Course 2 – Machine Learning Data Lifecycle in Production

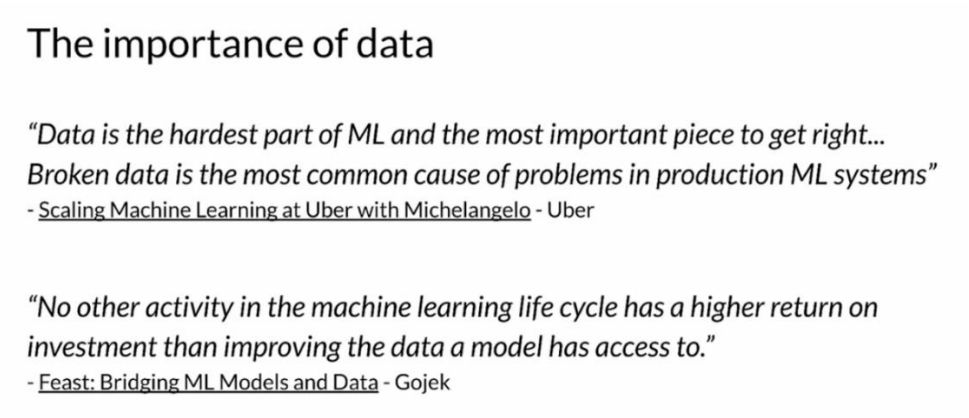
In the second course of Machine Learning Engineering for Production Specialization, you will build data pipelines by gathering, cleaning, and validating datasets and assessing data quality; implement feature engineering, transformation, and selection with TensorFlow Extended and get the most predictive power out of your data; and establish the data lifecycle by leveraging data lineage and provenance metadata tools and follow data evolution with enterprise data schemas.

Understanding machine learning and deep learning concepts is essential, but if you're looking to build an effective AI career, you need production engineering capabilities as well. Machine learning engineering for production combines the foundational concepts of machine learning with the functional expertise of modern software development and engineering roles to help you develop production-ready skills.

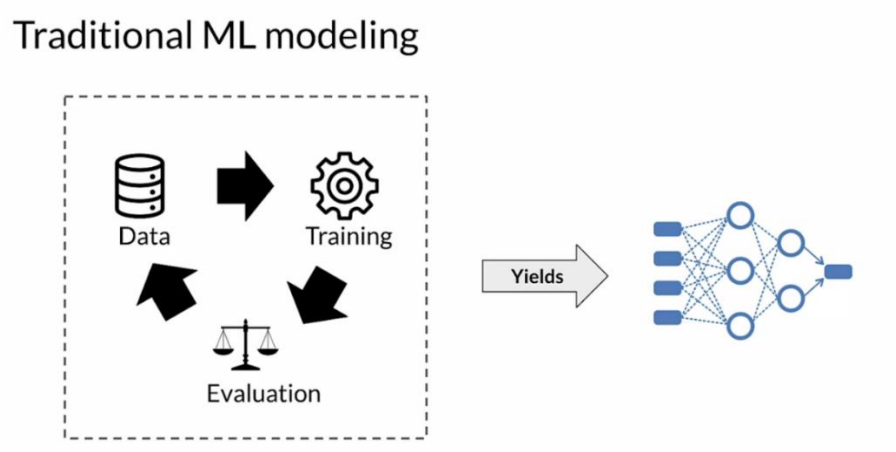
Week 1: Introduction to MLEP

Contents

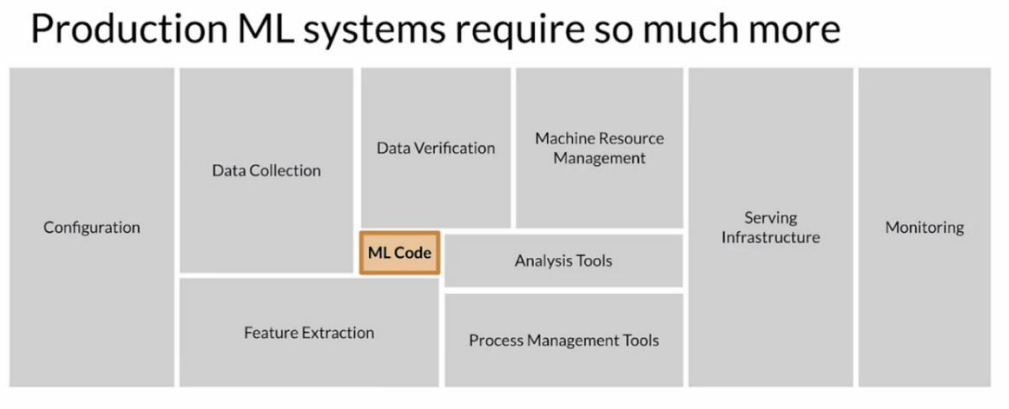
Introduction



- In production environments, you discover some interesting things about the importance of data.
- Here are two quotes from ML practitioners involved in businesses where Data and ML is mission-critical.
- First, from Uber, data is the hardest part of ML and the most important piece to get right. Broken data is the most common cause of problems in production ML Systems
- Next from Gojek, no other activity in the machine learning lifecycle has a higher return on investment than improving the data a model has access to.
- The truth is that if you go to just about any production ML team and ask them about the importance of data, you'll get similar answers.
- So that's why we're talking about data because it's incredibly important to success and the issues for data in production environments are very different from the academic or research environments that you might be familiar with.



- In an academic or research setting, modelling is really fairly simple. Well, maybe not simple, but perhaps less complicated.
- Typically you have some dataset, often standard dataset that are supplied to you and already cleaned and labelled, which you're going to use to train your model and evaluate the results.
- The end result yielded is a model that makes good predictions.
- So you'll probably go through a few iterations to fully optimize the model. Once you're satisfied with the results, then typically you're done.

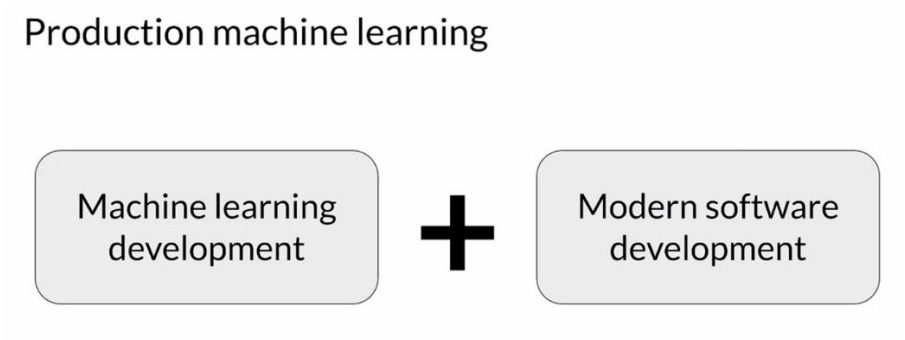


- Production ML requires a lot more than just a model.
- We found that the model is typically about 5% of the code required to put an ML application into production.
- Take a look at all the other boxes that are in this diagram and you'll get some idea about what we're going to be talking about.
- Fundamentally, we're not just talking about machine learning and modelling. We're talking about production ML applications and what it takes to create them, deploy them, maintain them, and improve them so that you can make them available to your users and your business

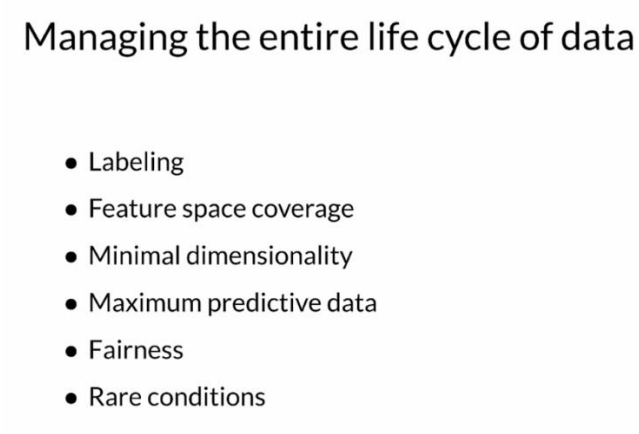
ML modeling vs production ML

	Academic/Research ML	Production ML
Data	Static	Dynamic - Shifting
Priority for design	Highest overall accuracy	Fast inference, good interpretability
Model training	Optimal tuning and training	Continuously assess and retrain
Fairness	Very important	Crucial
Challenge	High accuracy algorithm	Entire system

- Let's compare some of the differences between ML modelling in a research/academic setting and real production ML.
- To start with, in an academic or research environment, you're typically using a static dataset, whereas for production ML, real-world data is used, which is dynamic and usually shifting.
- The design priority for academic or research ML is usually the highest accuracy over the entire training set, but the design priority for production ML is fast inference and good interpretability (and of course, accuracy and cost).
- Model training for research ML is based on a single optimal result, with focus on the tuning and training necessary to achieve it. On the other hand, production ML requires continuous monitoring, assessment, and retraining.
- Interpretability and fairness is important for any ML modelling, but it's absolutely crucial for Production ML.
- Finally, while, the main challenge of academic and research ML is tuning for a high-accuracy model, the challenge for production ML is that plus everything else, i.e. the entire system.

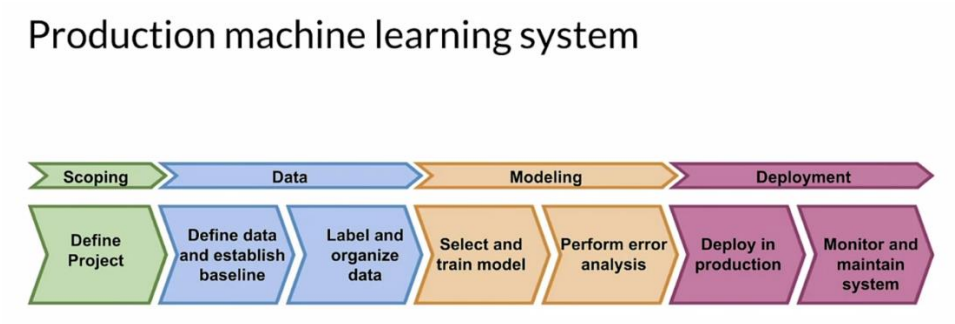


- It would be fair to say that you can look at Production Machine Learning as both machine learning itself and the knowledge and skillset required for modern software development.
- It really requires expertise in both areas to be successful, because you're not just producing a single result, you're developing a product or service that is often a mission-critical part of your offering.

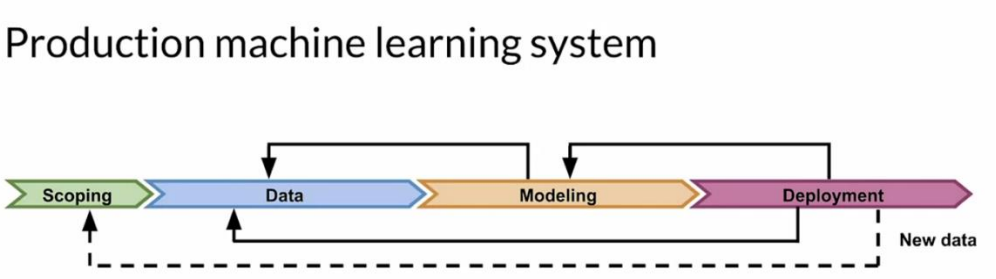


- ML development itself focuses on specific issues related with the data and the quality of predictions.
- For example, assuming that you're doing supervised learning, then you need to make sure that your labels are accurate, and you also need to make sure that your training dataset has examples which cover the same feature space as the request that your model will receive.
- You also want to reduce the dimensionality of your feature vector to optimize your system performance, while retaining or enhancing the predictive information on your data.
- Throughout all of this, you need to consider and measure the fairness of your data and model, especially for rare conditions, for example, in domains such as health care, where rare but important conditions may be absolutely critical to success.

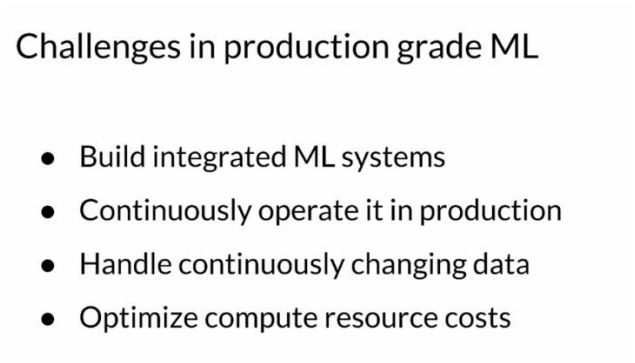
- On top of all of that, you're putting a piece of software into production that requires a system design, and that includes all of the things that are required for any production software deployment.
- Of course, this deployment has to be focused on ML and your application.
- Is your system scalable? Can you scale it both up and down? Can you extend it cleanly to add new stuff when you need to? Does it have a clear, well-defined configuration? Is it consistent? Can you reliably reproduce results? Is it hardened against attacks? Is the design modular, and does it follow modern software development principles?
- Can you test individual units? Can you do end-to-end testing? Can you continuously monitor the health and performance of your system and be alerted when there are problems? Have you adopted industry best practices?



- Using a model in real-world applications requires much more than just an understanding of machine learning algorithms.
- The first step is scoping, which focuses on defining the project needs and goals and the resources required to achieve them.
- Next, you start working on your data, which includes defining the features that you're going to use, as well as organizing and labeling your data. That may sometimes include measuring human level performance to set a baseline for comparison.
- Then you design and train a model. In this phase, error analysis will help you refine your model to suit your project's needs. After training your model, you deploy it so that it can be used to serve prediction requests.
- You might deploy your model on mobile devices, on a Cloud, or in IoT devices, or even in a web browser.
- Over time, real-world data continuously changes, which can result in a degradation of your model performance.
- You need to continuously monitor the performance of your model, and if you measure a drop in performance, you need to go back to model retraining and tuning, or revise your data.

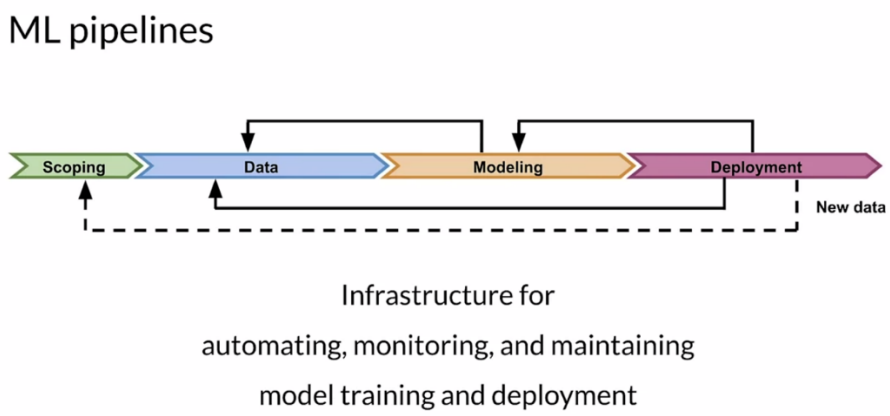


- During deployment, new data may affect your project design either positively or negatively and risk coping might be necessary.
- Ultimately, all these steps create your production ML system, which needs to run automatically such that you're continuously monitoring your model performance, ingesting new data, and retraining as needed, and then redeploying to maintain or improve your performance.

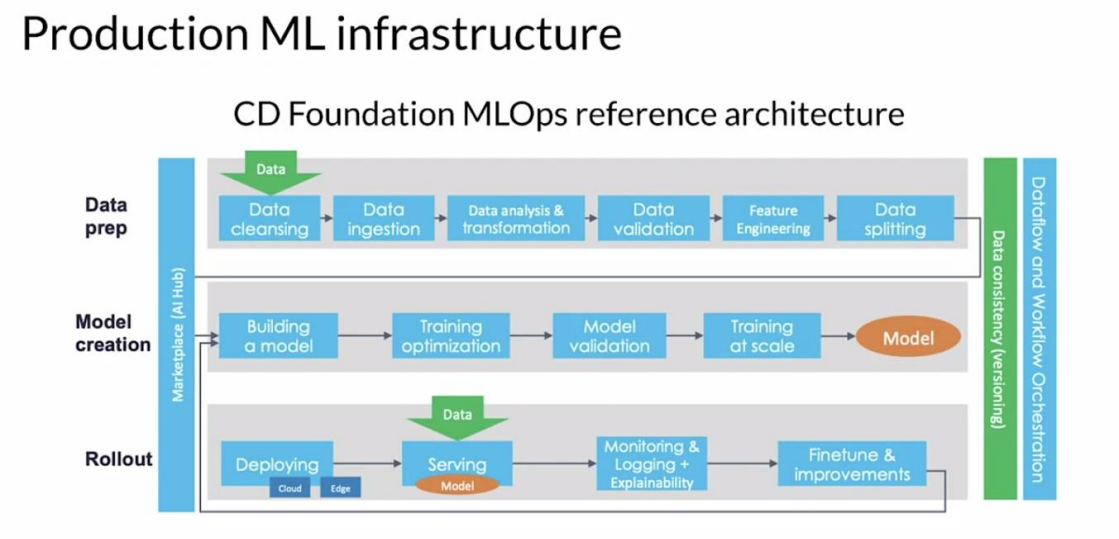


- The challenges when doing production ML are very different than academic or research ML, or in some sense they're the same but include a lot more.
- You're going to be building an integrated system specifically focused on ML use cases.
- You need to think about operating it continuously in production, and for online use cases, that means it has to stay available 24/7.
- You've got to think about and put systems in place, to handle a changing world and changing data, and of course, like any production system, you need to try to do all of this at the minimum cost while producing the maximum performance.
- It might seem daunting, but the good news is that there are well-established tools and methodologies for doing this.

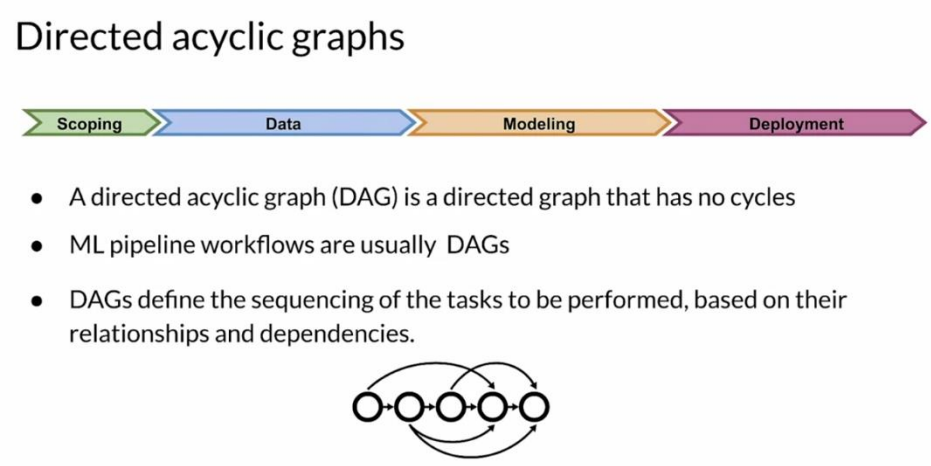
ML Pipelines



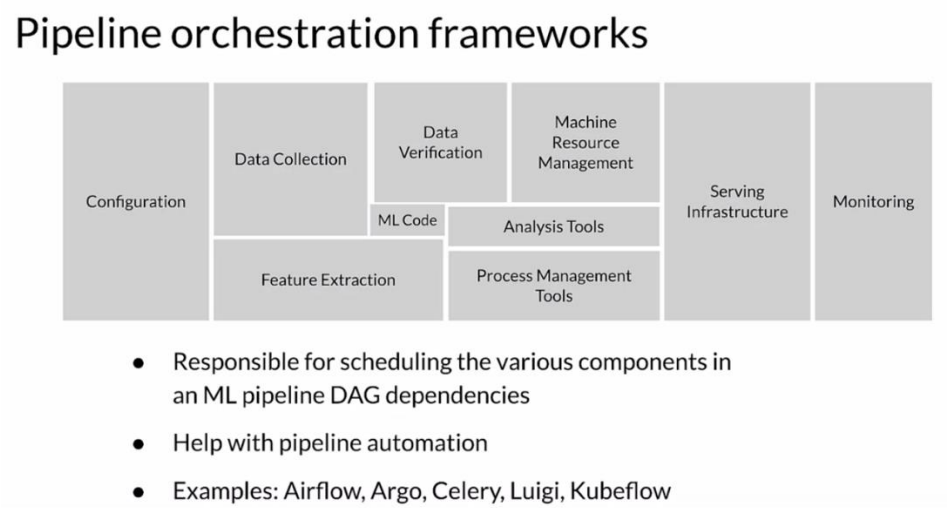
- In this lesson, we'll begin to introduce ML Pipelines and the concept of MLOps.
- We'll also see how pipeline orchestrators sequence and schedule ML tasks to implement the entire ML training process.
- We'll then look at the example of TensorFlow Extended or TFX, which is a widely adopted framework for creating ML Pipelines.
- What do we mean by the phrase ML Pipeline? Well, remember the iterative ML workflow that we talked about previously. **ML Pipeline is a software architecture** to implement exactly that.
- Automating, monitoring, and maintaining this ML workflow from data to a trained model.
- ML Pipelines form a key component of MLOps architectures.
- ML pipelines provide support for automating, monitoring and maintaining a model as you continue to train it over its lifetime.



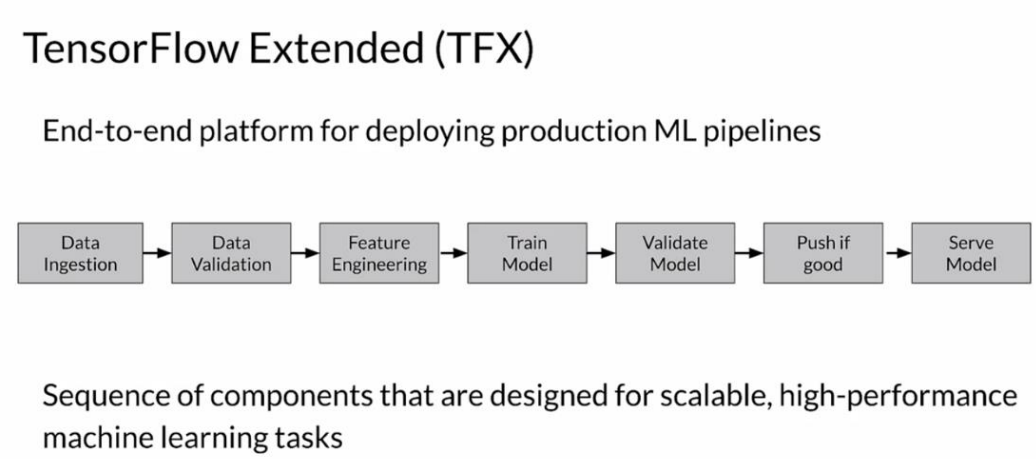
- This slide shows one version of what an ML Pipeline looks like, which was put together by an industry group, the CD Foundation.
- There are some differences between different pipeline architectures, but in general they look something like this.
- You'll notice that they basically mirror the ML development process. Starting with data ingestion and ending with a trained model. That's by design since they need to encapsulate and formalize that process.



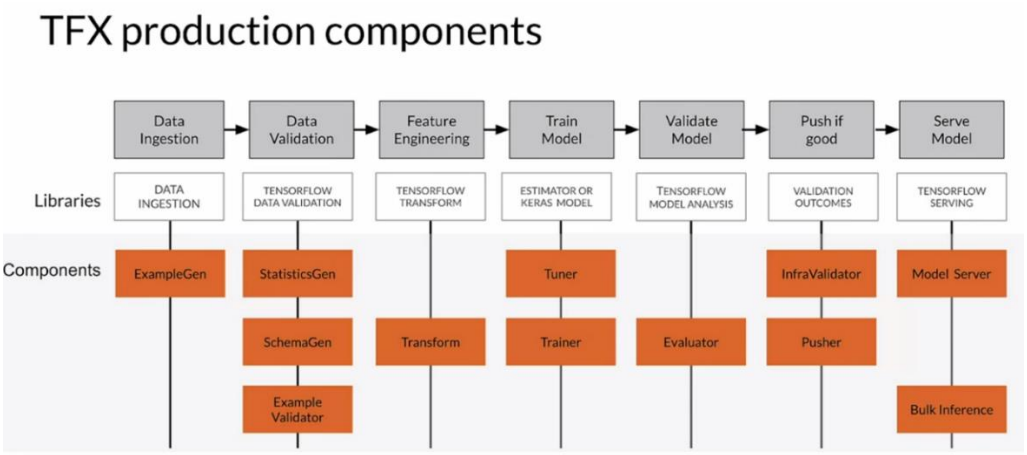
- ML Pipelines are almost always directed cyclic graph or DAGs, although in some advanced cases they can sometimes includes cycles.
- A DAG is a collection of all the tasks you want to run sequenced in a way that reflects their relationships and dependencies.
- Notice that in this graph the edges are directed and there are no cycles. That makes this graph a DAG



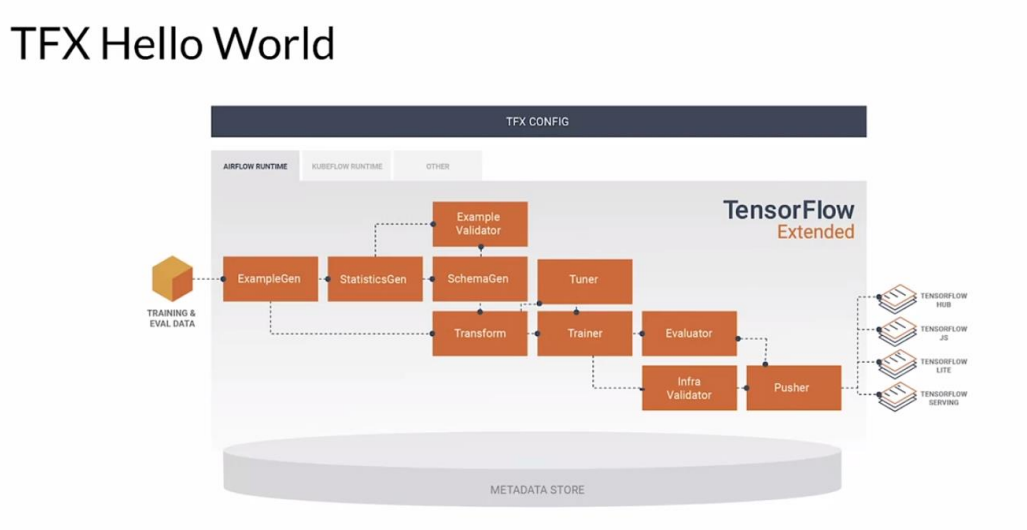
- Orchestrators are responsible for scheduling the various components in an ML Pipeline based on dependencies defined by a DAG.
- Orchestrators help with pipeline automation.
- Examples include Argo, Airflow, Celery, Luigi and Kubeflow.



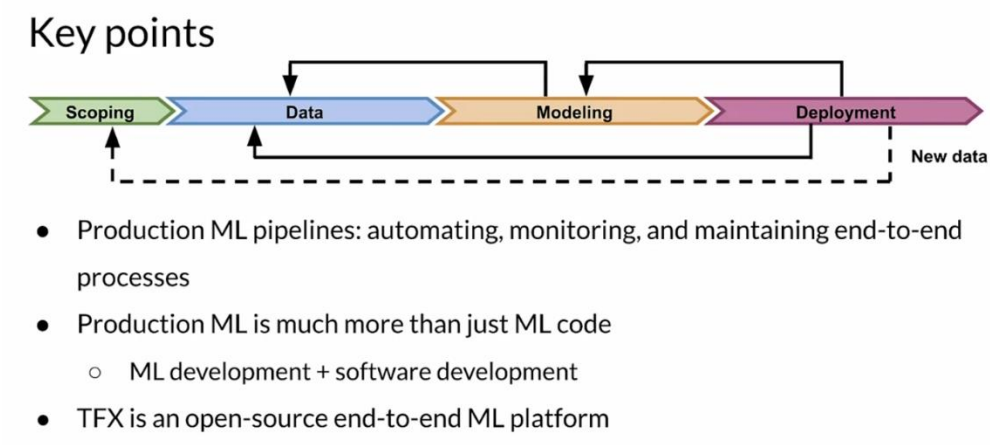
- TFX is an open-source end to end Machine Learning platform for deploying production ML pipelines, and it's what we use at Google.
- A **TFX Pipeline** is a sequence of scalable components that can handle large volumes of data.
- Starting on the left, we ingest data and then we move to Data Validation and we do some feature engineering.
- We then train a model, and we validate it. If it's better than what we already have in production, we're going to push it to production.
- Finally, we're serving predictions. The sequence of components are designed for scalable high performance Machine Learning tasks.



- In this course, you'll be using TFX to implement real ML Pipelines as you would for Production Systems.
- TFX in production components are built on top of open-source libraries, such as Tensorflow Data Validation, Tensorflow Transform and Tensorflow Model Analysis.
- Components in orange leverage those Libraries and form your DAG as you sequence these components and set up the dependency between them, which then forms your ML Pipeline.



- This is what we refer to as the Hello World of TFX. We start on the left with our data and we're going to ingest our data with a TFX component called ExampleGen. All the boxes in orange are TFX components.
- In fact, these are components that come with TFX when you just do a PIP install.
- Next, we generate statistics for our data. We want to know the ranges of our features, if they're numerical features, if they're categorical features, we want to know what are the valid categories and so forth
- Example Validator is used to look for problems in our data. SchemaGen is used to generate a schema for our data across our feature vector. Transform will do feature engineering.
- Tuner and Trainer are used to train a model and to tune the hyper-parameters for that model.
- Evaluator is used to do deep analysis of the performance of our model.
- Infra Validator is used to make sure that we can actually run predictions using our model on the infrastructure that we have. For example, do we have enough memory?
- If all of that passes and the model actually performs better than what we might already have in production, then Pusher pushes the model to Production. What does that mean? Well, we might be pushing to a repository like Tensorflow HUB and then using our model later for maybe transfer learning.
- We could push to TensorFlow JS, if we're going to use our model in a web browser or a Node.js application.
- We could push to TensorFlow Lite and use our model in a mobile application or on an IOT device.
- We could also push to TensorFlow Serving and UserModel on a server or maybe a serving cluster.



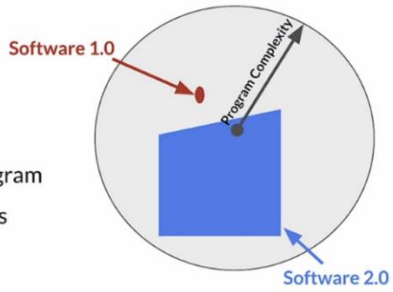
- The key points here, first of all, is that production ML pipelines are more than just ML code.
- They're ML development combined with software development, and are a formalized process for running that sequence of tasks end to end, in a maintainable and scalable way.

Importance of Data

- First let me tell you a story about an application that I was involved in.
- We were asked to create a model to predict the amount of time that it would take to get through an airport security checkpoint on different days at different times with different lines of different lengths and so forth.
- We need data, so we had to measure how long it took people to get through security checkpoints.
- We had one person standing at the beginning of the line to get into a security checkpoint and they would record the time that somebody entered.
- And then we had another person standing at the other end, the exit of the checkpoint, and actually they were far enough away from each other. They couldn't even see each other, and they would record the time that each person left.
- And in that way we would gradually build up a data set of labeled data that gave us the amount of time that people took to get through an airport security checkpoint. Well, as you can imagine, it was incredibly painful and expensive
- So when we talk about collecting data in the real world, hopefully you're not going to deal with a situation like that. But it will be a real world situation where you need to think about how you're going to get the data you need
- Unless, you're very lucky and somebody already has the data for you, which is great.

ML: Data is a first class citizen

- Software 1.0
  - Explicit instructions to the computer
- Software 2.0
  - Specify some goal on the behavior of a program
  - Find solution using optimization techniques
  - Good data is key for success
  - Code in Software = Data in ML



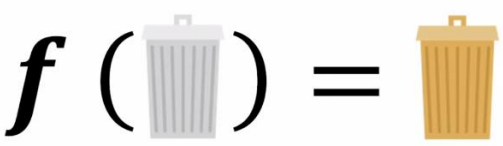
- In programming language design, a first class citizen is an entity which supports all the operations generally available to other entities. In ML, data is a first class citizen.
- So in software 1.0 was all about code, it's really just the instructions for the computer
- In software 2.0, we need to specify a goal and for the behavior of the program. The code is important, but not the only thing that we worry about,
- Optimization is really the driving force here, and can occur in a lot of different directions. We want to optimize for performance, obviously we also want to optimize for maintainability and scalability as well. And for ML, data quality is really critical for success, so in some ways you could look at it as, data is almost like the code in a software

Everything starts with data

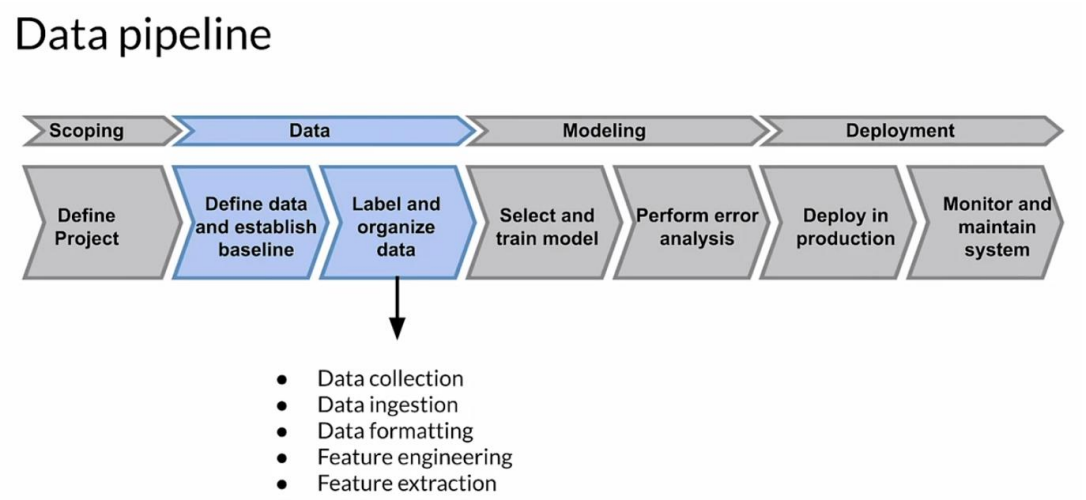
- Models aren't magic
- Meaningful data:
  - maximize predictive content
  - remove non-informative data
  - feature space coverage

- It's good for you to know that you could have mountains of data, but if it doesn't have predictive content, then you're just not going to be able to create a predictive model with it.
- You want to remove information from your model and features from your model that aren't predictive, because they're going to cause problems, and are certainly going to take a lot of compute resources
- You need to make sure that your training data is covering the same feature space as the as the prediction requests, so that your model has good information about the regions of that space to make predictions

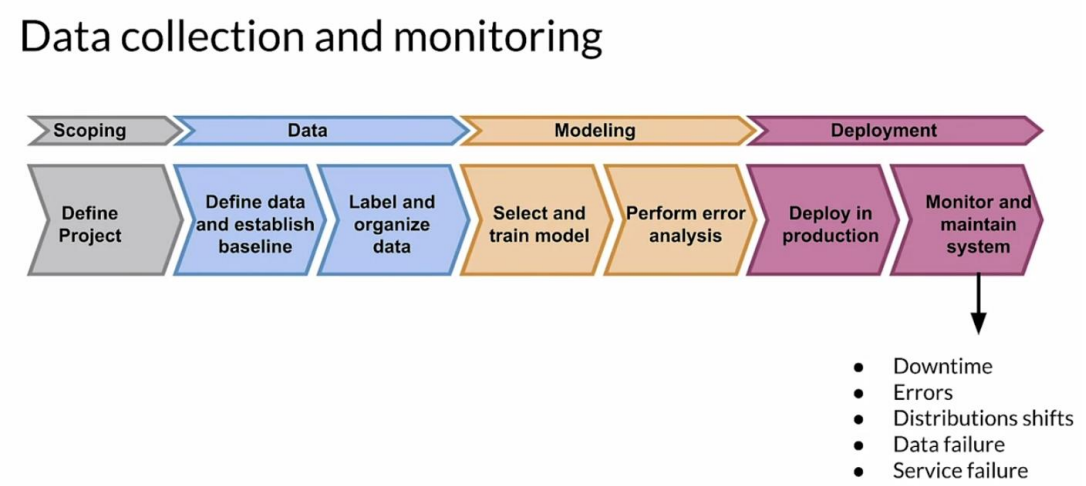
Garbage in, garbage out



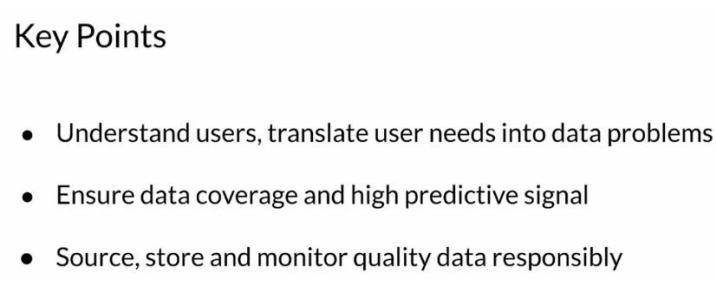
- Just like anything, it's garbage in and garbage out. So if your data is garbage, or if your data quality is low, your model and your application will be low quality.



- Data collection is an important and critical first step to building ML systems



- In deployment, you want to avoid problems with downtime
- You need to make sure that your training model can scale, and can serve predictions
- You need to think about different kinds of errors and what we'll talk about all of these.
- But it's this whole picture that you need to keep in your mind when you are developing the entire process, from ingestion through serving, and all has to be automated.
- It all has to be testable and maintainable and scale well and so forth, so you need to understand your users and you need to make sure that you're translating the user needs into data problems.
- You don't want to do a model that doesn't meet the needs of the user.



- Overall, you need to make sure that your data covers the same region of your feature space as the prediction request so as to maximize the predictive signal in that data.

- You need to worry about the data quality not just at the beginning but throughout the life of the application.
- Part of that is making sure that you are sourcing data responsibly and you're thinking about things like bias and fairness

Example Application – Suggesting Runs

Example application: Suggesting runs

Users	Runners
User Need	Run more often
User Actions	Complete run using the app
ML System Output	• What routes to suggest • When to suggest them
ML System Learning	• Patterns of behaviour around accepting run prompts • Completing runs • Improving consistency

- For this example, we're going to be looking at an application that suggests runs to runners.
- There's different runners with different level of fitness. The first step is really to try to understand the users
- This system is going to suggest runs based on the user's behavior and by leveraging observed patterns and preferences.
- The goal is to improve the consistency of running and for runners to complete those runs and to really be happy about that.

Key considerations

- Data availability and collection
  - What kind of/how much data is available?
  - How often does the new data come in?
  - Is it annotated?
    - If not, how hard/expensive is it to get it labeled?
- Translate user needs into data needs
  - Data needed
  - Features needed
  - Labels needed

- First of all, you need to consider data quality and data collection. What kind of data and how much data do you need? How often do you need new data? When do you expect things to change? Is that data annotated?
- We need to first understand the user, otherwise we run the risk of collecting a bunch of data that is really just garbage.
- But once we understand the user, then we need to translate the user needs into data needs. We're going to do that with identifying what the data is, what the features are and what the labels are.

Example dataset

		FEATURES			
	Runner ID	Run	Runner Time	Elevation	Fun
EXAMPLES	AV3DE	Boston Marathon	03:40:32	1,300 ft	Low
	X8KGF	Seattle Oktoberfest 5k	00:35:40	0 ft	High
	BH9IU	Houston Half-marathon	02:01:18	200 ft	Medium

LABELS

- Here's an example data set. We've got three different examples here for three different kinds of runs and we've got some features.
- The examples here are the Boston Marathon, the Seattle Oktoberfest 5K and the Houston Half-marathon.
- The features are the Run itself, the Runner Time and the Elevation, which is also important.
- Then the labels here are just going to be how the runner rates the fun level of those runs.

Get to know your data

- Identify data sources
- Check if they are refreshed
- Consistency for values, units, & data types
- Monitor outliers and errors

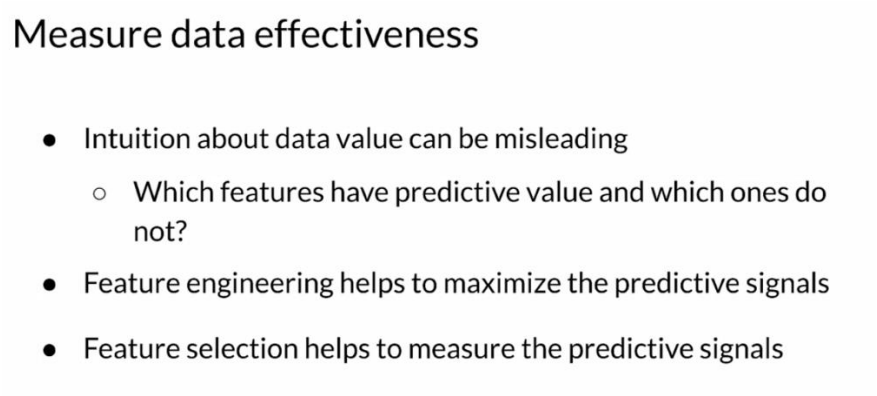
- You need to identify the data sources that you're going to use, where are you going to get this data and not just the first time but on an ongoing basis
- It's not just the training but you need to collect that same data, for you to do inference when you want to create a prediction.
- You need to think about, how often do I need to refresh my trainings at? Along the way when you're working with your data, you need to make sure that there's actually predictive value in your data. Make sure that you've eliminated features and data that does not have predictive value.
- There's also some more basic things like, is the data consistent? When you're expecting, say a float do you always get a float or is it mixed? Also look for things like outliers too or just errors.

Dataset issues

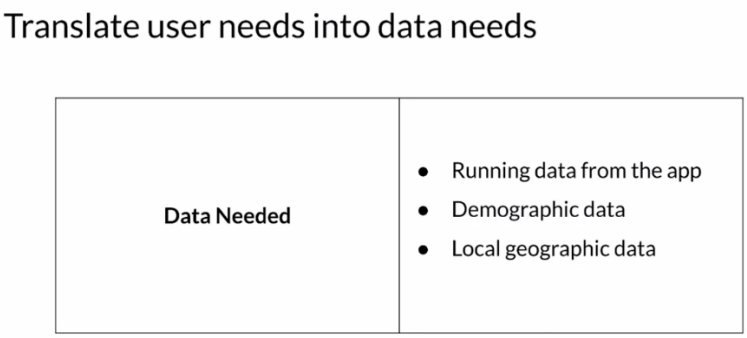
- Inconsistent formatting
  - Is zero “0”, “0.0”, or an indicator of a missing measurement
- Compounding errors from other ML Models
- Monitor data sources for system issues and outages

- There could be data issues coming from different measurements, different types and also simple things like the difference between an int and a float, or how missing value is encoded, that can all cause problems.
- In this example data set, if the elevation is zero feet, does that really mean that we're at sea level or does it mean that we don't have any elevation data for that record?

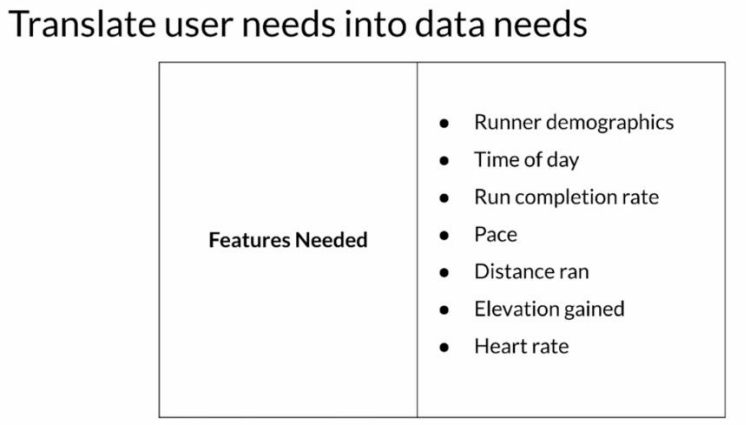
- If the output is coming from other ML models (maybe you're using an ensemble), there's errors in those that can compound when you try to use it in a downstream model.
- You also want to make sure that you're looking for errors and issues early in the process and monitoring the data sources for system issues and outages. Because we could potentially be operating this thing 24-7.



- You need some intuition about the data value, but your intuition can be misleading.
- You really need to make sure that you're looking at which data is really giving you the most information.
- Feature selection and feature engineering are really critical to shaping your data to be what you need it to be. The feature engineering helps you maximize predictive signals once you've identified where those are.
- Feature selection helps you measure where that predictive information is and focusing in on those features can give you the most value and help your model the most.



- You need to understand the user and the application
- In this case, we're looking at running data from an app. We can get demographic data when the user fills out their profile and we also probably can get some GPS data to give us some local geographic information. At a high level, that helps us understand the user.



- Then we need to translate that into features.

- The runner demographics, we need to express that as a feature or probably several features. Things like the time of day, how long it takes them to complete a run, their pace during the run, the distance and so forth
- We can also get information for elevation, and if we're working with an app that has some sensors like a heart rate monitor, that's great information to have to really feed into this app.

Translate user needs into data needs

Labels Needed	• Runner acceptance or rejection of app suggestions • User generated feedback regarding why suggestion was rejected • User rating of enjoyment of recommended runs

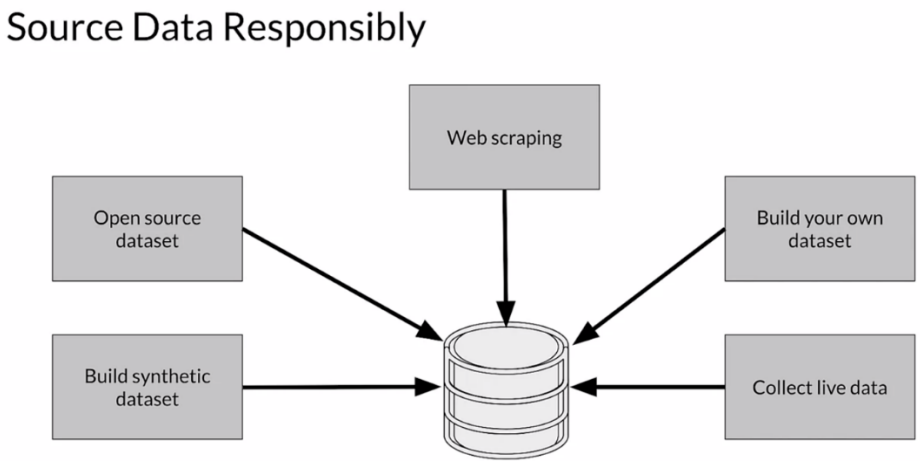
- Runner acceptance in this case is a label that we want to focus on. Runners that take our suggestions and use them. That tells us that the app successfully gave them a run that they wanted to do and conversely, if they rejected it.
- User-generated feedback. You need to think about, first of all, how they're going to give you that feedback in a structured way that you can use to help with training your model.
- Then you know things like user rating. In this case, it relates to the enjoyment of the recommended runs.

Key points

- Understand your user, translate their needs into data problems
  - What kind of/how much data is available
  - What are the details and issues of your data
  - What are your predictive features
  - What are the labels you are tracking
  - What are your metrics

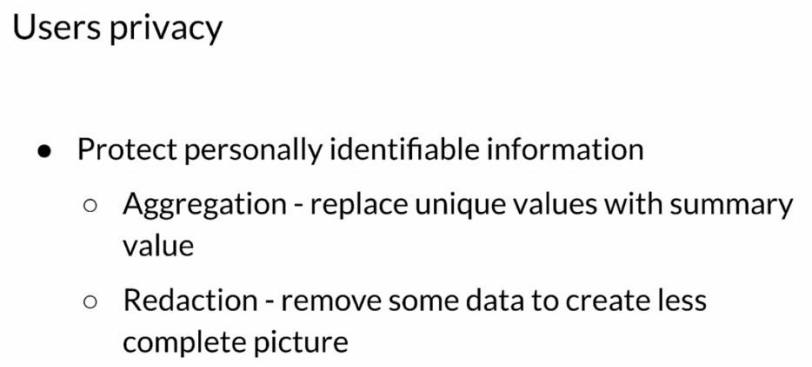
- Key points: understand your user and translate their needs into data problems and well-defined features that give you predictive information.
- Questions to ask include: What kind of data can you get? What's available? What are the details and issues for your data? Where's the predictive information in your data? What are the labels? We need to make sure that we're training a model to predict the right thing. We need to make sure our labels are the correct ones for our goals. What are the metrics to use to measure performance of our model?

- Here's an example. These images here show one standard open source image classifier trained on the open images data set that does not properly apply wedding related labels to images of wedding traditions from different parts of the world.
- On the far left, the classifier's label prediction is recorded as ceremony, with the labels wedding, bride, man, group, woman, dress. So that's pretty correct.
- The next one is bride ceremony, wedding, dress and woman. Again we know that's correct at least in the west, that's what a bride typically looks like.
- But the one on the end, the one on the right. Well, that's for an African wedding ceremony, but it's incorrectly labeled as simply person or people. Well, it is person and people, but it's also a ceremony and there's a bride and a groom and address and so forth. So this is a classic case it's an example it's often cited of a problem with bias in the data set.



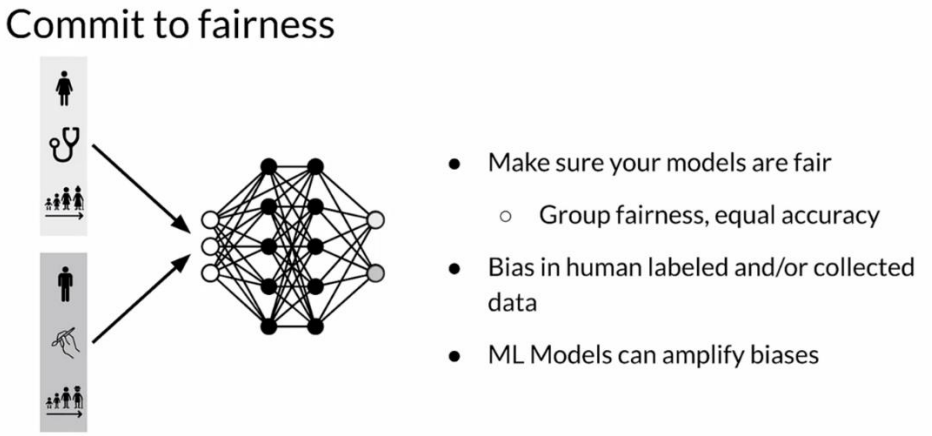
- So an ML system, the data may come from different sources and you need to think about those sources as well. It's not just the data that you have but where did you get it from?
- So you might be building synthetic data, you might be doing web scraping or collecting live data, especially when you're running inference.
- You're often almost always really going to be building your own data set, although sometimes you can be using an open source data set. It just depends on what's available and what you need.

- Data security refers to the policies and methods to secure personal data or what's often referred to as PII, Personally Identifiable Information.
- Data privacy is about proper usage, collection retention, deletion and storage of that data.
- So data collection is not just about your model. You need to think about your users and treat that data as something given to you, of which you are a steward responsible of managing that data responsibly.
- Users really should have control over which data is being collected. And it's important to establish mechanisms to prevent your system from revealing a user's data inadvertently
- How you handle your data privacy and data security depends on the nature of your data as well as the operating conditions and regulations and policies, with things like GDPR being important here.

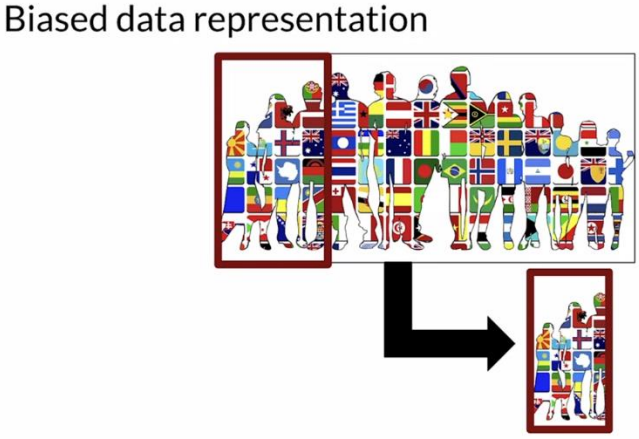


- Users' privacy is also really key. So you need to protect personally identifiable information or data
- Aggregation really helps with that, if you can aggregate the data so that you cannot identify individual people within it
- You need to consider any laws or regulations regarding user privacy in the places where you're going to be using your model
- Another way is redaction. So in a lot of cases you need to give users a way to remove some of the data, which will create a less complete picture but is part of being responsible with your data.

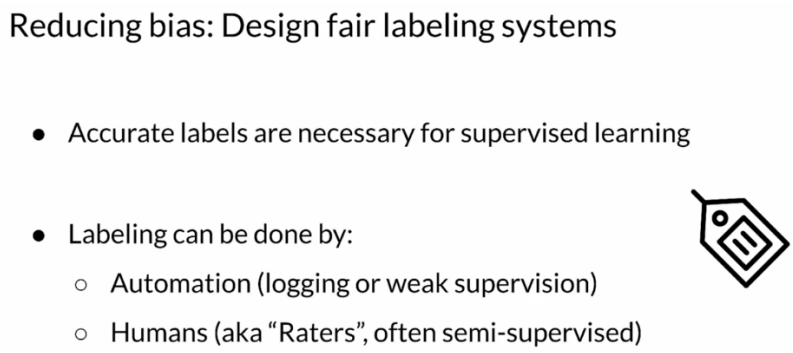
- So ML Systems can fail users in a lot of different ways and we need to strike a balance between being fair and accurate and transparent and explainable.
- Some of the ways that ML Systems can fail are through things like representational harm.
- Representational harm is where a system will amplify or reflect a negative stereotype about particular groups.
- Opportunity denial is when a system makes predictions that have negative real life consequences that could result in lasting impacts.
- Disproportionate product failure is where the effectiveness of your model is really skewed so that the outputs happen more frequently for particular groups of users, you get skewed outputs more frequently. Essentially can think of as errors more frequently.
- Harm by disadvantage is where a system will infer disadvantageous associations between different demographic characteristics and the user behaviors around that.



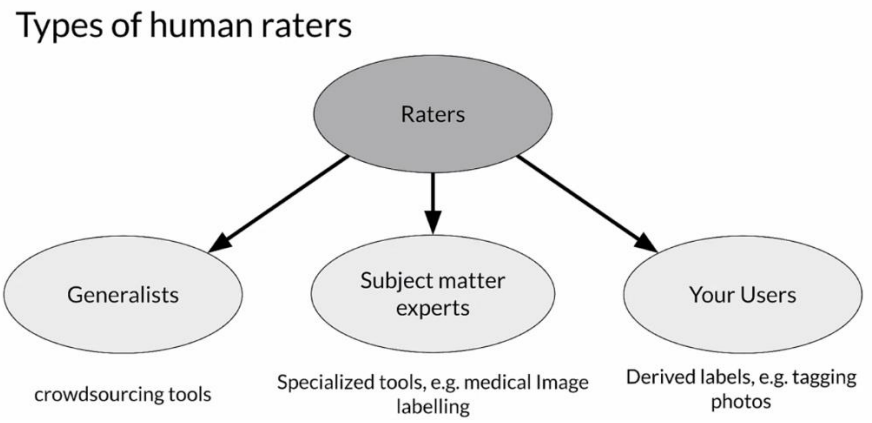
- So fairness is important and you should really commit to it from the beginning. So what does it mean?
- Well, being fair means that you're going to identify if some groups of people get a different experience than others in a problematic way.
- For instance, let's assume particular gender, occupation or age fields are part of your data and you use it to train a model that predicts whether someone would be a reliable new employee. So it's involved in hiring.
- You need to really check to make sure that your model does not consistently predict different experiences for some groups in a problematic way by ensuring group fairness.
- What that means is demographic parity and that things are equalized across different groups.
- And you need to make sure the accuracy as well is equal or or as close as you can get it.
- The data collected and labeled by humans will reflect their biases in many cases and their personal experiences so you have to account for that.
- Diversifying your user base is a good way to move towards fairness, but it's not a guarantee.
- ML Systems can **amplify** biases, so you need to be aware of that and be careful about it



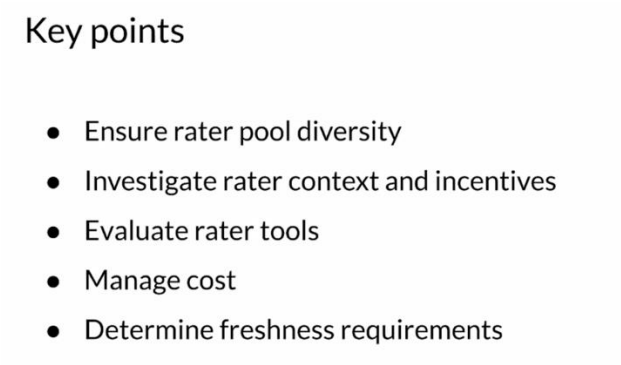
- Biases can also arise when you have disproportionate representation of some groups within your data or no representation at all.
- So looking at the graphic here, what we're trying to show is that part of the people that are shown here are in your data, but there's a whole lot of other people who are not.
- Those groups that might be stereotyped are the ones that are not in your data, and they might be presented in a less positive way, or they may just get a bad experience a lot more often.



- To reduce bias for supervised learning, you need accurate labels to train your model and serve predictions.
- The labels are usually coming from two broad sources. Most of the time they're going to be coming from automated systems or human Raters
- Humans are able to label data in different ways. And the more complicated the data is, the more you may require an expert to look at that data

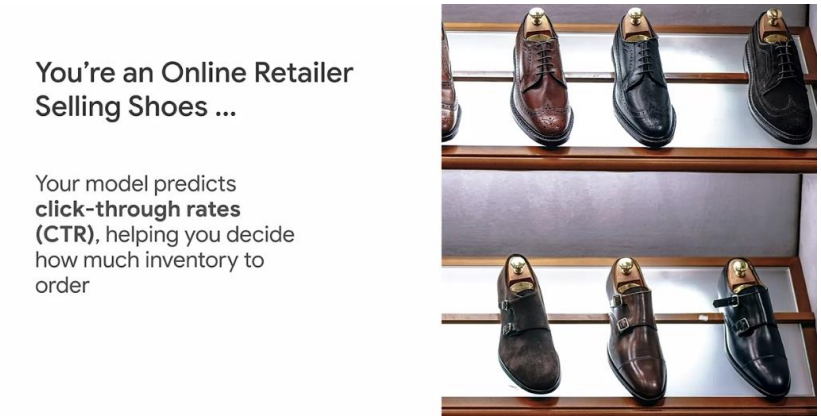


- Who are raters? Well, they could be generalists and these are people who just pretty average people who are going to be adding labels through a variety of crowdsourcing tools. And these are cases where it's fairly easy for people to recognize the correct label.
- So for example, if you want a human to recognize the difference between a cat and a dog, that's something most people can do by looking at the image.
- But in some cases you really need a subject matter expert. So in those cases you're often using specialized tools and an example of that is looking at X-rays for diagnosis. It's not something that just anyone can do. You need to make sure that you're working with an expert and that labelling tends to get pretty expensive
- You can also use your users. So this sort of feedback that we looked at for our running app. This can often be very valuable if you can find a way to work without in your application and it's going to give you this ongoing stream of labels for your data if you can make that work.

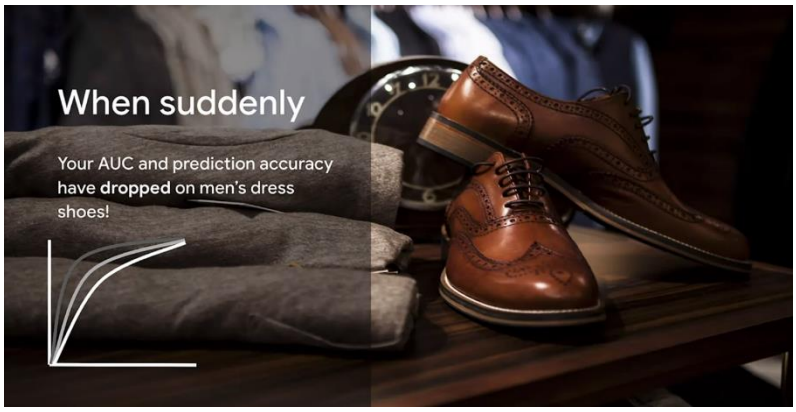


- First of all, always account for fair Raters and fair representation in your data set to avoid potential biases.
- Take into account who those labelers are, and what their incentives are, because if you **design the incentives incorrectly, you could get a lot of garbage in your data.**
- The cost is certainly always going to be an important consideration. So if you can find a way to do it with a high level of quality but at less cost, that's great.
- Finally data freshness too. You're going to be working with data and depending on how the context changes around the application and the data that you have, you're going to need to refresh that data on some regular basis and detect when you need to do that.

Case Study – Degraded Model Performance



- Imagine that you're an online retailer and you're selling shoes. You have a model that predicts click through rates which helps you to decide how much inventory to order.



- Suddenly the AUC and prediction accuracy have dropped, not on everything but on a particular part of your inventory, Men's dress shoes. Why?

Case study: taking action

- How to detect problems early on?
- What are the possible causes?
- What can be done to solve these?

- Unfortunately, if you don't put good practices into place in the production setting, you're probably going to find out either when you order way too many shoes or not enough shoes. And that's not a situation that you want to be in in a business. This is going to cost you money
- So you need to think about how you're going to detect problems like that early, and what the possible causes are so that you can look for those and monitor your system.
- And then try to have methods and systems in place to deal with those problems when they happen because they probably will happen at some point.

What causes problems?

Kinds of problems:

- Slow - example: drift
- Fast - example: bad sensor, bad software update

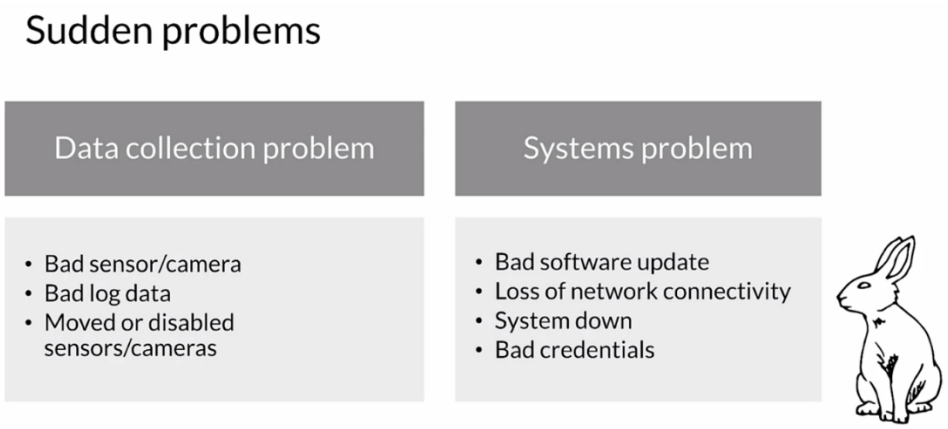
- But what kinds of problems? Well, there's different kinds and they tend to fall into two different categories.
- There are slow problems. So for example, your data will drift over time as the world changes and seasons pass and you have holidays and competitors enter etc.
- And then you have fast problems that are really part of your system. So you have a sensor that goes bad or you have a software update that gets applied and suddenly things are wonky.
- So you need to really be monitoring your system well and looking for both of those problems and thinking about remediation in both cases

Gradual problems

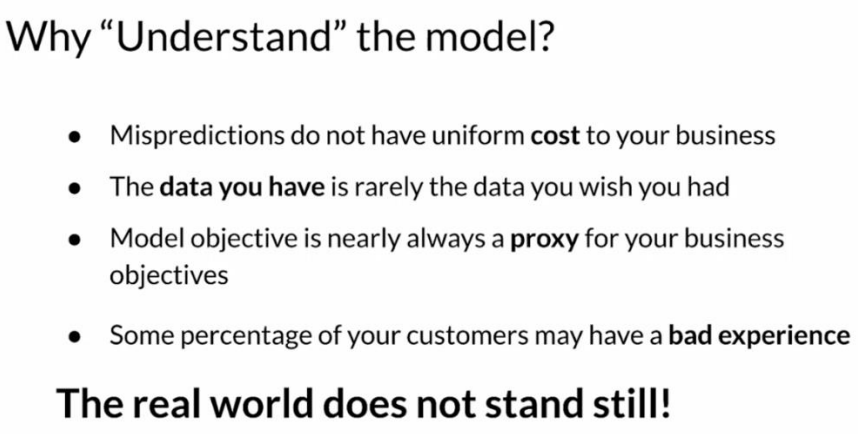
Data changes	• Trend and seasonality • Distribution of features changes • Relative importance of features changes
World changes	• Styles change • Scope and processes change • Competitors change • Business expands to other geos

- For gradual problems, they tend to fall into two groups, but they're interrelated too so it's not like there's a hard line between them.
- Trend and seasonality is an example, especially in time series where you will have trends and you will have seasonality in most cases. You could argue about whether that's really a change in data or a change in the world.
- Same thing with the distribution of the features. And the relative importance of features can change too. If you haven't retrained your model, the accuracy starts to decay.

- The world also changes constantly. And in production settings that has to become part of the systems that you design and the processes that you have in place.
- Things like we're working in retail and we're just looking at an example with shoes, fashion styles change. So, maybe last year black shoes were really fashionable for men's dress shoes and now it's brown shoes or what have you.
- The scope and the processes change. So your understanding of those processes and how they happen in the world will affect how your model views the results of those processes.
- Competitors change and your business also changes. So you may take on new products, or you may stop selling some other products
- This tends to be very domain specific, but in general, in nearly all domains, changes in the world affect your model performance.



- For sudden problems, there's things that you're probably familiar with. In data collection problems, things like a bad sensor or bad camera, the log data suddenly changes and you have a different format.
- In terms of systems problems, you're going to have software updates all the time and if they change something significant that you weren't aware of, that can be a real problem.
- Network connectivity. Sometimes the network or system goes down sometimes it does.
- And bad credentials, like if your sign in credentials time out or something changes about them, all those things can cause a sudden problem and you need to deal with it.



- It is important to understand your model and how it's sensitive to different changes in the world.
- A big issue here is that mispredictions don't have uniform cost to your business. Some mispredictions will have very little effect on your business. Other mispredictions could have huge effects.
- So understanding that and as you're monitoring things, looking for things that have larger impacts is important.
- The data you have as you collect data is rarely the data that you wish you had. So I've personally seen cases where we were using sensor data from wifi devices and it was not great data. It was really noisy and but it was all we had. So we had to work with what we had.

- The model objective is almost always a proxy for what you're really trying to get to. In many cases, sometimes you can design a model and you have the data to design a model for exactly the thing that you're trying to do.
- But often, as in the case of the shoes example that we just looked at, we were predicting click through rates as a proxy for deciding how much inventory to order.
- Some percentage of your customers will have a bad experience. You want that percentage to be as small as possible
- You want to understand which customers those are going to be, so that you can try to design ways to mitigate that and improve the situation for all of your customers.
- But the bottom line here that you will deal with constantly is that the real world does not stand still. The one constant in the world is change.

Data and Concept Change in Production ML

Detecting problems with deployed models

- Data and scope changes
- Monitor models and validate data to find problems early
- Changing ground truth: **label** new training data

- You need to monitor your models and you need to validate the results and the data of your models, so as to find problems.
- You want to try to find problems early especially with system problems that happen quickly like a bad sensor or things like that
- But a fundamental issue is changing ground truth. What that means is that you need to label new data over the life of your application. It depends on the domain that you're working in and the kinds of problems that you're trying to solve

Easy problems

- Ground truth changes slowly (months, years)
- Model retraining driven by:
  - Model improvements, better data
  - Changes in software and/or systems
- Labeling
  - Curated datasets
  - Crowd-based

- There are easy problems. Things like trying to recognize images of cats and dogs.
- In this case, the ground truth really changes pretty slowly. Model retraining in those cases is usually driven by model improvements
- You could have changes in software too. You could be upgrading things or using a different library, that kind of thing, or systems.
- Labeling in this case is fairly simple, you're going to work with maybe a curated dataset that you're getting from some public domain source, or a source that your organization has been using for awhile
- You could also look at crowd-based data as well.

Harder problems

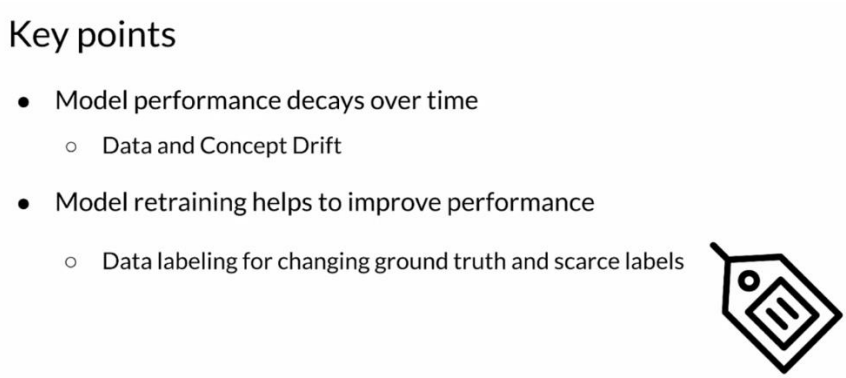
- Ground truth changes faster (weeks)
- Model retraining driven by:
  - **Declining model performance**
  - Model improvements, better data
  - Changes in software and/or system
- Labeling
  - Direct feedback
  - Crowd-based

- Then we get into a little harder problems where the ground truth really changes faster.
- Things like styles (e.g. shoes), but there's a lot of things in the world that changes in a matter of maybe weeks.
- Model retraining in that case is usually driven by declining model performance, which you need to measure if you're going to be aware of.
- You can also have model improvements to implement, or you can also have better data, and of course, the software and systems you're running on can also change.
- For labeling, if you can get direct feedback from either your systems or from your users, that's great.
- Crowd-based human labeling is another feasible way of doing this, since you do have probably weeks to respond. You can take a pass through human raters to do that.

Really hard problems

- Ground truth changes very fast (days, hours, min)
- Model retraining driven by:
  - **Declining model performance**
  - Model improvements, better data
  - Changes in software and/or system
- Labeling
  - Direct feedback
  - Weak supervision

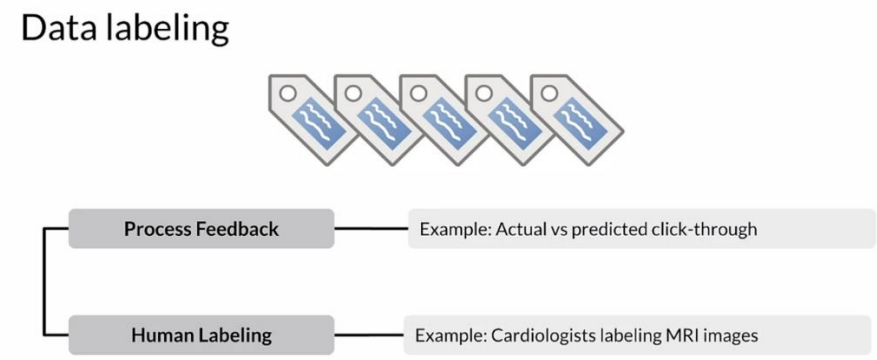
- In this case, ground truth changes very fast, like in the order of days or hours or even minutes.
- Things like stock markets fall into this category, they change very quickly.
- In this case, declining model performance is definitely going to be a driver for when you need to retrain your model.
- You can also have things like model improvements and changes in software and so forth, but those tend to be things that you'll be working on offline while you're keeping your application running.
- It's really model performance where you really need well-defined processes to deal with those changes.
- Labeling in this case becomes very challenging. Direct feedback is great if you can do it in your domain.
- If not, you need to look at things like week supervision that we'll talk about. But it's really challenging in these kinds of domains. They tend to be high-value domains. Things like predicting markets where there's significant incentive to doing these predictions.



- The key points of what we're talking about here, includes the point that model performance decays over time. It may decay slowly over time, in things like cats and dogs, that doesn't change very quickly, or it may change very fast, things like markets.
- Model retraining will help you improve or maintain your performance.
- Data labeling, assuming you're doing supervised learning, which is pretty common, is a key part of that. You really need to think about how you're going to approach that in your particular problem, in your particular domain and with the systems that you have available to you.

Process Feedback and Human Labelling

- There are numerous ways to generate labels for your data. We'll focus on the first two, which are the most common, process feedback or direct labeling and human labeling



- You need labels if you're going to do supervised learning
- Two simple ways of doing that are processed feedback and human labeling.
- Let's look at some examples, for process feedback, a very typical example is click-through rates. Actual versus predicted click-through rates.
- Suppose you have recommendations that you are giving to a user, did they actually click on the things that you recommend? If they did, you can label it positive, if they didn't you can label it negative.

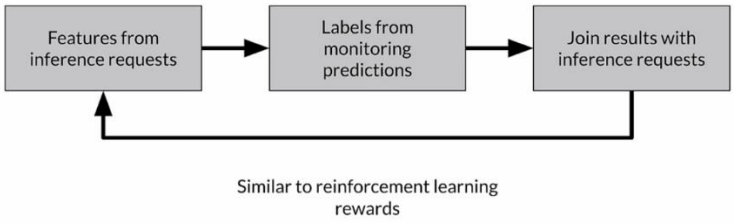
- Human labeling, you can have humans look at data and apply labels to them.
- For example, you can ask cardiologists to look at MRI images and apply labels to them

Why is labeling important in production ML?

- Using business/organisation available data
- Frequent model retraining
- Labeling ongoing and critical process
- Creating a training datasets requires labels

- Why is labeling important in production ML? Most businesses and organizations have a bunch of data, but if it's not labeled, you can't use it for supervised learning.
- If you can apply unsupervised techniques and get good results, that's great. But in many cases you really need supervised learning to solve the problems that you're trying to solve.
- What that means is that in most domains you're going to need to retrain at some point. It will depend as we've talked about on the domain that you're working in and the type of problem.
- Some will just need to retrain on an infrequent basis, and some might need to retrain several times a day.
- Labeling is an ongoing and often critical process in your application and your business

Direct labeling: continuous creation of training dataset

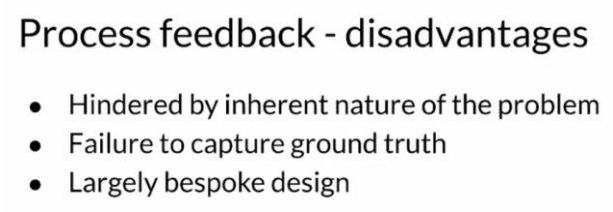


- Direct labeling is a way of continuously creating new training data to use to retrain your model.
- You're taking the features themselves from the inference requests that your model is getting. The predictions that your model is being asked to make and the features that are provided for that.
- You get labels for those inference requests by monitoring systems and use the feedback from those systems to label that data.
- One of the things that you need to solve is to join the results that you get from monitoring those systems with the original inference request which could be hours or days apart.
- You might've run batches on Monday and you're getting feedback on Friday. You need to make sure that you can do those joins to apply those labels.
- In some ways you can think about this as similar to reinforcement learning, where instead of applying rewards based on action, you're applying labels based on a prediction. It's a similar feedback loop.

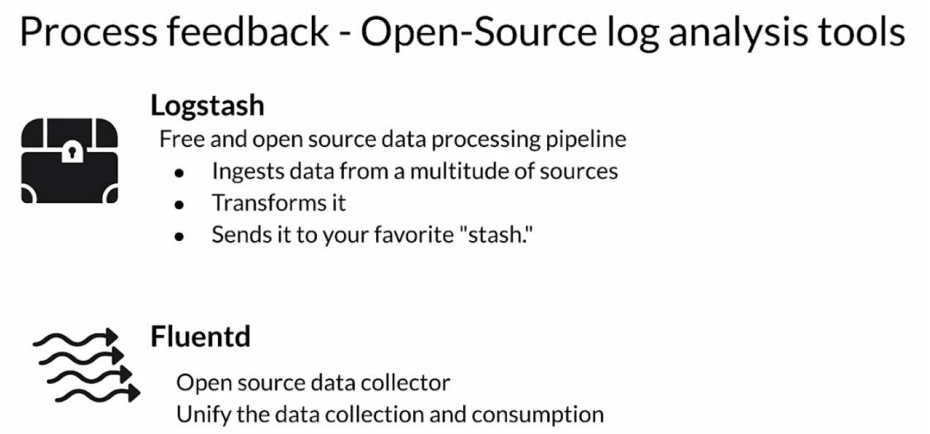
Process feedback - advantages

- Training dataset continuous creation
- Labels evolve quickly
- Captures strong label signals

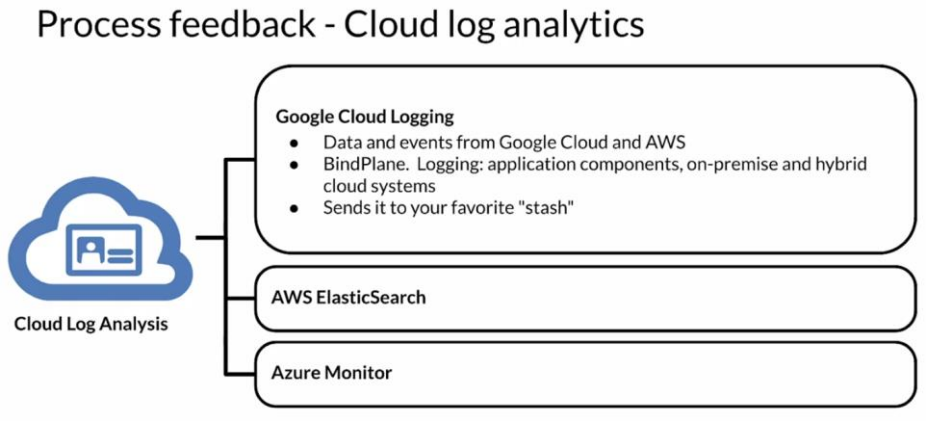
- The advantages of direct labeling is great, if your system and your domain is set up in a way that you can do that. It's often the best answer because you get back labels that you are monitoring, and you are constantly getting new training data.
- The signals that you get from your labels are really strong. You're getting things like click-throughs, if the user clicked or didn't click, it's a very strong signal.



- In many domains for many problems, unfortunately direct labelling just isn't possible.
- Personally in the problems I've been asked to solve, I found very few where I was able to do that, so that's an issue.
- The other big thing is that it tends to be very custom-designed. Your systems will be unique, but it'd be great if it was a little bit more off the shelf

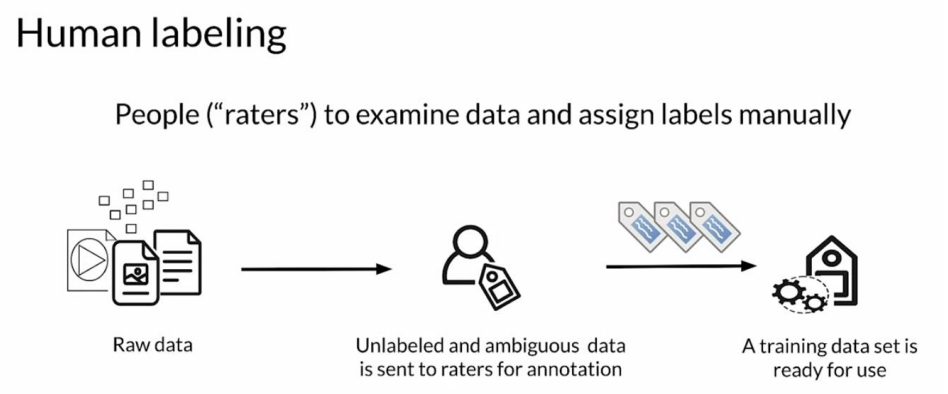


- One of the tools that you can apply is log analysis tools. Because often when you're doing process feedback, the data is coming from the log files. You're monitoring systems and populating log files.
- One good open source tool for doing that is **Logstash**. You can ingest from multiple sources for collecting and parsing and storing logs. You can index them in Elasticsearch, and you can push them to storage.
- It takes inputs from a variety of different sources and databases and so forth.
- Great tool and it's open source too.
- **Fluentd** is another good open source tool you can use to collect and parse. Fluentd comes from the Cloud Native Computing Foundation and it connects to a lot of different platforms.

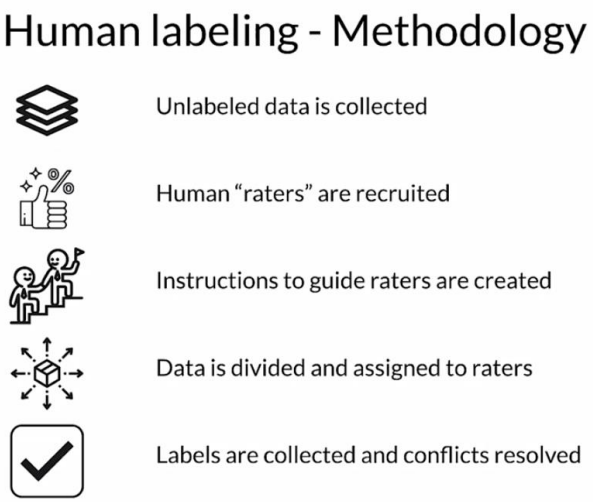


- When you're working on the Cloud, there's Cloud tools that are available, as well. If you're working on the Google Cloud, Google Cloud logging is a great tool to be able to log your data, either coming from Google Cloud or from AWS.
- BindPlane is great for applying on-premise or hybrid cloud systems. It's a very powerful service.

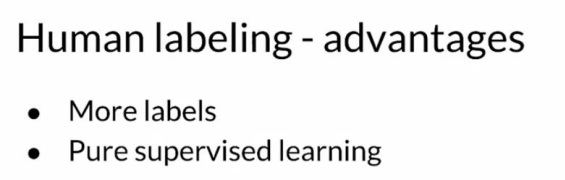
- For AWS, their version of Elasticsearch is available, so you can apply that. It's not really strictly a log analytics tool but you can apply it for login analytics.
- For Azure, you have Azure monitors. So regardless of which Cloud you're working on, there's probably log analytics tooling that you can use to do log analysis.



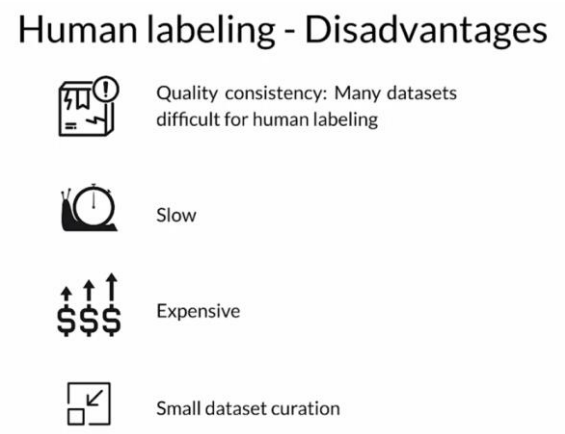
- In human labeling, you have people, humans and we refer to those as raters.
- We ask them to examine data and assign labels to it.
- You start with raw data and you give it to people and you ask them to apply labels to it. That's the way that you create a training data set that you're going to use to train or retrain your model.



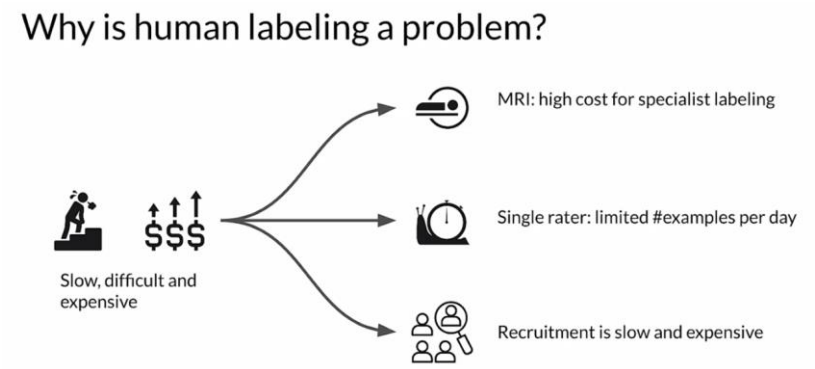
- You started with unlabeled data and then you need to recruit human raters
- There are several services that you can go to where they have pools of human raters already recruited.
- You need to provide them with instructions. Even if it's very simple, you need to still tell them what labels they should apply and what to look for to decide which label to apply.
- The data is then divided among different raters in the pool. Often you send the same examples to multiple raters, so that when there's disagreements, you're aware of that and you can work to resolve them.
- Then you collect the data and any of those conflicts that you have are resolved.



- Advantages of the human labeling. It allows you to generate labels which you need for supervised learning. It's a way to do that and it's actually a very common way to do that.

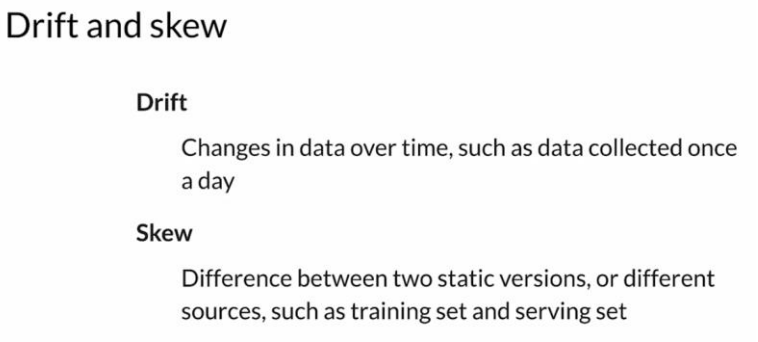


- You can have quality problems where different humans disagree on what the label should be.
- It can be very slow. You're asking people to look at each individual example and it can take time to do that
- In cases where the data's changing quickly, it is often just simply not feasible. Again, it can be very expensive even if you're using generalists to look at very simple data.
- In cases where you're asking experts to look at data, then it gets very expensive.
- Often what that means is that you end up with small data sets because the cost and the time involved results in difficulty in getting a very large data set.



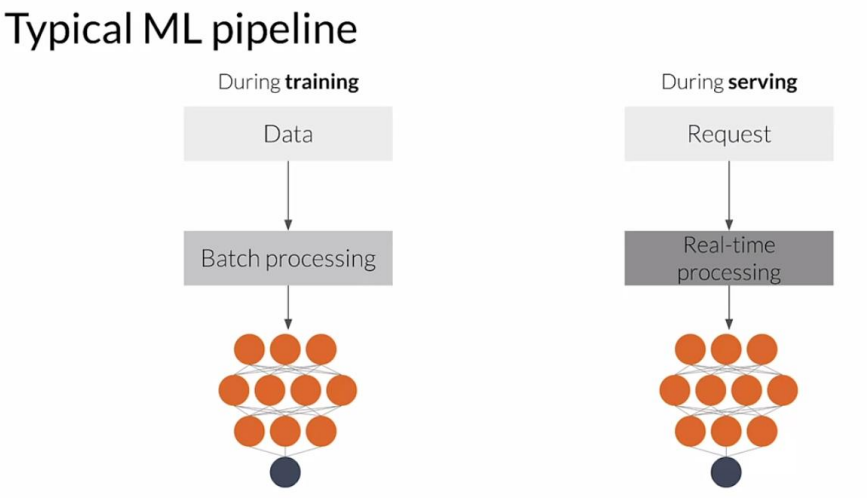
- Overall it can be slow, it can be difficult, and it can be expensive.
- If you're doing something like an MRI and you've got a specialist looking at it again, cost is a problem.
- A single rater can only do a certain number of examples per day, so you need to have a fairly large pool compared to the number of examples that you're trying to label.

Detecting Data Issues

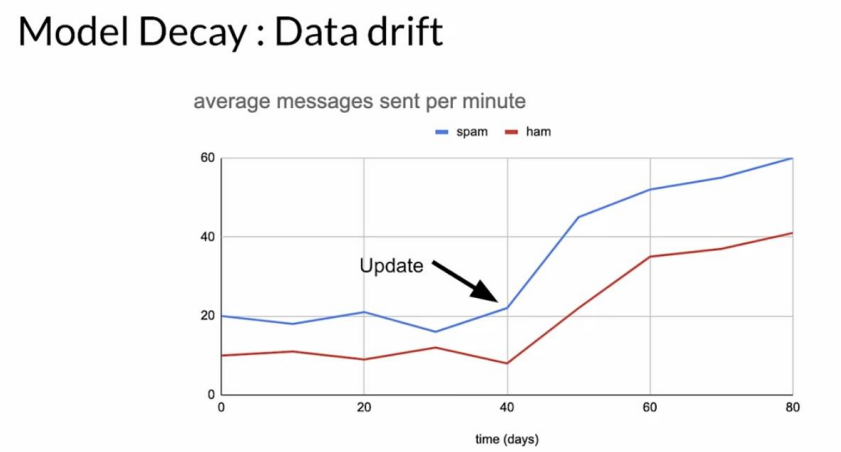


- Drift is changes in data over time. For example, data collected once a day over time, maybe a week later, a month later, there are changes that data has drifted.

- Skew is the difference between two static versions from different sources of conceptually the **same** dataset.
- For example, it could be the difference between your training set and the data that you're getting for prediction requests, your serving set. Those differences are referred to as skew.

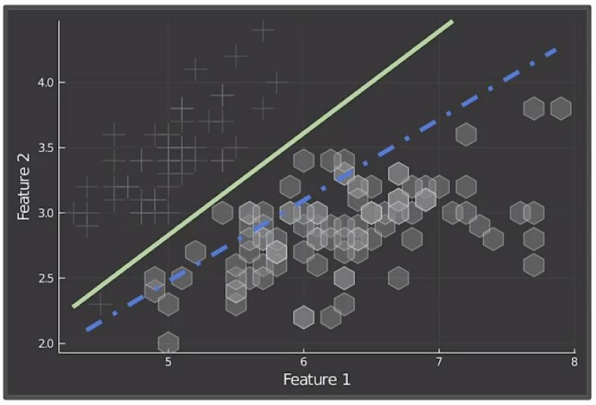


- In a typical ML pipeline, we have shows batch processing, but it could also be online processing.
- They have the same feature vector, but over time they will change.
- That means that model performance can either drop quickly due to things like system failure, or can decay over time due to changes in the data and changes in the world.
- We're going to focus on performance decay that arises due to issues between training and serving data.



- There's Data drift, which are changes in the data between training and serving typically and Concept drift, which are changes in the world changes in the ground truth.
- In model decay over time, an ML model will start to perform poorly in many cases and we refer to this as model decay. That's usually caused by drift, which is changes in a conceptual way.
- There are changes in the statistical properties of the features, or sometimes due to things like seasonality or trend or unexpected events.
- This example here, we're looking at a spam classifier app
- After a system update, both spammers and non-spammers start to send more messages.
- In this case, the data and the world has changed, and that causes unwanted misclassification

Performance decay : Concept drift



- Concept drift is a change in the statistical properties of the labels over time.
- At training, an ML model learns a mapping between the features and the labels. In a static world that's fine, that won't change.
- But in real-world, distribution and the labels' meaning will change. The model needs to change as well, as the mapping found during training will no longer be valid

Detecting data issues

- Detecting schema skew
  - Training and serving data do not conform to the same schema
- Detecting distribution skew
  - Dataset shift → covariate or concept shift
- Requires continuous evaluation

- As you previously see, there are many factors that cause changes over time, including upstream data changes and seasonality and evolving business processes.
- Schema skew occurs when the training and serving data do not conform to the same schema, which you might think could never happen but actually it can because you're collecting data and things change and suddenly you're getting an integer where are you expecting a float (i.e. schema changes).
- Or you're getting a string where you are expecting a category.
- Distributions skew is a divergence of training and serving datasets.
- The dataset shift can be really manifested by covariant and concept and other types of shifts. We'll talk about that in a second.
- Skew detection involves continuous evaluation of data coming to your server once you train your model. To detect these changes, you need continuous monitoring and evaluation of the data.

Detecting distribution skew

	Training	Serving
Joint	$P_{\text{train}}(y, x)$	$P_{\text{serve}}(y, x)$
Conditional	$P_{\text{train}}(y\|x)$	$P_{\text{serve}}(y\|x)$
Marginal	$P_{\text{train}}(x)$	$P_{\text{serve}}(x)$

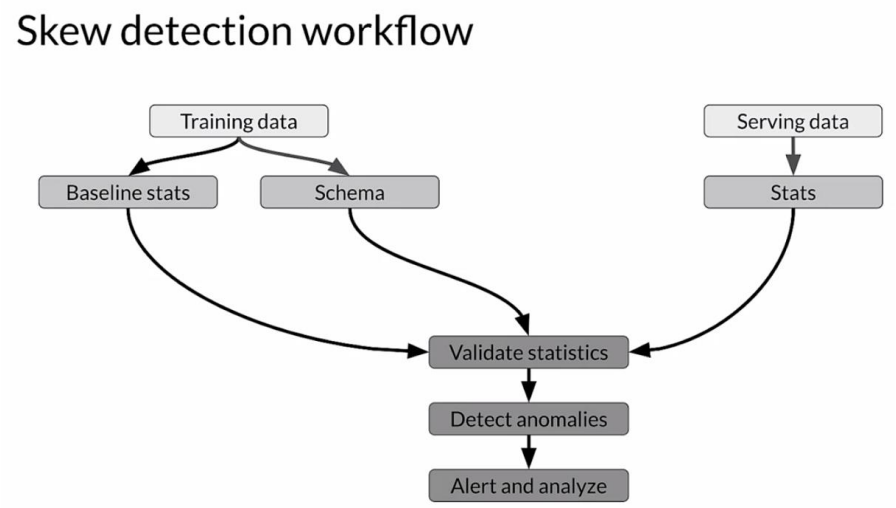
Dataset shift $P_{\text{train}}(y, x) \neq P_{\text{serve}}(y, x)$

Covariate shift $P_{\text{train}}(y|x) = P_{\text{serve}}(y|x)$, $P_{\text{train}}(x) \neq P_{\text{serve}}(x)$

Concept shift $P_{\text{train}}(y|x) \neq P_{\text{serve}}(y|x)$, $P_{\text{train}}(x) = P_{\text{serve}}(x)$

- Let's take a look at a more rigorous definition of the drift and skew that we're talking.
- **Dataset shift** occurs when the joint probability of x (features) and y (labels) is not the same during training and serving. The data has shifted over time.
- **Covariate shift** refers to the change in distribution of the input variables present in training and serving data. In other words, it's where the marginal distribution of x (features) is not the same during training and serving, but the conditional distribution remains unchanged.
- **Concept shift** refers to a change in the relationship between the input and output variables as opposed to the differences in the Data Distribution or input itself. In other words, it's when the conditional distribution of y (labels) given x (features) is not the same during training and serving, but the marginal distribution of x (features) remains unchanged.

Skew detection workflow



- There's a straightforward workflow to detect data skew
- The first stage is looking at training data and computing baseline statistics and a reference schema
- Then you do basically the same with your serving data, where you're going to generate the descriptive statistics
- Then you compare the two, and check for differences between the serving set and your training data i.e. you look for skew and drift
- Significant changes become anomalies and they'll trigger an alert. That alert goes to whoever's monitoring the system, and that can either be a human or another system to analyze the change and decide on the proper course of action. There's got to a remediation plan where you're going to fix and react to that problem

TensorFlow Data Validation (TFDV)

- Understand, validate, and monitor ML data at scale
- Used to analyze and validate petabytes of data at Google every day
- Proven track record in helping TFX users maintain the health of their ML pipelines

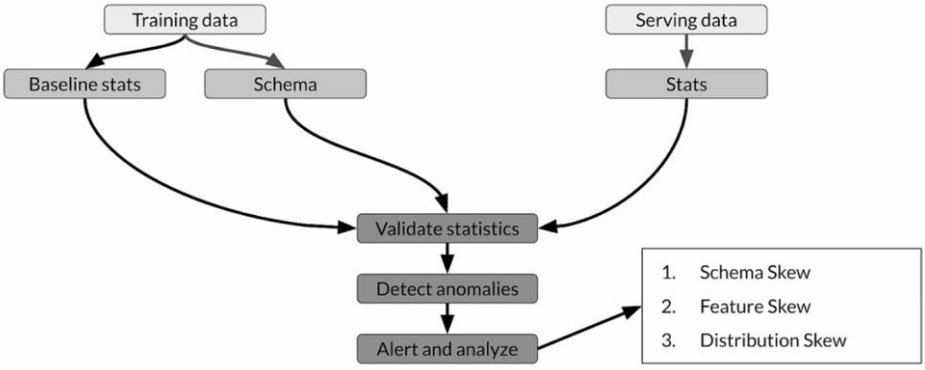
- Now that you've seen some of the data issues and detection workflows and got an idea of the importance of productions scale data validation, let's take a look at TensorFlow Data Validation, which is a library from Google as part of the TFX Ecosystem.
- It'll allow you to do data validation using Python and we'll do an exercise doing that.
- TensorFlow Data Validation (**TFDV**) helps developers understand, validate, and monitor ML data at scale.
- TFDV is used to analyze and validate petabytes of data at Google every day across hundreds or thousands of different applications that are currently in production.
- TFDV helps TFX users maintain the health of their ML pipelines.

TFDV capabilities

- Generates data statistics and browser visualizations
- Infers the data schema
- Performs validity checks against schema
- Detects training/serving skew

- TFDV helps generate data statistics and provides browser visualizations.
- It also helps infer the schema for your data, but you will need to make sure that that schema makes sense.
- Once it has these statistics and schema, it can look for problems are anomalies in your data.
- Then it will look at the training serving skew by comparing the data in your training and your serving datasets.
- One of the common use cases is continuously checking newly arriving data by validating it against the expectations which you have in the reference schema that you generated from your training data.
- The typical setup uses the schema, which is maintained over time, and the statistics are computed over new data. Those statistics are then used to validate the data against the original schema.

Skew detection - TFDV

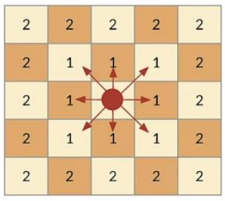


- Remember, we talked about skew detection and with TFDV you can easily detect three different types of skew, schema skew, feature skew, and distributions skew.

Skew - TFDV

- Supported for categorical features
- Expressed in terms of L-infinity distance (Chebyshev Distance):

$$D_{\text{Chebyshev}}(x, y) = \max_i(|x_i - y_i|)$$



- Set a threshold to receive warnings

- TFDV performs skew or drift detection on categorical features
- Skew is expressed in terms of an **L-infinity** distance, which is also known as **Chebyshev** distance.
- If you think of a chessboard, it's the distance metric that is the maximum absolute distance in one-dimension or two or n-dimensional points.
- You can set thresholds so that you'll receive warnings when the drift is higher than what you think is acceptable.

Schema skew

Serving and training data don't conform to same schema:

- For example, `int != float`

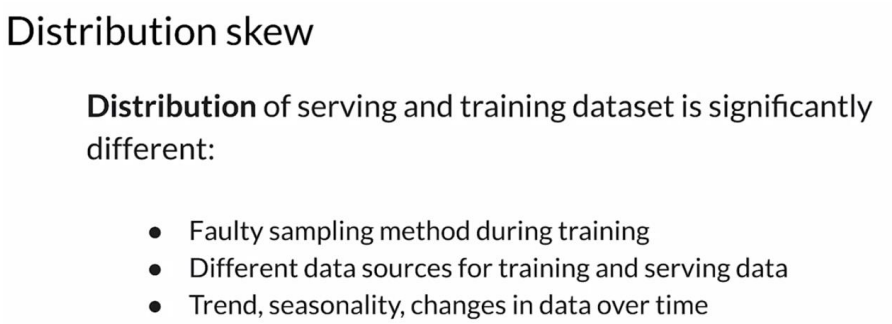
- Schema skew occurs when the serving and training data don't conform to the same schema.
- For example, it could be a change in type, an int, where you're expecting a float, which could be a change in the feature itself.

Feature skew

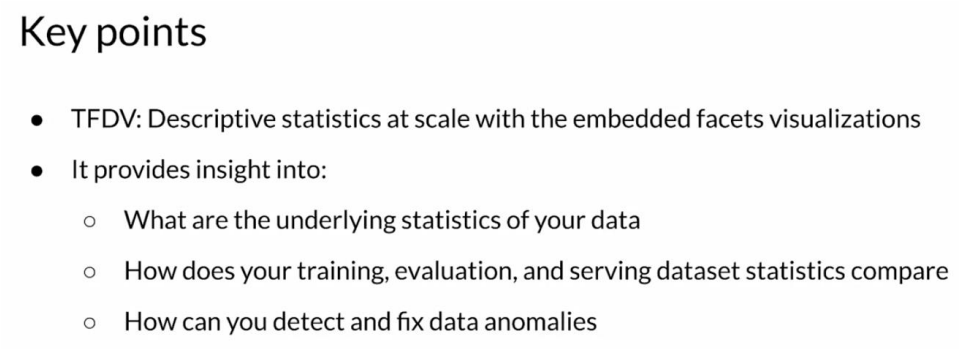
Training **feature values** are different than the serving **feature values:**

- Feature values are modified between training and serving time
- Transformation applied only in one of the two instances

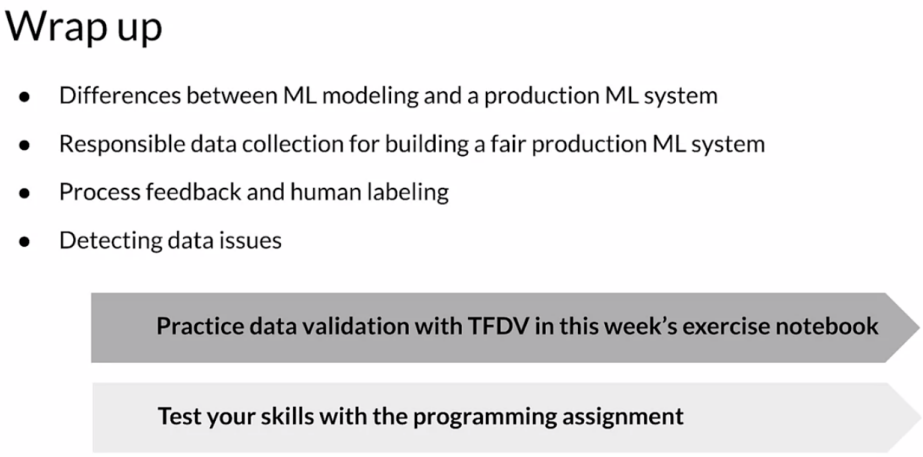
- Feature skew are changes in the feature values between training and serving.
- It could happen as the system uses different data sources during training and serving, or things change due to seasonality and trend as well.
- Sometimes that simply because you have two different code paths when you're trying to do the same transformations, resulting in you getting different results as a consequence.



- Distribution skew is changes in the distribution of individual features in the dataset.
- Features that's in training might have a range of 0-100 when you're training it, and then at serving time, you're seeing data between 5-600.
- That would be a change in the distribution for that feature.
- You may have things like changes is the mean or the median or the standard deviation, all of those are changes in distribution.
- Depending on how severe it is, it may or may not be a problem. The question is, does it affect your model performance enough that you need to make changes to try to account for it?



- TFDV provides you with descriptive statistics at scale. Remember, we could be working with petabytes of data.
- It provides some visualizations as well to help you monitor and really understand that data, for you to understand the underlying statistics for your data and do comparisons. How does your training and evaluation and serving datasets compare?
- Just in terms of statistics, for example, do they have the same mean? How can you calculate and fix or detect rather and fix data anomalies?



Further Reading

- MLops
- Data 1st class citizen
- Runners app
- Rules of ML
- Bias in datasets
- Logstash
- Fluentd
- Google Cloud Logging
- AWS ElasticSearch
- Azure Monitor
- TFDV
- Chebyshev distance
- Konstantinos, Katsiapis, Karmarkar, A., Altay, A., Zaks, A., Polyzotis, N., … Li, Z. (2020). Towards ML Engineering: A brief history of TensorFlow Extended (TFX). http://arxiv.org/abs/2010.02013
- Paleyes, A., Urma, R.-G., & Lawrence, N. D. (2020). Challenges in deploying machine learning: A survey of case studies. http://arxiv.org/abs/2011.09926
- ML code fraction: Sculley, D., Holt, G., Golovin, D., Davydov, E., & Phillips, T. (n.d.). Hidden technical debt in machine learning systems. Retrieved April 28, 2021, from Nips.cc https://papers.nips.cc/paper/2015/file/86df7dcfd896fcaf2674f757a2463eba-Paper.pdf