for the data science model needs to run live in production, it needs to be written in such a way that it can run non-interactively, and it needs to be integrated with the production software. Data scientists don't typically have the training to code this part of the system. Your software engineers might be able to help here, or you might need support from a data engineer or another role (see the Other possible roles section). Either way, it's best to plan for this early so you don't get blocked by a model that can't be integrated with your software product.

Do you have enough computing resources for each environment?

Data analysis models typically process large data sets to produce meaningful results. This requires computing (CPU, GPU and RAM) and data storage (typically in TB) resources to be available in each environment. Work with your data scientists to estimate how much will be needed and find out what is available. It's also worth learning about the channels to acquire more processing power: Is there a lengthy procurement process that goes through months of contracting and approvals, or will your manager simply write you a check?

Do you need any third-party licensed software?

There might be a need for third-party software for analysis, visualization and ETL. This can often involve a lengthy approval and procurement process, so it's best to request this early. In some cases, the cost of the software can make the project unviable from a business perspective, so it might be necessary to consider alternative solutions.

Is your training data biased?

It's important to analyze your training data set for possible bias that can lead to harmful real-life consequences. Perform an initial analysis to check for objective bias (such as demographics that don't proportionately represent your target population) or biased content (e.g., historical texts that perpetuate stereotypes common at the time). The impact of any findings will depend on the context of your model. Consider objectively whether any skewed data would have meaningful impacts on the results of your model and look for ways to correct anything that would, or find alternative data sets. Biased data sets are a common reason that models can unintentionally produce harmful results, so these mitigations should be undertaken early and often.

Data science backlog

When starting to work with a data science team, you'll need to figure out how to break their work into individual, valuable stories that can be prioritized and tracked in a backlog.

Separate backlogs

Because data science and software engineering are different disciplines, and people in one role won't be picking up stories intended for the other, the data science team needs a separate backlog from the software engineering team. **Note:** Whether that is an entirely different backlog instance or a separate track of work in a single backlog depends on your backlog management tool.

Starting a backlog

Data scientists often work by themselves and might not be used to sharing their work via a backlog and collaborating on the definition and prioritization of their tasks. But they do have different tasks to work on in their data science model, and those tasks can often be prioritized to meet objectives.

A good way to start building a backlog for your data science teammates is to have a conversation with them about what they are working on now and what tasks they have in mind for the near future. Then, have a conversation about how those tasks should be prioritized and see if you can come to an agreement. At that point, you can put them in the backlog and flesh the stories out with some explanations of what work is being done and why. All you are doing at this point is externalizing the backlog that is already in your team's head for transparency. Once you get that built, you can turn more attention into the story writing, agreeing on a definition of "done," and performing acceptance tests.

Types of stories

On a traditional software project, the types of stories you have include:

- **Features** – Changes to the software that provide value to the user or business
- **Technical tasks** – Also known as chores or enablers; work that helps the team deliver value but doesn't deliver user/business value directly
- **Spikes** – Research tasks completed in preparation to carry out other types of stories
- **Bugs** – Capturing scenarios where the product isn't behaving as expected

You might have other types of stories depending on the configuration of your backlog management tool, but most will fall into these four categories. Data science tasks include another type of work that deviates significantly from these stories in the way they are written and accepted. We refer to these as experiment stories. These capture the various alterations data scientists try to improve the quality of their model.

Story writing for data science

Writing stories for a data science model is different from a traditional software project. Because most work on the model will be an experiment, the traditional format of explaining the user value and the expected results doesn't fit.

Traditional stories

A typical user story has two parts: the story statement explaining the user's perspective, and the acceptance criteria describing how to test the story once it's coded. An example might look like the following.

Story statement:

```
AS an online shopper

I WANT to remove an item from my cart

SO THAT I can change my mind about buying something
```

Acceptance criteria:

```
GIVEN you have items in your cart

WHEN you go to the checkout screen

THEN you see a trash icon next to each item
```

```
WHEN you click on it

THEN the item is removed from your cart

AND the price and any tax are subtracted from your total
```

Story statements told from the user perspective might apply to the overall data science model, but the team has to perform smaller experiments behind the scenes to achieve the goal. For example, consider a story about improving the accuracy of an aspect of the model (e.g., "AS a traveler I WANT to see more accurate price predictions SO THAT I know if I should buy now or wait to get a better price"). The data science team might need to go through several iterations of experimentation to achieve this goal, each of which needs to be discussed, prioritized and evaluated. Because these experiments can be independently tested and prioritized, there is value in calling them out as separate stories.

Acceptance criteria based on inputs and expected outputs don't translate well to working on a data science model because we don't know exactly what the outputs will be. If we knew, we wouldn't need a data science model to figure it out.

Story statements for experiments

When carrying out experiment stories, it can be clearer to write the story from the perspective of the team and describe what you're hoping to get out of the experiment.

Story statement:

```
AS the price prediction team

WE WANT to see what happens if we lower the "hotspot threshold"

SO THAT we can reduce the number of outliers that skew our results too high
```

The reason this story statement is helpful is that it's transparent. Rather than trying to force it into a user-facing narrative, it explains directly what you want to do and why.

Hypothesis statements

To emphasize the scientific aspect of experimentation and data science, it can also be helpful to write a hypothesis statement.

```
WE HYPOTHESIZE THAT lowering the "hotspot threshold" by average price will result in more
accurate predictions.

WE WILL TEST THIS BY running our latest data (v371) set through the model at different thresholds
(e.g., 200%, 175%. 150%, 125%).

WE WILL KNOW WE'RE RIGHT IF the results below 200% are closer to our latest target result set (v19).
We will pay special attention to the set of EMEA routes that we have been having trouble with.
```

(See the lean hypothesis template for a more detailed template.)

Pair with data scientists on writing this section as they should have opinions about how to test and review. There is no need to go into too much detail—the adage "a user story is a placeholder for a conversation" applies here as with any other story. The data scientists on your team can work out the implementation details, similar to how software engineers work out the implementation details of a typical user story.

Acceptance criteria for experiments

The definition of done for an experiment is that the team decides what actions to take based on the result set. Define the decisions you want to make upfront as this can keep the team focused on what to look for and how much confidence they need to be able to move forward.

```
Decisions to be made:
1. Should we lower the hotspot threshold?
2. If so, what should we lower it to?
```

In reality, you might not make any of these decisions; rather, analysis might prompt you to run another experiment. In this example, you might realize the results are better for routes on certain airlines or in certain regions than others, so you can run a subsequent experiment to determine if you want to vary your thresholds. But having the target decision defined from the beginning provides the clarity and focus to keep the team moving forward instead of getting stuck in analysis paralysis.

Accepting experiment stories

Reviewing the results of the experiment and making decisions about how to proceed is an activity where the balanced team working together truly shines. A session involving either the whole team or at least representatives from each role allows everyone to ask questions and make suggestions from their unique vantage points. This helps everyone understand the model better and feel invested in its continued tuning and training. You might even hold a non-binding up-down vote (everyone simultaneously votes thumbs up or thumbs down) on each decision to understand how the team feels about the potential directions.

User testing for data science

Exploratory user research looks similar to that on traditional products: trying to understand the users' needs, what motivates them, and what problems you might be able to solve for them. But it gets a little more interesting when it comes to prototype testing. Designers on a data science team might make prototypes and gather feedback for the model's user interface, but they should also be getting feedback on the model itself.

Even if you have no front end for the model yet, it's critical to have users test the inputs and outputs of the data. You can do this by asking them what they would like to look up results for and having your data science team run the query and tell them the results. This way you can test:

- Are the users looking up what you expected them to?
- Do they understand the results?
- Do they trust the results?
- Do they look right to them?

In the example of the flight price predictor, you can ask Tina to perform a search in the actual production app, then tell her what the price prediction model would say. Tina searches for a flight from Boston to San Juan. Already you might realize that, while you limited your initial scope to the United States, you didn't include Puerto Rico. Or you included it, and you tell Tina that it tells her "Buy," but she responds that she flies this route often and the prices look higher than usual. This can prompt you to dig into why the model is giving that prediction and see if there is a defect. Finally, you can ask her what she would do based on the result, and she might tell you she doesn't trust it: "Of course it says 'buy.' You want my money." This is valuable feedback that can make or break your success. Don't wait until you're in production to get it.

Testing strategies for data science

Testing includes evaluating and verifying that a data science product or application meets expectations in terms of its behavior and output. Some aspects of testing can differ from the way Tanzu Labs is used to operating due to differences in the disciplines of software engineering and data science (see the Data science backlog section). Nevertheless, many Tanzu Labs testing practices can be applied to such projects in familiar ways.

Using test-driven development on a data science project

Test-driven development (TDD) can be used on a data science project for many parts of the system.

The model is often treated as an external dependency fronted by an application that interacts with user input. As such, this application can be developed using TDD similar to any other application. What then for the project components that do not resemble a traditional application?

Apply TDD anywhere the inputs and outputs are verifiable. Beyond considering what can be tested in a traditional application, a data science project might need to broaden the scope of its tests to cover data invariance scenarios. Use TDD to enforce that invariants are held during the app development lifecycle by codifying the range of valid, possible inputs. Store these tests alongside code in a repository and automate them to run every time something changes.

Whether or not you can use TDD on the data science model depends on how the model is being productized.

When integrating a data science model into the application, identify a rational interface for the way(s) the application can make use of it. What are the inputs to the model, and what does it return? Is the application going to be involved in the process of training the model, or will it merely use it? Encapsulating the model is a software engineer's specialization similar to how building the model is a data scientist's specialization.

In many cases, your goal can be to treat the model as a remote dependency. The application proper can be concerned with gathering user input, validating and normalizing it, constructing a query, sending the query to the model for execution, waiting for the result, performing basic interpretation of the result (e.g., column mapping), presenting it to the user, and detecting errors at any step in this process. Each of those steps can be tested using the familiar techniques of ordinary software development.

Here are four types of tests commonly applicable to the data scientist's workflow:

- Computational integrity:
  - These tests check that the logic of the code is consistent with what is expected by the model based on known sample data points used as inputs.
  - These might be most similar to tests written in software engineering.
  - For example, you have a sample data point that you know produces "buy" as output rather than a "wait" prediction in the latest version of the model. Pass when the test asserts "buy" and fail otherwise, using the sample data point as input.
- Directional expectation:
  - These tests check that the model logic is directionally in agreement with your assumptions, business intuition, and/or domain knowledge.
  - These are related to the importance of explainable AI. Note that this type of testing can be more broadly applicable to certain groups of algorithms where the direction effects remain constant (e.g., regression models with fixed coefficients).
  - For example, suppose that domain knowledge and data tell you that purchasing tickets for travel on Christmas Eve is most expensive during the week before Christmas. You have a sample data point where the date of purchase for travel is in May and the model prediction is "buy." You make a small change to this data point by adjusting the date of purchase to the week before Christmas and input it into the model (with the expectation of the model prediction to remain at "buy"). Pass when the model prediction continues to be "buy," and fail if the model prediction switches to "wait."
- Data integrity:
  - These tests check that the input data is in a format expected by the model.
  - This can include tests checking that the data types/formats are consistent with what the model expects, along with tests checking that the range of values in the input data are reasonable.
  - For example, assume that inputs to the "buy vs. wait" model are four columns of data in the following order: day of week, time of day, departing location, destination. Pass when the test asserts that the order of data is correct, and fail otherwise.
- Model drift:
  - These tests check that the model continues to produce outputs at a consistent level of quality.
  - The aim is to know when it would be appropriate for a data scientist retrain a deployed model so it is able to adapt and stay fresh.
  - For example, assume that the "buy vs. wait" model had a 75–85 percent accuracy at the time of deployment. Pass when the test asserts that the average accuracy over the past week of data remains above 70 percent, and fail otherwise.

These types of tests are by no means an exhaustive listing. We encourage your team to look for opportunities to understand parts of a data science workflow that can be turned into software, and then test them the way you would any software.

Beyond TDD

Even when inputs and outputs are not verifiable, testing strategies remain available. Although you might not know exactly what the outputs of your model will be, you might know enough to discern if something is wrong or looks out of place. Put some automated checks in place for errors and warnings, or "red flags" and "yellow flags," to show results that deserve your attention.

Red flags define error cases that tell you something has gone wrong. For the example of the flight price predictor, some red flags might be:

• The predicted price for a flight is USD $0.

• A route that you usually get results for suddenly does not return any results.

• 100 percent of the results appear as "Buy now."

These scenarios can be based on issues with the model that you already fixed and want to ensure aren't reintroduced.

Yellow flags indicate unusual results that should be looked at to see if they are valid or not. For the example of the flight price predictor, some yellow flags might be:

• A change in the predicted price of more than 100 percent from the previous result set

• More than 50 percent of the results that were "Wait" flipping to "Buy now"

• A result of "Wait" when a route is 50 percent of its average price

You can test these by running the same set of sample data through the model to ensure any variation is a result of changes to the model, not the inputs.

Automated red/yellow tests can be written more as fitness functions (as described in Building Evolutionary Architectures) rather than as unit or integration tests. Fitness functions can then be used to evolve the system around the model while defining expectations about the model itself, whether or not inputs and outputs are well-understood. Fitness functions, similar to unit and integration tests, should live in a repository. They can be included as a step in a pipeline or executed regularly in an environment on an interval.

Other considerations

• Testing must be done on a large sample of input and output analysis. You cannot perform limited testing and claim the feature works.

• While supervised models can have deterministic outputs, some unsupervised models might have some randomness built in. In these cases, use sanity checks on a range of values rather than testing for exact values.

• A/B testing, blue/green deployments, and canary deployments are as important for data science products as for traditional products.

• Keep in mind that interactive, web-based notebooks (such as Jupyter) don't support the concept of automated testing and should be used for demo or exploratory purposes.

Versioning strategies

Version control is the creation and management of multiple product releases that all have the same general function but are improved, updated or customized.

Overall approach

Many data scientists are not accustomed to using a version control system. However, versioning is crucial, especially once you start iterating on a model.

Compared to an ordinary software project, you might find that there are more things to check into version control, and they might need to change independently of each other. Having a clear versioning strategy from the beginning will help you.

Sometimes it is valuable for various versions of data science files and artifacts (see the next section) to be used in arbitrary combinations, which means you might want to use multiple repositories. Advanced techniques, such as subrepositories, can be helpful in that case.

Things to potentially version

- **The application consuming the model** – If your model is integrated with a software application, the application will presumably be versioned as would any other.
- **The model** – You might want to set a version for the overall model in addition to the more granular pieces. This can be especially helpful for those consuming the model (e.g., downstream teams) that only want to know the version they are on and don't need more specific information about what is contained in that version.
- **Inputs and data sets** – You can run your model on different sets of data (e.g., a quarterly report that you swap out every three months once the new data set is available). It might be helpful to indicate a version of the data set you are using (for example, DS14 or Q1FY2023).
- **Parameters** – If you are experimenting with setting different parameters, you might want to version each set of parameters (for example, PA12 or Min5Max10).
- **Formulation** – If you are trying different formulations for your model, subsequently or concurrently, you might want to version the formulations. For example, if you are experimenting between two different formulations, you can version them as FM1 and FM2.
- **Results** – If your results are deterministic, it's enough to version the combination of other factors previously listed (e.g., DS14PA12FM2). However, if the results are nondeterministic and you want to compare different sets of results, you can append each result set with its own identifier (e.g., DS14PA12FM2+RS01).
- **Experiments** – You will be running experiments to improve your model, and not all of these experiments will be successful. Use branches for experiments that might not necessarily get merged into the main code base.
- **Your entire system** – Like any modular system, you might find it helpful to roll up certain combinations of your application, model, formulation and so on into a single identifier, such as your system version or product version, or only your version. This is often the best version to show your users. A single, unified version helps you understand the state of your end-to-end system when you talk to your users about their experiences with your product.

Special considerations

A version control repository might not be the best place to store artifacts relevant to end users. An artifact store might be more suitable for things that need to vary at runtime rather than at deploy time. Carefully consider when and to whom versioning is useful.

You might want your data science colleagues to be able to deploy updates to the model independently of the software engineering team. If so, this ability should be part of your CD pipeline.

Some material can be in formats that Git doesn't handle well by default. Preventing problems in this situation can require advanced Git usage. Familiarize yourself with some of the options exposed through gitattributes, such as marking certain file extensions as binary so it won't attempt to merge them.

Knowing when to release

In lean product management, the endeavor is to release software very early on in the product lifecycle so you can learn, de-risking both the viability of the product and any technical hurdles on your path to production. This approach also applies to data products, but more thought might need to be put into what's "ready enough."

On one hand, there can be a tendency among data scientists to keep iterating on a model in a vacuum, striving for an imaginary perfection months or years beyond when the product is viable for the business. On the other hand, a faulty data product can have a much bigger impact than a typical software MVP. A shoddy, early stage software product carries the risks of failing to attract users and damaging the business's reputation—risks easily understood and mitigated (e.g., by releasing a beta to a small group of friendly early adopters instead of in a big bang release). While there are higher risk scenarios (e.g., a fintech app that moves billions of dollars), these aren't typically the first use cases addressed by a new app in beta.

A data science product, on the other hand, can give out blatantly wrong information that risks doing real harm to humans. A healthcare data product might cause someone to misdiagnose and not get critical treatment. A financial model might cause someone to invest poorly and lose their life savings. Sticking a few disclaimers on your user interface might prevent you from legal liability, but it won't prevent human damage, reputational loss, and poor business outcomes.

The tension between these two factors—avoid needless perfectionism but don't release something counterproductive—requires strategic thought around when to push to production. There is no clear cut answer, but there are some lines of questioning the team should prepare.

Questions to ask

If we released this, would we be doing harm to our users?

This is the first question to ask. Although doing good things, and not bad things, for your users should be baked into your product goals and anti-goals, asking this question explicitly will help you think of scenarios to discuss as a team. While you can't guarantee you will never have a negative outcome for your users, considering the possible scenarios before you release is due diligence.

In the case of the flight prediction widget, you might consider if it would be doing harm to Tina to predict that she should buy a ticket but the price drops dramatically the following week. Discussing among the team, you might decide that, while that would be an unfortunate outcome, it wouldn't likely lead to undue financial hardship. Even if the price isn't optimal, she knows how much she is spending (which might not be the case in an app predicting medical costs) and therefore shouldn't have any unexpected burdens due to the prediction alone.

If we released this, would we be meeting our goals and avoiding our anti-goals?

Look at your list of goals and anti-goals, and go through each one. In the example of the flight price predictor, the goals are "Increase loyalty to our site" and "Save Tina money." The anti-goals are "We lose money because Tina doesn't book as many flights" and "Tina misses important life events because she waited too long to book." Let's say you are trying to improve the accuracy of your buy/wait predictions. Currently, they are correct for 60 percent of predictions, and you are trying to decide whether to release as is or continue to work on the accuracy.

Goal: Increase loyalty to our site. A large subset of users will think it's cool and they'll never even find out if the prediction they saw was accurate or not. But if a user got an inaccurate prediction and found out about it, would it decrease loyalty? It can depend on how much the price difference was. You can check numbers and find out that the price differences are only off by 20 percent, so that lowers the risk of a big disaster. Additionally, you see the accuracy is greater for domestic flights than for international ones, so this feature might only be released for domestic flights for the time being to additionally lower risk.

Goal: Save Tina money. You can decide to run through a handful of sample scenarios using the 10 most purchased (not searched for, but actually purchased) routes, either buying now or waiting one week to purchase. In most of these sample cases, the user following the predictions would save money. There are a few cases where they would lose money, but only small amounts. You decide this will meet the goal.

Anti-goal: We lose money because Tina doesn't book as many flights. This might be a risk if Tina sees the "Wait" result. On the other hand, this might be a great opportunity for an A/B test. You can release the feature to a limited set of users and measure sales compared to a control group. If they are significantly lower, you can pull the feature or make changes to try to correct.

Anti-goal: Tina misses important life events because she waits too long to book. This is harder to measure, but the team hypothesizes it is lower risk. You decide to accept the risk for now.

These are largely gut checks and discussions about handling risk, but there are also more objective things you can look at.

What hard metrics do we think we need to meet?

As you refine your model, you will probably be able to nail down more specific acceptance criteria that indicate if you are getting useful results. In this example, a few indicators might be:

- 0 flights are predicted to be USD $0 (flights aren't free, so we know this would be a mistake)
- 75 percent of predictions for the 10 most popular routes are correct
- 60 percent of predictions overall are correct
- 75 percent of incorrect predictions are off by less than 50 percent of cost

Taken as a whole, this combination of gut checks and objective criteria can be a well-rounded approach to deciding when to release and when to wait and keep optimizing.

Further resources

Articles

- Forbes – Three Keys To a Harmonious Relationship Between Data Science And Data Engineering
- Frog – Data Science and Design Thinking Belong Together
- Towards Data Science – How to Work with a Data Scientist

Book

- O'Reilly – Deep Learning for Natural Language Processing

Guidebooks

- IBM – IBM Data Science - Best Practices
- IBM – IBM SPSS Modeler CRISP-DM Guide (data mining)

Infographic

- Illustrated Machine Learning – Machine Learning Engineering: Introduction

Courses

- Coursera – Sequence Models (NLP)
- Coursera – What is Data Science? (IBM)
- Fast.ai – Practical Deep Learning for Coders
- Pluralsight – Data Science: The Big Picture
- Pluralsight – Practical Python for Beginners

Video

- Khan Academy – Statistics and probability

Author

Amanda White

Contributors

Prashanth "PB" Belathur, Cassandra Jaime, Woo Jung, Mark Kollasch, Jon Lorenz, Tyson McNulty, Silvana Moiceanu, Sammota Mwakalobo, Harish Rao, Scott Steele