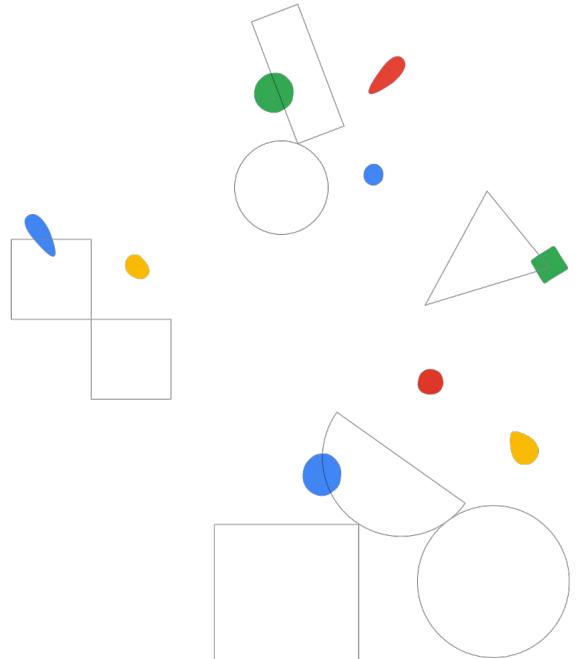




Specialization: Advanced Machine Learning on Google Cloud (Part 1)



Specialization

Advanced Machine Learning on Google Cloud

Production Machine Learning Systems

Image Processing and
Generation with Google Cloud

Sequence Models for Time Series
and Natural Language Processing

Recommendation Systems
with TensorFlow on Google Cloud



Specialization

Advanced Machine Learning on Google Cloud

Production Machine Learning Systems

Image Processing and
Generation with Google Cloud

Sequence Models for Time Series
and Natural Language Processing

Recommendation Systems
with TensorFlow on Google Cloud



Specialization

Advanced Machine Learning on Google Cloud

Production Machine Learning Systems

Image Processing and
Generation with Google Cloud

Sequence Models for Time Series
and Natural Language Processing

Recommendation Systems
with TensorFlow on Google Cloud



Specialization

Advanced Machine Learning on Google Cloud

Production Machine Learning Systems

Image Processing and
Generation with Google Cloud

Sequence Models for Time Series
and Natural Language Processing

Recommendation Systems
with TensorFlow on Google Cloud



Specialization

Advanced Machine Learning on Google Cloud

Production Machine Learning Systems

Image Processing and
Generation with Google Cloud

Sequence Models for Time Series
and Natural Language Processing

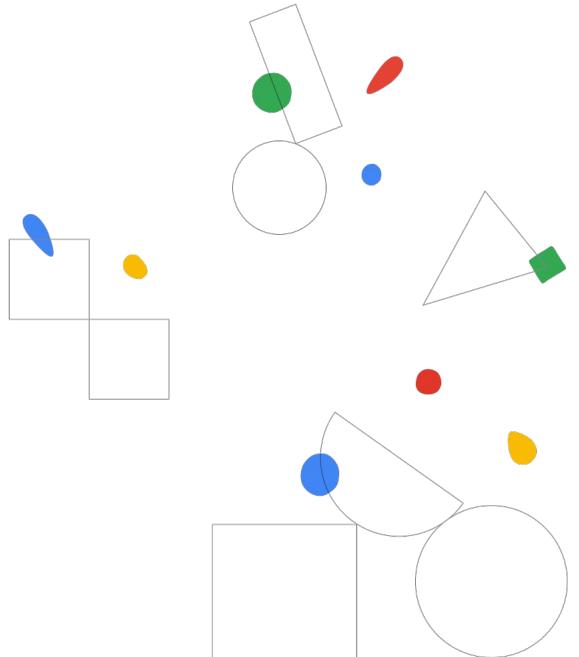
Recommendation Systems
with TensorFlow on Google Cloud





Welcome

Production ML systems



Specialization

Advanced Machine Learning on Google Cloud

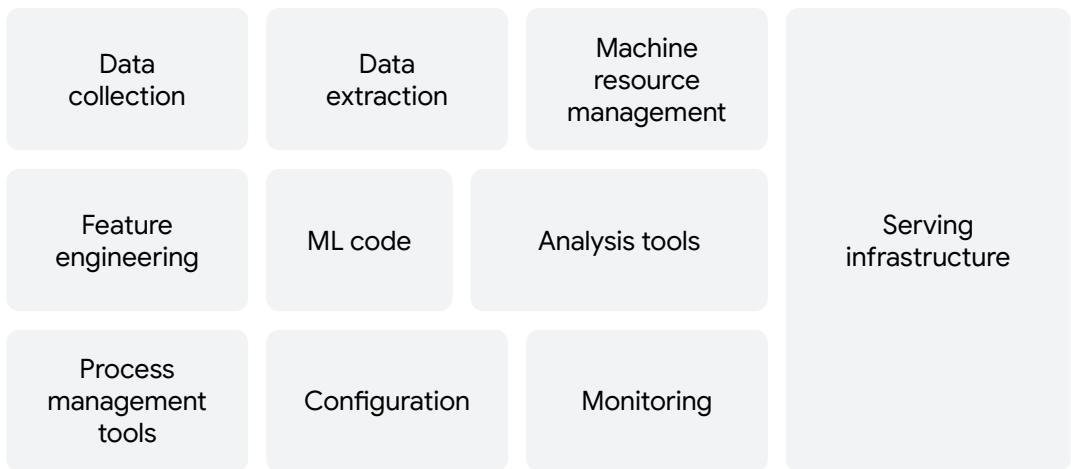
Production Machine Learning Systems

Image Processing and
Generation with Google Cloud

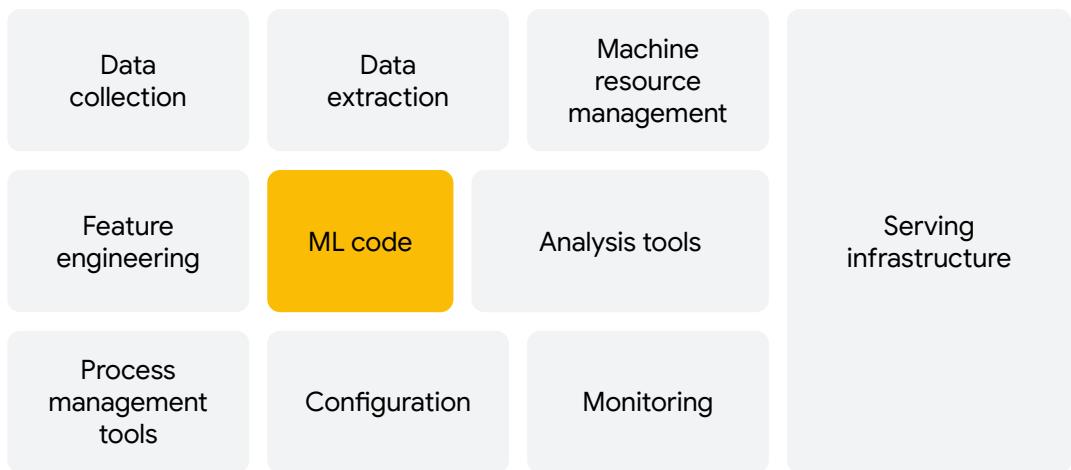
Sequence Models for Time Series
and Natural Language Processing

Recommendation Systems
with TensorFlow on Google Cloud





Real-world production ML systems are large ecosystems, of which the model code is just a small part. The rest consists of code that performs critical functions, like data extraction, feature engineering, monitoring, and a serving infrastructure.



This course is devoted to exploring the characteristics that make for a robust ML system beyond its ability to make good predictions.

Agenda



- | | |
|-----------|---------------------------------------|
| 01 | Architecting Production ML Systems |
| 02 | Designing Adaptable ML Systems |
| 03 | Designing High-Performance ML Systems |
| 04 | Building Hybrid ML Systems |



Agenda



- | | |
|-----------|---------------------------------------|
| 01 | Architecting Production ML Systems |
| 02 | Designing Adaptable ML Systems |
| 03 | Designing High-Performance ML Systems |
| 04 | Building Hybrid ML Systems |



Agenda



- | | |
|-----------|---------------------------------------|
| 01 | Architecting Production ML Systems |
| 02 | Designing Adaptable ML Systems |
| 03 | Designing High-Performance ML Systems |
| 04 | Building Hybrid ML Systems |



Agenda



- | | |
|-----------|---------------------------------------|
| 01 | Architecting Production ML Systems |
| 02 | Designing Adaptable ML Systems |
| 03 | Designing High-Performance ML Systems |
| 04 | Building Hybrid ML Systems |

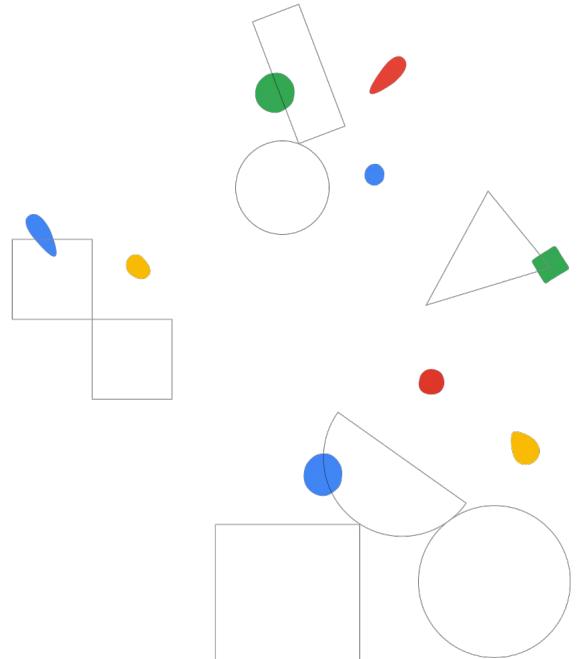




Architecting ML systems

Module 01

Architecting production ML systems



Welcome to
Architecting ML systems



Welcome to Architecting ML systems, the second module of the Production Machine Learning Systems course.

Objectives

- 1 What makes up an architecture



In this module, we'll explore what makes up an architecture

Objectives

- 1 What makes up an architecture
- 2 Why and how to make good systems design decisions



as well as why and how to make good systems design decisions.

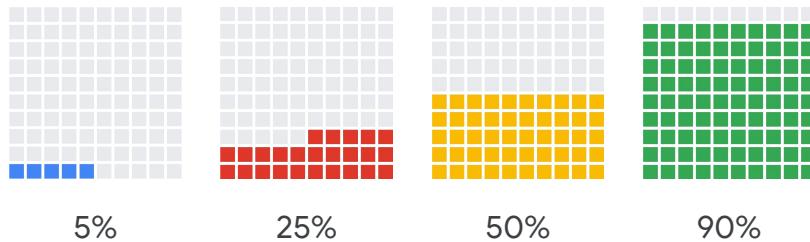
What percent of system code
does the ML model account for?



Let me ask you a question.

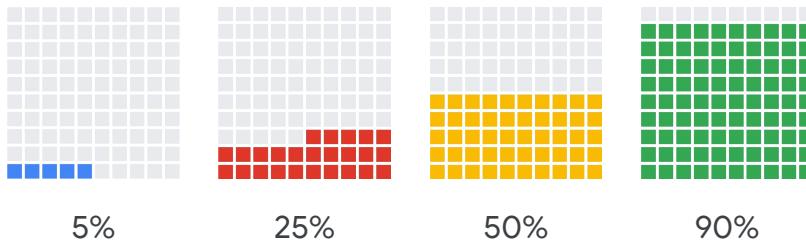
What percent of system code does the ML model account for?

What percent of system code
does the ML model account for?



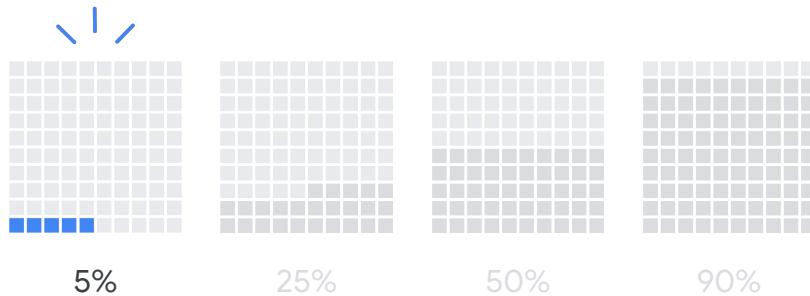
- (a) 5%
- (b) 25%
- (c) 50%
- (d) 90%

What percent of system code does the ML model account for?

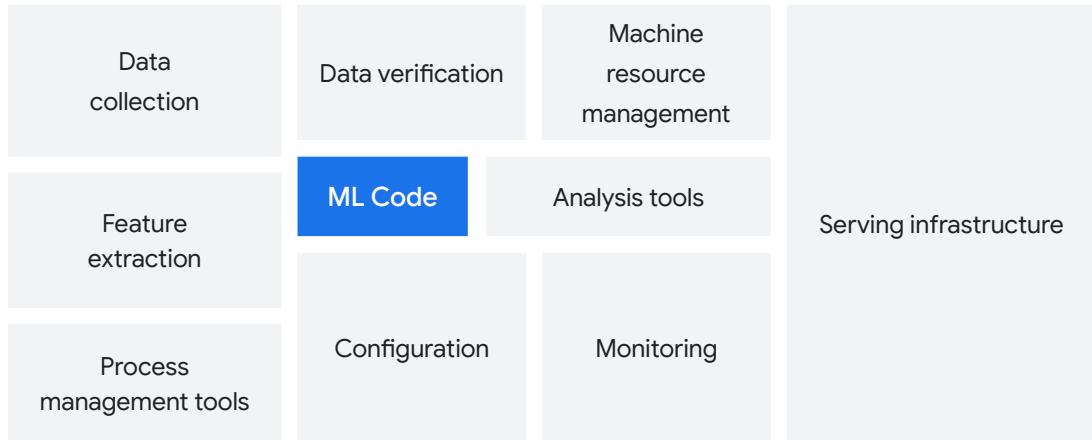


You'll recall from earlier in this specialization, we showed how time gets distributed among the different tasks necessary to launch an ML model and, surprisingly, modeling accounted for far less than most people expect. The same is true with respect to the code.

What percent of system code
does the ML model account for?



So, the answer is that ML model code typically accounts for about 5% of the overall code base.



The reason that ML models account for such a small percentage is that to keep a system running in production requires doing a lot more than just computing the model's outputs for a given set of inputs.

Learn how to...

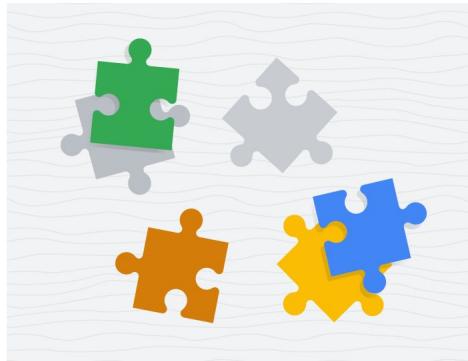
- ✓ Choose a training and serving paradigm
- ✓ Serve ML models scalably
- ✓ Design an architecture from scratch



In this module, you'll see what else a production ML system needs to do and how you can meet those needs.

Upon completing this module, you should acquire the knowledge to choose an appropriate training and serving paradigm, serve ML models scalably, and design an architecture from scratch.

Generic systems → Your ML Production System



And while our focus is on Google Cloud, it's important that you always try and reuse generic systems when possible—many of which are open-source frameworks.



TensorFlow



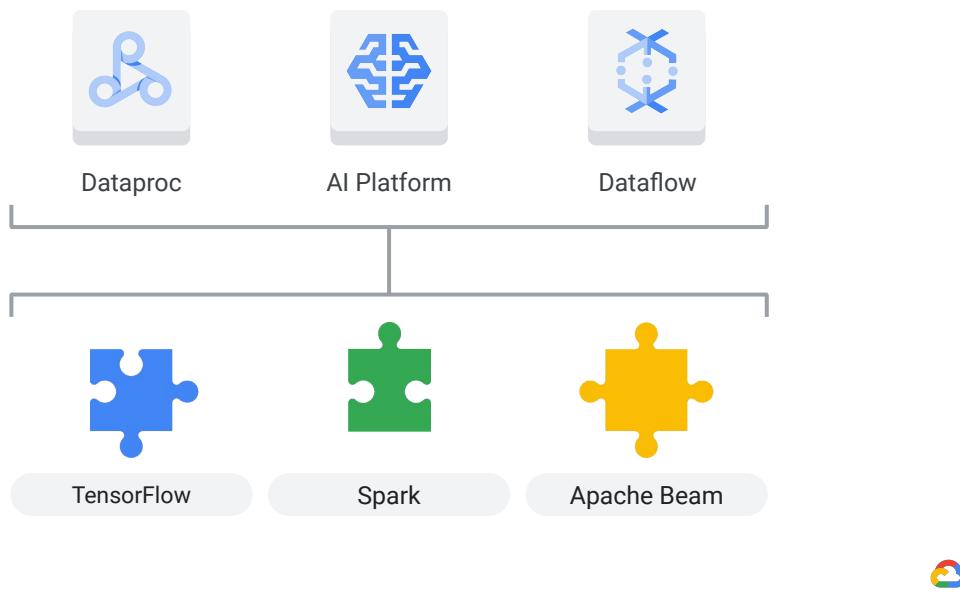
Spark



Apache Beam



What's true of software frameworks like TensorFlow, Spark, or Apache Beam



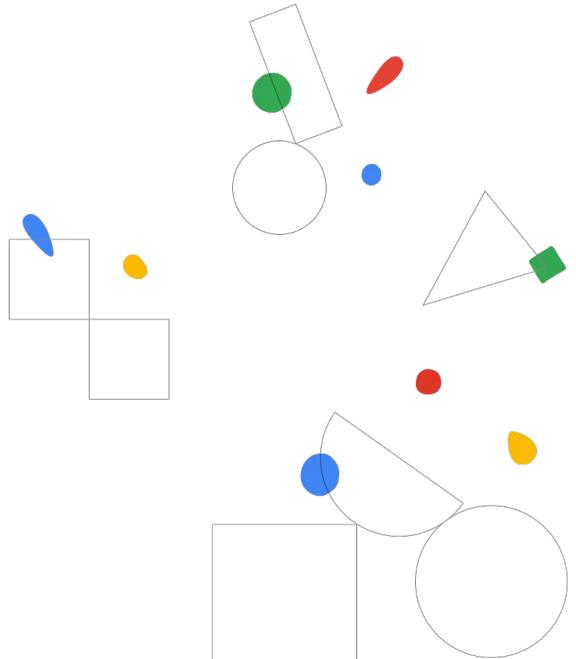
is also true of the underlying infrastructure on which you execute them. Rather than spend time and effort provisioning infrastructure, you can use managed services such as Dataproc, AI Platform, or Dataflow to execute your Spark, TensorFlow, and Beam code.

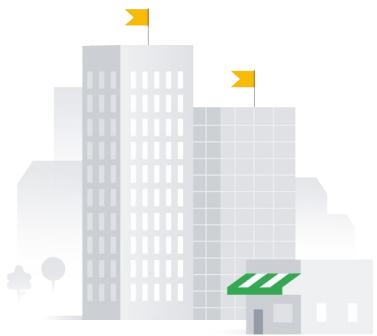


Data extraction, analysis, and preparation

Module 01

Architecting production ML systems





After you

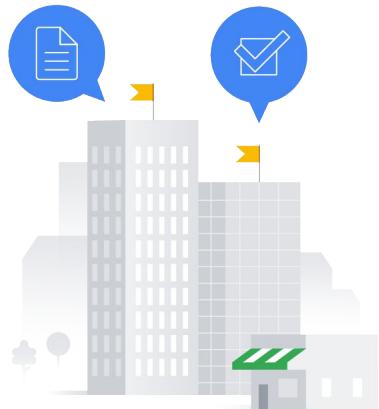


Define business use case



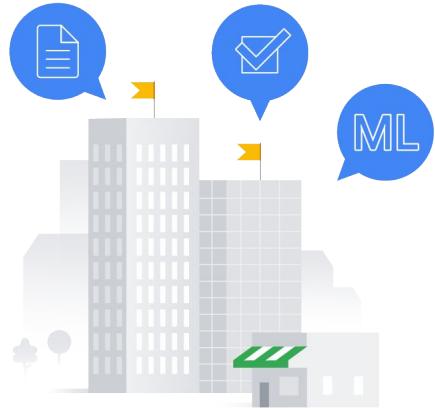
define the business use case and

-  Define business use case
-  Establish success criteria



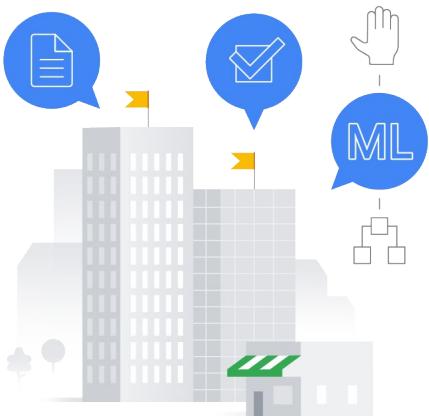
establish the success criteria,

- Define business use case
- Establish success criteria
- Deliver ML model to production



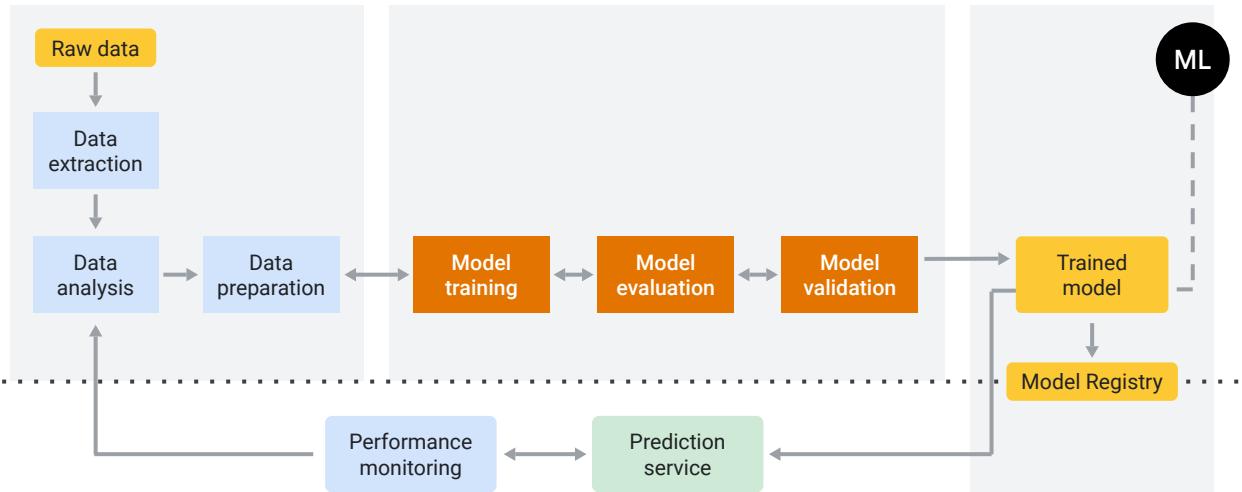
the process of delivering an ML model to production typically involves several steps,

- Define business use case
- Establish success criteria
- Deliver ML model to production



which can be completed manually or by an automated pipeline.

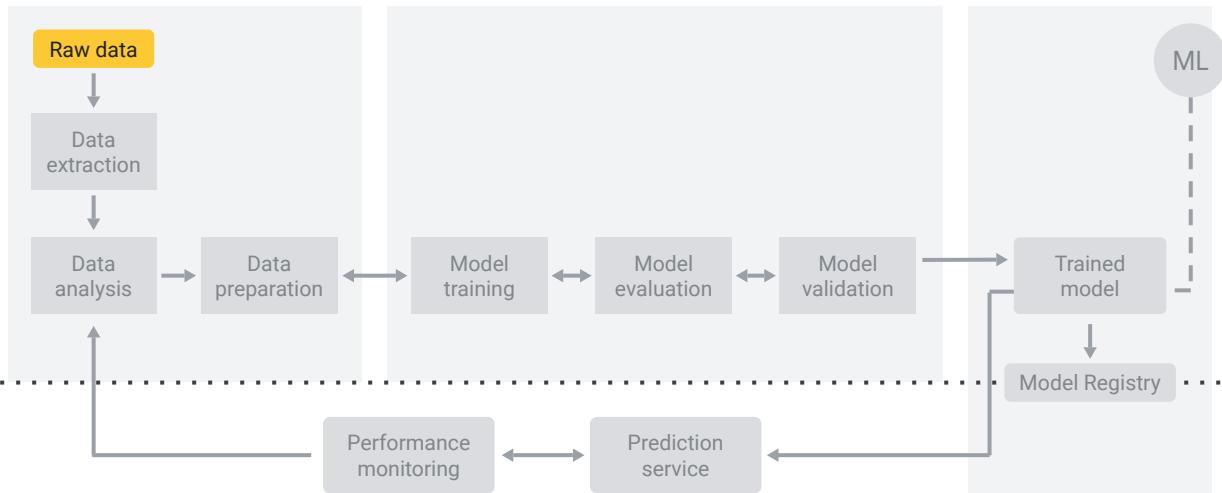
Staging/pre-production/production environments



Experimentation/development/test environments

The first three steps deal with data.

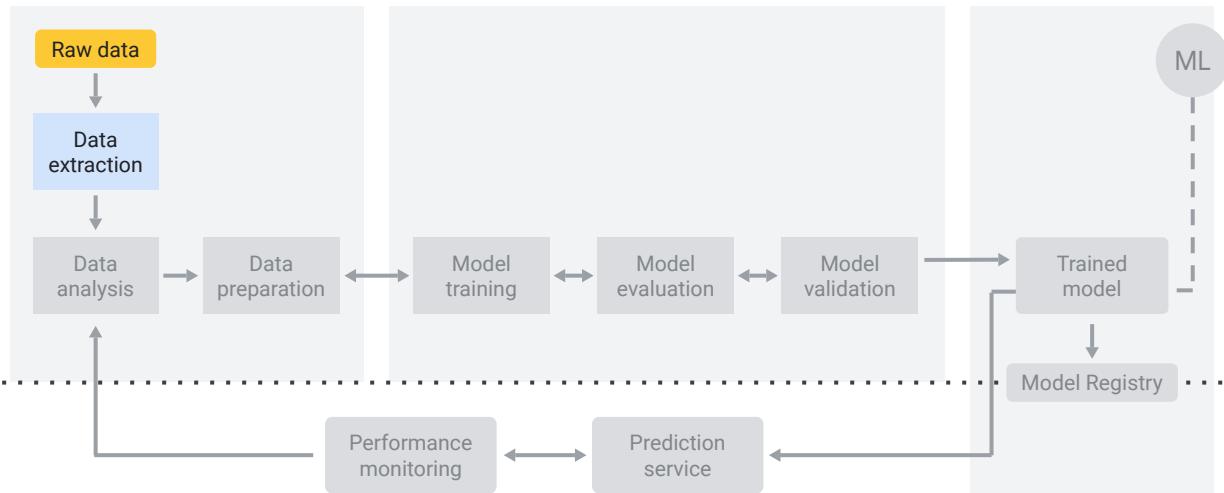
Staging/pre-production/production environments



Experimentation/development/test environments

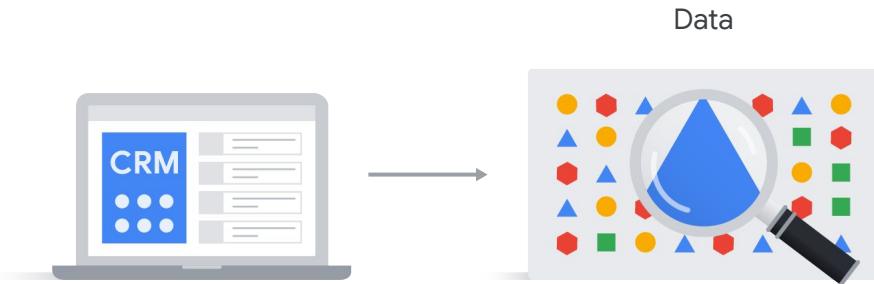
Data needs to be ingested, which means it's extracted from a **raw data** source.

Staging/pre-production/production environments

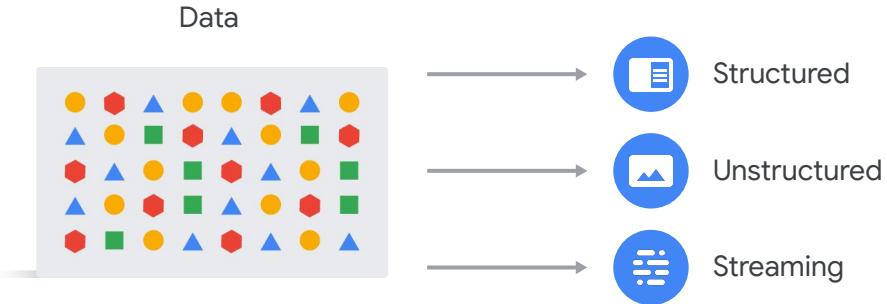


Experimentation/development/test environments

With **data extraction**, you retrieve data from various sources. Those sources can be "streaming", in "real-time", or "batch".

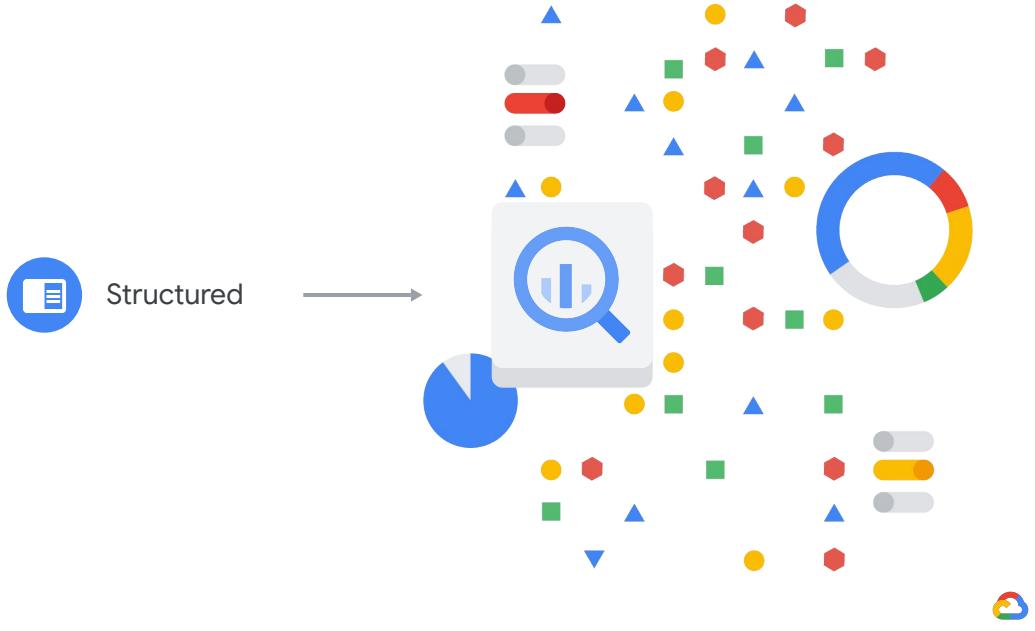


For example, you may extract data from a customer relationship management system, or CRM, to analyze customer behavior.

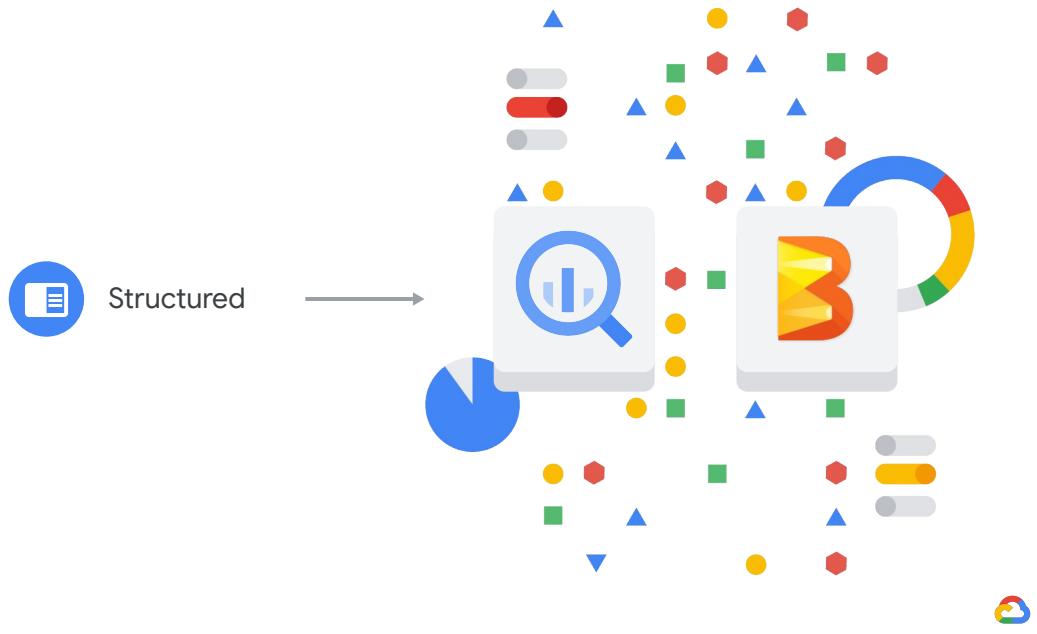


This data may be “structured”, where the file type is a .csv, .txt, JSON, or . XML format, or, you may have an “unstructured” data source where you have images of your customers, or text comments from chat sessions with your customers.

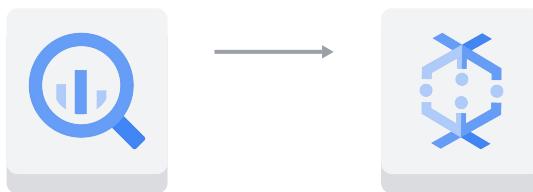
You may have to extract “streaming” data from your company’s transportation vehicles that are equipped with sensors that transmit data in real time.



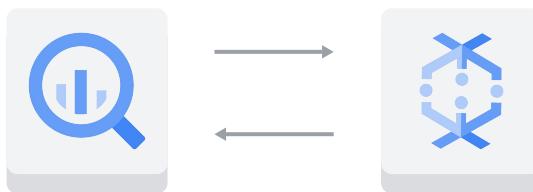
If the data you want to train your model on or get predictions for is structured, you might retrieve it from a data warehouse – such as BigQuery.



Or, you can use Apache Beam's IO module.

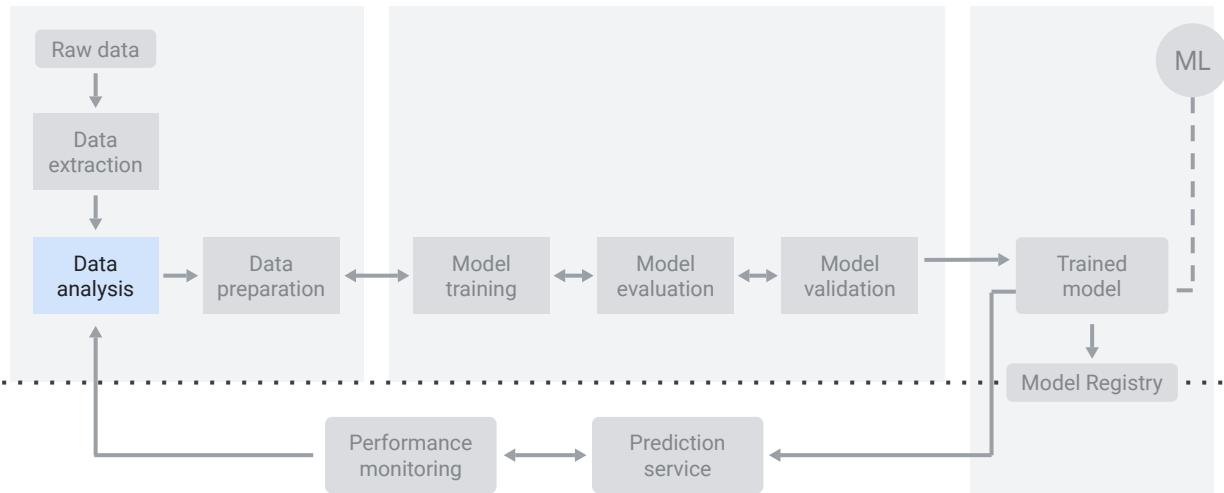


In this Dataflow example we're loading data from BigQuery, calling predict on every record,



and then writing the results back into BigQuery.

Staging/pre-production/production environments



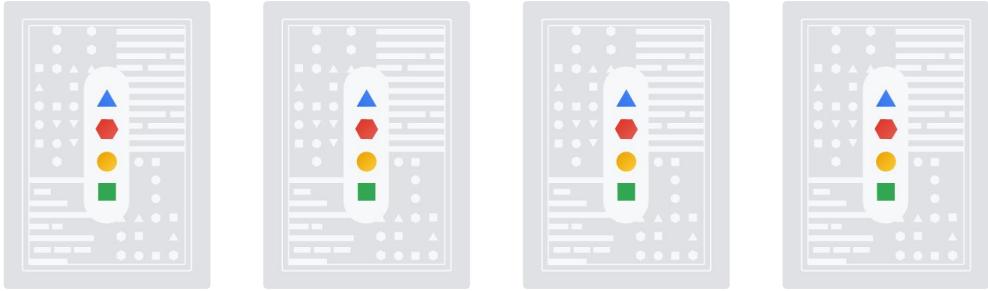
Experimentation/development/test environments

In **data analysis**, you analyze the data you've extracted.

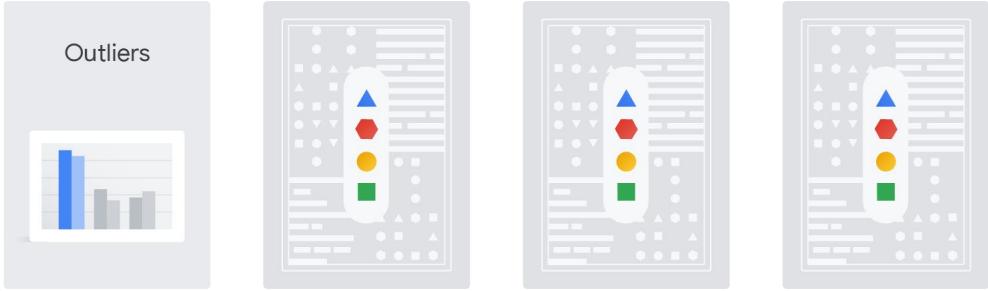
Exploratory data analysis (EDA)



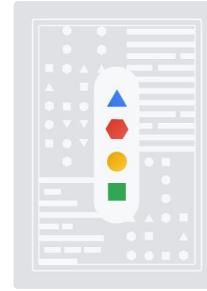
For example, you can use exploratory data analysis (or EDA).



This involves using graphics and basic sample statistics to explore your data



such as looking for outliers or anomalies, trends, and data distributions. [not sure if time allows to turn them one by one but I have added the breakdown just in case]



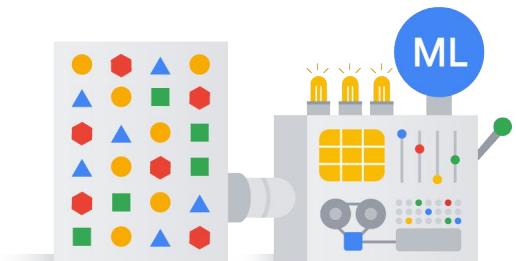
such as looking for outliers or anomalies, trends, and data distributions.



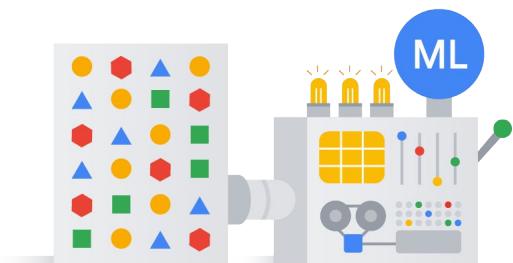
such as looking for outliers or anomalies, trends, and data distributions.



such as looking for outliers or anomalies, trends, and data distributions.



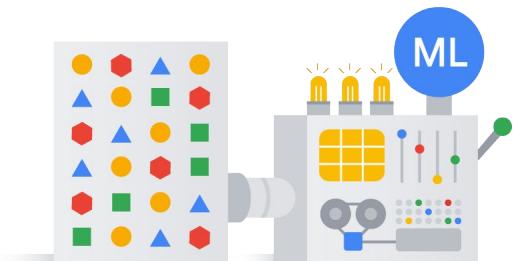
It may not be apparent how changes in the distribution of your data could affect your model, so let's consider a scenario.



Number	Product Name
112	Blue T-Shirt
231	Dog Frisbee
1333	Mobile Phone Charger



In this scenario, an upstream data source encodes a categorical feature using a number, such as a product number.



Number	Number	Product Name
112	231	Blue T-Shirt
231	231231	Dog Frisbee
1333	112	Mobile Phone Charger



One day, the product numbering convention changes and now the customer uses a totally different mapping, using some old numbers and some new numbers.

How would you know that
this had happened?

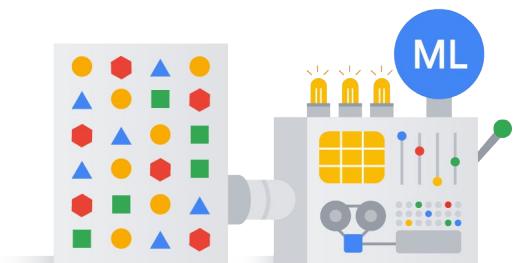


How would you know that this had happened?

How would you
debug your ML model?



How would you debug your ML model?

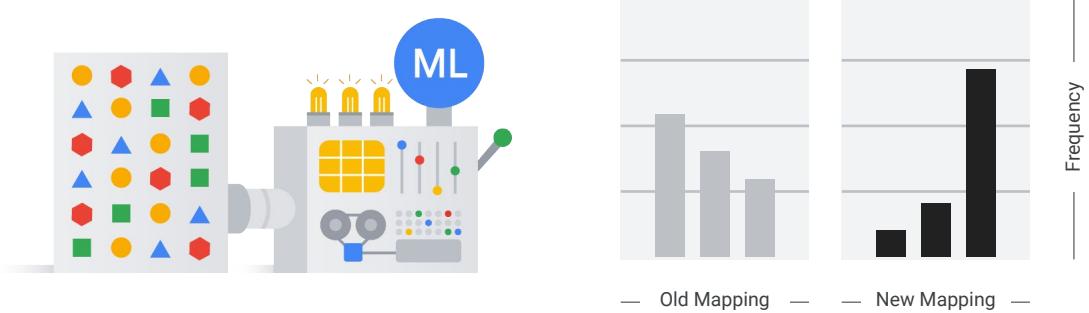


Number	Number	Product Name
112	231	Blue T-Shirt
231	231231	Dog Frisbee
1333	112	Mobile Phone Charger



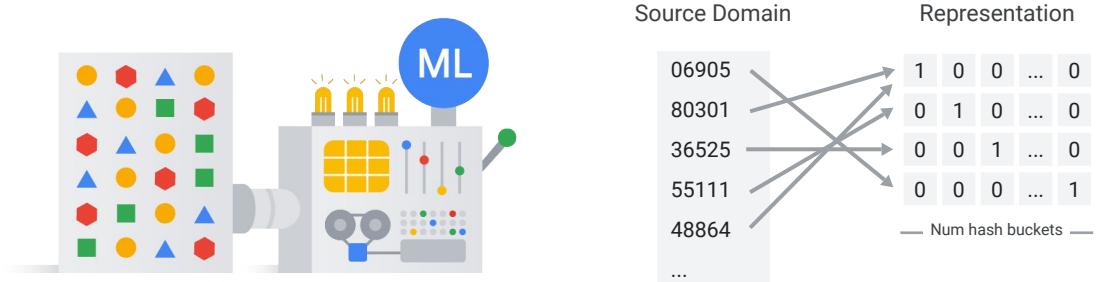
The output of your model would tell you if there's a drop in performance, but it won't tell you why. The raw inputs themselves would appear valid because we're still getting numbers.

In order to recognize this change, you would need to look at changes in the distribution of your inputs.

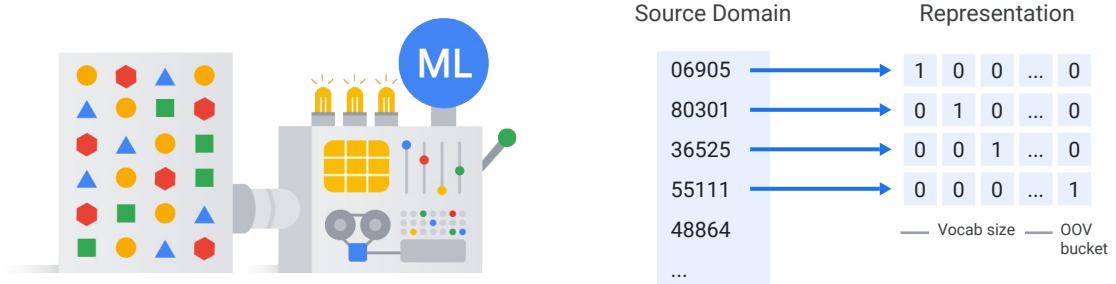


In doing so, you might find that while earlier, the most commonly occurring value might have been a 4, in the new distribution, 4 might never even occur and the most commonly occurring value might be a 10.

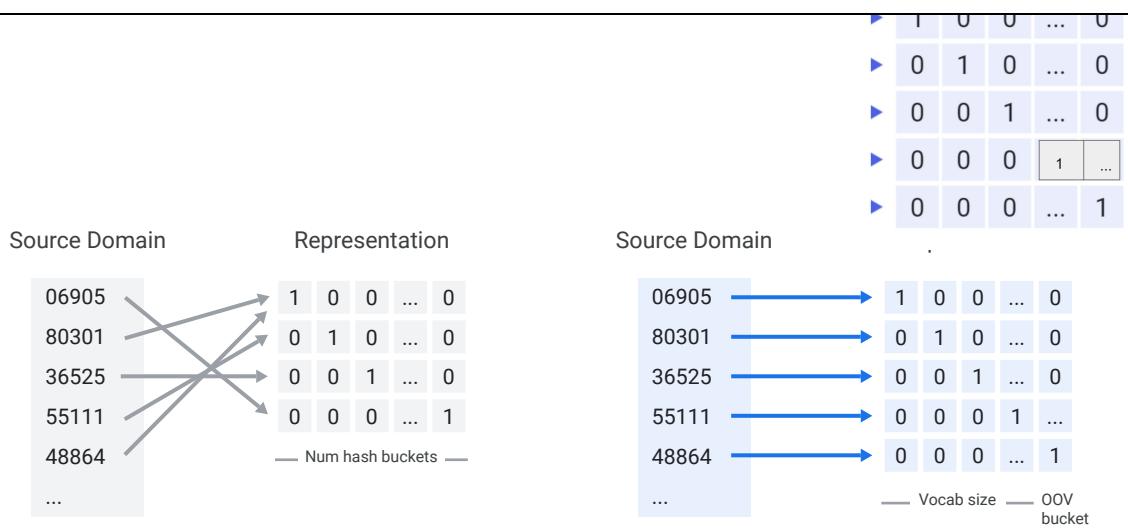




Depending on how you implemented your feature columns, these new values might be mapped to one component of a one-hot encoded vector or to many components.

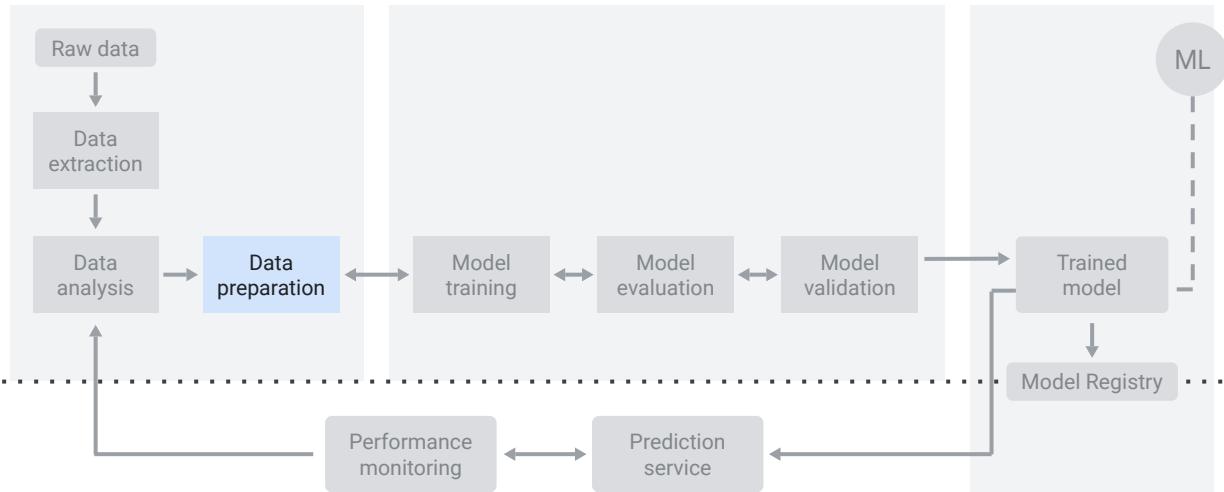


If, for example, you used a categorical column with a hash bucket, the new values would be distributed according to the hash function, and so one hash bucket might now get more and different values than before. If you used a vocabulary, then the new values would map to OOV buckets.



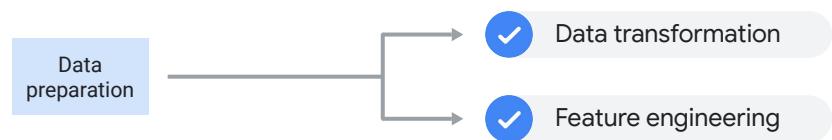
But what's important is that, for a given tensor, its relationship to the label before, and its relationship to the label now, are likely to be very different.

Staging/pre-production/production environments



Experimentation/development/test environments

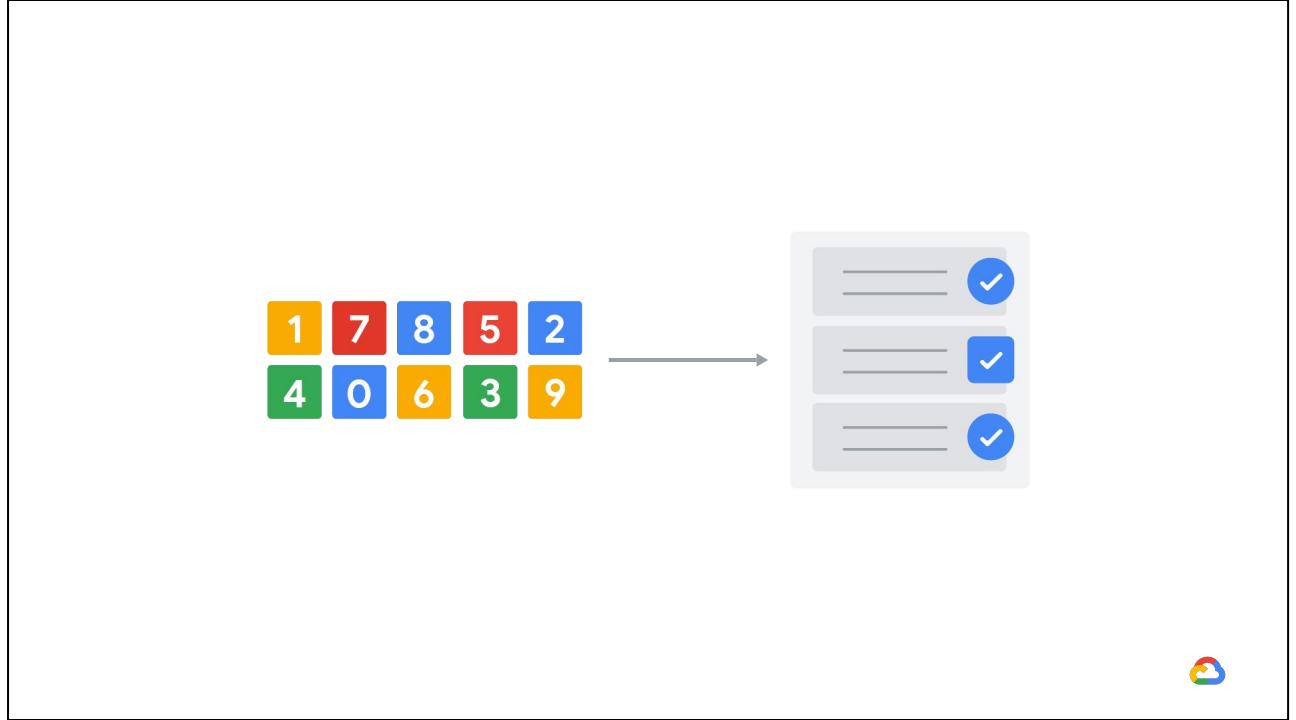
So, after you've extracted and analyzed your data, the next step in the process is **data preparation**.



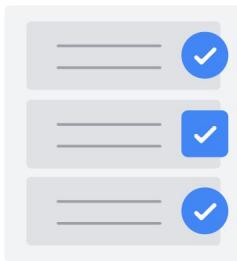
Data preparation includes data transformation and feature engineering, which is the process of changing, or converting, the format, structure, or values of data you've extracted, into another format or structure.



Most ML models require categorical data

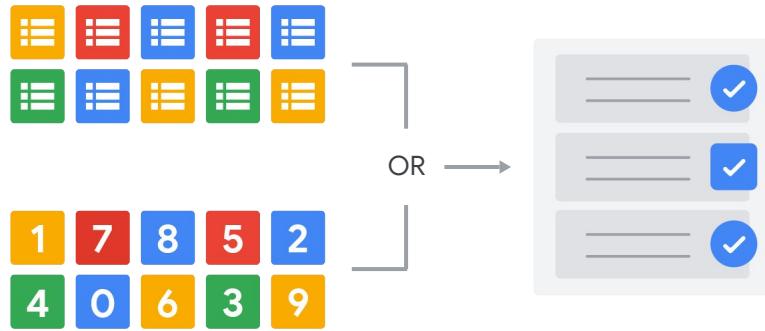


1 7 8 5 2
4 0 6 3 9



to be in a numerical format,





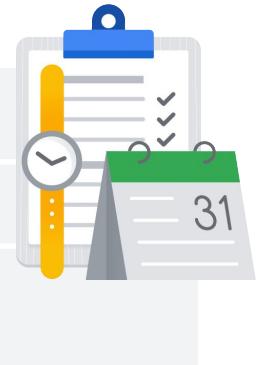
but some models work either with numeric or categorical features,



while others can handle mixed-type features.

Preprocessing for dates using SQL in BigQuery ML

```
EXTRACT(DAYOFWEEK FROM pickup_datetime) AS dayofweek,
```



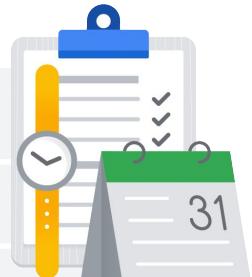
For example, here are three types of preprocessing for dates using SQL in BigQuery ML, where we are:

Extracting the parts of the date into different columns: Year, month, day, etc.

Preprocessing for dates using SQL in BigQuery ML

```
EXTRACT(DAYOFWEEK FROM pickup_datetime) AS dayofweek,
```

```
EXTRACT(HOUR FROM pickup_datetime) AS hourofday,
```



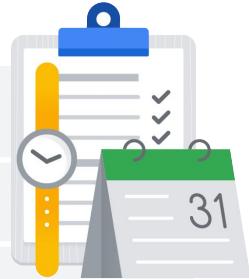
Extracting the time period between the current date and columns in terms of years, months, days, etc.

Preprocessing for dates using SQL in BigQuery ML

```
EXTRACT(DAYOFWEEK FROM pickup_datetime) AS dayofweek,
```

```
EXTRACT(HOUR FROM pickup_datetime) AS hourofday,
```

```
CONCAT(CAST(EXTRACT(DAYOFWEEK FROM pickup_datetime) AS STRING),  
       CAST(EXTRACT(HOUR FROM pickup_datetime) AS STRING)) AS hourofday,
```



And extracting some specific features from the date: Name of the weekday, weekend or not, holiday or not, etc.

Example



The diagram illustrates the process of extracting data from an SQL database. On the left, there is a blue cylinder icon with the word "SQL" written on it. An arrow points from this icon to the right, where a screenshot of a Data Studio interface is shown. The interface features a sidebar with three toggle switches and a main area displaying a table. The table has two columns: "dayofweek" and "hourofday". The data is as follows:

	dayofweek	hourofday
1	Week 6	4 AM
2	Week 5	3 AM
3	Week 4	2 AM
4	Week 7	1 AM
5	Week 3	12 AM

Data Studio output: from EXTRACT dates/time queries



Now here is an example of the dayofweek and hourofday queries extracted using SQL and visualized as a table in Data Studio.

For all non-numeric columns, other than
TIMESTAMP, BigQuery ML performs a
one-hot encoding transformation



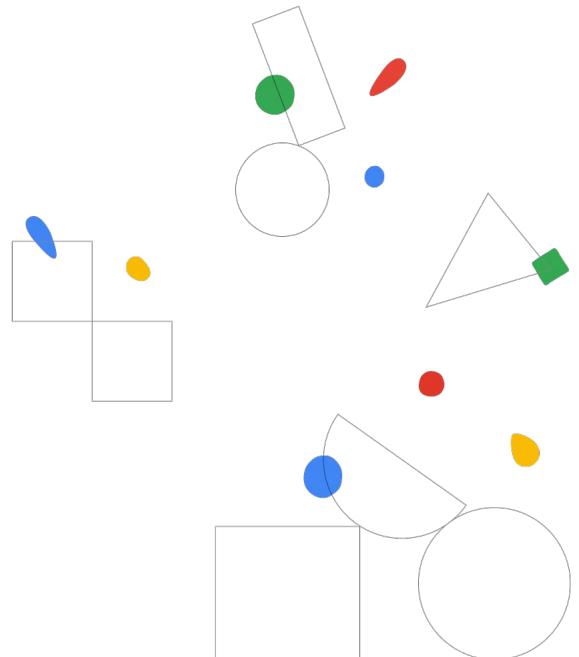
Please note that for all non-numeric columns, other than TIMESTAMP, BigQuery ML performs a one-hot encoding transformation. This transformation generates a separate feature for each unique value in the column.



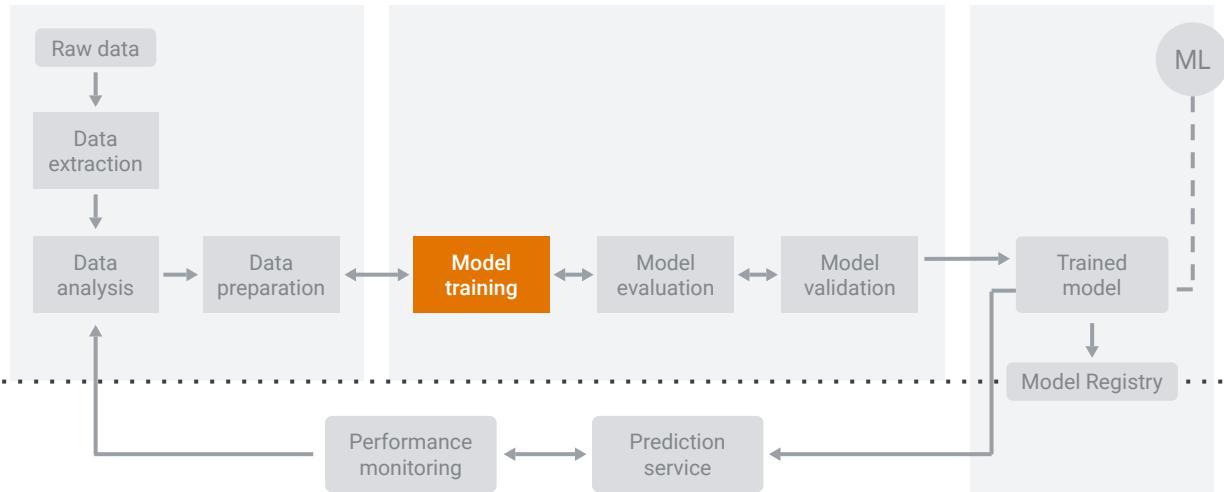
Model training, evaluation, and validation

Module 01

Architecting production ML systems

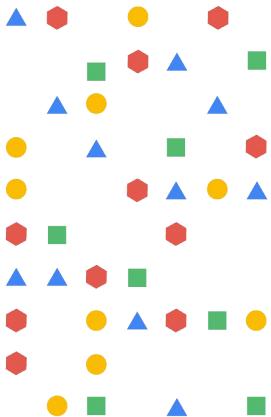


Staging/pre-production/production environments

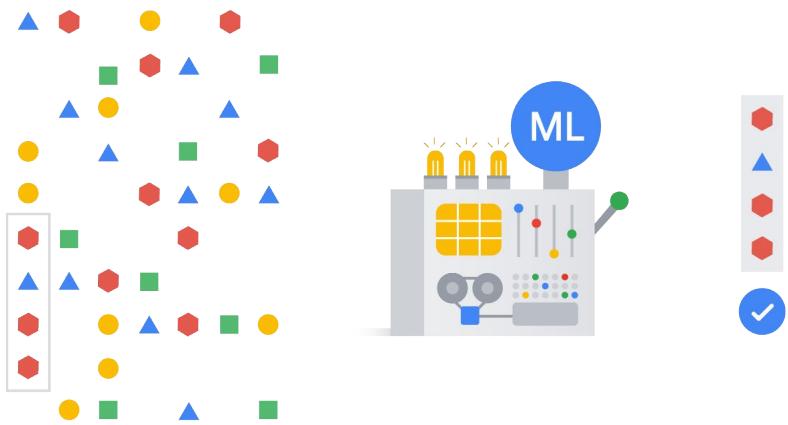


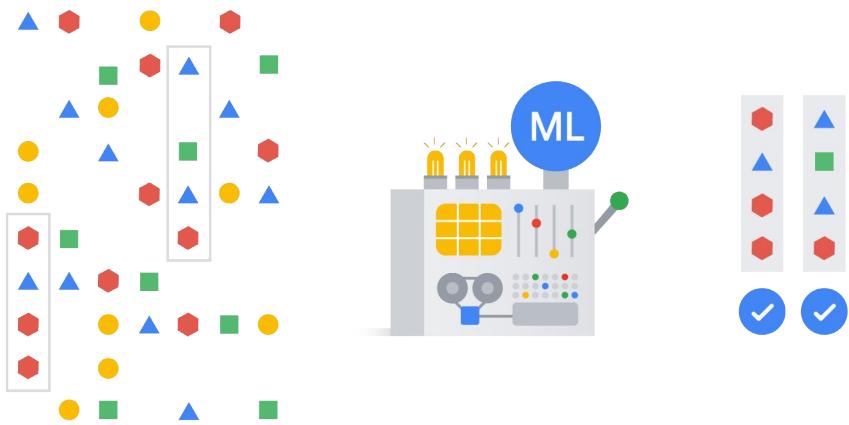
Experimentation/development/test environments

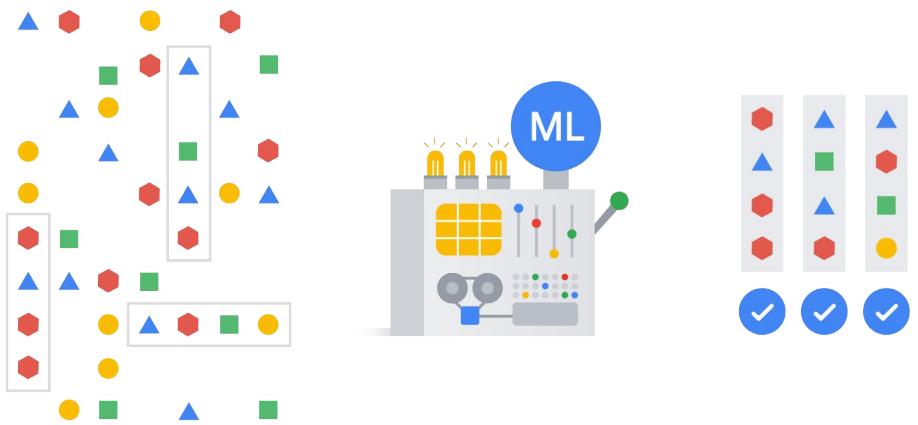
The next step in our workflow is choosing a model that you'll train. In **model training**, you implement different machine learning algorithms with the prepared data to train various ML models.

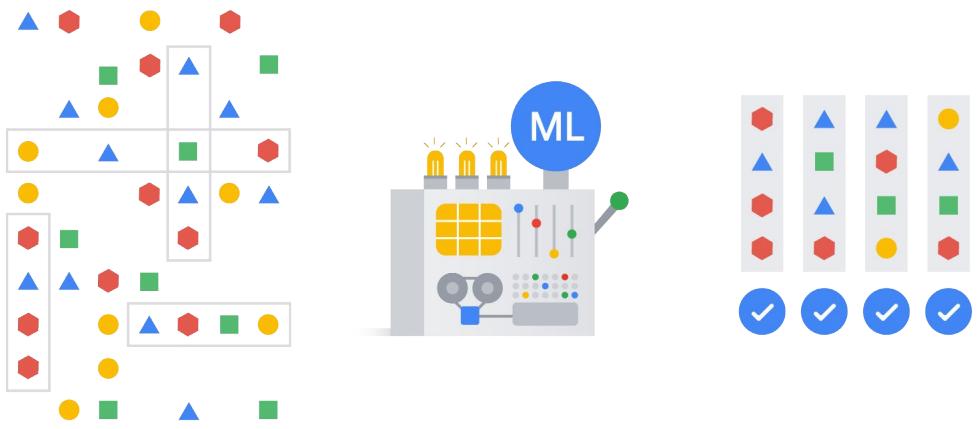


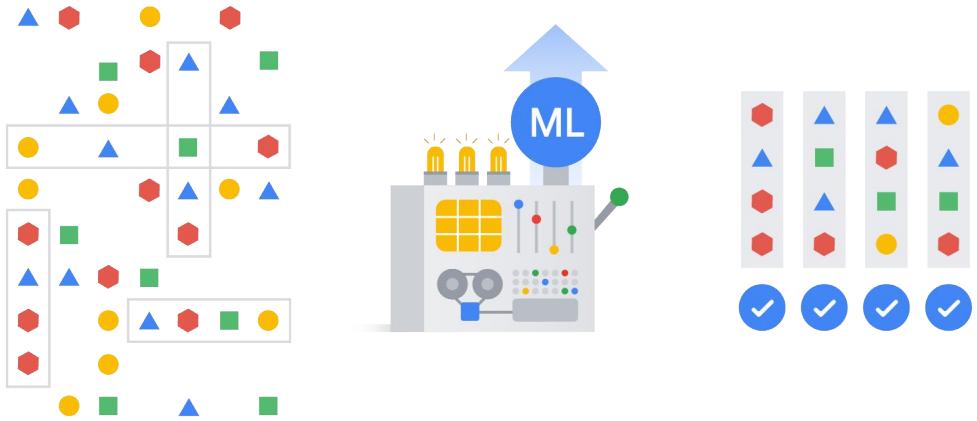
[SLIDES 78-82] Essentially, model training is the process of feeding an ML algorithm with data to help identify and learn good values for the feature set.





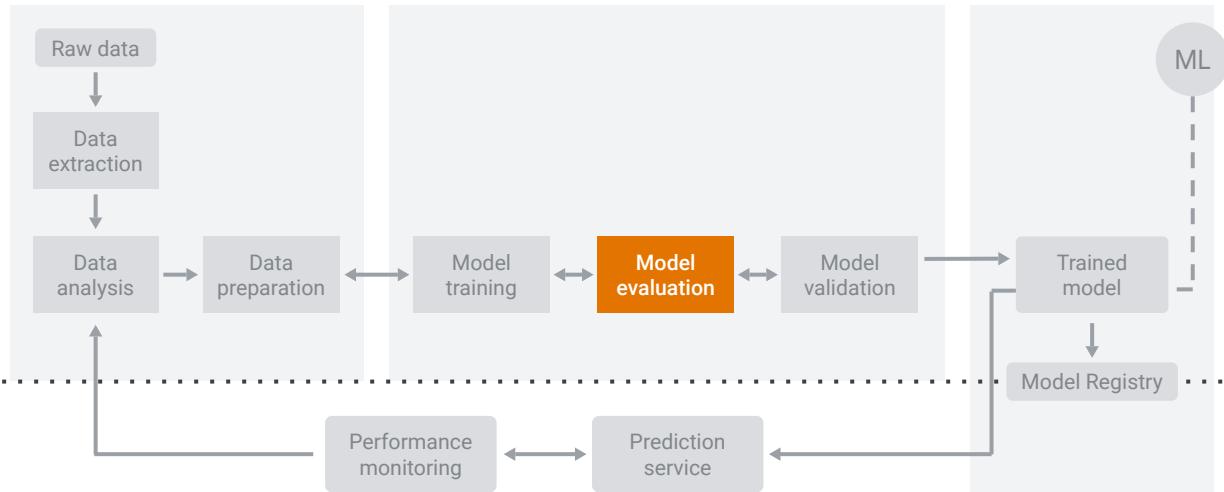






So, you use your data to incrementally improve your model's predictive ability.

Staging/pre-production/production environments



Experimentation/development/test environments

Model evaluation aims to estimate the generalization accuracy of a model on future, unseen, or out-of-sample data. While training a model is a key step in the pipeline, how the model generalizes on unseen data is an equally important aspect that should be considered in every machine learning pipeline.

We need to know whether the model
actually works and, consequently,
if we can trust its predictions



This means that we need to know whether the model actually works and, consequently, if we can trust its predictions.



Could the model be merely memorizing the data it's fed with,



and therefore unable to make good predictions on future samples, or samples that it hasn't seen before, such as the test set?





Model evaluation consists of a person or group of people evaluating or assessing the model with respect to some business-relevant metric, like AUC (area under the curve) or cost-weighted error. If the model meets their criteria,

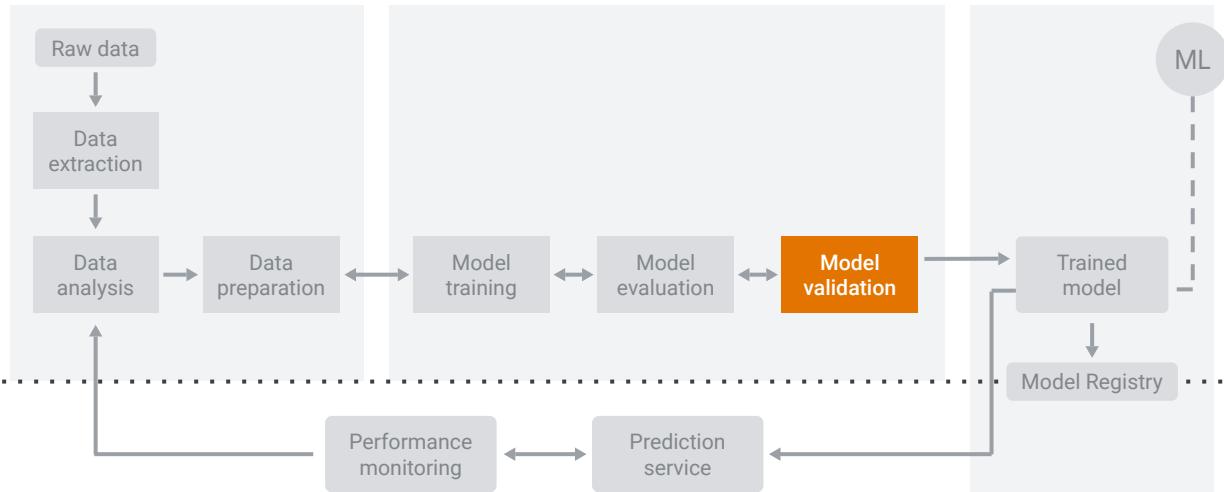


then the model is moved from the assessment phase to development. For example, in the development phase, you may want to make modifications to hyperparameter values to increase the model's performance, which correlates to improvements in evaluation results.



After development, the model is then ready for a live experiment or real-world test of the model.

Staging/pre-production/production environments



Experimentation/development/test environments

In contrast to the Model Evaluation component, which is performed by humans, the **Model Validation** component evaluates the model against fixed thresholds and alerts engineers when things go awry. One common test is to look at performance by slice.



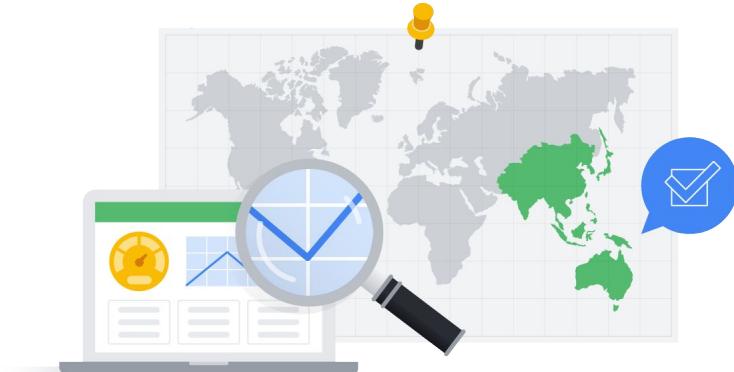
Let's say, for example, business stakeholders care strongly



about a particular geographic market region.



An alert can be set to notify engineers when the accuracy by country begins to skew downward.



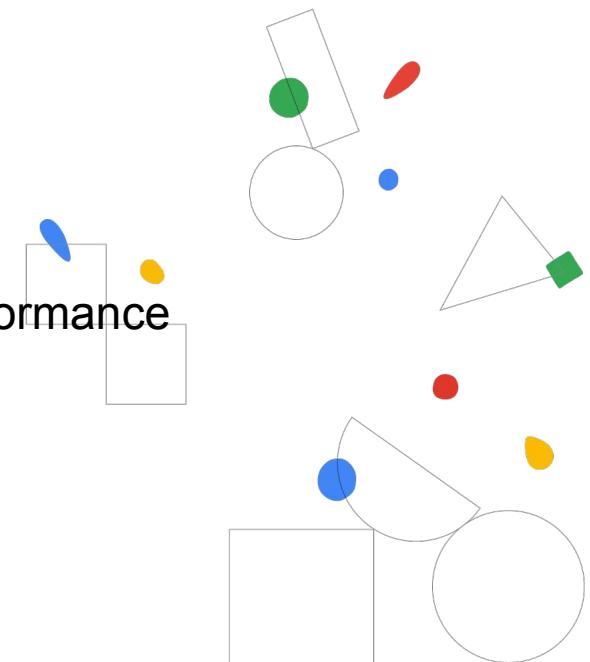
The model evaluation and validation components have one responsibility: to ensure that the models are 'good' before moving them into a production environment.



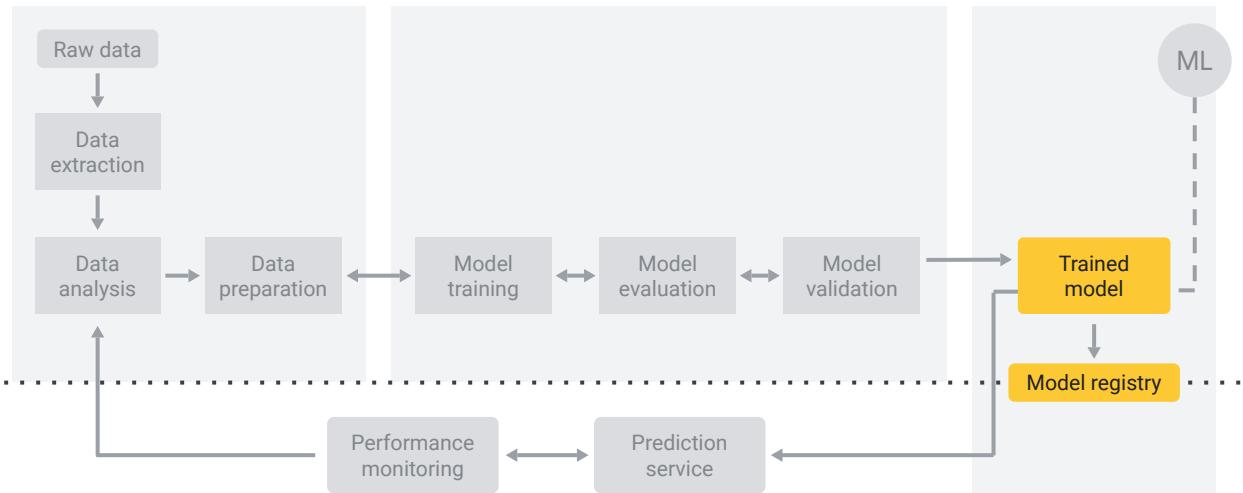
Trained model, prediction service, and performance monitoring

Module 01

Architecting production ML systems



Staging/pre-production/production environments



Experimentation/development/test environments

The output of model validation is a **trained model** that can be pushed to the **model registry**. A machine learning model registry is a centralized tracking system that stores lineage, versioning, and related metadata for published machine learning models.



A registry may capture governance data required for auditing purposes,



Who trained and published a model



such as who trained and published a model,

Who trained and published a model

Which datasets were used for training



which datasets were used for training,

- Who trained and published a model
- Which datasets were used for training
- The values of metrics measuring predictive performance



the values of metrics measuring predictive performance,

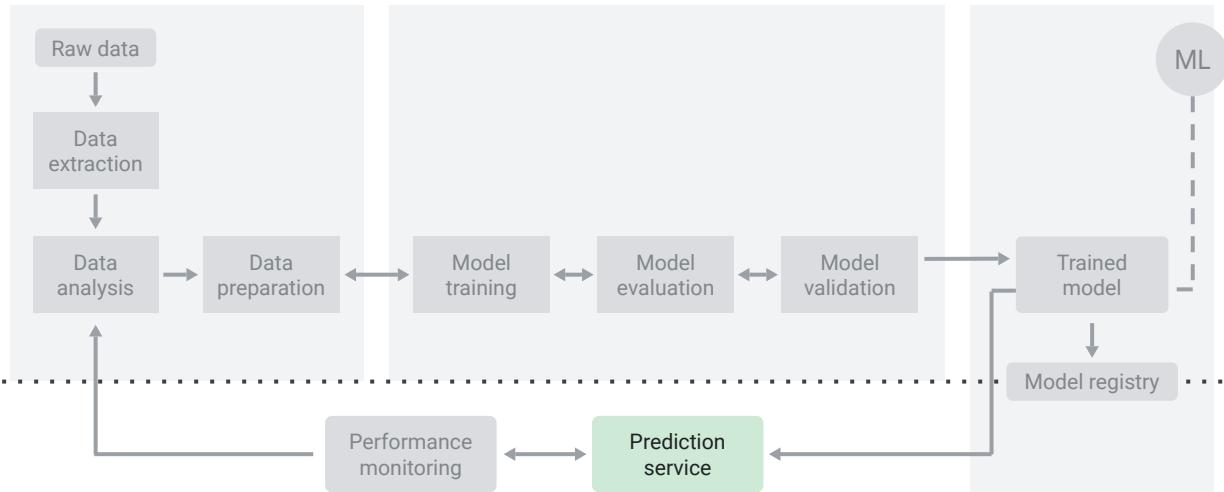
- Who trained and published a model
- Which datasets were used for training
- The values of metrics measuring predictive performance
- When the model was deployed to production



and when the model was deployed to production.

It's a place to find, publish, and use ML models or model pipeline components.

Staging/pre-production/production environments



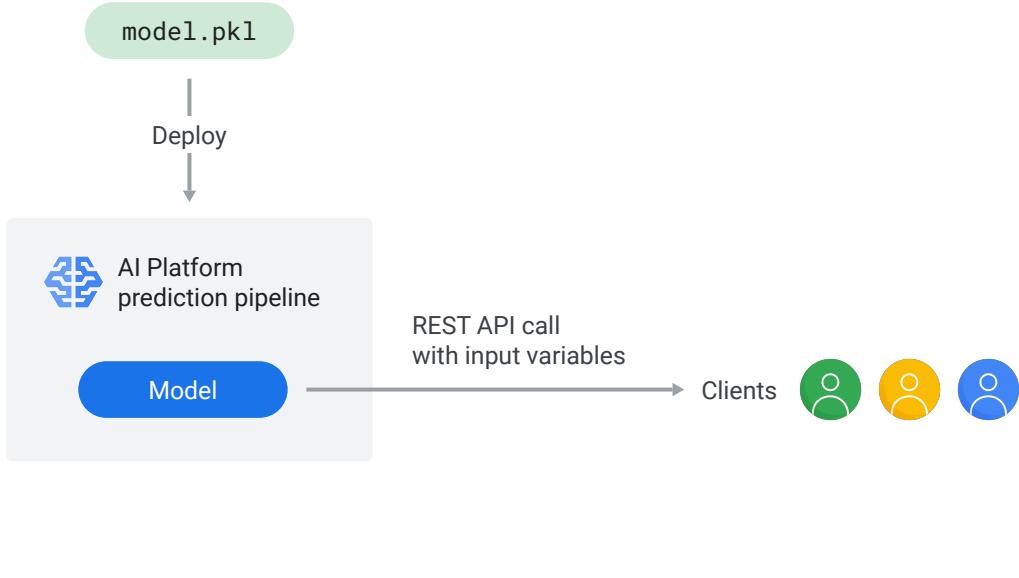
Experimentation/development/test environments

Machine learning uses data to answer questions. So prediction, or inference, is the step where we get to answer the questions we posed—whether it be a business problem or an academic research problem. The trained model is served as a **prediction service** to production.

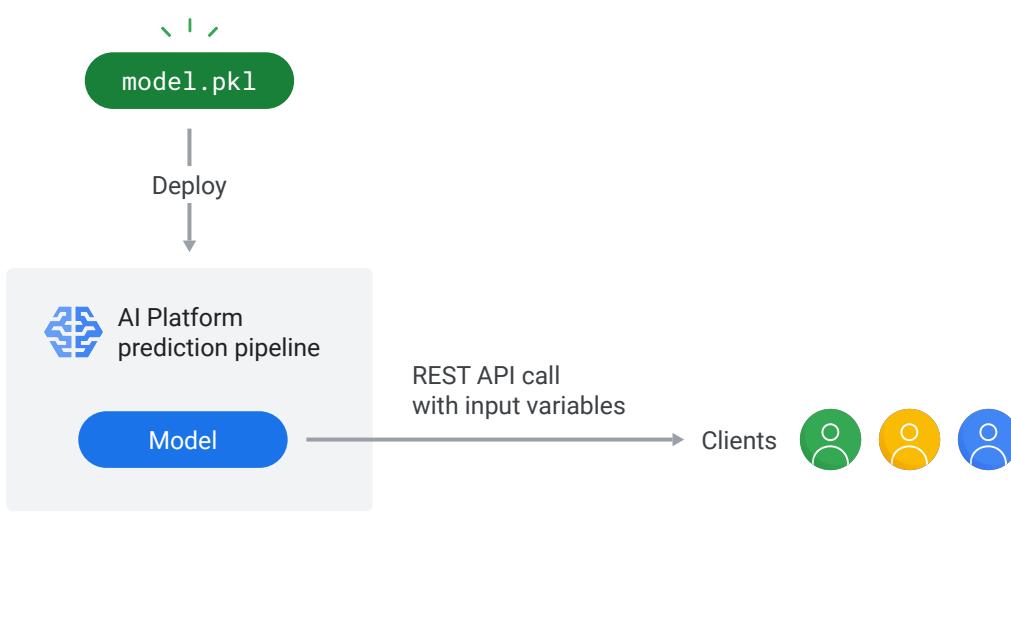
The process is concerned only
with deploying the trained model
as a prediction service rather than
deploying the entire ML system



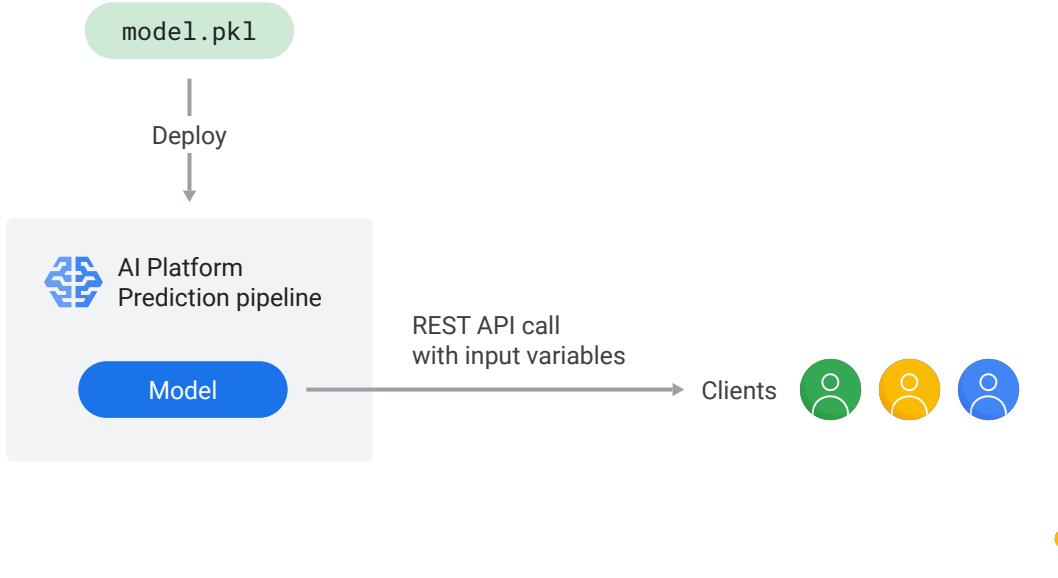
It's important to note that the process is concerned only with deploying the trained model as a prediction service, for example, a microservice with a REST API, rather than deploying the entire ML system.



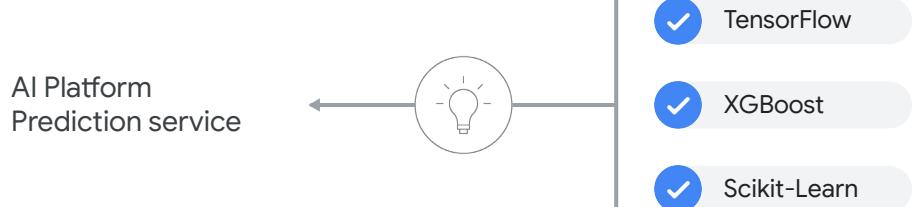
For example, Google's AI Platform Prediction service has an API for serving predictions from machine learning models.



In this particular example, AI Platform Prediction retrieves the trained model and saves it as a pickle in Cloud Storage. Pickle is the standard way of serializing objects in Python.

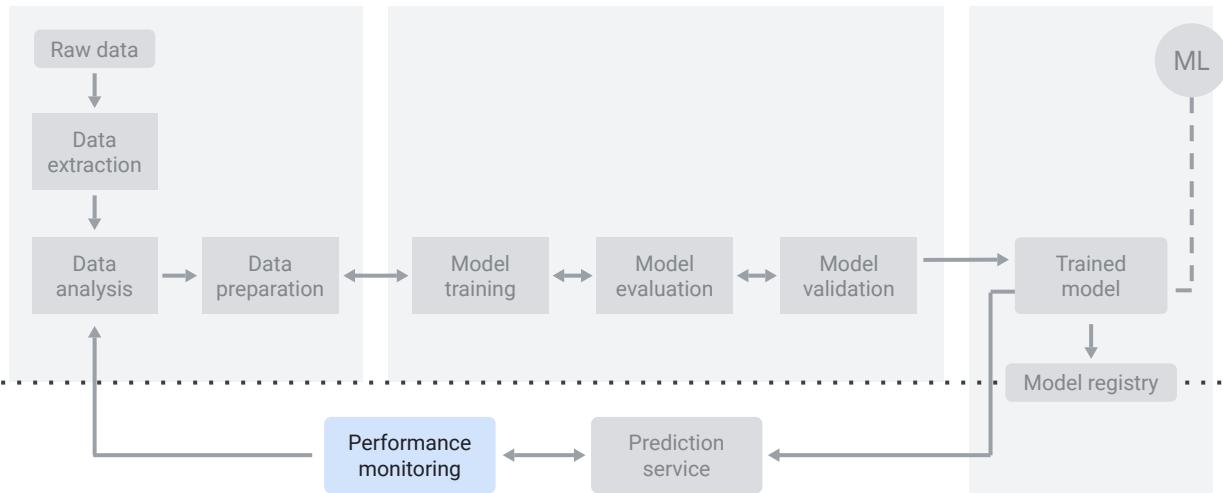


Trained models deployed in AI Platform Prediction service are exposed as REST endpoints that can be invoked from any standard client that supports HTTP, such as a JupyterLab notebook.



The AI Platform Prediction service can host models trained in popular machine learning frameworks including TensorFlow, XGBoost, and Scikit-Learn.

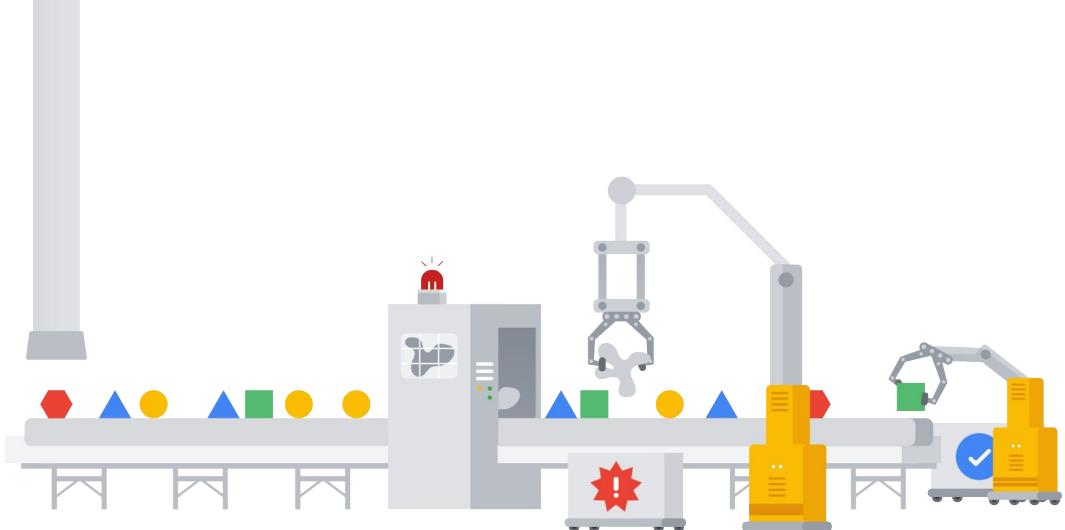
Staging/pre-production/production environments



Experimentation/development/test environments

As a best practice, you need a way to actively monitor the quality of your model in production. **Monitoring** lets you detect model performance degradation or model staleness.

The output of monitoring for these changes then feeds into the data analysis component, which could serve as a trigger to execute the pipeline or to execute a new experimental cycle.



For example, monitoring should be designed to detect data skews, which occur when your model training data is not representative of the live data. That is to say, the data that we used to train the model in the research or production environment does not represent the data that we actually get in our live system, and this leads to model staleness.



- Traffic patterns
- Error rates
- Latency
- Resource utilization



To understand other performance metrics, you can configure Google's Cloud Monitoring to monitor your model's



- Traffic patterns
- Error rates
- Latency
- Resource utilization



traffic patterns,



- Traffic patterns
- Error rates
- Latency
- Resource utilization



error rates,



- Traffic patterns
- Error rates
- Latency
- Resource utilization



latency, and



- Traffic patterns
- Error rates
- Latency
- Resource utilization



resource utilization.

Spot problems with your models
and find the right machine type
to optimize latency and cost

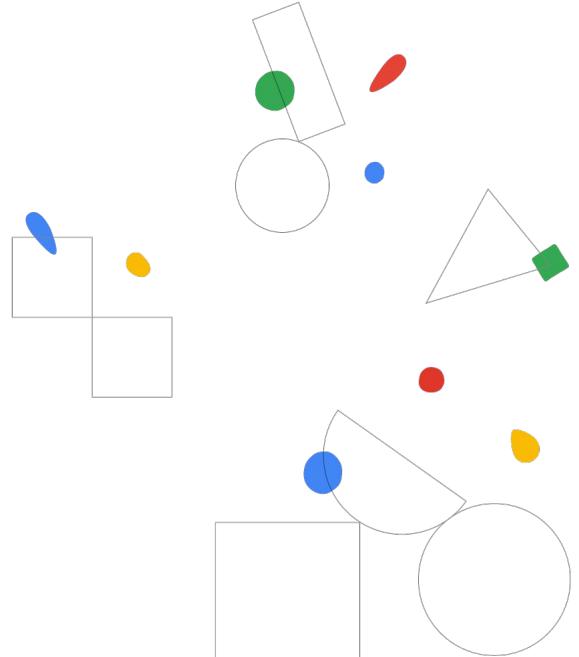


This can help spot problems with your models and find the right machine type to optimize latency and cost.

Training design decisions

Module 01

Architecting production ML systems



One of the key decisions you'll need to make about your production ML system concerns training.



Here's a question.



Physics



Here's a riddle.

How is physics





Physics



Fashion



unlike fashion? If we assume that science is about discovering relationships that already exist in the world, then the answer is that



Physics

Constant



Fashion

Changeable



physics is constant whereas fashion isn't.



Physics

Constant



Fashion

Changeable



To see some proof, just look at some old pictures of yourself.

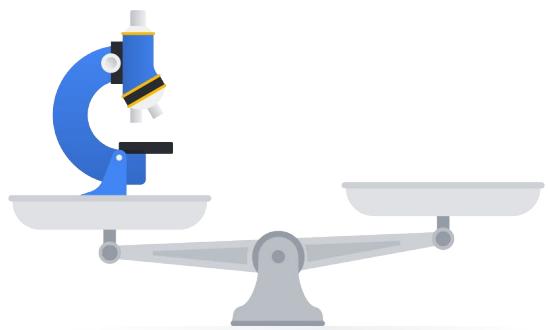
Why is this relevant?



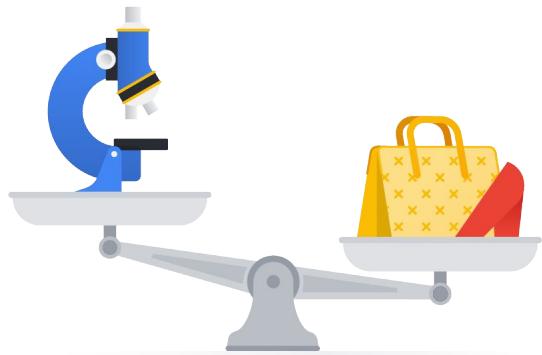
Now, you might be asking, why is this relevant?



Well, when making decisions about training, you have to decide whether the phenomenon you're modelling is



more like physics



or like fashion..

Static training

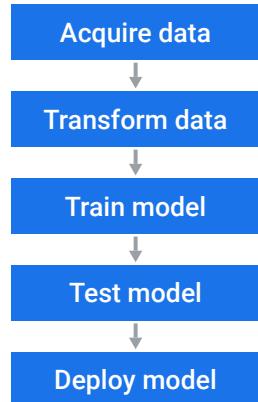
Dynamic training



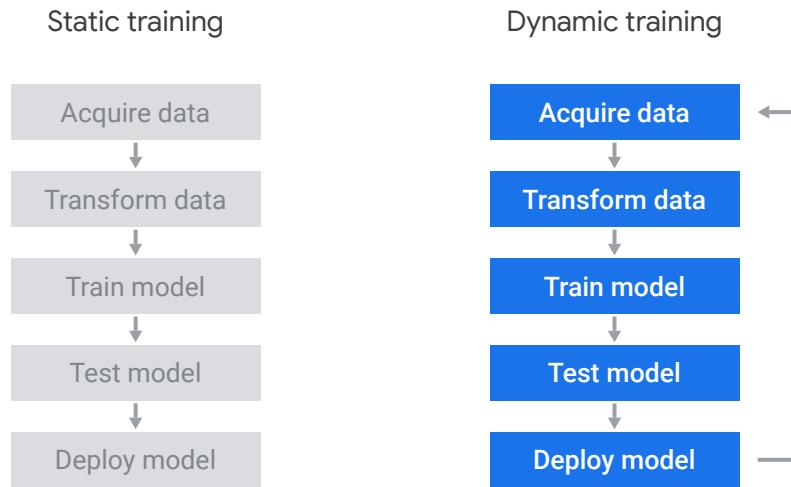
When training your model, there are two paradigms; static training and dynamic training.

Static training

Dynamic training



In static training, we do what we did in the last specialization. We gather our data, we partition it, we train our model, and then we deploy it.



In dynamic training, we do this repeatedly as more data arrives. This leads to the fundamental trade-off between static and dynamic.

Static training

Dynamic training

 Simpler to build and test

 Likely to become stale



Static is simpler to build and test, but likely to become stale.

Static training

 Simpler to build and test

 Likely to become stale

Dynamic training

 Harder to build and test

 Will adapt to changes



Whereas dynamic is harder to build and test, but will adapt to changes. And the tendency to become or not become stale is what was being alluded to earlier when we contrasted physics and fashion.

If the relationship you're trying to model
is one that's constant then a **statically
trained model** may be sufficient

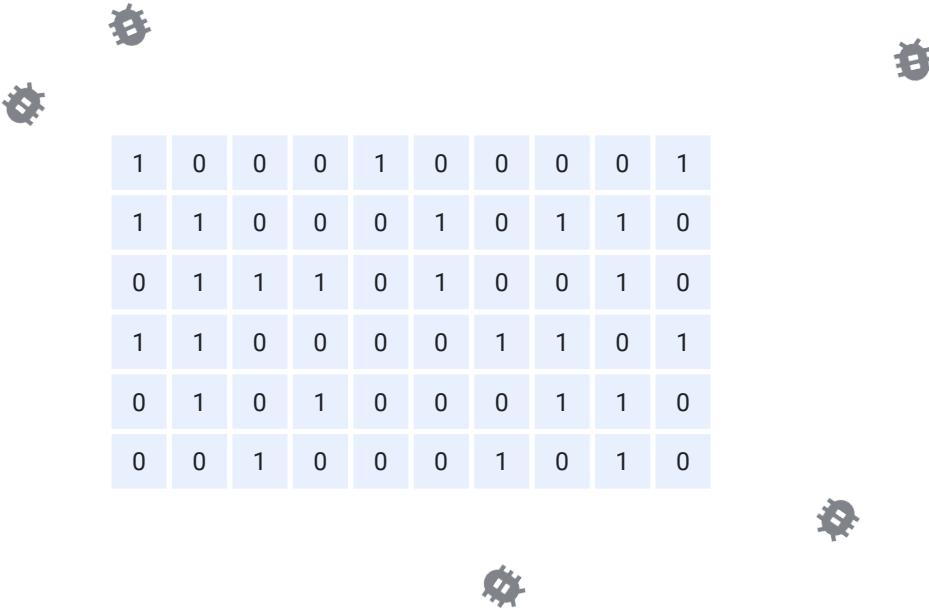


If the relationship you're trying to model is one that's constant, like physics, then a statically trained model may be sufficient.

If the relationship you're trying to model is one that changes, then the **dynamically trained model** might be more appropriate



If the relationship you're trying to model is one that changes then the dynamically trained model might be more appropriate.



1	0	0	0	1	0	0	0	0	1
1	1	0	0	0	1	0	1	1	0
0	1	1	1	0	1	0	0	1	0
1	1	0	0	0	0	1	1	0	1
0	1	0	1	0	0	0	1	1	0
0	0	1	0	0	0	1	0	1	0

Part of the reason the dynamic is harder to build and test is that new data may have all sorts of bugs in it. And that's something we'll talk about more deeply in a later module on designing adaptable ML systems.



- Monitoring
- Model rollback
- Data quarantine capabilities



Engineering might also be harder because we need more monitoring, model rollback, and data quarantine capabilities.

Problem

Training style

Predict whether email is spam

Android voice to text

Shopping ad conversion rate



Let's explore some use cases and think about which sort of training style would be most appropriate. The first use case concerns spam detection, and the question you should ask yourself is, "how fresh does spam detection need to be?"

Problem

Predict whether email is spam

Training style

Static

Android voice to text

Shopping ad conversion rate



You could do this as static, but spammers are a crafty and determined bunch. They will probably discover ways of passing whatever filter you impose within a short time.

Problem

Training style

Predict whether email is spam

Static

Dynamic

How quickly spammers change

Android voice to text

Shopping ad conversion rate



So, dynamic is likely to be more effective over time.

Problem

Predict whether email is spam

Training style

Static

Dynamic

How quickly spammers change

Android voice to text

Shopping ad conversion rate



What about Android Voice-to-Text? Note that this question has some subtlety.

Problem

Predict whether email is spam

Training style

Static

Dynamic

How quickly spammers change

Android voice to text

Static

Shopping ad conversion rate



For a global model, training offline is probably fine.

Problem	Training style	
Predict whether email is spam	Static Dynamic	How quickly spammers change
Android voice to text	Static Dynamic	Global vs personalized
Shopping ad conversion rate		



But, if you want to personalize the voice recognition, you may need to do something online, or at least different, on the phone.

So this could be static or dynamic, depending on whether you want global or personalized transcription.

Problem	Training style	
Predict whether email is spam	Static Dynamic	How quickly spammers change
Android voice to text	Static Dynamic	Global vs personalized
Shopping ad conversion rate		



What about ad conversion rate?

The interesting subtlety here is that conversions may come in very late. For example, if I'm shopping for a car online, I'm unlikely to buy for a very long time.

Problem	Training style	
Predict whether email is spam	Static Dynamic	How quickly spammers change
Android voice to text	Static Dynamic	Global vs personalized
Shopping ad conversion rate	Dynamic	



This system could use dynamic training, then regularly going back at different intervals to catch up on new conversion data that has arrived for the past. So in practice, most of the time, you'll need to use dynamic,

Problem	Training style	
Predict whether email is spam	Static Dynamic	How quickly spammers change
Android voice to text	Static Dynamic	Global vs personalized
Shopping ad conversion rate	Dynamic Static	



but you might start with static because it's simpler.

Static training



In a reference architecture for static training, models are trained once and then pushed to AI Platform.

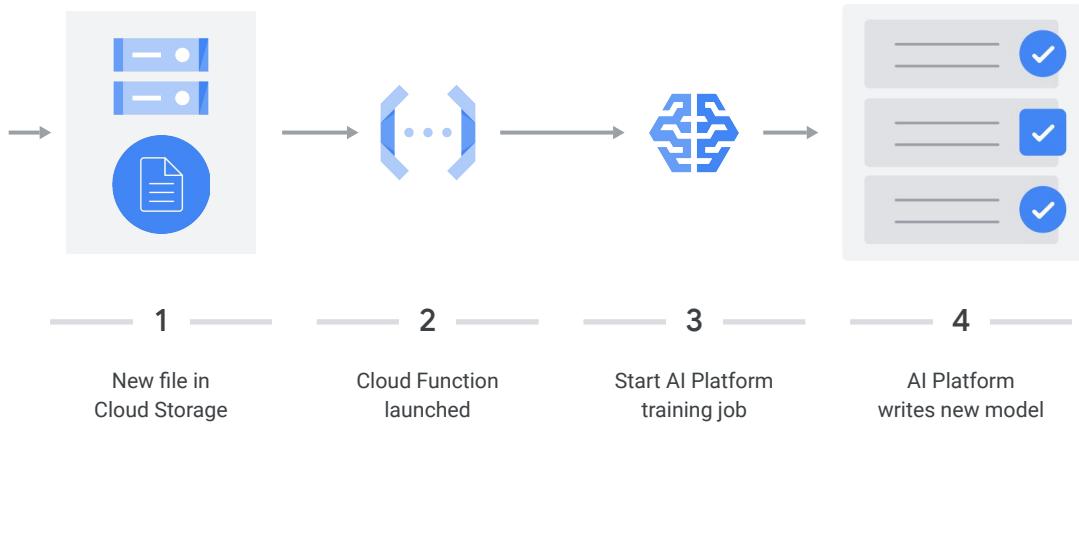
Static training



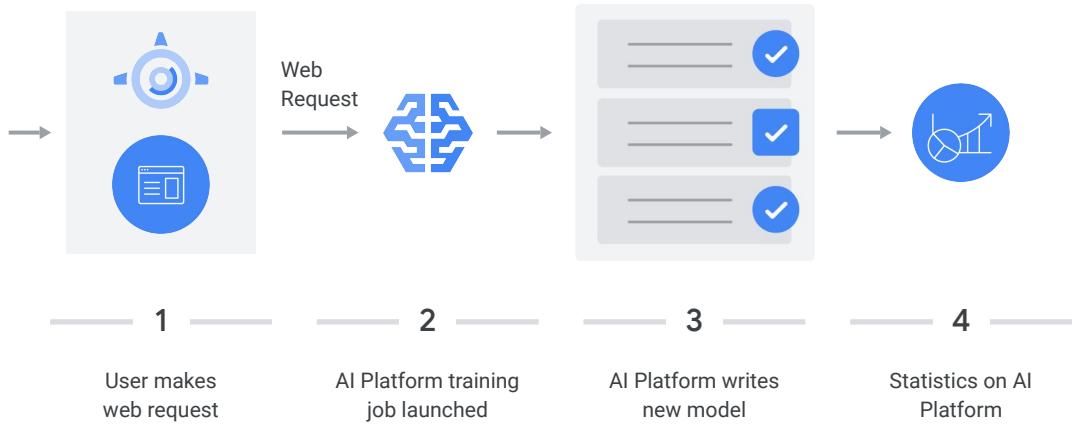
Dynamic training



Now, for dynamic training, there are three potential architectures to explore, Cloud Functions, App Engine, or Cloud Dataflow.



In a general architecture for dynamic training using Cloud functions, a new data file appears in Cloud storage and then the Cloud function is launched. After that, the Cloud function starts the AI Platform training job, and then the AI Platform writes out a new model.



In a general architecture for dynamic training using App Engine, when a user makes a web request, perhaps from a dashboard to AppEngine, an AI Platform training job is launched, and the AI Platform job writes a new model to Cloud storage. From there, the statistics of the training job are displayed to the user when the job is complete.



It's possible that the Dataflow pipeline is also invoking the model for predictions.

Here, a streaming topic is ingested into Pub/Sub from subscribers. Messages are then aggregated with Dataflow and aggregated data is stored into BigQuery.

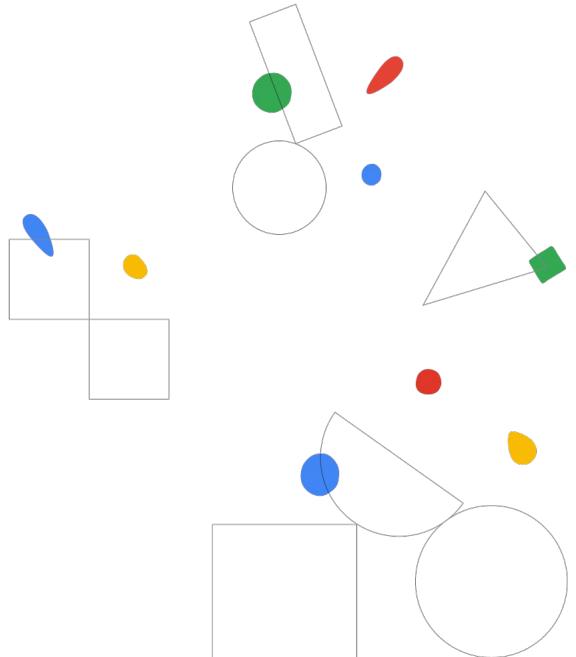
AI Platform is launched on the arrival of new data in BigQuery and then an updated model is deployed.



Serving design decisions

Module 01

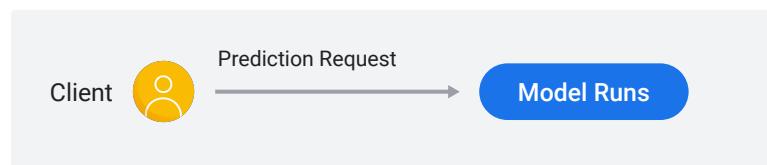
Architecting production ML systems



Static training



Dynamic training



In designing our serving architecture, one of our goals is to minimize average latency.

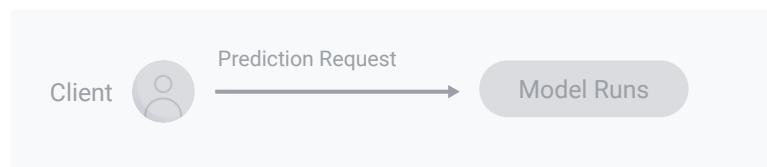
Just like in operating systems, where we don't want to be bottlenecked by slow disk I/O, when serving models, we don't want to be bottlenecked by slow-to-decide models.

Remarkably, the solution for serving models is very similar to what we do to optimize I/O performance: we use a cache. In this case, rather than faster memory, we'll use a table.

Static training



Dynamic training

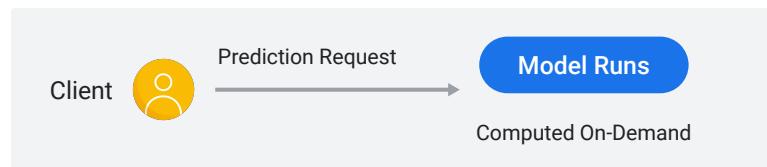


Static serving then computes the label ahead of time and serves by looking it up in the table.

Static training



Dynamic training



Dynamic serving, in contrast, computes the label on-demand.

Static training

- Space intensive
- Higher storage cost
- Low, fixed latency
- Lower maintenance

Dynamic training

- Compute intensive
- Lower storage cost
- Variable latency
- Higher maintenance



There's a space-time tradeoff.

Static serving is space-intensive, resulting in higher storage costs, because we store pre-computed predictions with a low, fixed latency and lower maintenance costs.

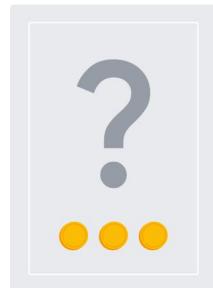
Dynamic serving, however, is compute-intensive. It has lower storage costs, higher maintenance, and variable latency.



The choice whether to use static or dynamic serving is determined by considering how important latency, storage, and CPU costs are.

Peakedness

Cardinality



Sometimes, it can be hard to express the relative importance of these three areas. As a result, it might be helpful to consider static and dynamic serving through another lens: peakedness and cardinality.

Peakedness in a data distribution is
the degree to which data values are
concentrated around the mean

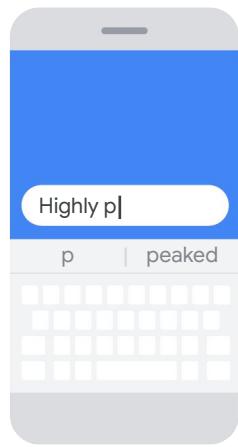


Peakedness in a data distribution is the degree to which data values are concentrated around the mean,

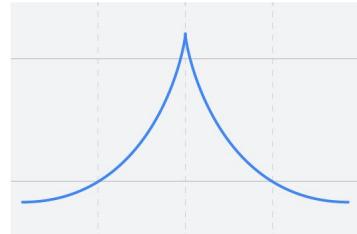
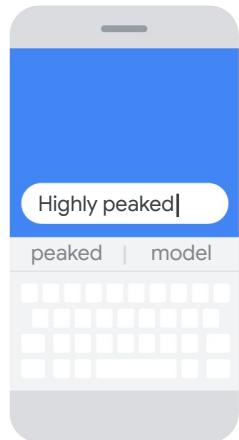
Peakedness refers to how concentrated the distribution of the prediction workload is



or in this case, how concentrated the distribution of the prediction workload is.



SLIDES 197-199 - For example, a model that predicts the next word given the current word, which you might find in your mobile phone keyboard app, would be highly peaked because a small number of words account for the majority of words used.



Highly peaked model

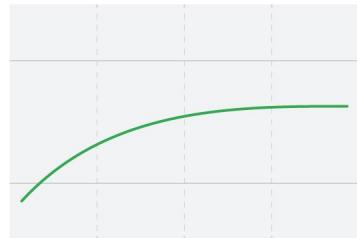


A small number of words account
for the majority of words used





SLIDES 200-202 - In contrast, a model that predicted quarterly revenue for all sales verticals in order to populate a report would be run on the same verticals, and with the same frequency for each, and so it would be very flat.



Flat model



A report would be run on the same verticals, and with the same frequency

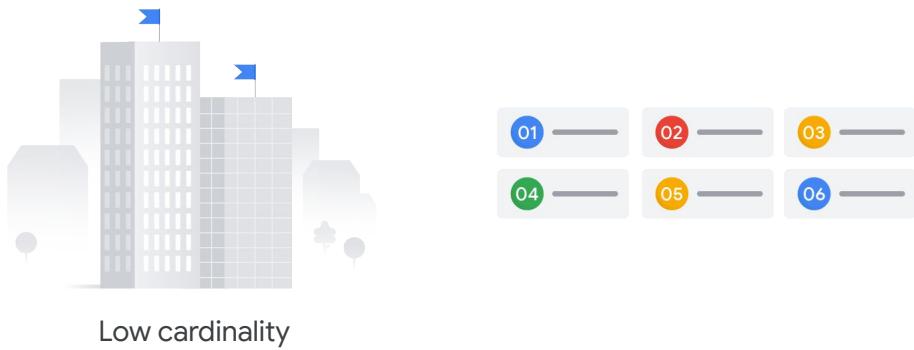


Cardinality refers to the number of values in a set

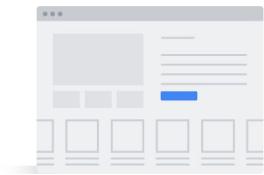


Cardinality refers to the number of values in a set.

In this case, the set is composed of all the possible things we might have to make predictions for.



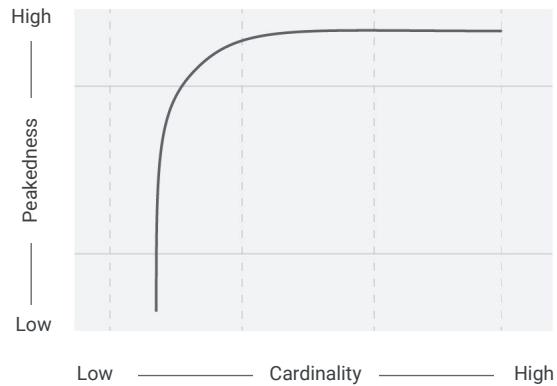
So, a model predicting sales revenue given organization division number would have fairly low cardinality.



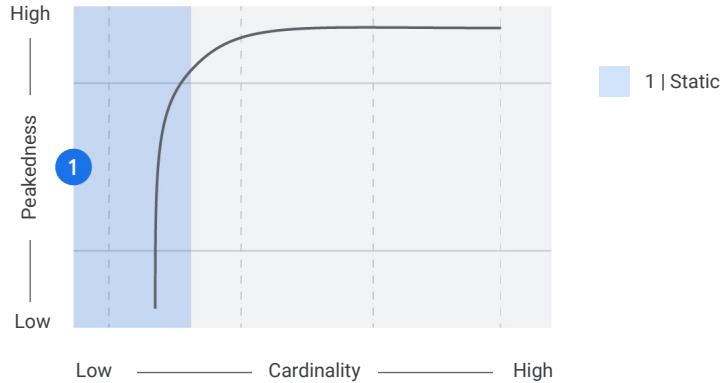
High cardinality



A model predicting lifetime value given a user for an ecommerce platform would be high cardinality because the number of users, and the number of characteristics of each user, are likely to be quite large.



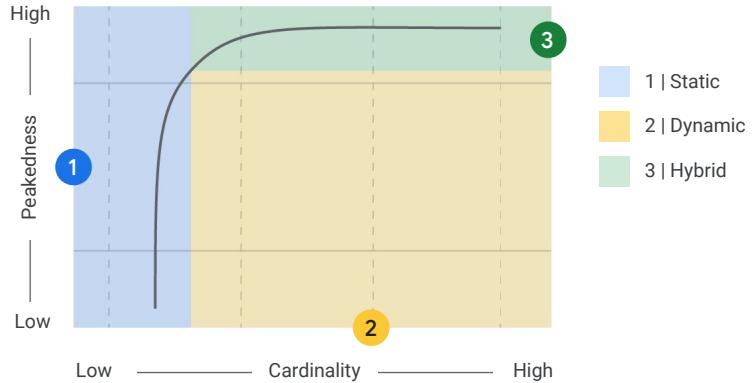
Taken together, peakedness and cardinality create a space.



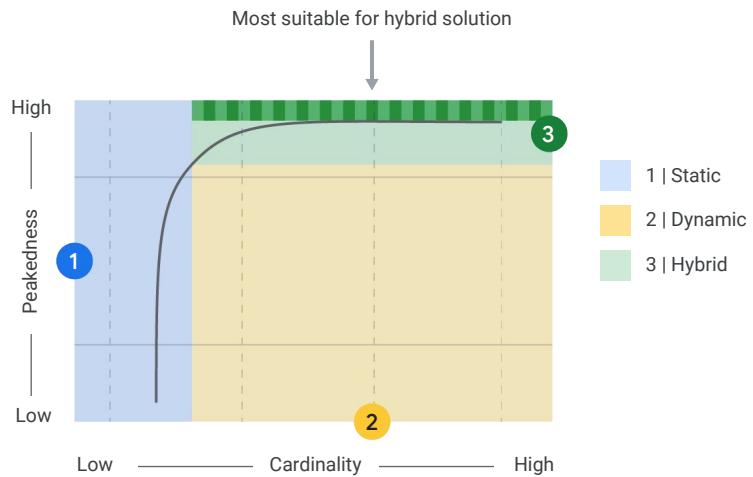
When the cardinality is sufficiently low, we can store the entire expected prediction workload, for example, the predicted sales revenue for all divisions, in a table and use static serving.



When the cardinality is high, because the size of the input space is large, and the workload is not very peaked, you probably want to use dynamic training.



In practice, though, you often choose a hybrid of static and dynamic, where you statically cache some of the predictions while responding on-demand for the long tail. This works best when the distribution is sufficiently peaked.



The striped area above the curve and not inside the blue rectangle is suitable for a hybrid solution, with the most frequently requested predictions cached and the tail computed on demand.

Problem

Predict whether email is spam

Training style

Android voice to text

Shopping ad conversion rate



Let's try to estimate training and inference needs for the same use cases that we saw in the previous lesson.

Problem	Training style
Predict whether email is spam	
Android voice to text	
Shopping ad conversion rate	



The first use case is predicting whether an email is spam.

What inference style is needed?

Well, first we need to consider how peaked the distribution is. The answer is *not at all*; most emails are likely to be different, although they may be very similar if generated programmatically.

Depending on the choice of representation, the cardinality might be enormous.

Problem

Predict whether email is spam

Training style

Dynamic

Android voice to text

Shopping ad conversion rate



So, this would be **dynamic**.

Problem	Training style
Predict whether email is spam	Dynamic
Android voice to text	
Shopping ad conversion rate	



The second use case is Android voice-to-text.

This is again subtle. Inference is almost certainly online, since there's such a long tail of possible voice clips. But maybe with sufficient signal processing, some key phrases like "okay google" may have precomputed answers.

Problem	Training style
Predict whether email is spam	Dynamic
Android voice to text	Dynamic Hybrid
Shopping ad conversion rate	



So, this would be **dynamic or hybrid**.

Problem	Training style
Predict whether email is spam	Dynamic
Android voice to text	Dynamic Hybrid
Shopping ad conversion rate	



And the third use case is shopping ad conversion rate.

The set of all ads doesn't change much from day to day. Assuming users are comfortable waiting for a short while after uploading their ads, this could be done statically, and then a batch script could be run at regular intervals throughout the day.

Problem	Training style
Predict whether email is spam	Dynamic
Android voice to text	Dynamic Hybrid
Shopping ad conversion rate	Static



And the third use case is shopping ad conversion rate.

The set of all ads doesn't change much from day to day. Assuming users are comfortable waiting for a short while after uploading their ads, this could be done statically, and then a batch script could be run at regular intervals throughout the day.

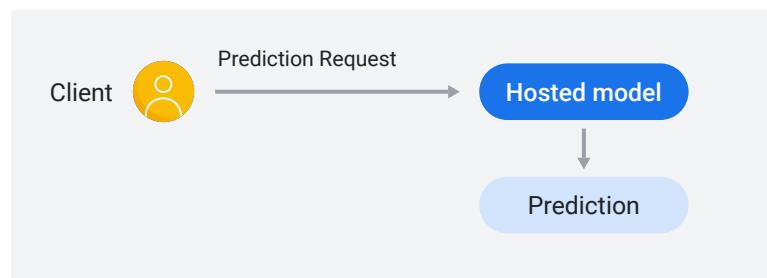
This would be **static**.

In practice, you'll often
use a **hybrid** approach.



In practice, you'll often use a hybrid approach.

Dynamic serving



```
gcloud ai-platform predict --model $MODEL_NAME \
    --version $VERSION_NAME \
    --json-instances $INPUT_DATA_FILE
```



You might not have realized it, but dynamic serving is what we have learned so far.

Think back to the architecture of the systems we've used to make predictions: a model that lived in AI Platform was sent one or more instances and returned predictions for each.

Static serving

1

2

3



If you wanted to build a static serving system, you would need to make three design changes.

Static serving

1

Change the call to AI Platform from an online prediction job to a batch prediction job

2

3



First, you would need to change your call to AI Platform from an online prediction job to a batch prediction job.

Static serving

- 1 Change the call to AI Platform from an online prediction job to a batch prediction job
- 2 Ensure the model accepted and passed through keys as input
- 3



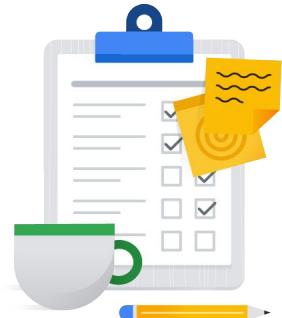
Second, you'd need to make sure that your model accepted and passed through keys as input. These keys are what will allow you to join your requests to predictions at serving time.

Static serving

- 1 Change the call to AI Platform from an online prediction job to a batch prediction job
- 2 Ensure the model accepted and passed through keys as input
- 3 Write the predictions to a data warehouse



And third, you would write the predictions to a data warehouse, like BigQuery and create an API to read from it.



- Submitting a batch prediction job
- Enabling pass-through features
- Loading data into BigQuery



Although the details for each of these instructions are beyond the scope of this lesson, we've provided links in the course resources on:

1. Submitting a batch prediction job:
<https://cloud.google.com/ai-platform/prediction/docs/batch-predict>
2. Enabling pass-through features in your model:
<https://towardsdatascience.com/how-to-extend-a-canned-tensorflow-estimator-to-add-more-evaluation-metrics-and-to-pass-through-ddf66cd3047d>
3. And loading data into BigQuery:
<https://cloud.google.com/bigquery/docs/loading-data>

Agenda

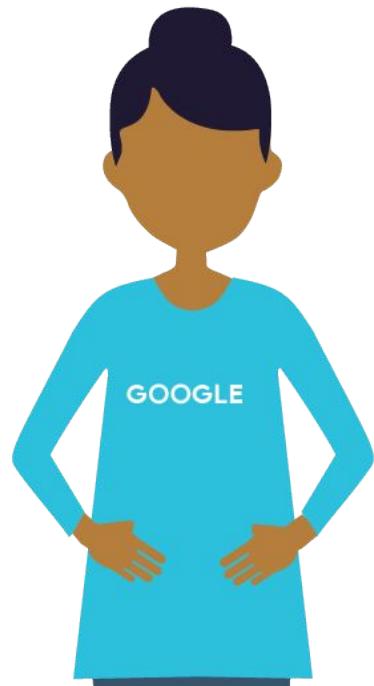
What's in a Production ML System

Training Design Decisions

Serving Design Decisions

AI Platform Notebooks

Designing an Architecture from Scratch



In this section, we'll apply what we've learned to a new use case.



Lab: Build a system that predicts the traffic levels on roads



Let's pretend the head of a municipal transit system has contacted you to build a system that predicts the traffic levels on roads.

As part of your preparation for this task, you're trying to thoroughly understand the business constraints in order to make the appropriate system design tradeoffs.

Image cc0: <https://pixabay.com/en/architecture-buildings-cars-city-1837176/>



Lab: Build a system that predicts the traffic levels on roads

Available data: Traffic sensors deployed all over the city



The data available consist of sensors deployed all over the city which record whenever a car passes by. For each sensor, we know where it is. We also know the characteristics of the road that it's on.

Image cc0: <https://pixabay.com/en/junction-city-aerial-view-urban-984045/>



Lab: Build a system that predicts
the traffic levels on roads

What sort of training architecture is
appropriate?



What sort of training architecture is appropriate?



Lab: Build a system that predicts the traffic levels on roads

What is the relationship between the features and labels like?



What is the relationship between the features and labels like? Is it more like physics equations (fixed) or fashion trends (ever changing)?

It's more like fashion trends. Cities are complex systems; if a train stops service, people will still need to get home. Technology is also always changing: on-demand taxi services have reshaped urban transit in ways we didn't anticipate a decade ago. There are also episodic changes like sports events and parades for example. For dynamic relationships, we need to use dynamic training.



Lab: Build a system that predicts
the traffic levels on roads

Which sort of serving architecture is
appropriate?



Which sort of serving architecture is appropriate?



Lab: Build a system that predicts the traffic levels on roads

Is the distribution of prediction requests likely to be more peaked or more flat?



Is the distribution of prediction requests likely to be more peaked or more flat?
Peaked

The distribution of demand is peaked because it is likely to be dominated by requests on most heavily-trafficked roads.



Lab: Build a system that predicts the traffic levels on roads

Is the cardinality of the set of all prediction requests likely to be low, moderate, high, need more info?



Is the cardinality of the set of all prediction requests likely to be low, moderate, high, need more info?

Need more info



Lab: Build a system that predicts the traffic levels on roads

Is the cardinality of the set of all prediction requests likely to be low, moderate, high, need more info?

What does it depend on?

- A) Historical traffic data
- B) Problem framing
- C) Variance of Traffic Levels



Why? What does it depend upon?

- Historical traffic data
- Problem framing
- Variance of Traffic Levels



Lab: Build a system that predicts the traffic levels on roads

Is the cardinality of the set of all prediction requests likely to be low, moderate, high, need more info?

What does it depend on?

- A) Historical traffic data
- B) Problem framing
- C) Variance of Traffic Levels



Answer: A and B. Not C.

The reason that the cardinality depends upon the framing of the problem is that we don't know whether the task is to make predictions for every minute, hour or day. Similarly, we don't know how big a region of space each prediction should correspond to; it could be anything from a few feet to a few blocks.

As we learned in the first specialization, machine learning is all about generalization, the leap of faith to unseen input. What we don't know is whether our users want to generalize in space (i.e., by making predictions far away from the sensors), in time (by making predictions in the future with finer granularity than the historical data) or both. In all likelihood, you'd start conservatively which corresponds to a lower, rather than a higher, cardinality.

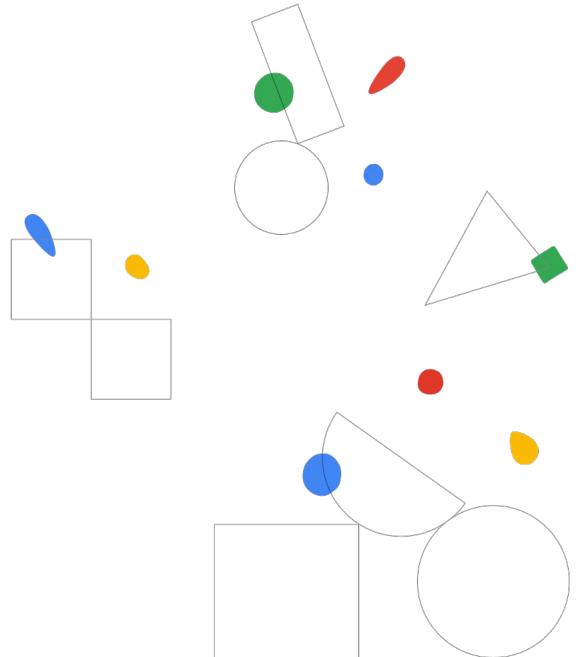
Variance of traffic levels wouldn't matter because that's a label, not a feature.



Using Vertex AI

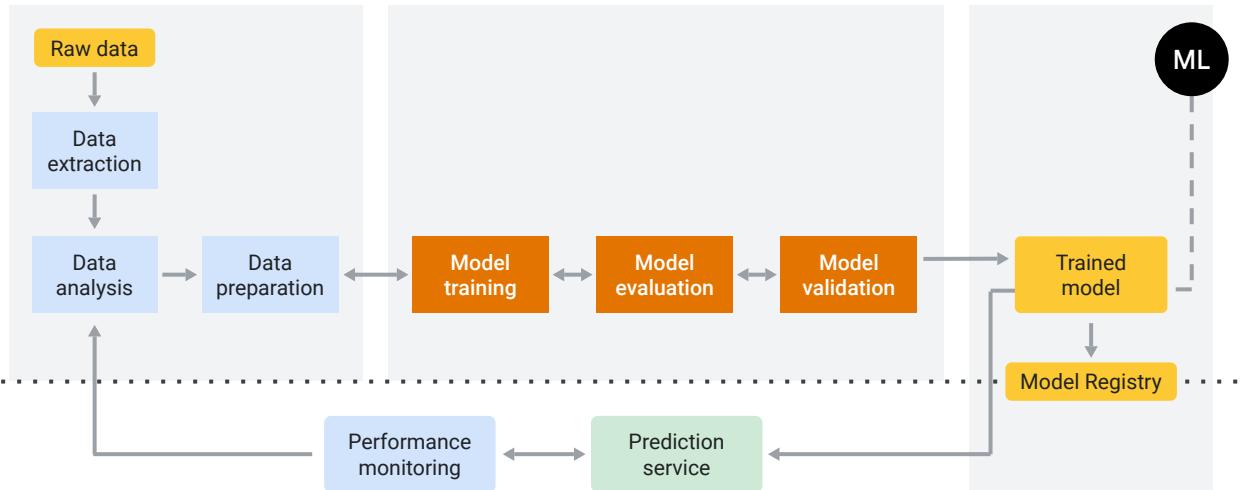
Module 01

Architecting production ML systems



As you've seen from previous videos, the machine learning ecosystem requires decision making at every stage.

Staging/pre-production/production environments

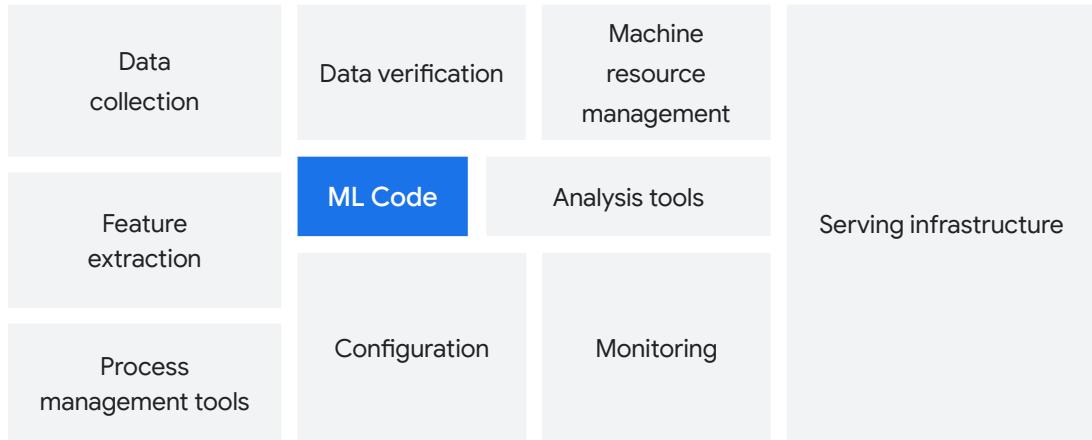


Experimentation/development/test environments



You need to determine how to handle and prepare data, and also how to design, build, evaluate, train, and monitor a model's performance.

Decisions around workflow processes, how to implement or execute those processes, and the management of the workflow itself are required to solve machine learning problems.



One of the most interesting details about the ML ecosystem is that ML code accounts for only a small percentage of it.

To keep a system running in production requires a lot more than just computing the model's outputs for a given set of inputs.

This means that each component of the ML ecosystem requires not only decisions and processes, but also people.

A lack of staff with the **right expertise**, a lack of **production-ready data**, and a lack of an **integrated development environment**

International Data Corporation, May 2020



According to the International Data Corporation, in 2020, a lack of staff with the right expertise, a lack of production-ready data, and a lack of an integrated development environment were reported as primary reasons that machine learning technologies fail.

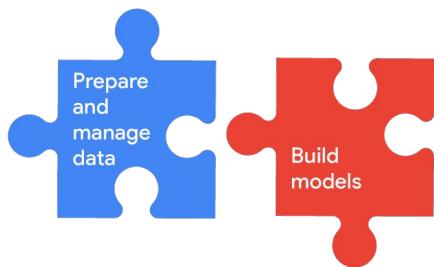
So, how do you ensure success for your machine learning or AI use case?



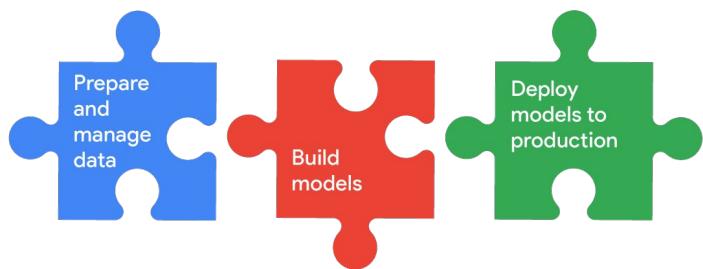
So, how do you ensure success for your machine learning or AI use case?



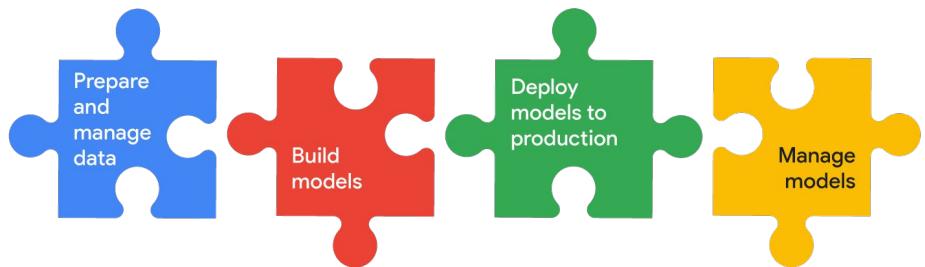
And how can you or your team prepare and manage your data,



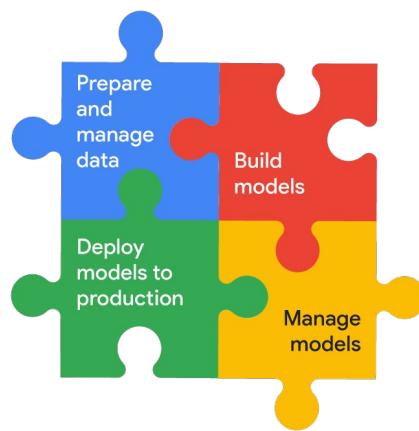
build your models,



deploy them into production,



and then manage them?



A solution is to use a unified platform that brings all the components of the machine learning ecosystem and workflow together.



Vertex AI



And in this case, that platform is Vertex AI.

Vertex AI brings together the Google Cloud services for building ML under one unified user interface and application programming interface, or API.

☰ Google Cloud Platform

Search products and resources

HOME PREVIEW DASHBOARD ACTIVITY RECOMMENDATIONS CUSTOMISE

Quick access ①

123 Dashboard – Vertex AI – cloud-trainin... Accessed just now

123 Datasets – Vertex AI – cloud-training... Accessed 2 minutes ago

123 Notebooks – Vertex AI – cloud-trainin... Accessed 1 minute ago

Google Cloud Getting started

Project info

Project name: hello_world

Project ID: 123

Project number: 1234567

Go to project settings

Resources

App Engine: 8 versions

Compute Engine

App Engine

Summary (count/sec)

0.06/s

0.03/s

0

15:00 15:15 15:30 15:45

http/server/response_count: 0

http/server/response_count: 0

Google Cloud Platform status

All services normal

Go to Cloud status dashboard

Billing

Estimated charges: USD \$2,095.31

For the billing period 1–6 Aug 2021

Take a tour of billing

View detailed charges

☰ Google Cloud Platform

Search products and resources

ACTIVITY RECOMMENDATIONS CUSTOMISE

Home > Recent Marketplace Billing APIs and services Support IAM & Admin Getting started Compliance Security Anthos

Recent

Datasets – Vertex AI – cloud-training... Accessed 2 minutes ago

Notebooks – Vertex AI – cloud-train... Accessed 1 minute ago

Google Cloud Getting started

App Engine

Summary (count/sec)

The chart displays a series of sharp, vertical blue peaks representing response counts. The x-axis is labeled with time points: 15:00, 15:15, 15:30, and 15:45. The y-axis has two scale bars: 0.06/s at the top and 0 at the bottom. The peaks occur approximately every 15 seconds, reaching a maximum height of about 0.06/s.

http/server/response_count: 0

Google Cloud Platform status

All services normal

Go to Cloud status dashboard

Billing

Estimated charges USD \$2,095.31 For the billing period 1–6 Aug 2021

Take a tour of billing

View detailed charges

≡ Google Cloud Platform

Search products and resources

ACTIVITY RECOMMENDATIONS CUSTOMISE

Home Recent ARTIFICIAL INTELLIGENCE

- Vertex AI
- AI Platform
- Data Labelling
- Demand AI
- Document AI
- Natural Language
- Recommendations AI
- Retail
- Speech-to-text
- Tables
- Talent solution

123 Datasets – Vertex AI – cloud-training... Accessed 2 minutes ago

123 Notebooks – Vertex AI – cloud-train... Accessed 1 minute ago

Google Cloud Getting started

App Engine

Summary (count/sec)

0.06/s

0.03/s

15:00 15:15 15:30 15:45

http/server/response_count: 0 http/server/response_count: 0

Google Cloud Platform status

All services normal

Go to Cloud status dashboard

Billing

Estimated charges USD \$2,095.31
For the billing period 1–6 Aug 2021

Take a tour of billing

View detailed charges

Google Cloud Platform

Search products and resources

Vertex AI

Dashboard

Datasets

Features

Labelling tasks

Notebooks

Pipelines

Training

Experiments

Models

Endpoints

Batch predictions

Metadata

Region: us-central1 (Iowa)

Get started with Vertex AI

Vertex AI empowers machine learning developers, data scientists and data engineers to take their projects from ideation to deployment, quickly and cost-effectively. [Learn more](#)



Recent datasets

chicago-taxi-tips	26 Jun 2021
ml_on_gc_test_2	9 Jun 2021
ml_on_gc_test	9 Jun 2021

Recent models

chicago-taxi-tips-classifier-v01	26 Jun 2021
imagedataset_162350007121_12Jun2021_2_202161213847	12 Jun 2021

Average precision: 0.988

Get predictions

After you train a model, you can use it to get predictions, either online as an endpoint or through batch requests

+ CREATE BATCH PREDICTION

With Vertex AI, you can access a dashboard, datasets, features, labeling tasks, notebooks, pipelines, training, experiments, models, endpoints, batch predictions, and metadata.

Let's take a closer look at the Datasets, Notebooks, Training, and Models sections of the Vertex AI navigation bar that help you to prepare your data and build and deploy your models.

Google Cloud Platform

Search products and resources

REFRESH

Vertex AI

Data sets + CREATE

Dashboard

Datasets

Features

Labelling tasks

Notebooks

Pipelines

Training

Experiments

Models

Endpoints

Batch predictions

Metadata

Region us-central1 (Iowa)

Filter Enter a property name

	Name	ID	Region	Type	Items	Labels	Last updated	Status	Metadata
<input type="checkbox"/>	<input checked="" type="checkbox"/> chicago-taxis-tips	8914813889728741376	us-central1	Tabular	-	-	26 June 2021	Created data set	
<input type="checkbox"/>	<input checked="" type="checkbox"/> ml_on_gc_test_2	4512404516485726208	us-central1	Tabular	-	-	9 June 2021	Created data set	
<input type="checkbox"/>	<input checked="" type="checkbox"/> ml_on_gc_test	5156419263199707136	us-central1	Text	4,420	-	9 June 2021	Finished importing data	

◀

The screenshot shows the Google Cloud Platform Vertex AI interface. On the left, there's a sidebar with various options like Dashboard, Datasets, Features, etc. The 'Datasets' option is selected. The main area shows a table of datasets. There are three entries in the table:

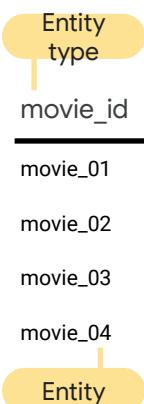
- chicago-taxis-tips: ID 8914813889728741376, Region us-central1, Type Tabular, Last updated 26 June 2021, Status Created data set.
- ml_on_gc_test_2: ID 4512404516485726208, Region us-central1, Type Tabular, Last updated 9 June 2021, Status Created data set.
- ml_on_gc_test: ID 5156419263199707136, Region us-central1, Type Text, Items 4,420, Last updated 9 June 2021, Status Finished importing data.

Vertex AI has a unified data preparation tool that supports image, tabular, text, and video content. Uploaded datasets are stored in a Cloud Storage bucket that acts as an input for both AutoML and custom training jobs.

row	movie_id	average_rating	title	genres	update_time
1	movie_01	4.9	The Shawshank Redemption	Drama	70-01-01 00:26:19
2	movie_02	4.2	The Shining	Horror	70-01-01 00:26:19
3	movie_03	4.5	Cinema Parasado	Romance	70-01-01 00:26:19
4	movie_04	4.6	The Dark Knight	Action	70-01-01 00:26:19



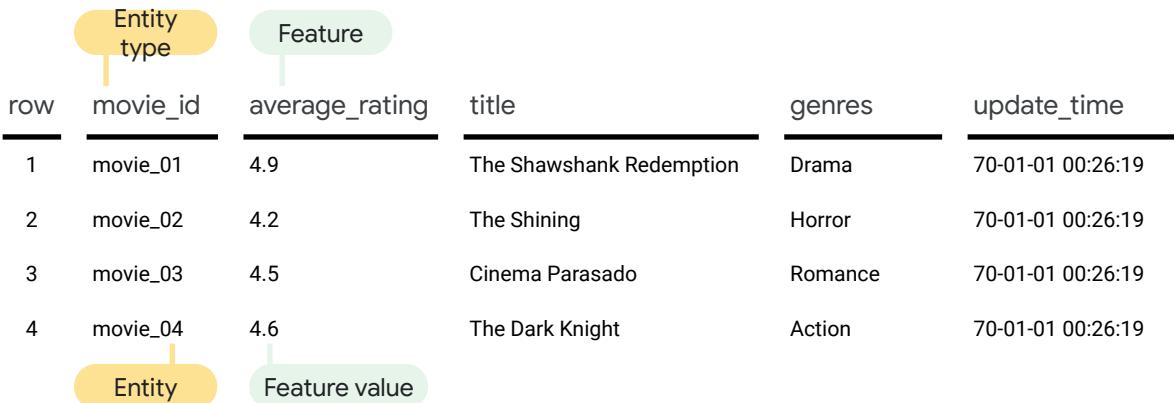
Let's explore an example where you have sample source data from a BigQuery table about movies and their features.



row	movie_id	average_rating	title	genres	update_time
1	movie_01	4.9	The Shawshank Redemption	Drama	70-01-01 00:26:19
2	movie_02	4.2	The Shining	Horror	70-01-01 00:26:19
3	movie_03	4.5	Cinema Parasado	Romance	70-01-01 00:26:19
4	movie_04	4.6	The Dark Knight	Action	70-01-01 00:26:19



First, there is a `movie_id` column header that can map to an entity type called *movie*. For each movie entity, features include an `average_rating`, `title`, and `genres`.



The diagram illustrates a dataset structure with annotations:

- Entity type:** A yellow oval at the top left pointing to the "row" header.
- Feature:** A green oval at the top center pointing to the "average_rating" header.
- Entity:** A yellow oval at the bottom left pointing to the "movie_id" header.
- Feature value:** A green oval at the bottom center pointing to the "4.9" value in the "average_rating" column.

row	movie_id	average_rating	title	genres	update_time
1	movie_01	4.9	The Shawshank Redemption	Drama	70-01-01 00:26:19
2	movie_02	4.2	The Shining	Horror	70-01-01 00:26:19
3	movie_03	4.5	Cinema Parasado	Romance	70-01-01 00:26:19
4	movie_04	4.6	The Dark Knight	Action	70-01-01 00:26:19



The values in each column map to specific instances of an entity type or features, which are called entities and feature values.

row	movie_id	average_rating	title	genres	update_time
1	movie_01	4.9	The Shawshank Redemption	Drama	70-01-01 00:26:19
2	movie_02	4.2	The Shining	Horror	70-01-01 00:26:19
3	movie_03	4.5	Cinema Parasado	Romance	70-01-01 00:26:19
4	movie_04	4.6	The Dark Knight	Action	70-01-01 00:26:19



The update_time column indicates when the feature values were generated.

In the featurestore, the timestamps are an attribute of the feature values, not a separate resource type. If all feature values were generated at the same time, you don't need to have a timestamp column. You can specify the timestamp as part of your ingestion request.



row	movie_id	average_rating	title	genres
1	movie_01	4.9	The Shawshank Redemption	Drama
2	movie_02	4.2	The Shining	Horror
3	movie_03	4.5	Cinema Parasado	Romance
4	movie_04	4.6	The Dark Knight	Action



When you use the data to train a model, Vertex AI examines the source data type and feature values and infers how it will use that feature in model training.



Transformation

row	movie_id	average_rating	title	genres
1	movie_01	4.9	The Shawshank Redemption	Drama
2	movie_02	4.2	The Shining	Horror
3	movie_03	4.5	Cinema Parasado	Romance
4	movie_04	4.6	The Dark Knight	Action



This is called the *transformation* for that feature. If needed, you can specify a different supported transformation for any feature.

Google Cloud Platform

Search products and resources

Vertex AI

Notebooks NEW INSTANCE REFRESH START STOP RESET DELETE SHOW INFO PANEL

MANAGED NOTEBOOKS PREVIEW INSTANCES EXECUTIONS PREVIEW SCHEDULES PREVIEW SCHEDULED RUNS

Dashboard Datasets Features Labelling tasks

Create and use Jupyter Notebooks with a notebook instance. Notebook instances have JupyterLab pre-installed and are configured with GPU-enabled machine-learning frameworks. [Learn more](#)

Filter Enter property name or value

Instance name	Zone	Environment version	Auto-upgrade	Environment	Machine
asl2	us-west1-b	M34	—	TensorFlow:2.0	4 vCPUs RAM ▾
cloud-training-demos1234-notebook	us-central1-a	M69	—	TensorFlow:2.4	8 vCPUs RAM ▾
tensorflow-2-1-20200624-204504	us-central1-a	M49	—	TensorFlow:2.1	4 vCPUs RAM ▾
tensorflow-2-1-2021-july-16	us-central1-a	M75	—	TensorFlow:2.1	4 vCPUs RAM ▾
tensorflow-2-1-20210619-094520	us-central1-a	M71	—	TensorFlow:2.1	4 vCPUs RAM ▾
tensorflow-2-3-20210225-215102	us-west1-b	Mnightly-2021-02-12-debian-10-test	—	TensorFlow:2.3	4 vCPUs RAM ▾

After you prepare your dataset, you can develop models using Notebooks. Notebooks is a managed service that offers an integrated and secure JupyterLab environment for data scientists and machine learning developers to experiment, develop, and deploy models into production.

Notebooks enable you to create and manage virtual machine (VM) instances because they come pre-installed with the latest data science and machine learning frameworks.

They also come with a pre-installed suite of deep learning packages, including support for the TensorFlow and PyTorch frameworks. Either can be configured for CPU-only or GPU-enabled instances.

With regard to security, Notebooks instances are protected by Google Cloud authentication and authorization and are available using a Notebooks instance URL, which is part of the metadata of the VM.

Google Cloud Platform

Search products and resources

REFRESH

Vertex AI

Training + CREATE

TRAINING PIPELINES CUSTOM JOBS HYPERPARAMETER TUNING JOBS

Dashboard

Datasets

Features

Labelling tasks

Notebooks

Pipelines

Training

Experiments

Models

Endpoints

Batch predictions

Metadata

Region us-central1 (Iowa)

Filter Enter a property name

Name	ID	Job type	Model type	Status	Created	Elapsed time
imagedataset_162350071212_202161213847	8913542854287032320	Training pipeline	Image classification (Single-label)	Succeeded	12 Jun 2021, 14:09:56	23 min 44 sec

Now let's shift our focus to training.

With Vertex AI, you can train and compare models using AutoML or custom code training, with all models stored in one central model repository.

Training pipelines are the primary model training workflow in Vertex AI, which can use training pipelines to create an AutoML-trained model or a custom-trained model.

The screenshot shows the Google Cloud Platform Vertex AI Training interface. On the left, there's a sidebar with various options like Dashboard, Datasets, Features, Labelling tasks, Notebooks, Pipelines, Experiments, Models, Endpoints, Batch predictions, and Metadata. The 'Training' option is selected. At the top right, there are buttons for '+ CREATE', 'REFRESH', and a search bar. The main area is titled 'CUSTOM JOBS' and shows three entries:

Name	ID	Job type	Model type	Status	Created	Elapsed time
xgb_train_test_user_071521_1821	7190124348445818880	Custom job	—	Succeeded	15 Jul 2021, 23:23:56	2 min 1 sec
chicago-taxi-tips-classifier-v01_trainer_20210626015459	2136135588489723904	Custom job	—	Succeeded	26 Jun 2021, 02:54:59	2 min 31 sec
xgb_train_test_user_061921_0658	7644477737414950912	Custom job	—	Succeeded	19 Jun 2021, 14:58:23	2 min 1 sec

For custom-trained models, training pipelines orchestrate custom training jobs and hyperparameter tuning in conjunction with steps like adding a dataset or uploading the model to Vertex AI for prediction serving.

Custom jobs specify how Vertex AI runs custom training code, including worker pools, machine types, and settings related to a Python training application and custom container.

The screenshot shows the Google Cloud Platform Vertex AI interface. The left sidebar lists various options like Dashboard, Datasets, Features, Labelling tasks, Notebooks, Pipelines, Experiments, Models, Endpoints, Batch predictions, and Metadata. The 'Training' option is selected. The main area has tabs for TRAINING PIPELINES, CUSTOM JOBS, and HYPERPARAMETER TUNING JOBS, with HYPERPARAMETER TUNING JOBS being the active tab. A descriptive text explains that hyperparameter tuning searches for the best combination of hyperparameter values by optimising metric values across a series of trials. It notes that hyperparameter tuning is only used by custom-trained models and not AutoML models, with a link to 'Learn more'. A dropdown menu for 'Region' is set to 'us-central1 (Iowa)'. A 'Filter' input field allows entering a property name. A table header row includes columns for Name, ID, Job type, Model type, Status, Created, and Elapsed time. Below the table, it says 'No rows to display'.

Alternatively, hyperparameter tuning searches for the best combination of hyperparameter values by optimizing metric values across a series of trials.

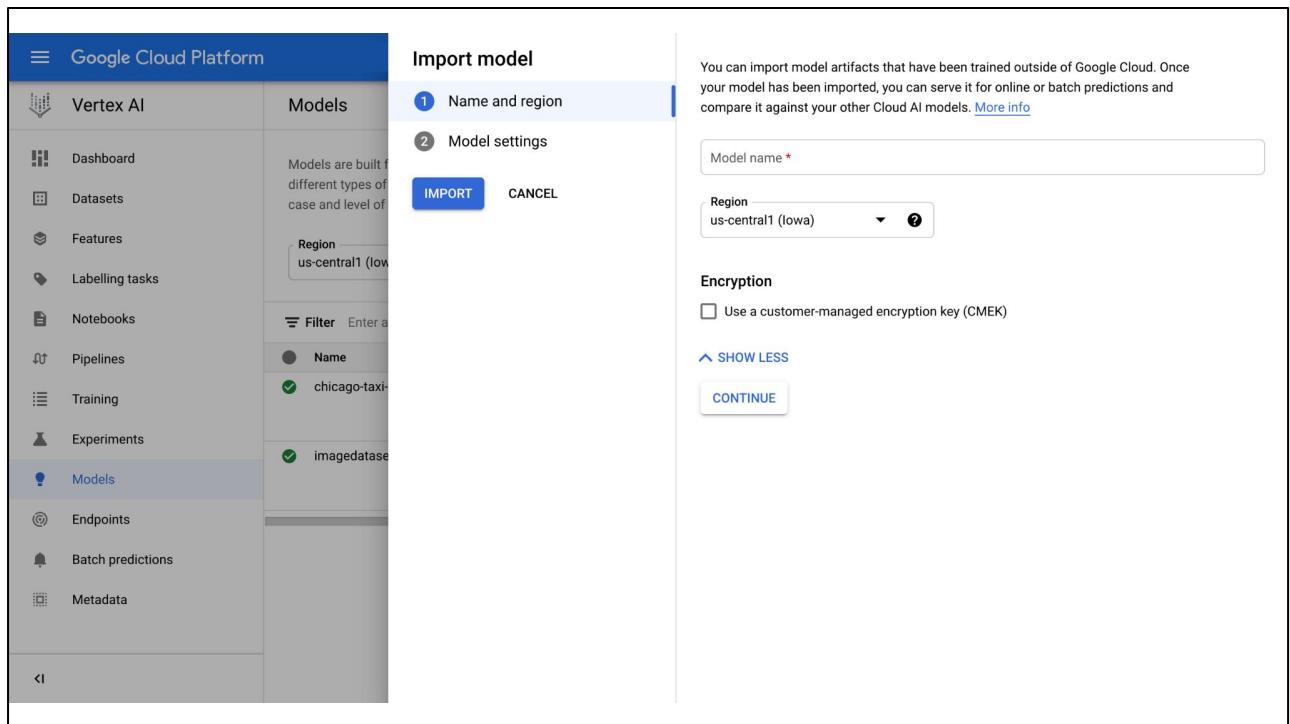
Both custom jobs and hyperparameter tuning, however, are only used by custom-trained models. They are not used by AutoML models.

The screenshot shows the Google Cloud Platform Vertex AI interface. On the left is a sidebar with various options: Dashboard, Datasets, Features, Labelling tasks, Notebooks, Pipelines, Training, Experiments, Models (which is selected), Endpoints, Batch predictions, and Metadata. The main area is titled 'Models' and contains a 'CREATE' button and an 'IMPORT' button. Below this is a section about building models from datasets or unmanaged data sources, mentioning many different types of machine learning models available. A 'Region' dropdown is set to 'us-central1 (Iowa)'. A 'Filter' input field allows entering a property name. A table lists two models:

Name	ID	Data	Endpoints	Region	Type	Created	N
chicago-taxi-tips-classifier-v01	6897412362899292160	—	0	us-central1	Imported Custom training	26 Jun 2021, 03:18:39	⋮
imagedataset_1623500071212_202161213847	80510639432269824	—	0	us-central1	Image classification AutoML	12 Jun 2021, 14:09:56	⋮

Next up are models.

Models are built from datasets or unmanaged data sources. Many different types of machine learning models are available with Vertex AI. The right choice will depend on the use case and your level of experience with machine learning.



A new model can be trained, or an existing model can be imported.

After the model has been imported into Vertex AI, it can be deployed to an endpoint and then used to request predictions.

The screenshot shows the Google Cloud Platform Vertex AI interface. On the left, there's a sidebar with various options like Dashboard, Datasets, Features, Labeling tasks, Notebooks, Pipelines, Training, Experiments, Models (which is selected), Endpoints, Batch predictions, and Metadata. The main area is titled "Train new model" and has four steps: 1. Training method (selected), 2. Model details, 3. Training options, and 4. Compute and pricing. Step 1 is currently active, showing a dropdown for "Dataset" set to "chicago-taxi-tips" and another dropdown for "Objective" set to "Classification". A note below says to refer to the pricing guide for more details. Below the steps are two radio button options: "AutoML" (selected) and "Custom training (advanced)". The "AutoML" option is described as training high-quality models with minimal effort and machine learning expertise. The "Custom training (advanced)" option is described as running TensorFlow, scikit-learn, and XGBoost training applications in the cloud. At the bottom right of the dialog is a "CONTINUE" button.

AutoML can be used to train a new model with minimal technical effort. It can be used to quickly prototype models and explore new datasets before investing in development.

For example, you might use AutoML to determine the good features in a dataset.

The screenshot shows the Google Cloud Platform Vertex AI interface. On the left, there's a sidebar with various options like Dashboard, Datasets, Features, Labeling tasks, Notebooks, Pipelines, Training, Experiments, Models (which is selected), Endpoints, Batch predictions, and Metadata. The main area is titled 'Train new model' and has a step-by-step process:

- 1 Training method
- 2 Model details
- 3 Training container
- 4 Hyperparameters (optional)
- 5 Compute and pricing
- 6 Prediction container (optional)

For step 1, 'Training method', the 'Custom training (advanced)' option is selected. The form fields show 'Dataset *' set to 'chicago-taxi-tips' and 'Objective *' set to 'Classification'. A note below says: 'Please refer to the pricing guide for more details (and available deployment options) for each method.' There are two radio button options: 'AutoML' (unselected) and 'Custom training (advanced)' (selected). Below the radio buttons is a note: 'Train high quality models with minimal effort and machine learning expertise. Just specify how long you want to train. [Learn more](#)'.

At the bottom of the 'Training method' section are 'START TRAINING' and 'CANCEL' buttons. To the right, there's a 'CONTINUE' button.

Generally speaking, custom training is used to create a training application optimized for a targeted outcome, because it allows for complete control over training application functionality. You can target any objective, use any algorithm, develop your own loss functions or metrics, or carry out any other customization.

The screenshot shows the Google Cloud Platform interface for Vertex AI. The left sidebar lists various options: Dashboard, Datasets, Features, Labelling tasks, Notebooks, Pipelines, Training, Experiments, Models, Endpoints (which is selected and highlighted in blue), Batch predictions, and Metadata. The main content area is titled 'Endpoints' and features a 'CREATE ENDPOINT' button. Below this is a detailed description of what endpoints are: machine learning models made available for online prediction requests. It mentions that endpoints are useful for timely predictions from many users and can also request batch predictions if immediate results aren't needed. A note states that at least one machine-learning model is required to create an endpoint, with a link to 'Learn more'. A 'Region' dropdown is set to 'us-central1 (Iowa)'. A 'Filter' input field allows entering a property name. A table lists existing endpoints:

	Name	ID	Models	Region	Monitoring	Most recent alerts	Last updated	API	Notification	Metadata
<input type="checkbox"/>	hello endpoint	3988412459059773440	0	us-central1	Disabled	—	27 Jul 2021, 08:00:01	—	—	—

And finally, let's explore Endpoints.

Endpoints are machine learning models made available for online prediction requests. An endpoint is an HTTPS endpoint that clients can call to receive the inferencing (scoring) output of a trained model. They can provide timely predictions from many users, for example, in response to an application request. They can also request batch predictions if immediate results aren't required.

Multiple models can be deployed to an endpoint, and a single model can be deployed to multiple endpoints to split traffic. You might deploy a single model to multiple endpoints to test out a new model before serving it to all traffic.

Either way, it's important to emphasize that a model must be deployed to an endpoint before that model can be used to serve online predictions.

The screenshot shows the Google Cloud Platform Vertex AI interface. On the left, there's a sidebar with various options like Dashboard, Datasets, Features, Labeling tasks, Notebooks, Pipelines, Training, Experiments, Models, Endpoints (which is selected and highlighted in blue), Batch predictions, and Metadata. The main area is titled "New endpoint" and is divided into two tabs: "Define your endpoint" (selected) and "Model settings". Under "Define your endpoint", there are fields for "Endpoint name" (with a red asterisk indicating it's required) and "Region" (set to "us-central1 (Iowa)"). Below these are sections for "Location" (Region dropdown) and "Access" (radio buttons for "Standard" and "Private"). The "Standard" option is selected. The "Access" section includes a note about AutoML and custom-trained models being available for standard endpoints, and a link to "private services access" for private endpoints. There's also a "CONTINUE" button at the bottom. The overall interface is clean and modern, typical of Google's cloud services.

To make that happen, you must define an endpoint in Vertex AI by giving it a name and location and deciding whether the access is Standard, which makes the endpoint available for prediction serving through a REST API.



Vertex AI

cloud.google.com/vertex-ai



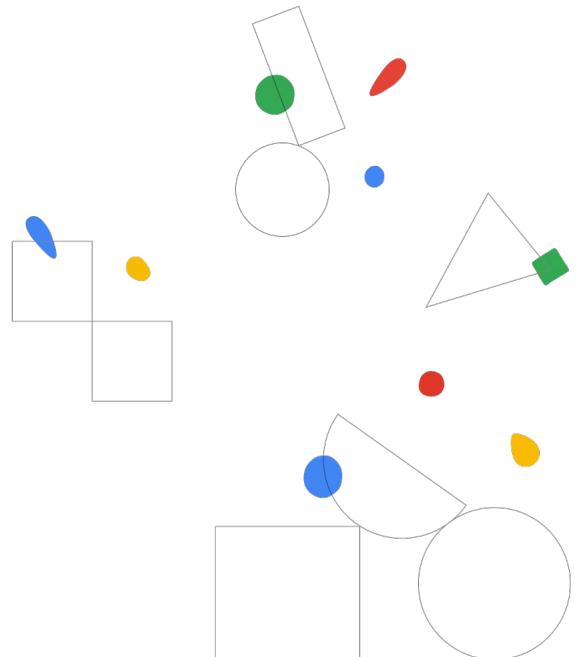
This has been a brief introduction to Vertex AI, Google Cloud's unified ML platform.
For more information, please see cloud.google.com/vertex-ai.



Lab

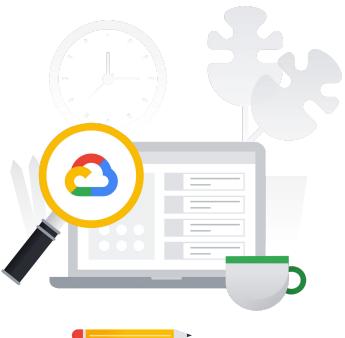
Structured Data Prediction using AI Platform

Module 01
Architecting Production ML Systems



This lab provides hands-on practice using Google Cloud's AI Platform

Lab objectives

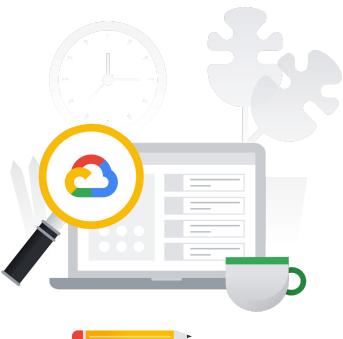


Create a BigQuery Dataset



You'll start by creating a BigQuery Dataset,

Lab objectives



Create a BigQuery Dataset

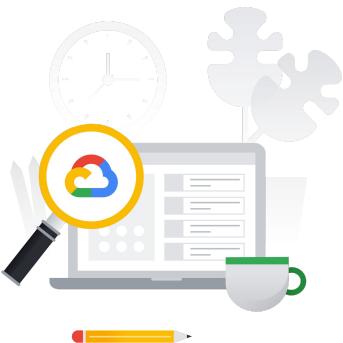


Export the data into a GCS bucket and train using Cloud AI Platform



then Export the data into a GCS bucket and train using Cloud AI Platform.

Lab objectives



- ✓ Create a BigQuery Dataset.
- ✓ Export the data into a GCS bucket and train using Cloud AI Platform.
- ✓ Deploy the trained model using Cloud AI Platform



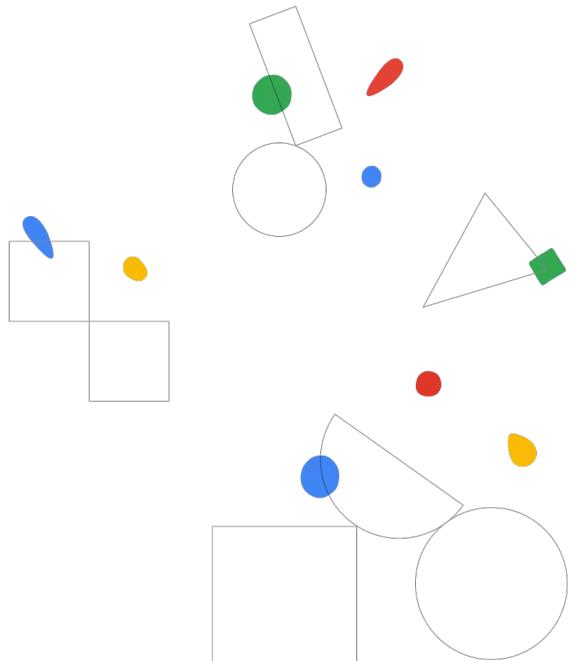
And finally, deploy the trained model using Cloud AI Platform.



Introduction

Module 02

Designing adaptable ML systems



Welcome to
Designing adaptable
machine learning systems



Welcome to Designing Adaptable ML systems.

Objectives

1

Recognize the ways that a model
is dependent on data



In this module, we'll explore how to:

recognize the ways that a model is dependent on data,

make cost-conscious engineering decisions,

know when to roll back a model to an earlier version,

debug the causes of observed model behavior,

and implement a pipeline that is immune to one type of dependency.

Objectives

- 1 Recognize the ways that a model is dependent on data
- 2 Make cost-conscious engineering decisions



In this module, we'll explore how to:

recognize the ways that a model is dependent on data,

make cost-conscious engineering decisions,

know when to roll back a model to an earlier version,

debug the causes of observed model behavior,

and implement a pipeline that is immune to one type of dependency.

Objectives

- 1 Recognize the ways that a model is dependent on data
- 2 Make cost-conscious engineering decisions
- 3 Know when to roll back a model to an earlier version



In this module, we'll explore how to:

recognize the ways that a model is dependent on data,

make cost-conscious engineering decisions,

know when to roll back a model to an earlier version,

debug the causes of observed model behavior,

and implement a pipeline that is immune to one type of dependency.

Objectives

- 1 Recognize the ways that a model is dependent on data.
- 2 Make cost-conscious engineering decisions.
- 3 Know when to roll back a model to an earlier version
- 4 Debug the causes of observed model behavior



In this module, we'll explore how to:

recognize the ways that a model is dependent on data,

make cost-conscious engineering decisions,

know when to roll back a model to an earlier version,

debug the causes of observed model behavior,

and implement a pipeline that is immune to one type of dependency.

Objectives

- 1 Recognize the ways that a model is dependent on data
- 2 Make cost-conscious engineering decisions
- 3 Know when to roll back a model to an earlier version
- 4 Debug the causes of observed model behavior
- 5 Implement a pipeline that is immune to one type of dependency



In this module, we'll explore how to:

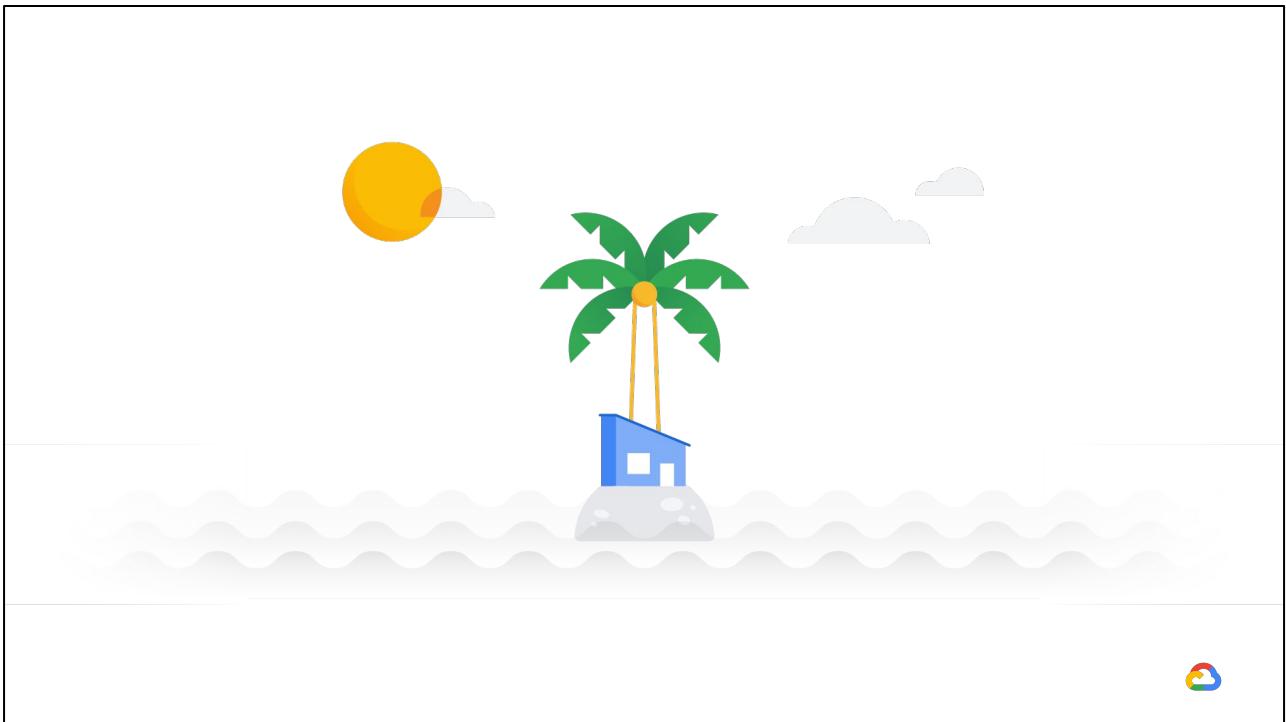
recognize the ways that a model is dependent on data,

make cost-conscious engineering decisions,

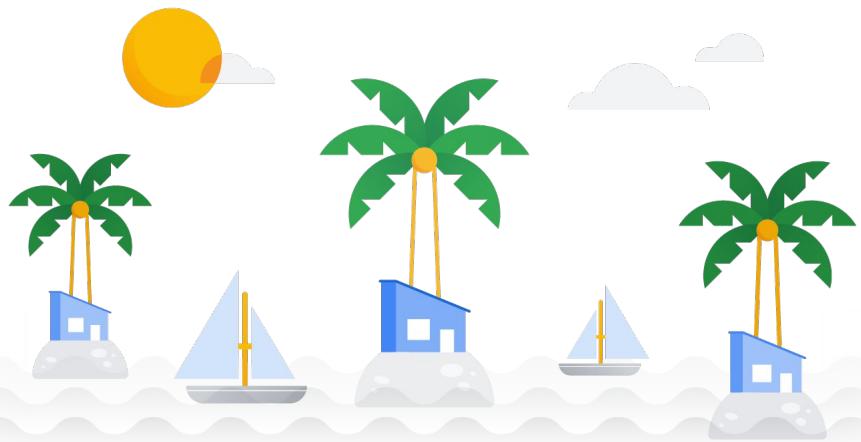
know when to roll back a model to an earlier version,

debug the causes of observed model behavior,

and implement a pipeline that is immune to one type of dependency.

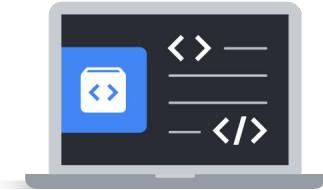


In the 16th century, John Donne famously wrote in one of his poems that no man is an island.



He meant that human beings need to be part of a community to thrive.

Software today is modular

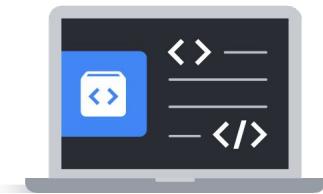


Monolithic design

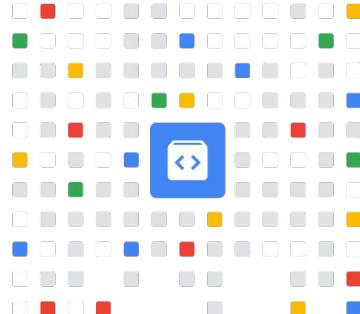


In software engineering terms, we would say that few software programs adopt a monolithic island-like design.

Software today is modular



Monolithic design

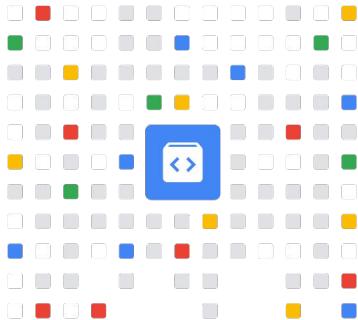


Modular design



Instead, most software today is modular, and depends on other software.

Modular programs are more maintainable



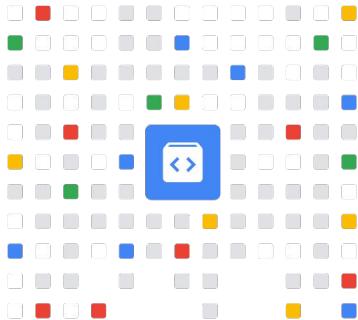
More maintainable

Modular design



Modular programs are more maintainable,

Modular programs are more maintainable



More maintainable

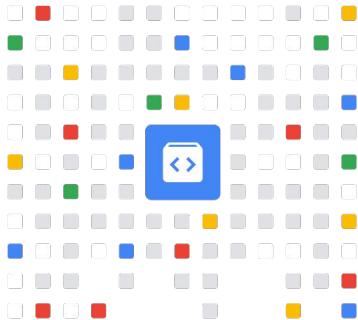
Easier to reuse

Modular design



as well as easier to reuse,

Modular programs are more maintainable



More maintainable

Easier to reuse

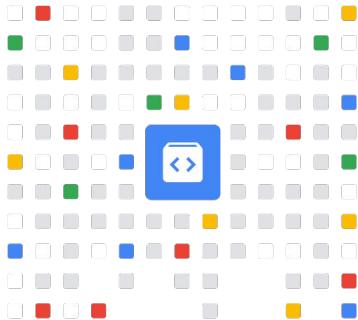
Easier to test

Modular design



test,

Modular programs are more maintainable



More maintainable

Easier to reuse

Easier to test

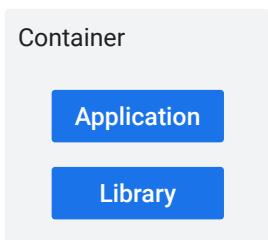
Easier to fix

Modular design



and fix because they allow engineers to focus on small pieces of code rather than the entire program.

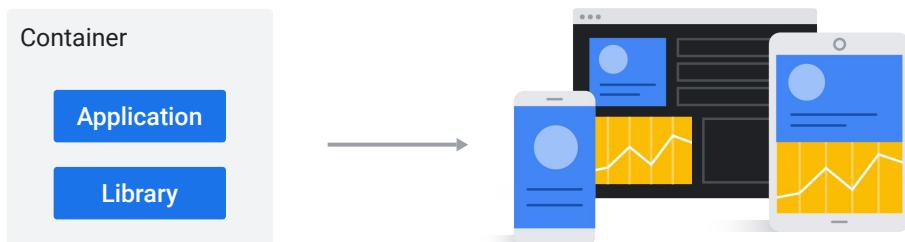
Containers



Containers make it easier to manage modular programs.

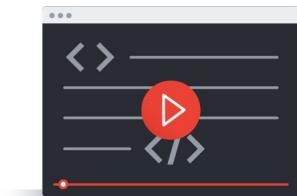
A container is an abstraction that packages applications and libraries together

Containers



so that the applications can run on a greater variety of hardware and operating systems. This ultimately makes hosting large applications better.

Kubernetes



Getting started
with Google
Kubernetes Engine



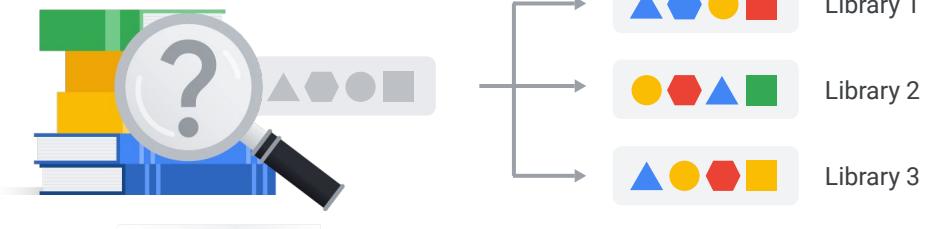
To learn more about Kubernetes, Google's open source container orchestration software, check out the getting started with Google Kubernetes engine course.

Identify a specific version of a library



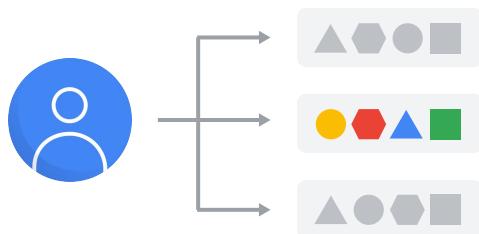
But what if there was no way to identify a specific version of a library,

Identify a specific version of a library



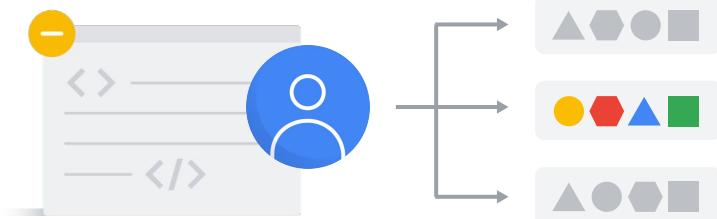
and you had to rely on finding similar libraries at run-time?

Identify a specific version of a library



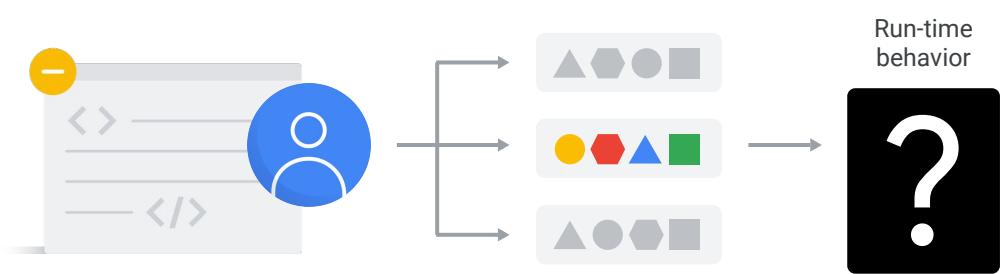
Furthermore, what if someone else got to choose which version got run

Identify a specific version of a library



and they didn't know or really care about your program?

Identify a specific version of a library



There would be no way of knowing what the run-time behavior would look like.



This is precisely the case for ML



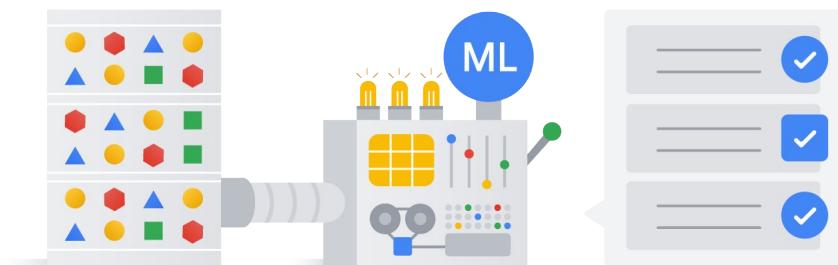
Unfortunately, this is precisely the case for machine learning, because the

This is precisely the case for ML



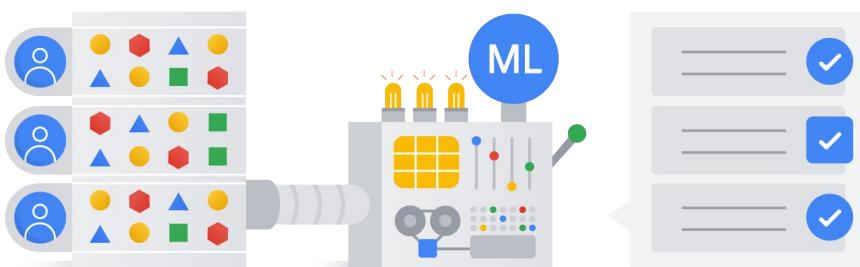
run-time instructions, for example, the model weights, depend on the data that the model was trained on.

This is precisely the case for ML



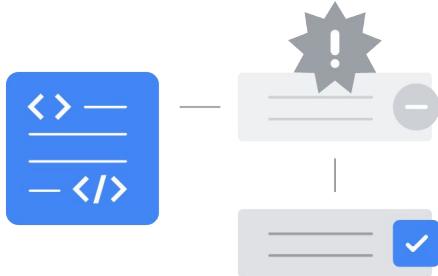
Additionally, similar data will yield similar instructions.

This is precisely the case for ML



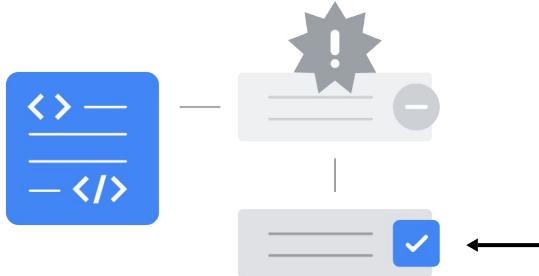
And finally other people including other teams and our users create our data.

Mismanaged dependencies



Just like in traditional software engineering, mismanaged dependencies

Mismanaged dependencies



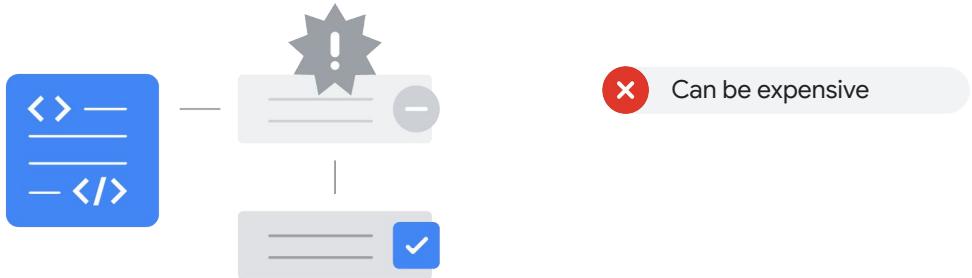
say, code that assumes one set of instructions will be called when another

Mismanaged dependencies



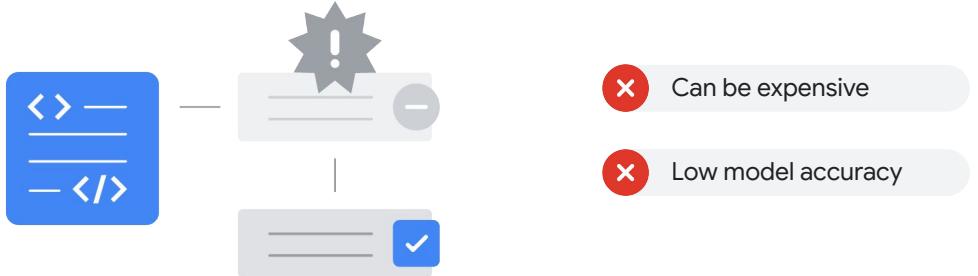
end up being called instead

Mismanaged dependencies



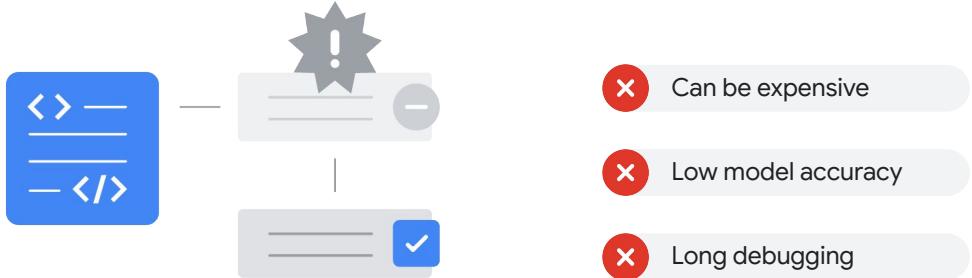
Just like in traditional software engineering, mismanaged dependencies -- say, code that assumes one set of instructions will be called when another end up being called instead--can be expensive.

Mismanaged dependencies



Your models' accuracy might go down or become unstable.

Mismanaged dependencies



Sometimes, the errors are subtle and your team may end up spending a large proportion of its time debugging.

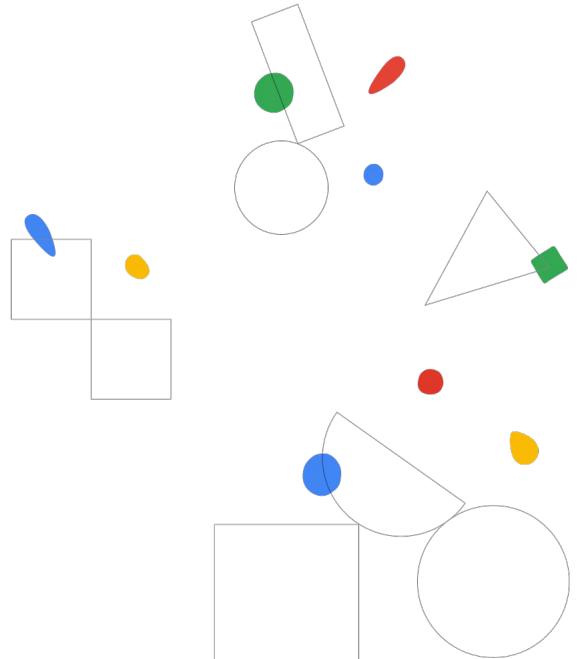
With a better understanding of how to manage data dependencies, many problems can be either **detected quickly** or **circumvented entirely**



Adapting to data

Module 02

Designing adaptable ML systems



Adapting to change

When it comes to adapting to change, consider which of these four is most likely to change?



When it comes to adapting to change, consider which of these four is more likely to change?

Adapting to change

A

An upstream model

When it comes to adapting to change, consider which of these four is most likely to change?



An upstream model,

Adapting to change

A An upstream model

B A data source maintained by another team

When it comes to adapting to change, consider which of these four is most likely to change?



a data source maintained by another team,

Adapting to change

When it comes to adapting to change, consider which of these four is most likely to change?

A An upstream model

B A data source maintained by another team

C The relationship between features and labels



the relationship between features and labels,

Adapting to change

When it comes to adapting to change, consider which of these four is most likely to change?

- A An upstream model
- B A data source maintained by another team
- C The relationship between features and labels
- D The distributions of inputs



or the distributions of inputs. The answer is

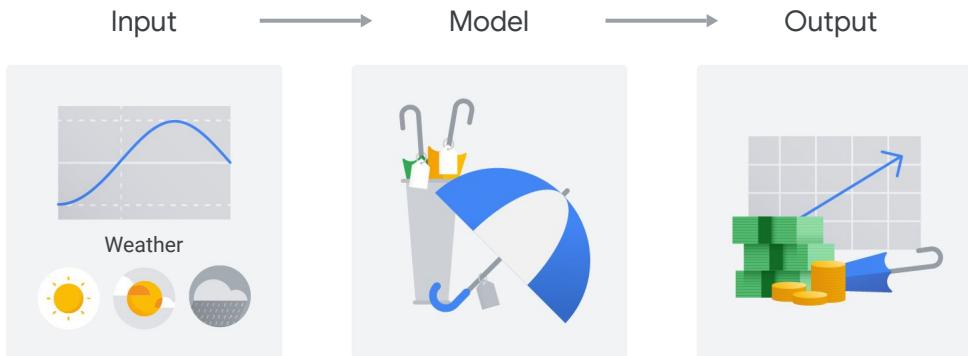
Adapting to change

When it comes to adapting to change, consider which of these four is most likely to change?

- An upstream model
- A data source maintained by another team
- The relationship between features and labels
- The distributions of inputs

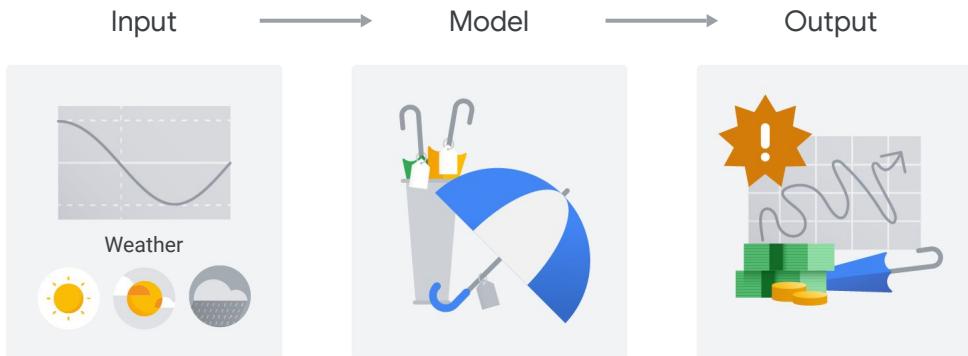


that all of them can, and often do, change. Let's see how this happens, and what to do about it with a couple example scenarios.



Let's say that you've created a model to predict demand for umbrellas that accepts as input an output from a more specialized weather prediction model.

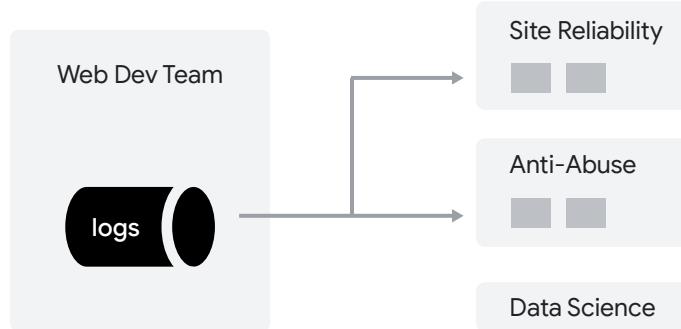
Unbeknownst to you and the owners, this model has been trained on the wrong years of data. Your model, however, is fit to the upstream model's outputs. What could go wrong?



One day, the model owners silently push a fix and the performance of your model, which expected the old model's distribution of data, drops.

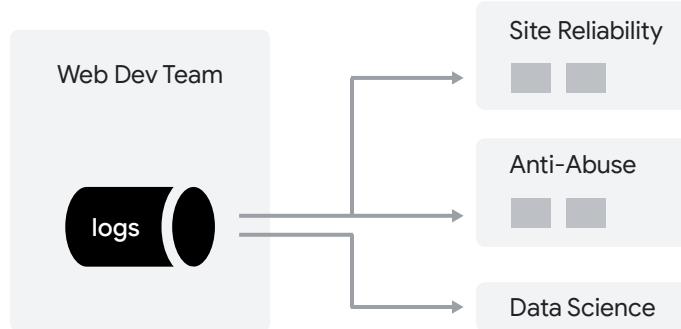
The old data had below-average rainfall and now you're under-predicting the days when you need an umbrella.

Decoupled upstream data producers



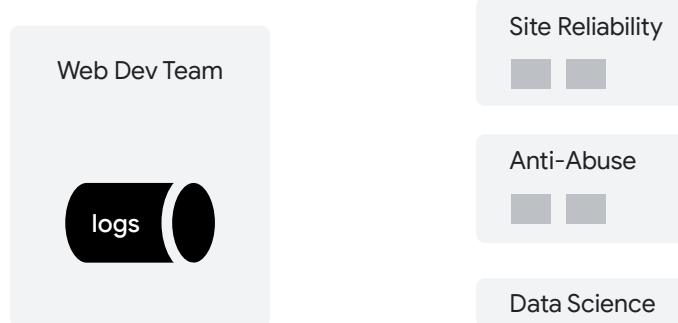
Here's another scenario. Let's say your small data science team

Decoupled upstream data producers



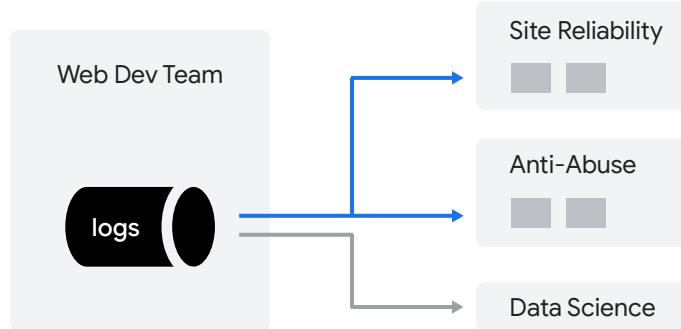
has convinced the web development team to let you ingest their traffic logs.

Decoupled upstream data producers



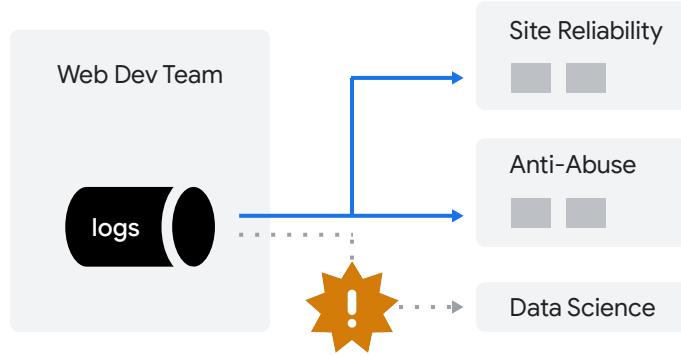
Later, the web development team refactors their code and changes their logging format,

Decoupled upstream data producers



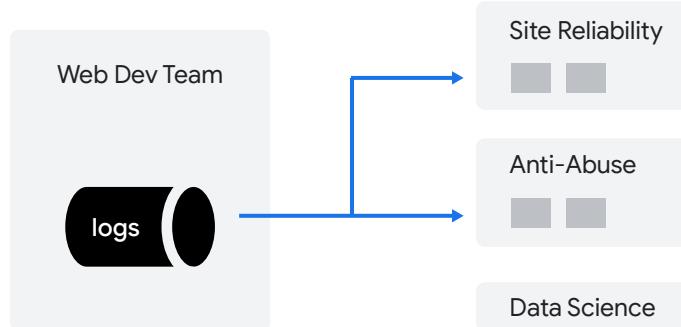
but continues publishing the old format.

Decoupled upstream data producers



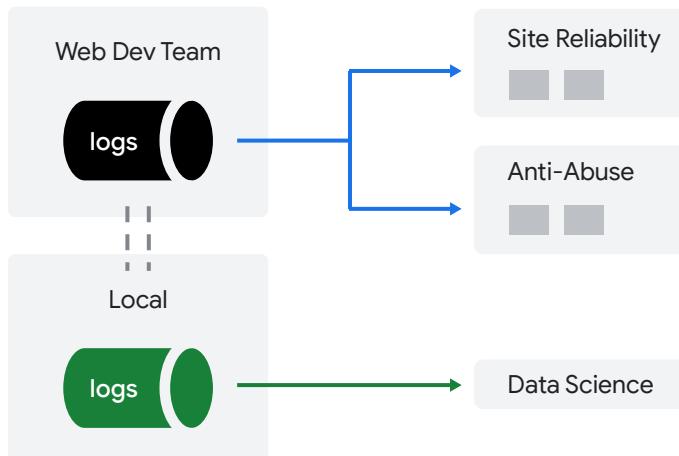
At some point, they stop publishing in the old format but they forget to tell your team. Your model's performance degrades after getting an unexpectedly high number of null features.

Decoupled upstream data producers

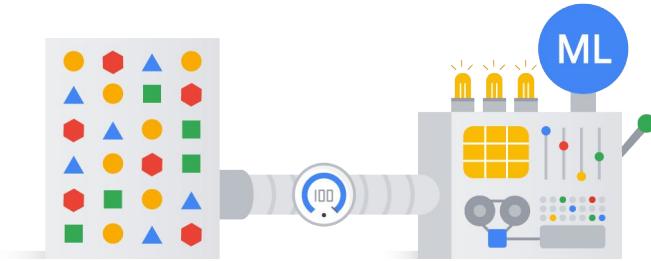


To fix this problem, first, you should stop consuming data from a source that doesn't notify downstream consumers.

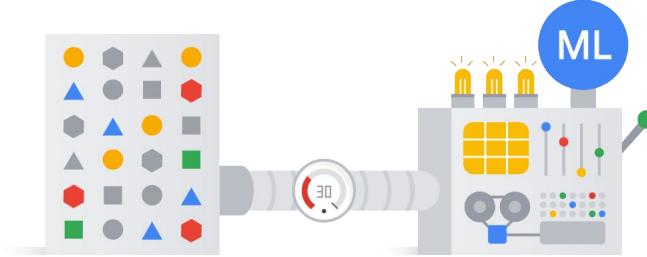
Decoupled upstream data producers



Second, you should consider making a local version of the upstream model and keeping it updated.

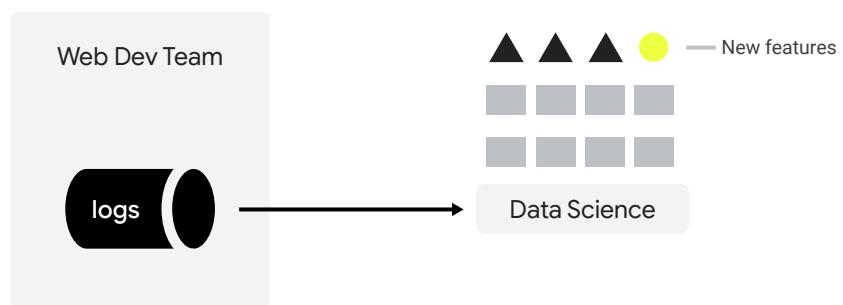


Sometimes, the set of features that the model has been trained on include



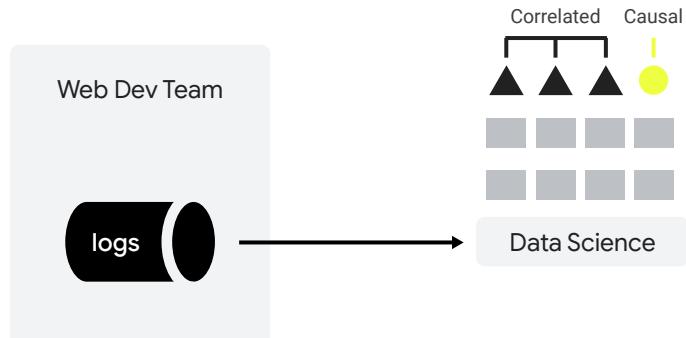
many that were added indiscriminately, which may worsen performance at times.

Underutilized data dependencies



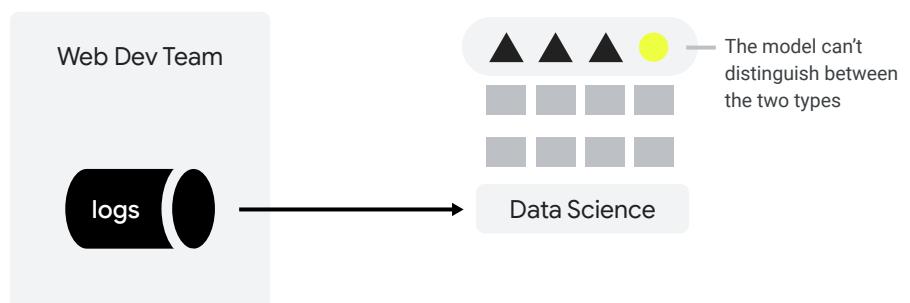
For example, under pressure during a sprint, your team decided to include a number of new features without understanding their relationship to the label.

Underutilized data dependencies



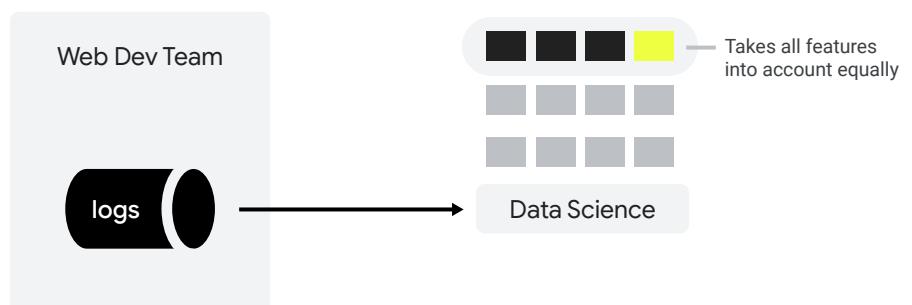
One of them is causal, while the others are merely correlated with the causal one.

Underutilized data dependencies



The model can't distinguish between the two types, and takes all features into account equally.

Underutilized data dependencies



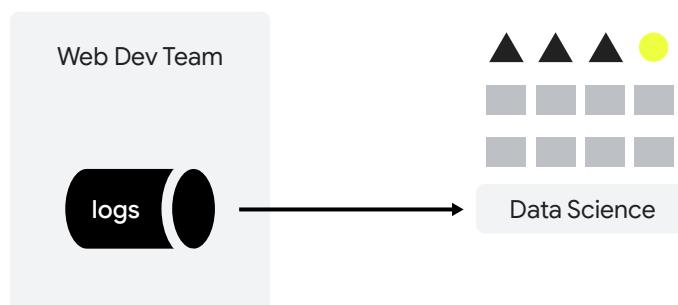
and takes all features into account equally.

Underutilized data dependencies



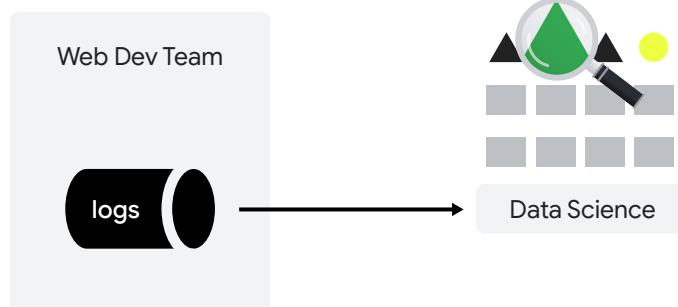
Months later, the correlated feature becomes decorrelated with the label and is thus no longer predictive. The model's performance suffers.

Underutilized data dependencies



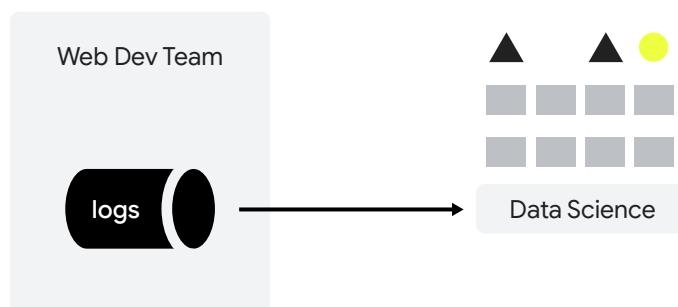
To address this,

Underutilized data dependencies



features should always be scrutinized before being added, and all features should be subjected to leave-one-out evaluations, to assess their importance.

Underutilized data dependencies

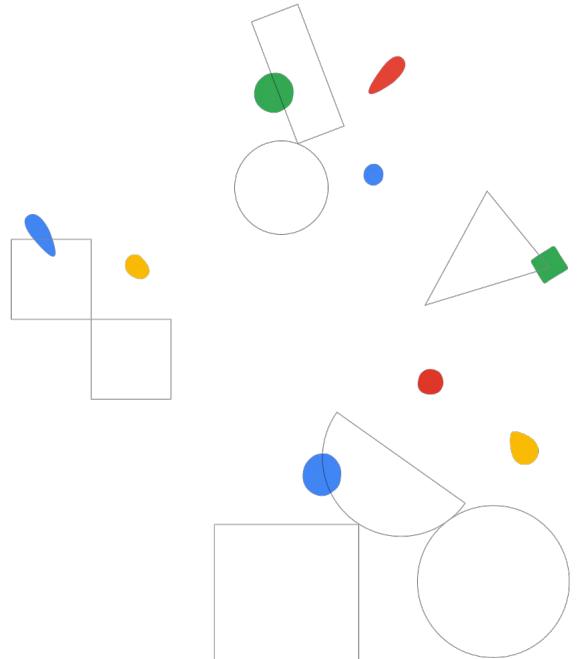


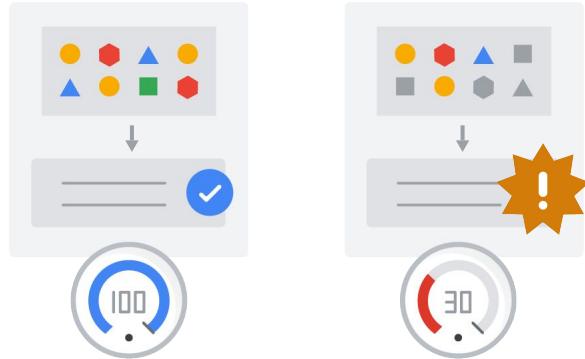
and all features should be subjected to leave-one-out evaluations, to assess their importance.

Changing distributions

Module 02

Designing adaptable ML systems





Earlier you saw how, in the context of ingesting an upstream model, our model's performance would degrade if it expected one input but ingested another.

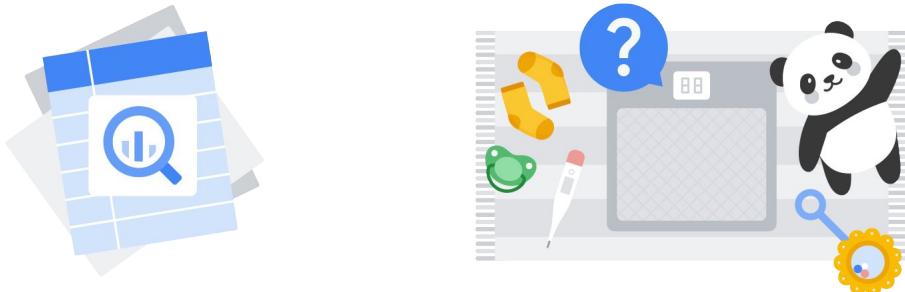
The statistical term for changes in the likelihood of observed values like model inputs is **changes in the distribution**



The statistical term for changes in the likelihood of observed values like model inputs is *changes in the distribution*.

Changes in label distribution

Natality database → Baby weight prediction

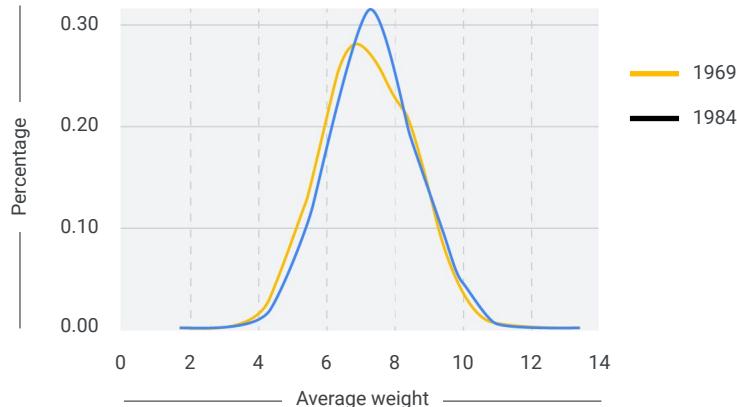


For example, sometimes the distribution of the label changes.

We've looked at the natality dataset in BigQuery and tried to predict baby weight.

Baby weight has actually changed over time. It peaked in the 1980s and has since been declining.

Changes in label distribution



In 1969, babies weighed significantly less than they did in 1984. When the distribution of the label changes, it could mean that the relationship between features and labels is changing as well.

At the very least, it's likely that our model's predictions, which will typically match the distribution of the labels in the training set, will be significantly less accurate.

Changes in feature distribution

Postal code



Population movement patterns



However, sometimes it's not the labels, but the features, that change their distribution.

For example, say you've trained your model to predict population movement patterns using postal code as a feature. Surprisingly, postal codes aren't fixed. Every year, governments release new ones and deprecate old ones.

Changes in feature distribution

Postal codes

99501

87506

63141

98723

23451

...



```
tf.feature_column.categorical_column_with_vocabulary_list(  
    'postal_code',  
    Vocabulary_list = ['99501', '87506', '63141', '98723', '23451']),
```

99501	87506	63141	98723	23451
-1	0	0	0	0
99501	87506	63141	98723	23451
0	-1	0	0	0
99501	87506	63141	98723	23451
0	0	-1	0	0



Now as a ML practitioner, you know that postal codes aren't really numbers. So you've chosen to represent them as categorical feature columns, but this might lead to problems. If you chose to specify a vocabulary, but set the number of out of vocab buckets to 0, and didn't specify a default, then the distribution may become skewed toward the default value, which is -1.

Changes in feature distribution



■ Features



And this might be problematic because the model may be forced to make predictions in regions of the feature space which were not well represented in the training data.

Changes in feature distribution



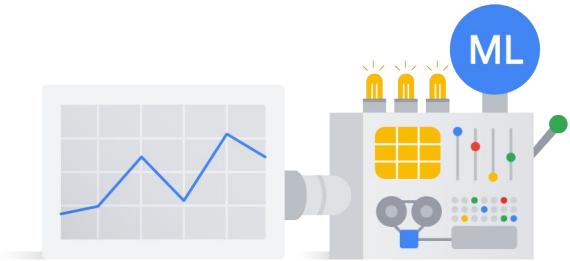
There's another name for when models are asked to make predictions on points in feature space that are far away from the training data, and that's extrapolation.

Extrapolation means to generalize outside the bounds of what we've previously seen.

Protect from changing distributions



Monitoring



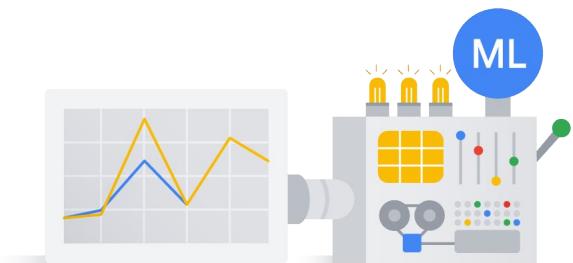
You can protect yourself from changing distributions using a few different methods.

The first thing you can do is be vigilant through monitoring. You can look at the descriptive summaries of your inputs

Protect from changing distributions



Monitoring



and compare them to what the model has seen. If, for example, the mean or the variance has changed substantially,

Protect from changing distributions



Monitoring



then you can analyze this new segment of the input space, to see if the relationships learned still hold.

Protect from changing distributions

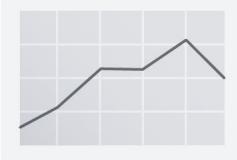


Monitoring

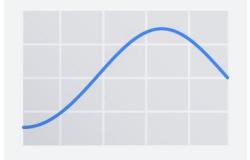


Check residuals

Prediction



Label

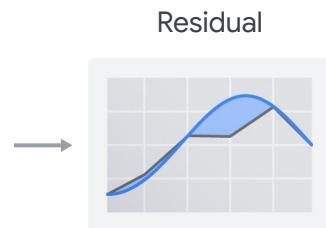
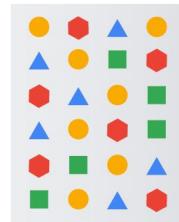


You can also look to see whether the model's residuals, that is the

Protect from changing distributions

 Monitoring

 Check residuals



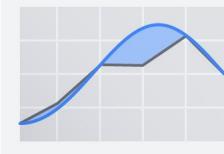
difference between its predictions and the labels, has changed as a function of your inputs.

Protect from changing distributions

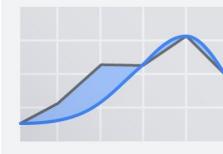
Monitoring

Check residuals

Before



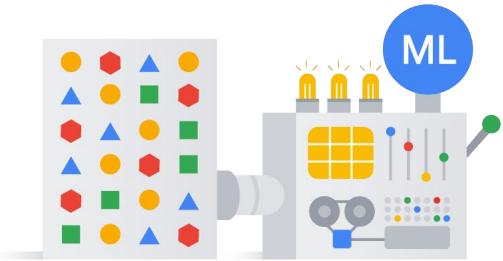
After



If, for example, you used to have small errors at one slice of the input and large in another, and now it's switched, this could be evidence of a change in the relationship.

Protect from changing distributions

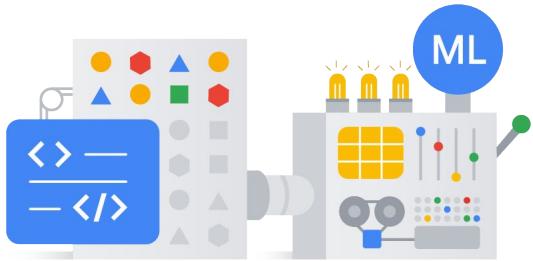
- Monitoring
- Check residuals
- Emphasize data recency



Finally, if you have reason to believe that the relationship is changing over time,

Protect from changing distributions

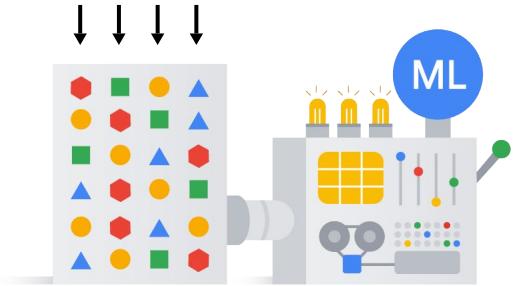
- Monitoring
- Check residuals
- Emphasize data recency



you can force the model to treat more recent observations as more important by writing a custom loss function,

Protect from changing distributions

- Monitoring
- Check residuals
- Emphasize data recency
- Regularly retrain your model



or by retraining the model on the most recent data.

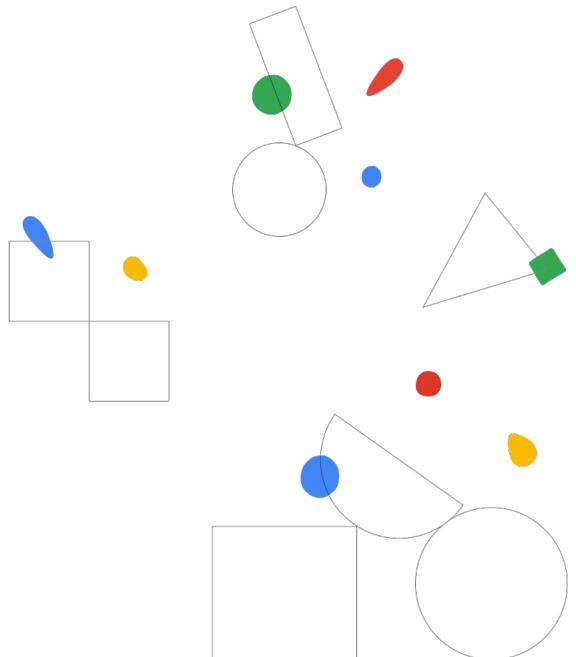


Lab

Adapting to Data

Module 02

Designing adaptable ML systems

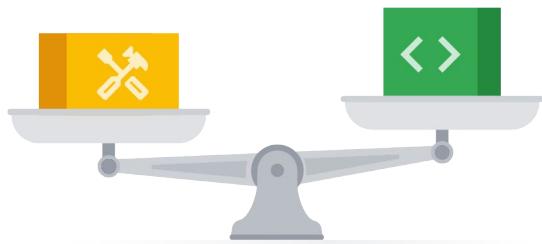




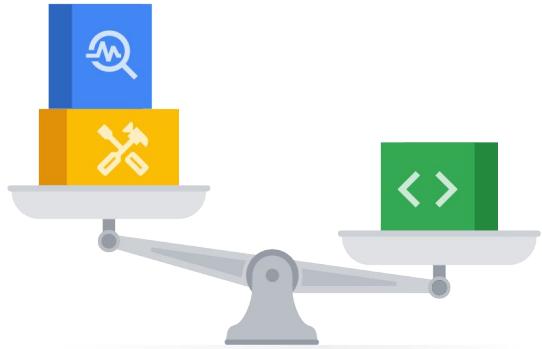
When leading a team of engineers,



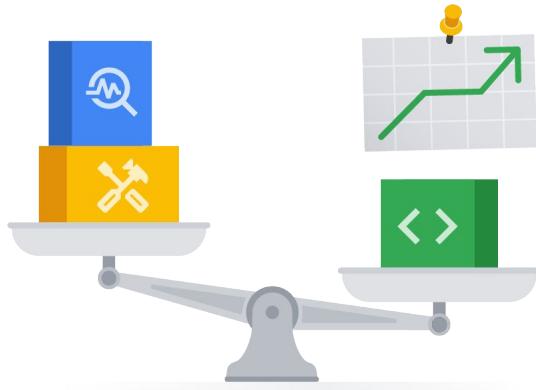
many decisions are informed by



technical debt and other sorts of



cost-benefit analyses.

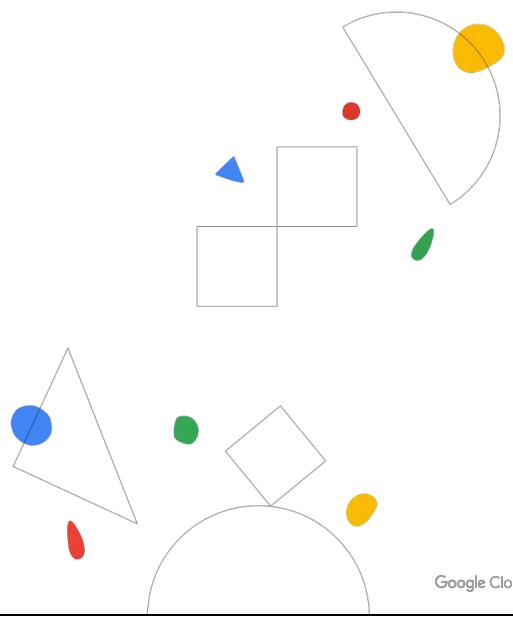


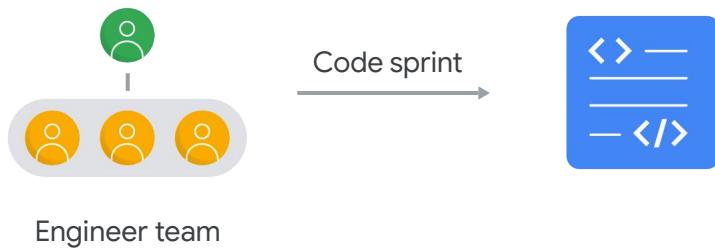
Very high rates of return



The best teams get very high rates of return on their investments. With that in mind, let's consider a few scenarios.

Scenario 1 Code Sprint





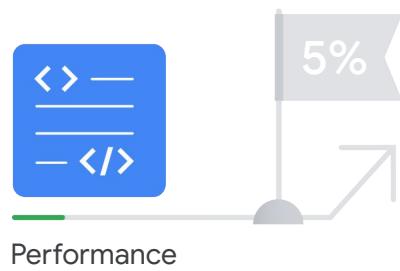
Engineer team



Let's imagine that you're the leader of a team of engineers and you are nearing the end of a code sprint.



One of the team's goals for the sprint is to increase performance on the model by 5%.



Performance



Currently, however, the best performing model is only marginally better than what was around before.

Ablation analysis

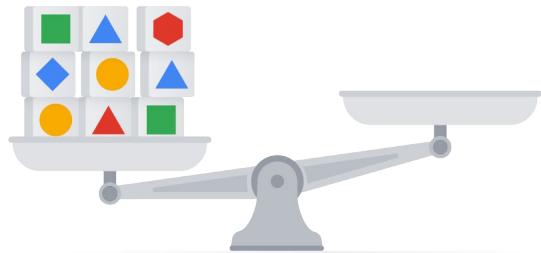


One of the engineers acknowledges this but still insists that it's worth spending time doing an extensive ablation analysis



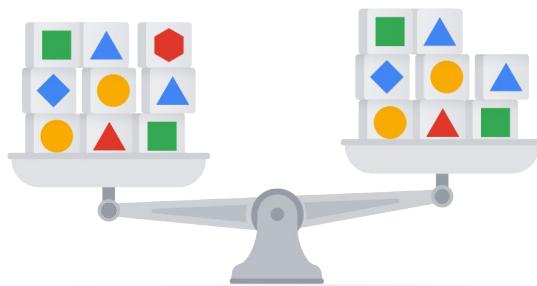
where the value

Model with feature



of an individual feature is computed by comparing it

Model with feature Model without feature



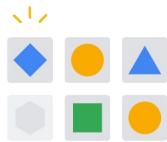
to a model trained without it. What might this engineer be concerned about?

Legacy

Bundled features



The engineer might be concerned about legacy and bundled features.



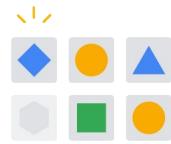
Legacy

Bundled features

Made redundant by
the implementation
of new features



Legacy features are older features that were added, because they were valuable at the time. But since then, better features have been added, which have made them redundant without our knowledge.



Legacy

Made redundant by
the implementation
of new features



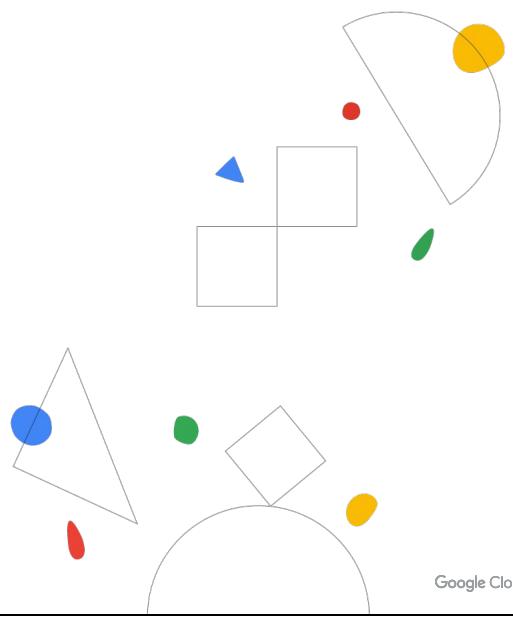
Bundled features

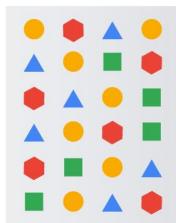
Added as part of a
bundle and not
valuable individually



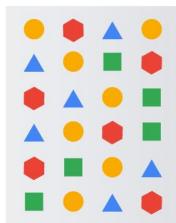
Bundled features on the other hand, are features that were added as part of a bundle, which collectively are valuable but individually may not be. Both of these features represent additional unnecessary data dependencies.

Scenario 2 A Gift Horse





In another scenario, another engineer has found a new data source that is very much related to the label.



Language 1



Python



The problem is that it's in a unique format and there's no parser written in Python, which is what the codebase is composed of.



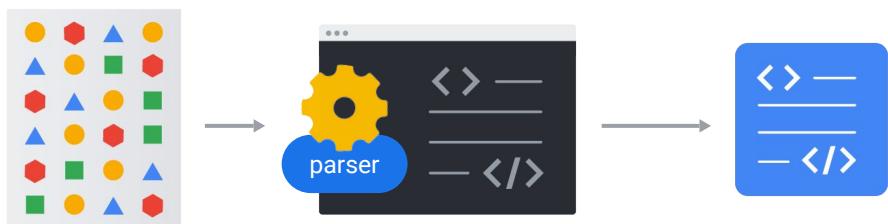
Thankfully, there is a parser on the web but it's closed source and written in a different language.

The engineer is thinking about the model performance.

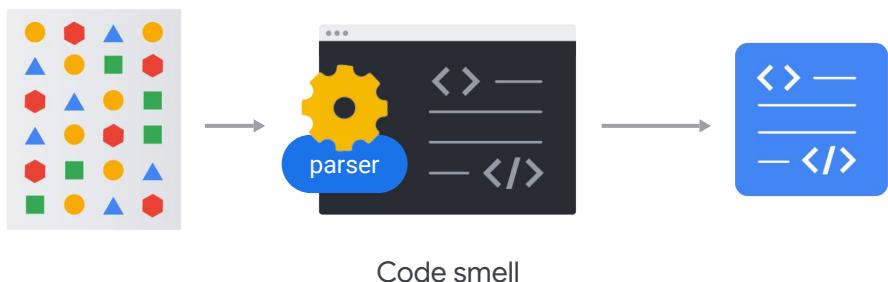
Something seems **wrong**



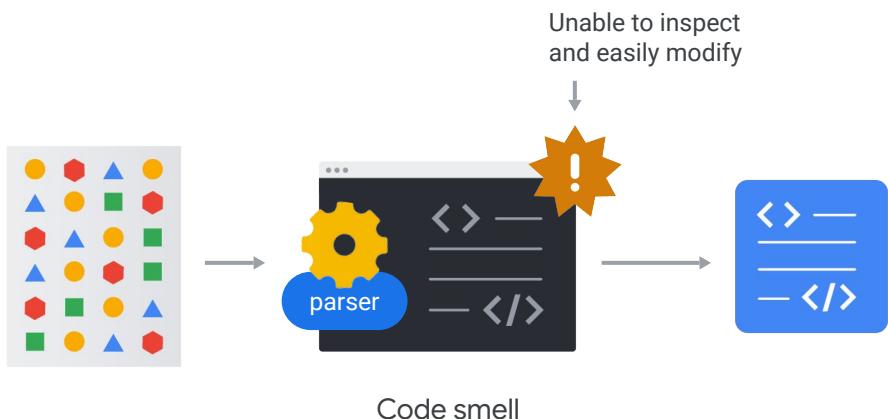
Something in the back of your mind seems wrong.



What is it? It's the smell.



No, really! There's a concept called code smell and it applies in ML as well.

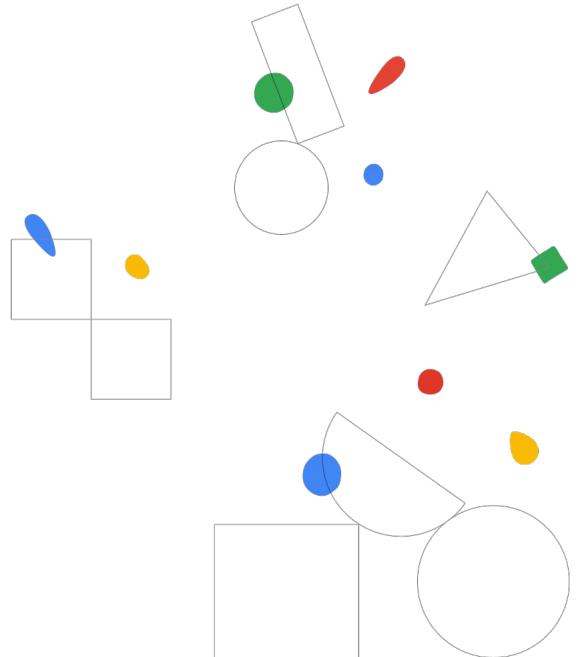


In this case, you might be thinking, "I wonder what introducing code that we can't inspect and are unable to easily modify into our testing in production frameworks will do."

Right and wrong decisions

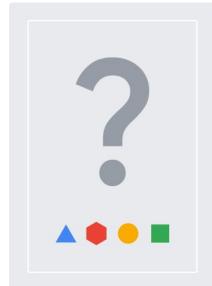
Module 02

Designing adaptable ML systems

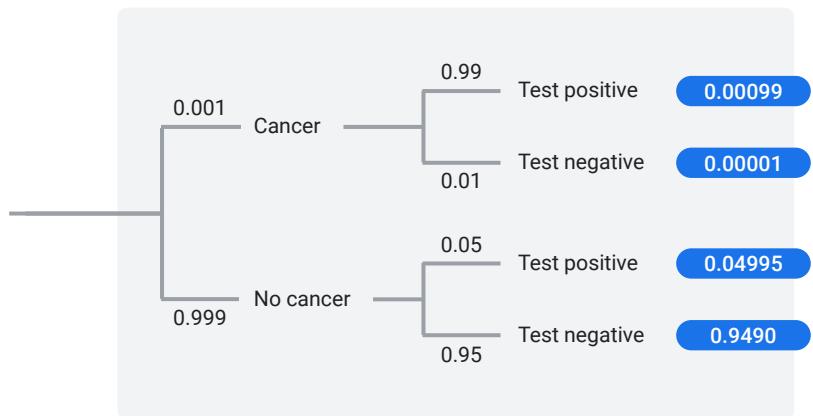
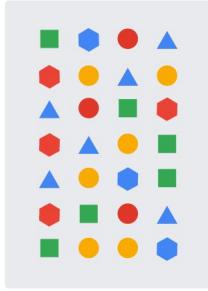


Costs vs. benefits

Right vs. wrong

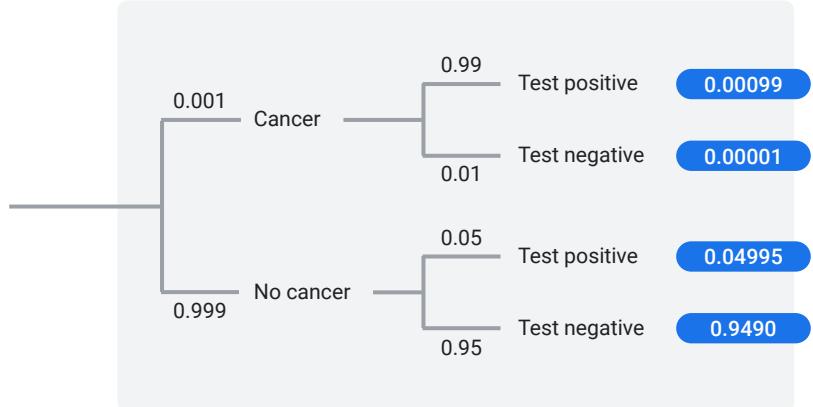


Some decisions about data are a matter of weighing cost vs. benefit, like short-term performance goals against long-term maintainability. Others, though, are about right and wrong.



For example, let's say that you've trained a model to predict "probability a patient has cancer" from medical records and

- ▲ Patient age
- ◆ Gender
- Prior medical conditions
- Hospital name
- ◆ Vital signs
- Test results

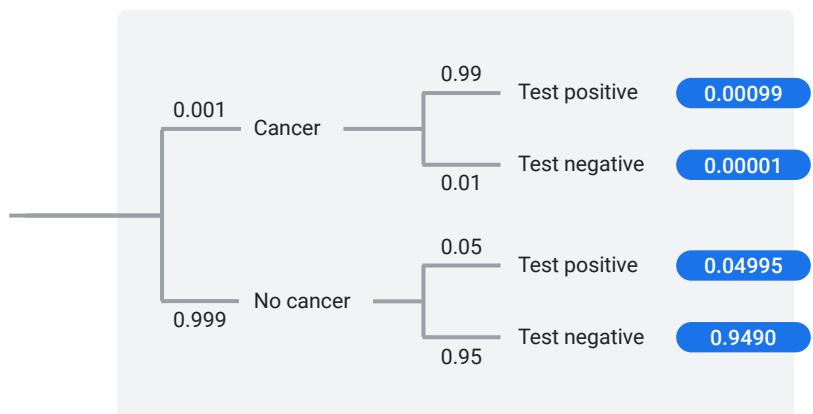
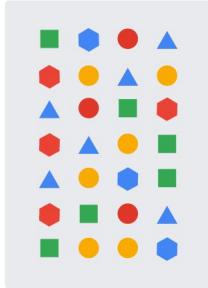


that you've selected patient age, gender, prior medical conditions, hospital name, vital signs, and test results as features. Your model had excellent performance on held-out test data but performed terribly on new patients.

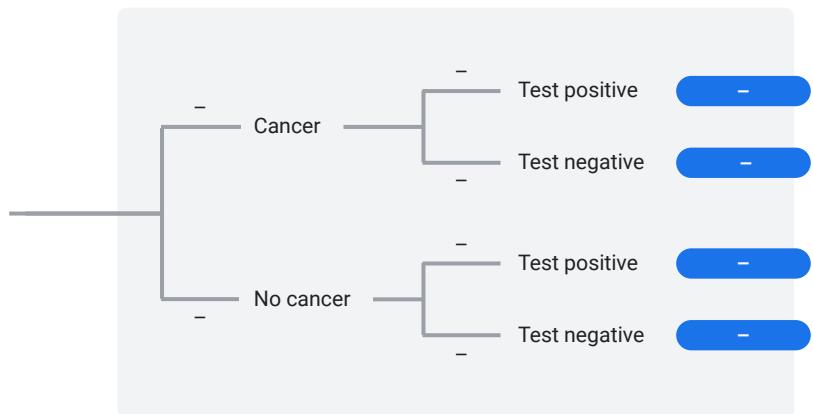
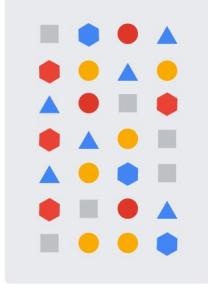
Any guesses as to **why**?



Any guesses as to why?

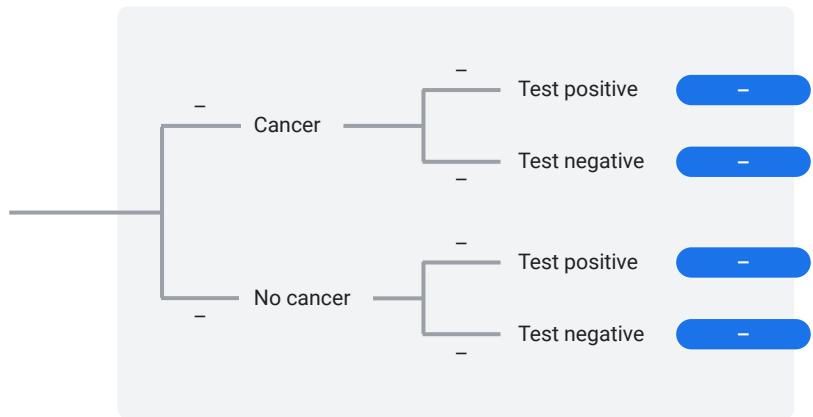


It turns out the model was trained using a feature



that wasn't legitimately available at decision time, and so, when the model was deployed into production, the distribution of this feature changed and it was no longer a reliable predictor.

- ▲ Patient age
- ◆ Gender
- Prior medical conditions
- Hospital name
- ◆ Vital signs
- Test results



In this case, that feature was 'hospital name'. You might think, 'hospital name'... How could that be predictive? Well, remember that there are some hospitals that focus on diseases like cancer. So, the model learned that 'hospital name' was very important.



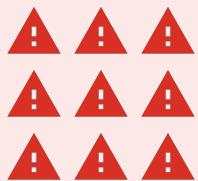
However, at decision time, this feature wasn't available to the model, because patients hadn't yet been assigned to a hospital, but rather than throwing an error, the model simply interpreted the hospital name as an empty string, which it was still capable of handling thanks to out-of-vocabulary buckets in its representations of words.

We refer to this idea where the label is somehow leaking into the training data as **data leakage**



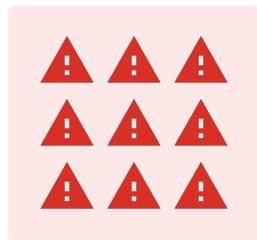
We refer to this idea where the label is somehow leaking into the training data as data leakage.

Data leakage



Data leakage is related to a broader class of problems we've seen before in the last specialization

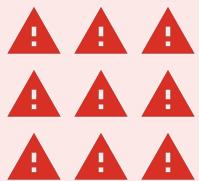
Data leakage → Models learning unacceptable strategies



where we talked about models learning unacceptable strategies.

Data leakage

→ Models learning unacceptable strategies



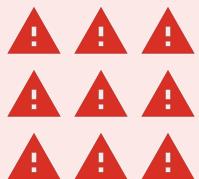
Predict the majority class



Previously, we learned that when there's class imbalances, a model might learn to predict the majority class.

Data leakage

→ Models learning unacceptable strategies



Predict the majority class

Use an unknown feature

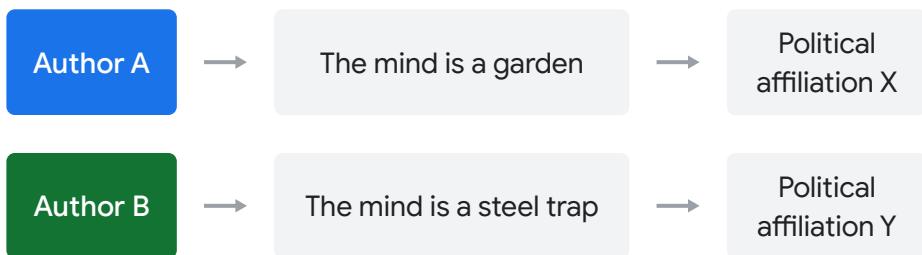


In this case, the model has learned to use a feature that wouldn't actually be known and which cannot be plausibly causally related to the label.



Here's a similar case.

A professor of 18th century literature believed that there was a relationship between how an author thought about the mind and their political affiliation.



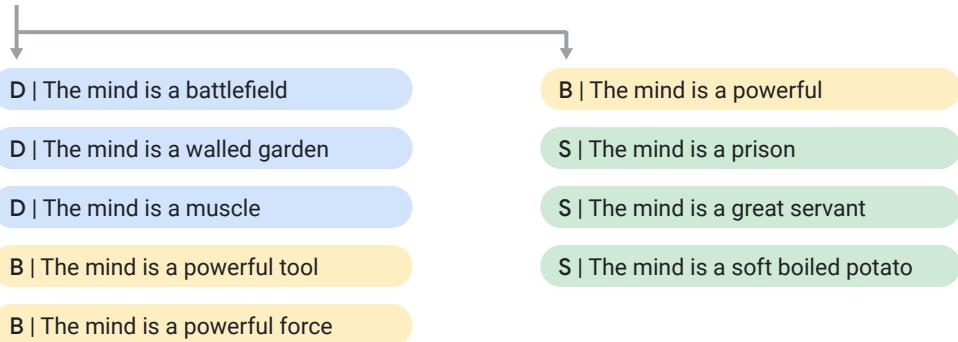
So, for example, perhaps authors who used language like “the mind is a garden” had one political affiliation and authors who used language like “the mind is a steel trap” another. Here’s what they did.

What if we were to naively test this hypothesis with machine learning?



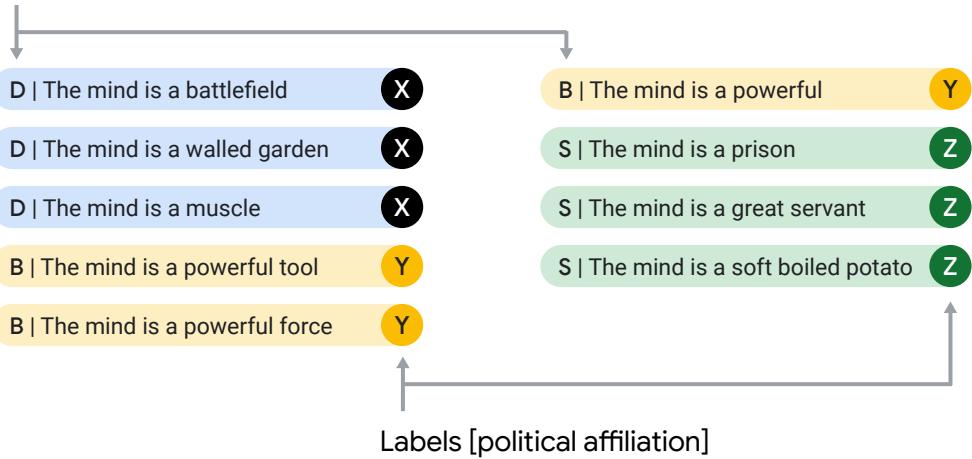
What if we were to naively test this hypothesis with machine learning? Some people tried that and they got some unexpected results.

Feature [mind metaphor]



They took all of the sentences in all of the works by a number of 18th century authors, extracted just the mind metaphors and set those as their features

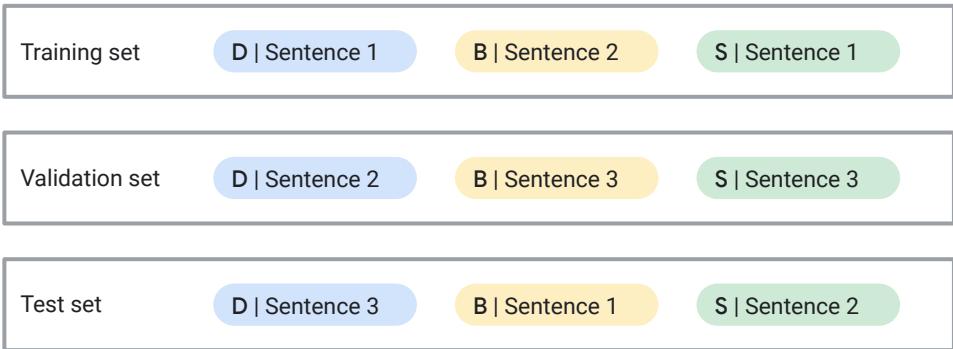
Feature [mind metaphor]



Labels [political affiliation]



and set those as their features and then used the political affiliations of the authors who wrote them as labels.

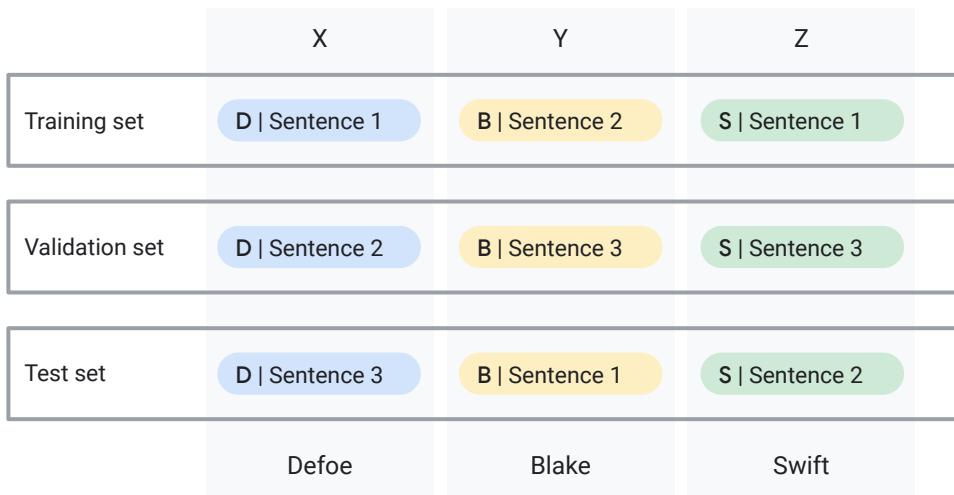


Then, they randomly assigned sentences to each of the training, validation, and test sets. And because they divided the data in this way, some sentences from each author were distributed to each of those three sets. And the resulting model was amazing! ... But suspiciously amazing.

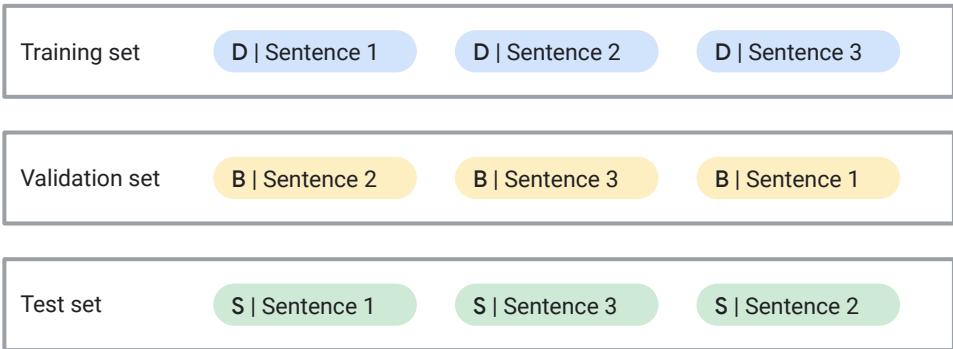
What might have gone wrong?



What might have gone wrong?



One way to think about it is that political affiliation is linked to that person. And if we wouldn't include '*person name*' in the feature set, we should not include it implicitly either.

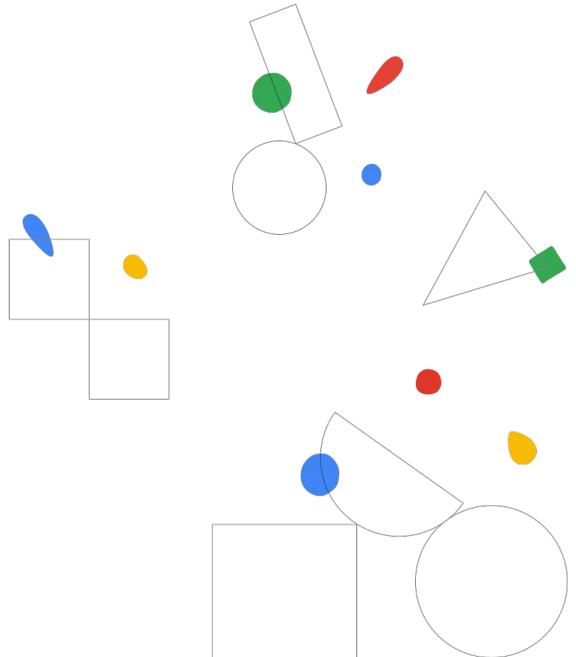


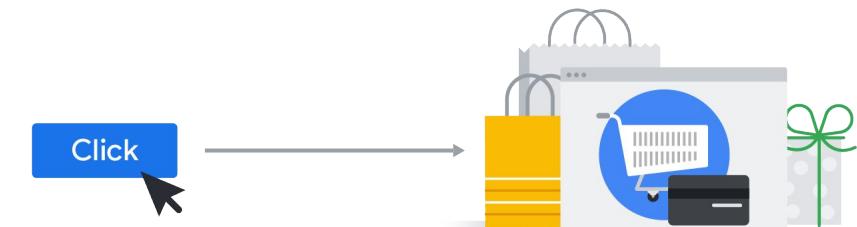
When the researchers changed the way they partitioned the data and instead partitioned it by author instead of by sentence, the model's accuracy dropped to something more reasonable.

System failure

Module 02

Designing adaptable ML systems





Here's another slightly different scenario.

You've trained a product recommendation model based on users' click and purchase behavior on your ecommerce site.

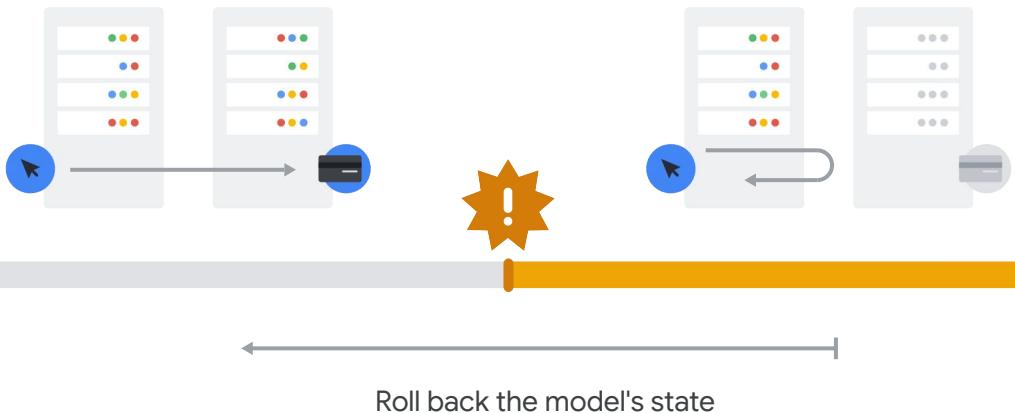


On Black Friday, your server responsible for transactions and payments goes down whilst the web server remains up and running, so the model thinks that no one who clicks is buying anything.

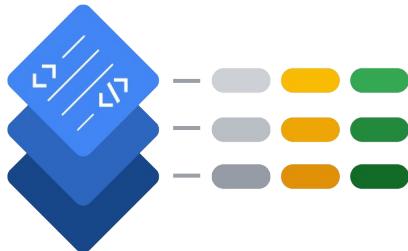
It's impossible to have models unlearn things that have already been learned



It's impossible to have models unlearn things that have already been learned but one thing you can do



is roll back the model's state to a time prior to the data pollution.



Automatically creates and saves models as well as their meta information.



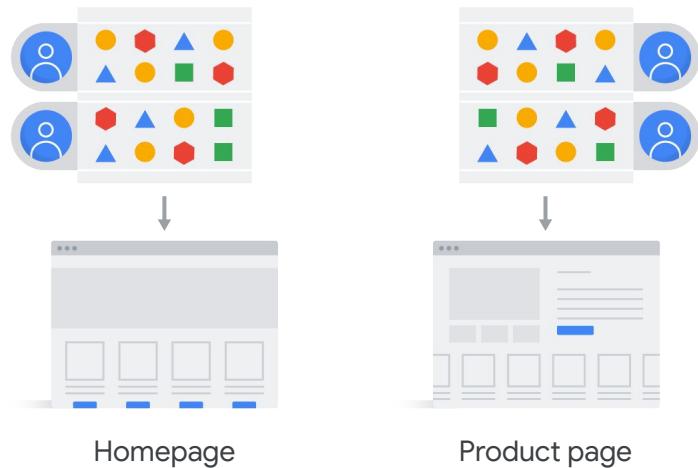
Of course, in order to do this, you will need infrastructure that automatically creates and saves models as well as their meta information.



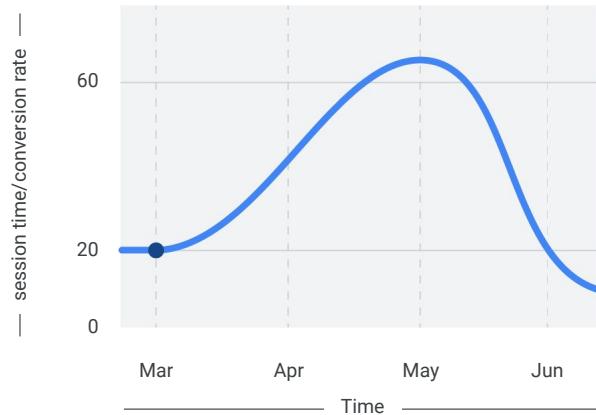
Here's another scenario.

You've trained a static product recommendation model which alone will determine which products users see when they are on the home page and when they are viewing individual products.





The model works by using purchasing behavior of other users.



After deploying it, user session time and conversion rate initially increase. But, in the months that follow the release of the model, conversion rate and user session time steadily decline to slightly below the levels they were at before the launch of the model.

What went wrong?



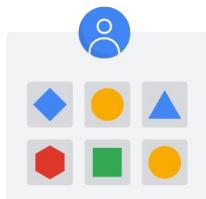
What went wrong?

Model is not updating to:



Well, your model is not updating to

Model is not updating to:

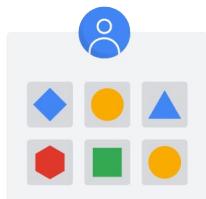


New users

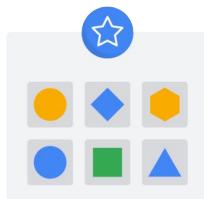


new users,

Model is not updating to:



New users

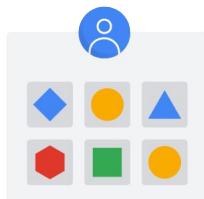


New products

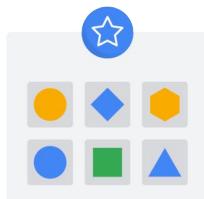


new products, and

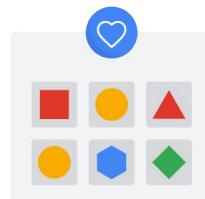
Model is not updating to:



New users



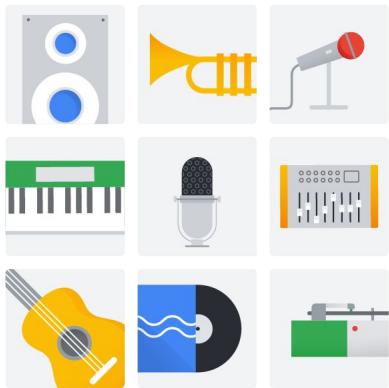
New products



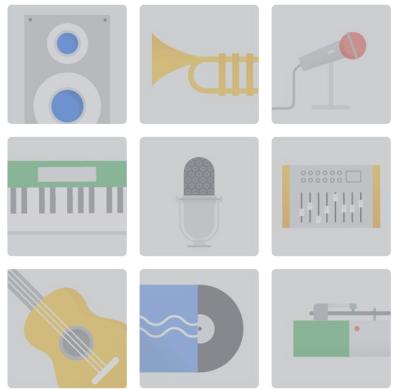
New patterns



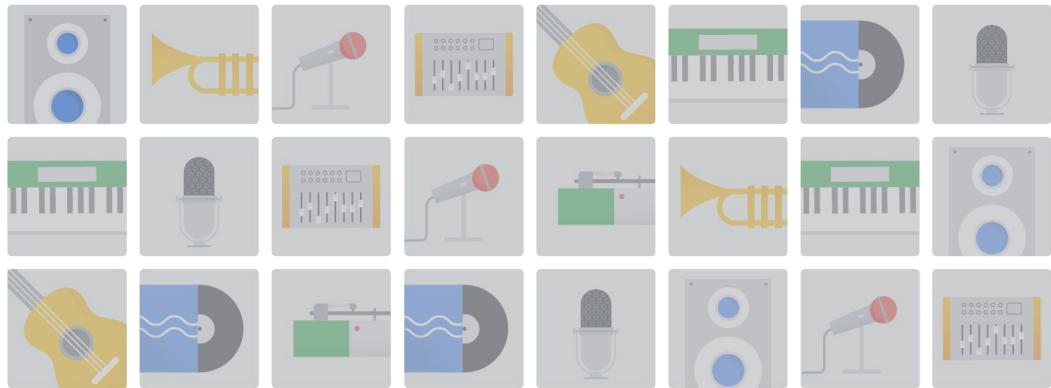
new patterns in user preference.



Because the model only knows about



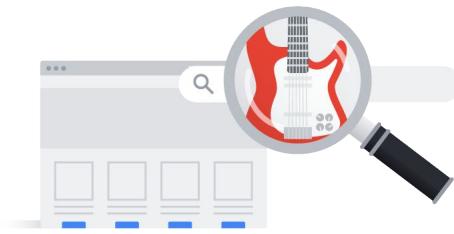
your older products,



it continues to recommend them long after they've fallen out of favor. Ultimately, users simply ignored



the recommendations altogether,



and made do with the site's search functionality.

“Cold start” problem

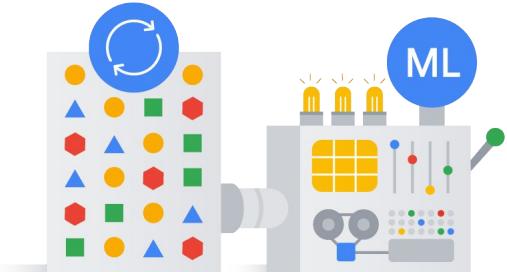


This “cold start” problem is common for this sort of recommendation model. We’ll talk more about recommendation systems later in the course.

Solution to a “cold start” problem

- Dynamically retrain the model

- Understand the limits of the model



The solution here is to dynamically retrain your model on newer data, and also to understand the limits of your model.



Here's one other scenario. You've deployed a statically-trained fraud detection model and its performance starts off good but quickly degrades.

What's gone wrong here?



What's gone wrong here?

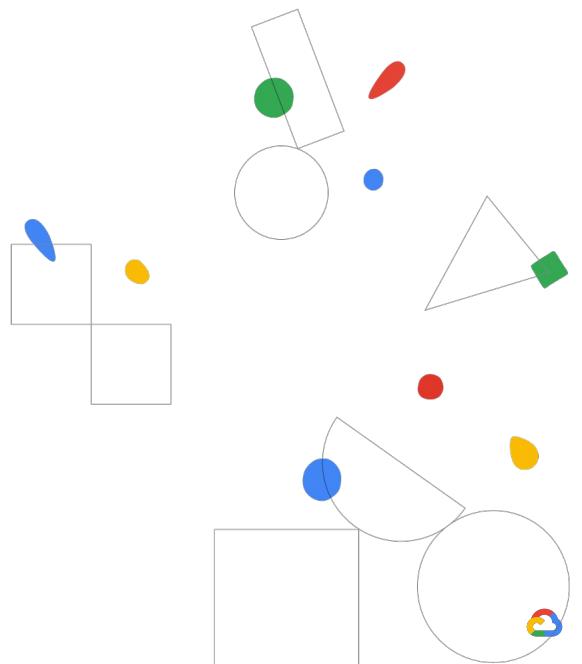


In adversarial environments, where one party is trying to beat another, it's particularly important to dynamically retrain the model, to keep up with the most recent strategies.

Concept drift

Module 02

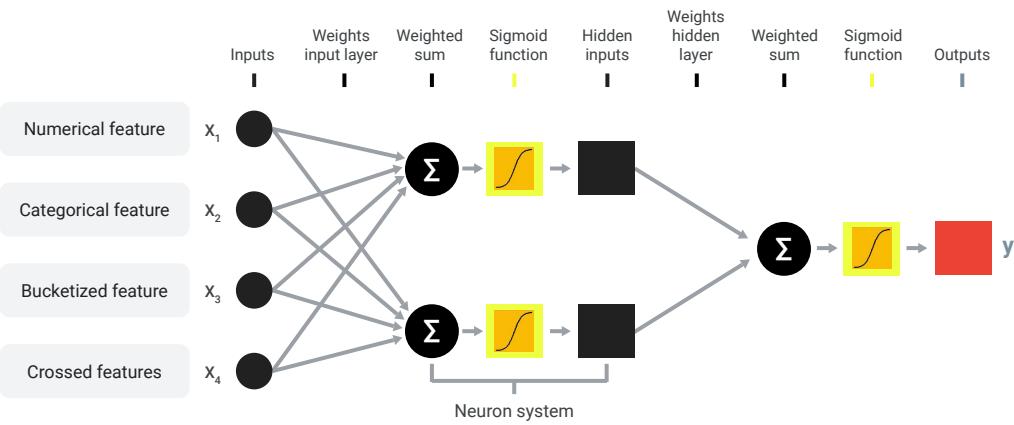
Designing adaptable ML systems



Why do machine learning models lose
their predictive power over time?



Why do machine learning models lose their predictive power over time?



You'll recall that machine learning models, such as neural networks, accept a feature vector and provide a prediction for our target variable. These models learn in a supervised fashion where a set of feature vectors with expected output is provided.

The traditional supervised learning assumes that the training and the application data come from the same distribution.

Machine learning algorithm assumptions

01 Instances are generated at random according to some probability distribution D .

02 Instances are independent and identically distributed.

03 That D is stationary with fixed distributions.

Linear regression

$$f(x) = \sum_{i=1}^n m_i x_i + b$$

Neural networks

$$f(x) = w_\theta + K \sum_{i=1}^n w_i x_i$$

Gradient descent

$$\theta_{ij} = \theta_j - \alpha \sum_{i=1}^n (h(x_i) - y) * x_i$$

Backpropagation

$$\Delta w_{ij}(n) = \eta \delta_j x_{ij} + \alpha \Delta w_{ij}(n-1)$$

Logistic regression

$$\text{Odds Ratio} = \log \left(\frac{P(a|c)}{1 - P(a|c)} \right)$$
$$\text{Prob}(y=1) = \frac{1}{1 + \exp(-\theta(\sum_{i=1}^n m_i x_i + b))}$$

Principal components analysis

$$x_j = x_i - \bar{x}$$
$$\text{Eigenvector} = \text{Eigenvalue} \cdot [x_1, \dots, x_n]$$
$$f(x) = (\text{Eigenvector})^T \cdot [x_{j1}, \dots, x_{jn}]$$



You'll also remember that traditional machine learning algorithms were developed with certain assumptions.

The first is that instances are generated at random according to some probability distribution D .

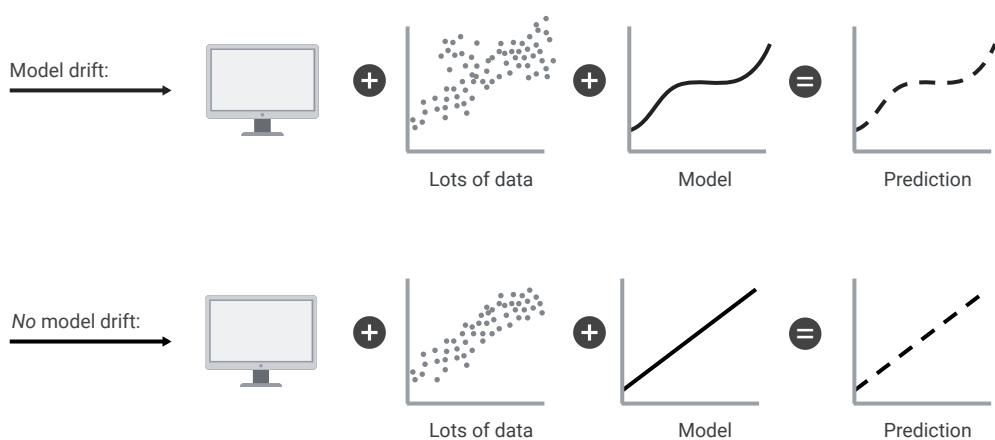
The second is that instances are independent and identically distributed.

And the third assumption is that D is stationary with fixed distributions.

Drift is the **change** in an entity
with respect to a baseline



Drift is the change in an entity with respect to a baseline.



In the case of production ML models, this is the change between the real-time production data and a baseline data set, likely the training set, that is representative of the task the model is intended to perform.

If your model were running in a static environment, using static or stationary data - for example data whose statistical properties do not change, then model drift wouldn't occur and your model would not lose any of its predictive power because the data you're predicting comes from the same distribution as the data used for training.

But production data can diverge or drift from the baseline data over time due to changes in the real world.

Types of drift in ML models

Data drift

A change in $P(X)$ is a shift in the model's input data distribution.

Concept drift

A change in $P(Y|X)$ is a shift in the actual relationship between the model inputs and the output.

Prediction drift

A change in $P(\hat{Y}|X)$ is a shift in the model's predictions.

Label drift

A change in $P(Y \text{ Ground Truth})$ is a shift in the model's output or label distribution.

There are several types of drift in ML models...

Data Drift or change in probability of X $P(X)$ is a shift in the model's input data distribution. For example, incomes of all applicants increase by 5%, but the economic fundamentals are the same.

Concept drift or change in probability of Y given X $P(Y|X)$ is a shift in the actual relationship between the model inputs and the output. An example of concept drift is when macroeconomic factors make lending riskier, and there is a higher standard to be eligible for a loan. In this case, an income level that was earlier considered creditworthy is no longer creditworthy.

Prediction drift or change in the predicted value of Y given X $P(\hat{Y} | X)$ is a shift in the model's predictions. For example, a larger proportion of credit-worthy applications when your product was launched in a more affluent area. Your model still holds, but your business may be unprepared for this scenario.

Label drift or change in the predicted value of Y as your target variable $P(Y \text{ Ground Truth})$ is a shift in the model's output or label distribution.

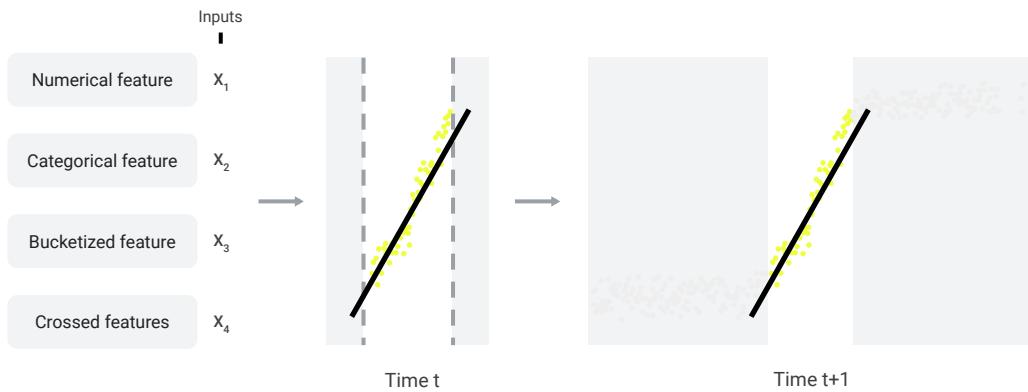


Describe changes in the data distribution of the inputs



Data drift, feature drift, population, or covariate shift are all names to describe changes in the data distribution of the inputs. When data shift occurs, or when you observe that the model performs worse on unknown data regions, that means that the input data has changed.

Data drift



The distribution of the variables is meaningfully different. As a result, the trained model is not relevant for this new data. It would still perform well on the data that is similar to the “old” one! The model is fine on the “old data”, but in practical terms, it became dramatically less useful since we are now dealing with a new feature space.

Indeed, the relationships between the model inputs and outputs have changed.

Concept drift

01. Stationary supervised learning

02. Learning under concept drift

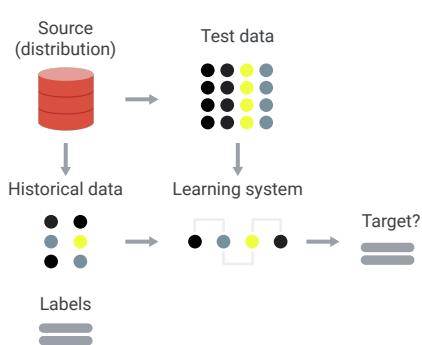


In contrast, concept drift occurs when there is a change in the relationship between the input feature and the label (or target).

Let's explore two examples of concept drift which highlight the change in the relationship between the input feature and the label.

Concept drift

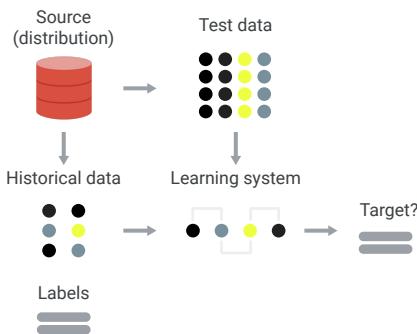
01. Stationary supervised learning



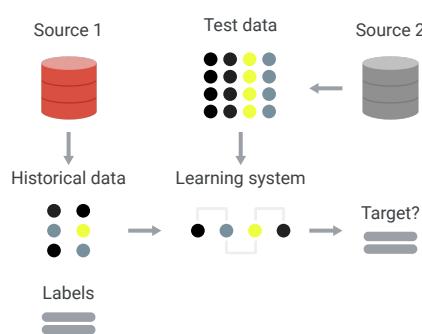
In this first example, stationary supervised learning, historical data is used to make predictions. You might recall that in supervised learning, a model is trained from historical data and that data is used to make predictions.

Concept drift

01. Stationary supervised learning



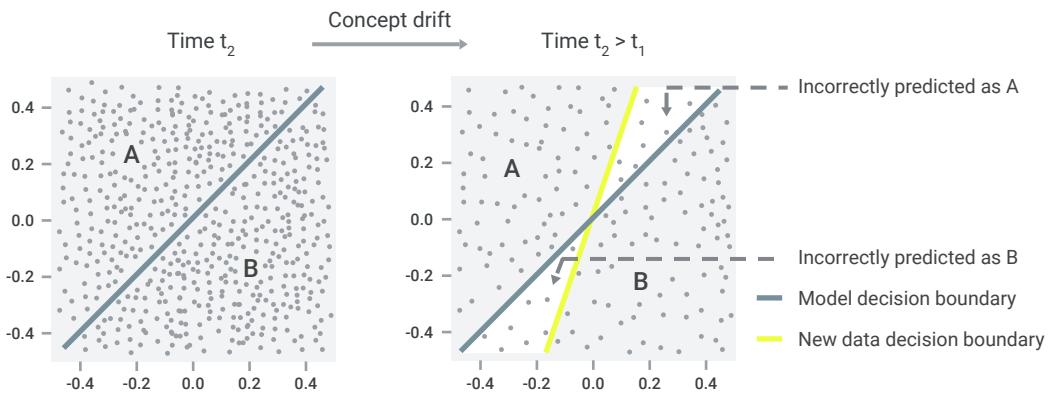
02. Learning under concept drift



This second example is supervised learning under concept drift, where a new, secondary data source is ingested to provide both historical data and new data to make predictions.

This new data could be in batch or real time. Whatever the form, it's important to know that the statistical properties of the target variable may change over time. As a result, an interpretation of the data changes with time, while the general distribution of the feature input may not.

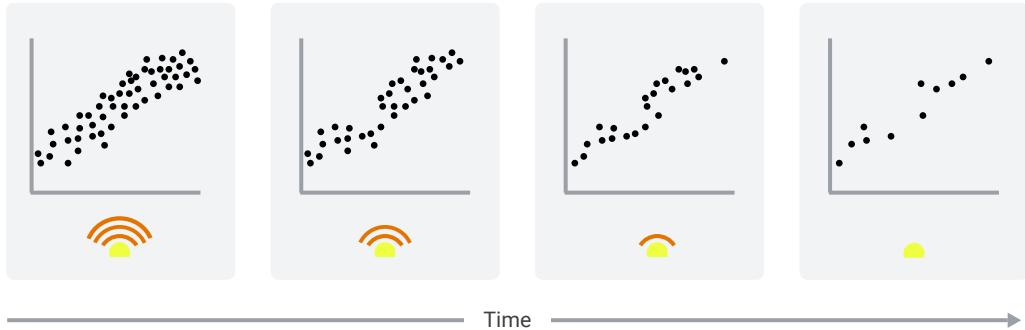
This illustrates concept drift, where the statistical properties of the class variable (the target we want to predict) changes over time.



In this supervised learning classification example, when the distribution of the label changes, it could mean that the relationship between features and labels is changing as well.

At the very least, it's likely that our model's predictions, which will typically match the distribution of the labels on the data on which it was trained, will be significantly less accurate.

Streaming data



Let's take a look at one other example. This one deals with streaming data.

Data flows continuously over time in dynamic environments - particularly for streaming data, such as e-commerce, user modeling, spam emails, fraud detection, and intrusion,

Changes in underlying data occur due to changing personal interests, changes in population, or adversary activities, or they can be attributed to a complex nature of the environment.

In this example, a sensor's measurement can drift due to a fault with the sensor or aging, changes in operation conditions or control command, and machine degradation as a result of wearing.

In these cases, the distribution of the feature inputs and the labels or targets may change - which will impact model performance and lead to model drift. There is no guarantee that future data will follow similar distributions of past data in a stream setting.

Concept drift

A probabilistic definition

$$\text{Concept} = P_t(X, y)$$

$$\text{Concept drift} = P_t(X, y) \neq P_{t+1}(X, y)$$



Concept drift occurs between times t and $t+1$ when the distributions change.

J. Lu, A. Liu, F. Dong, F. Gu, J. Gama and G. Zhang, "Learning under Concept Drift: A Review," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, pp. 2346-2363, 1 Dec. 2019, doi: 10.1109/TKDE.2018.2876857



Accordingly, concept drift at time t can be defined as the change of joint probability of X and y at time t . The joint probability $P_t(X, y)$ can be decomposed into two parts as the probability of X times the probability of y given X $P_t(X, y) = P_t(X) \times P_t(y|X)$.

Concept drift

A probabilistic definition

$$\text{Concept} = P_t(X, y)$$

An observation, which is a feature vector with its corresponding label.

$$\text{Concept drift} = P_t(X, y) \neq P_{t+1}(X, y)$$



Concept drift occurs between times t and $t+1$ when the distributions change.

J. Lu, A. Liu, F. Dong, F. Gu, J. Gama and G. Zhang, "Learning under Concept Drift: A Review," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, pp. 2346-2363, 1 Dec. 2019, doi: 10.1109/TKDE.2018.2876857



Let's be a little more careful here with the definition of concept drift. Let's use X to denote a feature vector and y to denote its corresponding label. Of course, when doing supervised learning, our goal is to understand the relationship between X and y .

Concept drift

A probabilistic definition

Concept = $P_t(X, y)$

A concept, which is the (joint probability) distribution of an observation.

Concept drift = $P_t(X, y) \neq P_{t+1}(X, y)$



Concept drift occurs between times t and $t+1$ when the distributions change.

J. Lu, A. Liu, F. Dong, F. Gu, J. Gama and G. Zhang, "Learning under Concept Drift: A Review," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, pp. 2346-2363, 1 Dec. 2019, doi: 10.1109/TKDE.2018.2876857



We will define a concept as a description of the distribution of our observations. More precisely, you can think of this as a joint probability distribution of our observations.

However, this concept could depend on time! Otherwise concept drift would be a non-issue, right?

Concept drift

A probabilistic definition

$$\text{Concept} = P_t(X, y)$$

$$\text{Concept drift} = P_t(X, y) \neq P_{t+1}(X, y)$$

$P(X,y)$ written for a certain time is $P_t(X,y)$.



Concept drift occurs between times t and $t+1$ when the distributions change.

J. Lu, A. Liu, F. Dong, F. Gu, J. Gama and G. Zhang, "Learning under Concept Drift: A Review," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, pp. 2346-2363, 1 Dec. 2019, doi: 10.1109/TKDE.2018.2876857



We'll use the notation probability of X and y at time t $P_t(X,y)$ when we want to consider the probability of X and y $P(X,y)$ at a specific time.

Concept drift

occurs when the distribution of our observations shifts over time, or that the joint probability distribution changes



Now it's easy to give a more rigorous description of concept drift. Simply put, concept drift occurs when the distribution of our observations shifts over time, or that the joint probability distribution we mentioned before changes.

Concept drift

A probabilistic definition

$$\text{Concept} = P_t(X, y)$$

$$\text{Concept drift} = P_t(X, y) \neq P_{t+1}(X, y) \quad P_t(X, y) = P_t(X) * P_t(y | X)$$



Concept drift occurs between times t and $t+1$ when the distributions change.

J. Lu, A. Liu, F. Dong, F. Gu, J. Gama and G. Zhang, "Learning under Concept Drift: A Review," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, pp. 2346-2363, 1 Dec. 2019, doi: 10.1109/TKDE.2018.2876857



We can break down this distribution into two parts using properties of joint distributions.

First we have the distribution of the feature space, probability of X $P(X)$, and then what we can think of as a description of our decision boundary, the probability of Y given X $P(y | X)$ (y given X).

Concept drift

A probabilistic definition

$$\text{Concept} = P_t(X, y)$$

$$\text{Decision boundary drift: } P_t(y | X) \neq P_{t+1}(y | X)$$

$$\text{Concept drift} = P_t(X, y) \neq P_{t+1}(X, y)$$

$$P_t(X, y) = P_t(X) * P_t(y | X)$$



Concept drift occurs between times t and $t+1$ when the distributions change.

J. Lu, A. Liu, F. Dong, F. Gu, J. Gama and G. Zhang, "Learning under Concept Drift: A Review," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, pp. 2346-2363, 1 Dec. 2019, doi: 10.1109/TKDE.2018.2876857



If the drift is occurring for the decision boundary, then we call this decision boundary drift.

Concept drift

A probabilistic definition

Feature space drift: $P_t(X) \neq P_{t+1}(X)$

Concept = $P_t(X, y)$

Decision boundary drift: $P_t(y | X) \neq P_{t+1}(y | X)$

Concept drift = $P_t(X, y) \neq P_{t+1}(X, y)$

$P_t(X, y) = P_t(X) * P_t(y | X)$



Concept drift occurs between times t and $t+1$ when the distributions change.

J. Lu, A. Liu, F. Dong, F. Gu, J. Gama and G. Zhang, "Learning under Concept Drift: A Review," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, pp. 2346-2363, 1 Dec. 2019, doi: 10.1109/TKDE.2018.2876857



Likewise, if the drift is occurring for the feature space, we call this feature space drift.

Concept drift

A probabilistic definition

$$\text{Concept} = P_t(X, y)$$

$$\text{Concept drift} = P_t(X, y) \neq P_{t+1}(X, y)$$

Both can occur at the same time

$$\text{Feature space drift: } P_t(X) \neq P_{t+1}(X)$$

$$\text{Decision boundary drift: } P_t(y | X) \neq P_{t+1}(y | X)$$

$$P_t(X,y) = P_t(X) * P_t(y | X)$$



Concept drift occurs between times t and $t+1$ when the distributions change.

J. Lu, A. Liu, F. Dong, F. Gu, J. Gama and G. Zhang, "Learning under Concept Drift: A Review," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, pp. 2346-2363, 1 Dec. 2019, doi: 10.1109/TKDE.2018.2876857



Of course, both can be happening at the same time, and this can make it complicated to understand where the changes are happening!

Concept drift can occur due to shifts in
feature space and/or the **decision boundary**

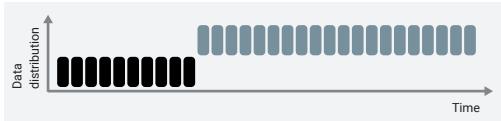
We need to be aware of these during production!



OK, we've veered off in a little bit more of a technical direction here, so what's the point we're trying to make?

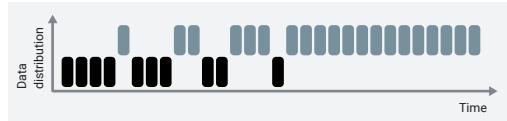
Concept drift can occur due to shifts in the feature space and / or the decision boundary, so we need to be aware of these during production.

If the data is changing, or if the relationship between the features and the label is changing, this is going to cause issues with our model.



Sudden drift

A new concept occurs within a short time.



Gradual drift

A new concept gradually replaces an old one over a period of time.



Incremental drift

An old concept incrementally changes to a new concept over a period of time.



Recurring concepts

An old concept may reoccur after some time.



There are different types of concept drift. Let's wrap up this video by taking a quick look at four of them:

In sudden drift a new concept occurs within a short time.

In gradual drift a new concept gradually replaces an old one over a period of time.

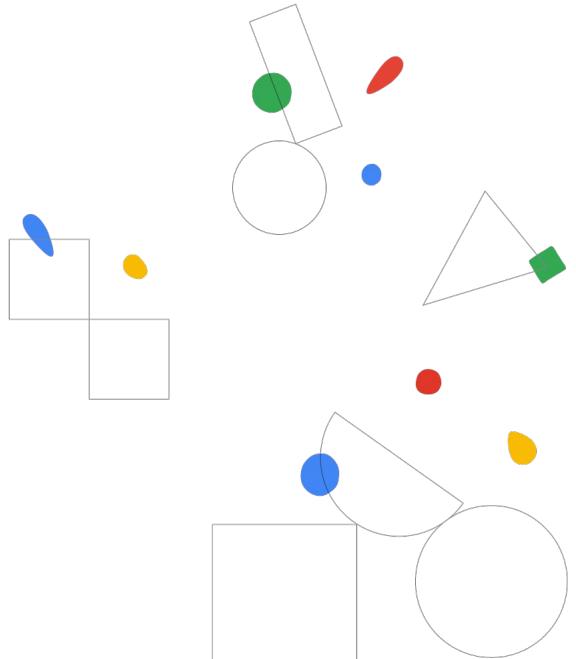
In incremental drift an old concept incrementally changes to a new concept over a period of time.

And in recurring concepts an old concept may reoccur after some time.

Actions to mitigate concept drift

Module 02

Designing adaptable ML systems

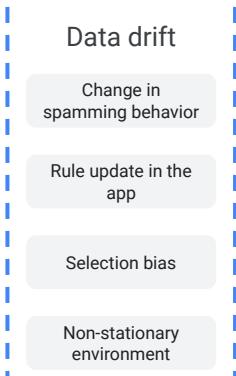


Both data drift and concept drift
lead to **model drift**



As previously mentioned, both data drift and concept drift lead to Model Drift.

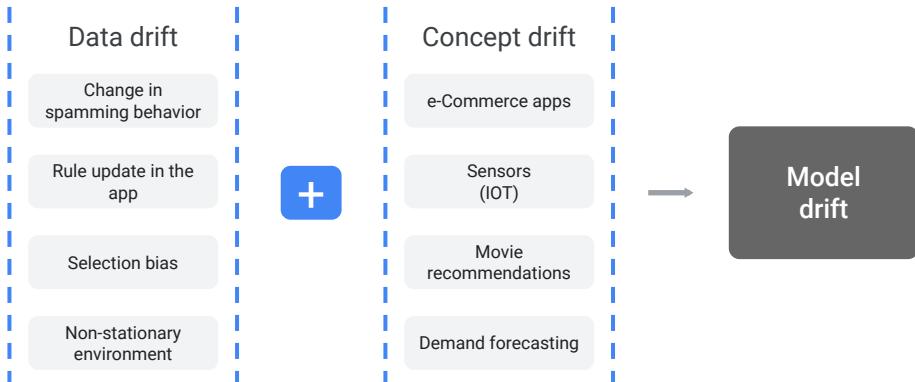
Examples of data drift and concept drift



Examples of data drift can include concepts of:

the change in the spamming behavior to try to fool the model, a rule update in the app—in other words, a change in the limit of user messages per minute, selection bias, and non-stationary environment—training data for a given season that has no power to generalize to another season.

Examples of data drift and concept drift



eCommerce apps are good examples of potential concept drift due to their reliance on personalization, for example, the fact that people's preferences ultimately do change over time.

Sensors may also be subject to concept drift due to the nature of the data they collect and how it may change over time.

Movie recommendations - again - similar to eCommerce apps - rely on user preferences - and they may change.

Demand forecasting heavily relies on time, and as we have seen, time is a major contributor to potential concept drift.

What if you diagnose data drift and concept drift?

Data drift

If you diagnose data drift, enough of the data needs to be labeled to introduce new classes and the model retrained.

Concept drift

If you diagnose concept drift, the old data needs to be relabeled and the model retrained.



So, what if you diagnose data drift?

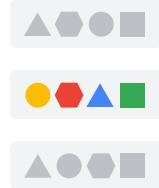
If you diagnose data drift, enough of the data needs to be labeled to introduce new classes and the model retrained.

What if you diagnose concept drift?

If you diagnose concept drift, the old data needs to be relabeled and the model retrained.



Design your systems to detect changes



Use an ensemble approach to train

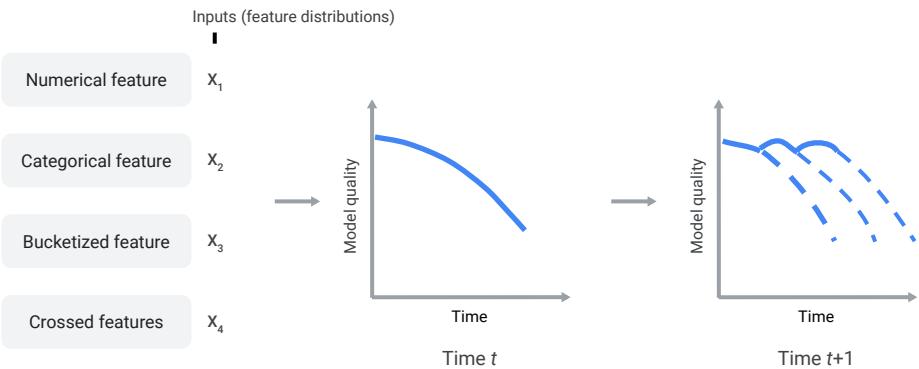


Also, for concept drift, you can design your systems to detect changes. Periodically updating your static model with more recent historical data, for example, is a common way to mitigate concept drift. You can either discard the static model completely or you can use the existing state as the starting point for a better model to update your model by using a sample of the most recent historical data.

You can also use an ensemble approach to train your new model in order to correct the predictions from prior models. The prior knowledge learnt from the old concept is used to improve the learning of the new concept. Ensembles which learnt the old concept with high diversity are trained by using low diversity on the new concept.

Concept drift

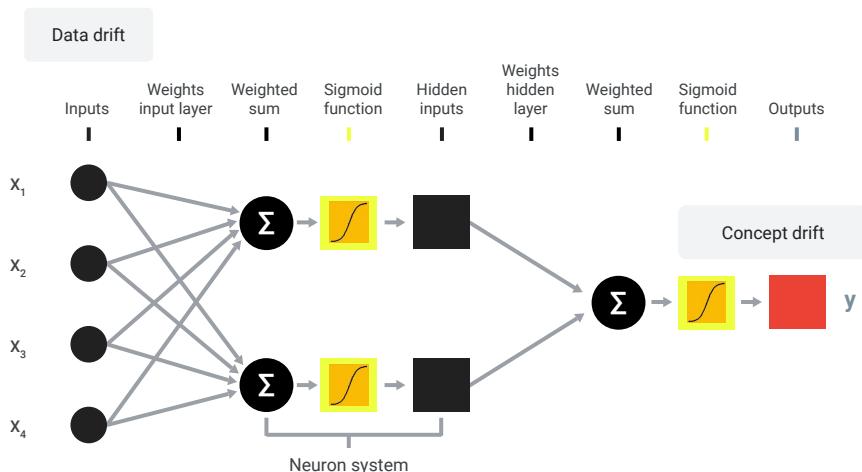
Change in relationships between the model inputs and the model output



Remember, concept drift is the change in relationships between the model inputs and the model output.

After your diagnosis and mitigation efforts, retraining or refreshing the model over time will help to maintain model quality.

Model drift = {Data drift, Concept drift}



As the world changes, your data may change. The change can be gradual, sudden, and seasonal. These changes will impact model performance.

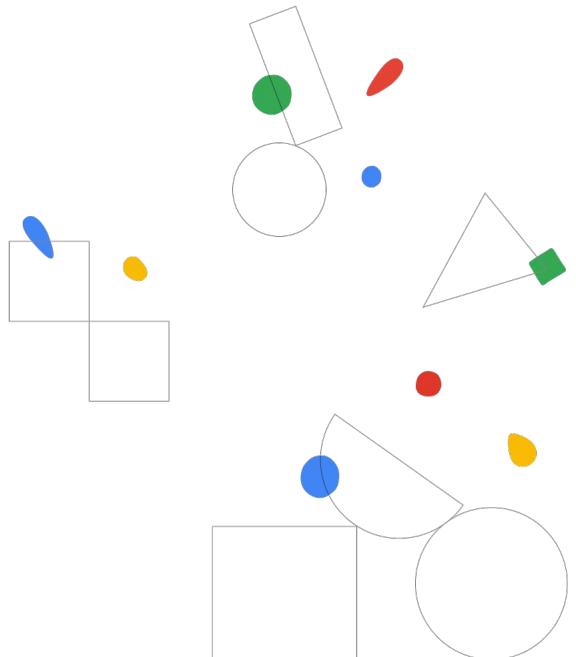
Thus, machine learning models can be expected to degrade or decay. Sometimes, the performance drop is due to low data quality, broken data pipelines, or technical bugs.



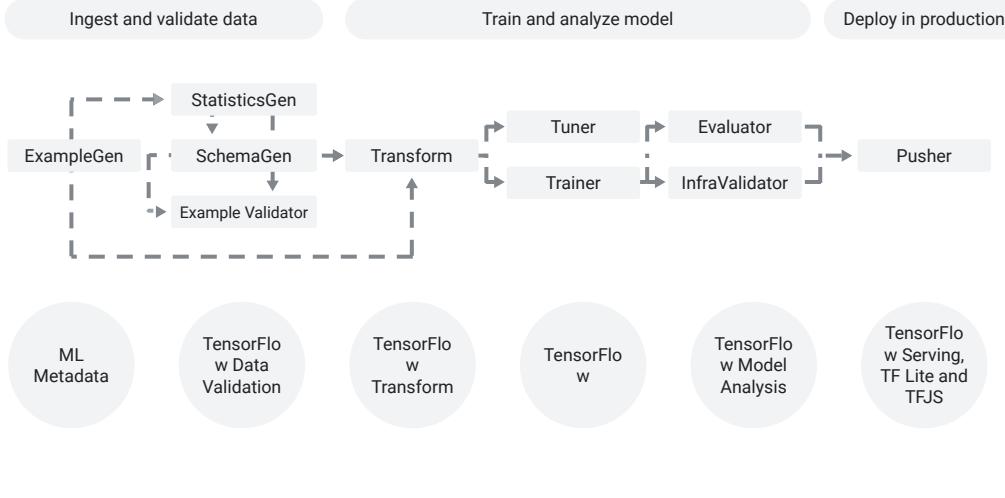
TensorFlow Data Validation

Module 02

Designing adaptable ML systems

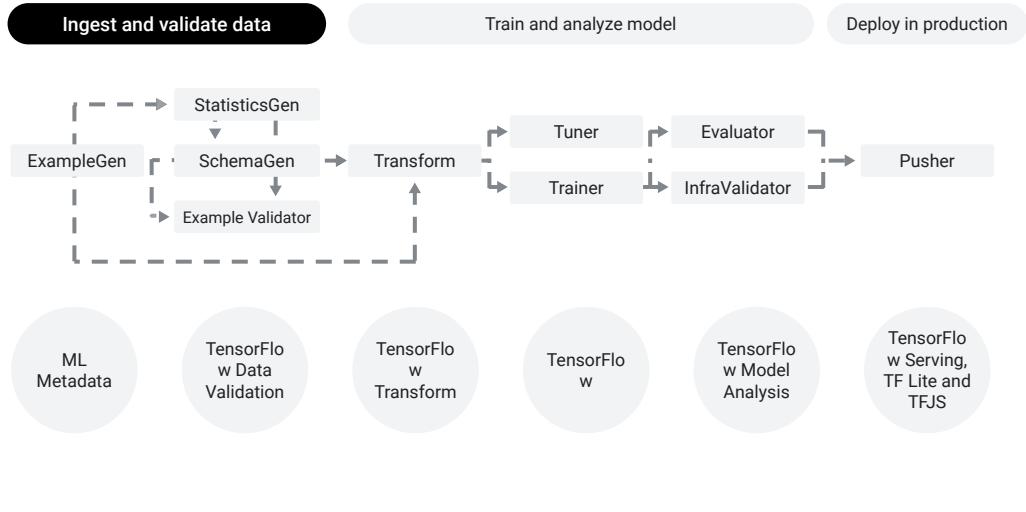


TensorFlow Data Validation (TFDV)



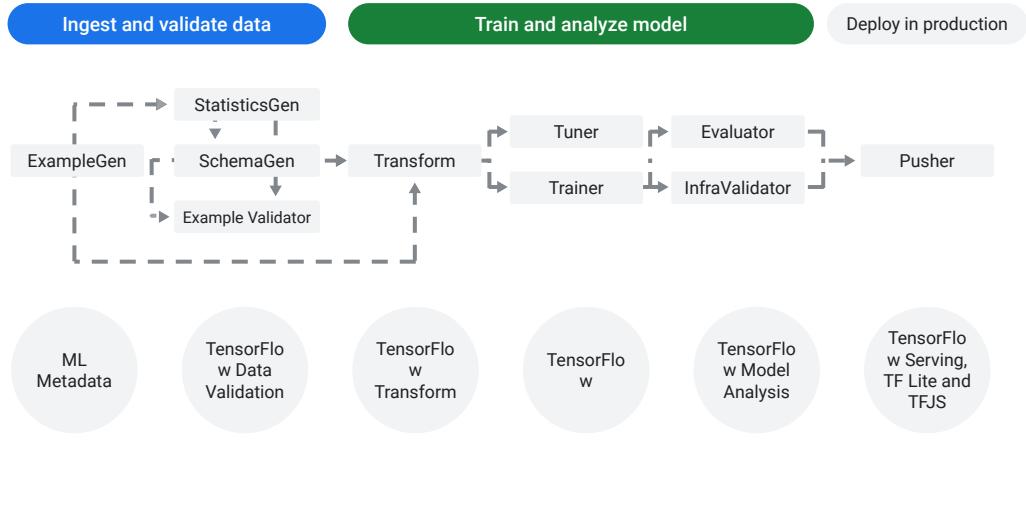
There are three phases in a pipeline.

TensorFlow Data Validation (TFDV)



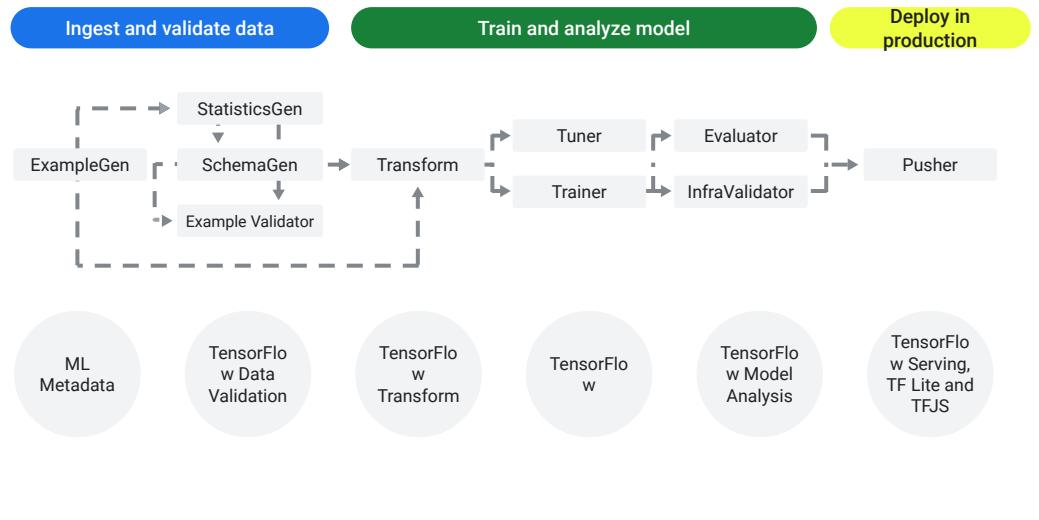
Data is ingested and validated,

TensorFlow Data Validation (TFDV)



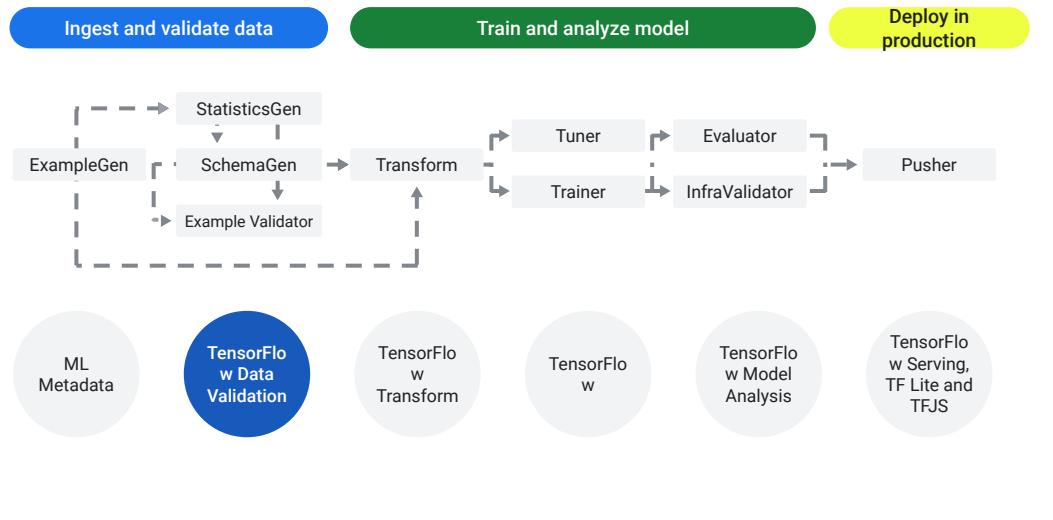
a model is trained and analyzed,

TensorFlow Data Validation (TFDV)



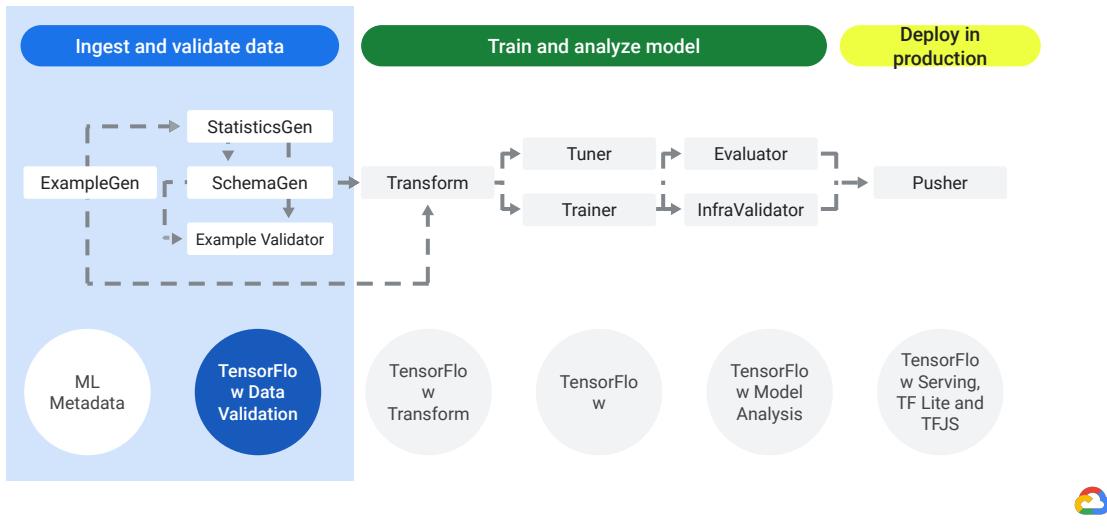
and the model is then deployed in production.

TensorFlow Data Validation (TFDV)



In this video, we'll provide an overview of TensorFlow Data Validation,

TensorFlow Data Validation (TFDV)



which is part of the ingest and validate data phase.

To learn more about the train and analyze the model phase, or how to deploy a model in production, please check out our ML Ops Course.



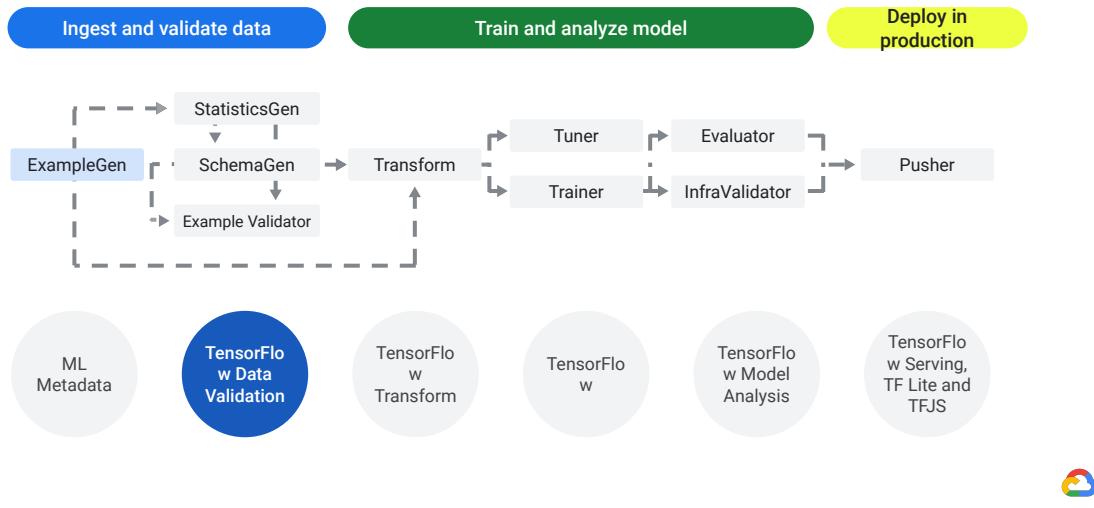
- ✓ Validation of continuously arriving data
- ✓ Training/serving skew detection



TensorFlow Data Validation is a library for analyzing and validating machine learning data.

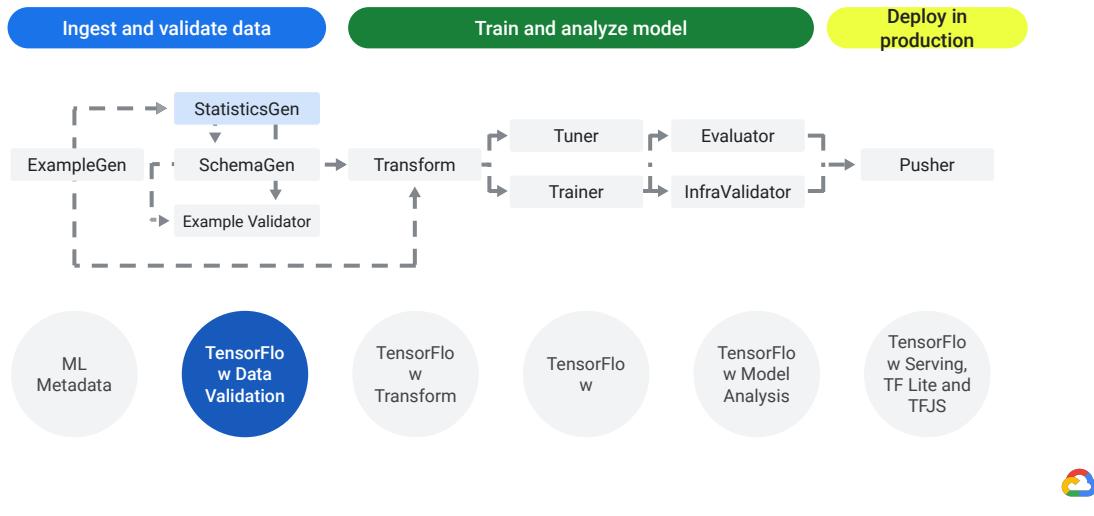
Two common use-cases of TensorFlow Data Validation within a TensorFlow Extended pipelines are validation of continuously arriving data and training/serving skew detection.

TensorFlow Data Validation (TFDV)



The pipeline begins with the **ExampleGen** component. This component takes raw data as input and generates TensorFlow examples, it can take many input formats (e.g. CSV, TF Record). It also does split the examples for you into Train/Eval. It then passes the result...

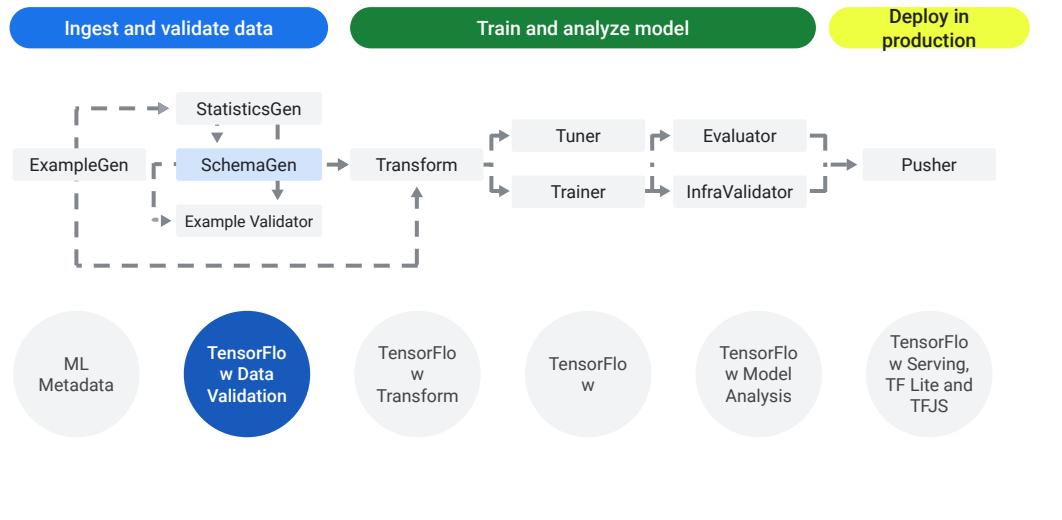
TensorFlow Data Validation (TFDV)



...to the StatisticsGen component. This brings us to the three main components of TensorFlow Data Validation.

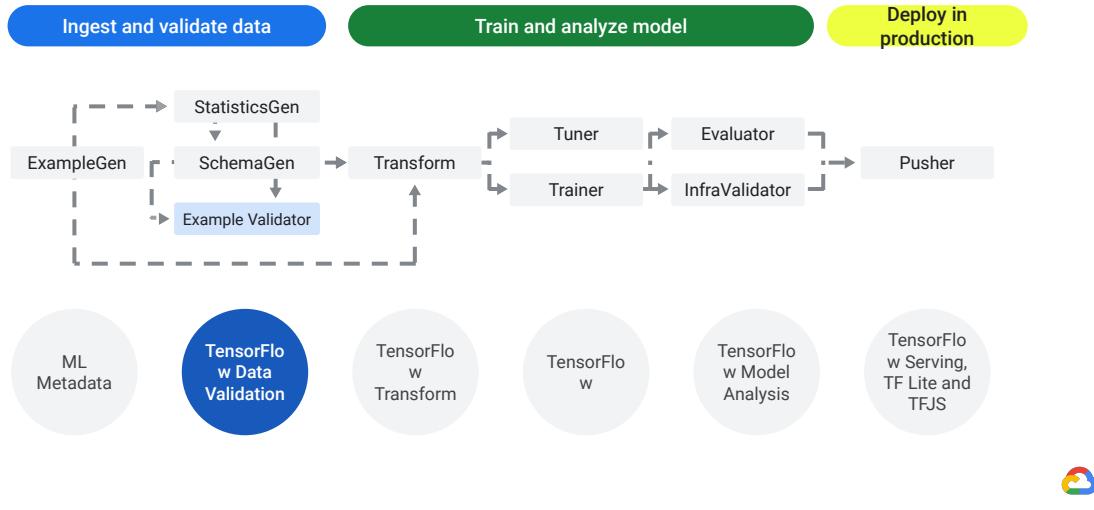
The Statistics Generation component - which generates statistics for feature analysis,

TensorFlow Data Validation (TFDV)



the Schema Generation component - which gives you a description of your data,

TensorFlow Data Validation (TFDV)



and the Example Validator component - which allows you to check for anomalies

We'll explore those three components in more depth in the next video, but first let's look at our use cases for TensorFlow Data Validation so that we can understand how these components work.

Reasons to analyze and transform your data

To find problems in your data.

Common problems include:

- Missing data
- Labels treated as features
- Features with values outside an expected range
- Data anomalies



There are many reasons and use cases where you may need to analyze and transform your data.

For example, when you're missing data, such as features with empty values, or when you have labels treated as features, so that your model gets to peek at the right answer during training.

You may also have features with values outside the range you expect or other data anomalies.

Reasons to analyze and transform your data

To find problems in your data.

Common problems include:

- Missing data
- Labels treated as features
- Features with values outside an expected range
- Data anomalies

To engineer more effective feature sets.

Identify:

- Informative features
- Redundant features
- Features that vary so widely in scale that they may slow learning
- Features with little or no unique predictive information



To engineer more effective feature sets, you should identify:

Especially informative features,

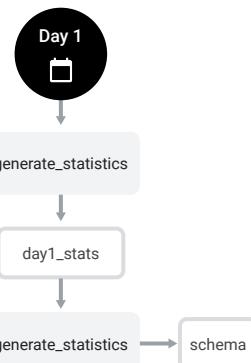
redundant features,

features that vary so widely in scale that they may slow learning,

and features with little or no unique predictive information.

TFDV

Check and analyze data



----- Time -----

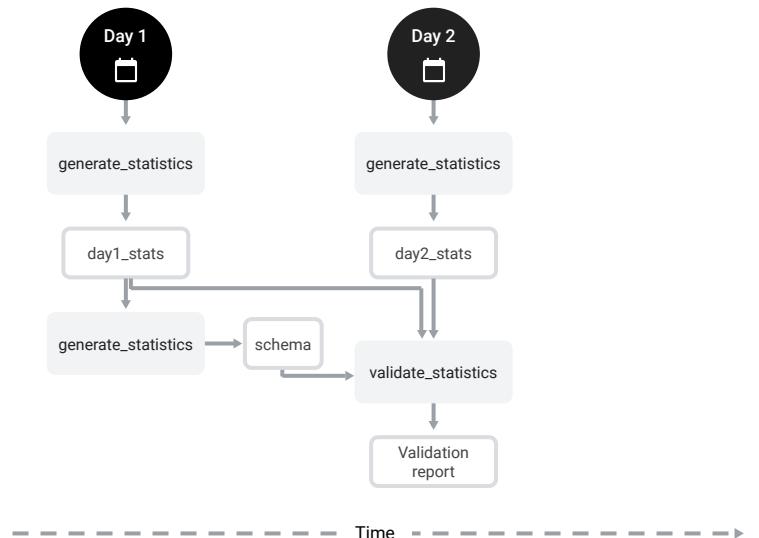


One use case for TensorFlow Data Validation is to validate continuously arriving data.

Let's say on day one you generate statistics based on data from day one.

TFDV

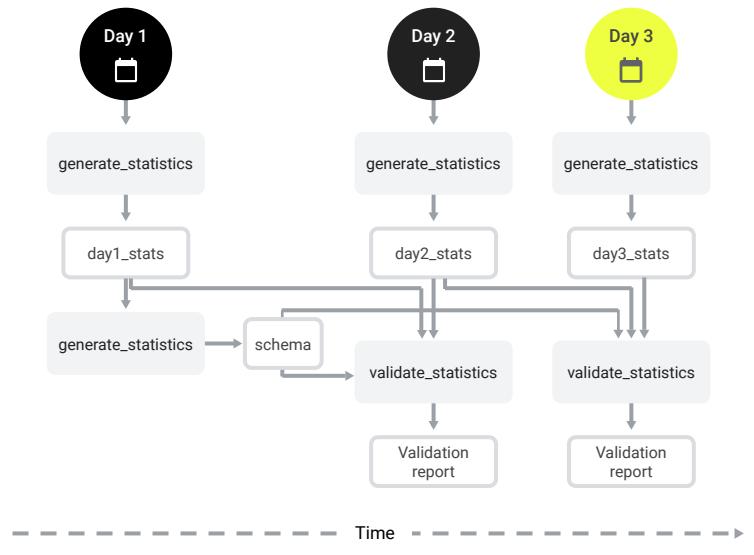
Check and analyze data



Then, you generate statistics based on day two data. From there, you can validate Day 2 statistics against day one statistics and generate a validation report.

TFDV

Check and analyze data



You can do the same for day three, validating day three statistics against statistics from both day two and day one.

TFDV: Check and analyze data

Skew

Skew happens when you generate training data differently than you generate the data you use to request predictions



TensorFlow Data Validation can also be used to detect distribution skew between training and serving data. Training-serving skew occurs when training data is generated differently from how the data used to request predictions is generated.

But what causes distribution skew? Possible causes might come from a change in how data is handled in training vs in production, or even a faulty sampling mechanism that only chooses a subsample of the serving data to train on.

TFDV: Check and analyze data

Skew

— Skew happens when you generate training data differently than you generate the data you use to request predictions

Training data

— Value = Average over **10 days**

Prediction data

— Value = Average over **the last month**



For example, if you use an average value, and for training purposes you average over 10 days, but when you request prediction, you average over the last month.

TFDV: Check and analyze data

Skew

Skew happens when you generate training data differently than you generate the data you use to request predictions

Training data

Value = Average over **10 days**

Prediction data

Value = Average over **the last month**

The difference

Should be reviewed



In general, any difference between how you generate your training data and your serving data (the data you use to generate predictions) should be reviewed to prevent training-serving skew.

TFDV: Check and analyze data



Training-serving skew can also occur based on your data distribution in your training,

TFDV: Check and analyze data



validation,

TFDV: Check and analyze data



and testing data splits.

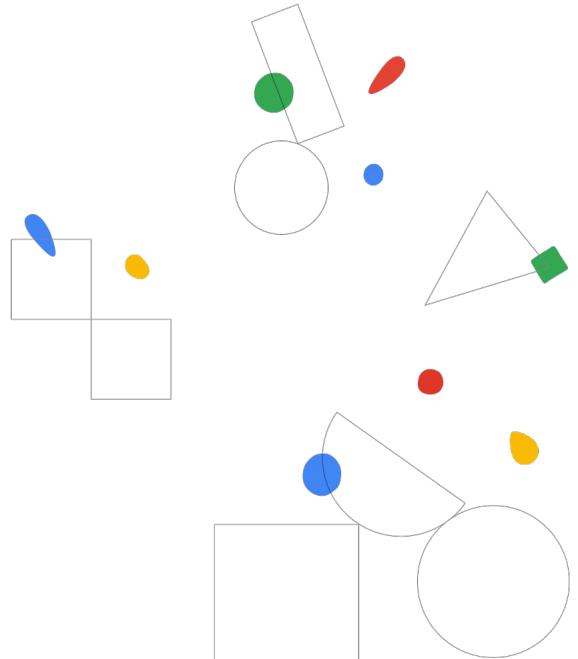
To summarize, distribution skew occurs when the distribution of feature values for training data is significantly different from serving data and one of the key causes for distribution skew is how data is handled or changed in training vs in production.



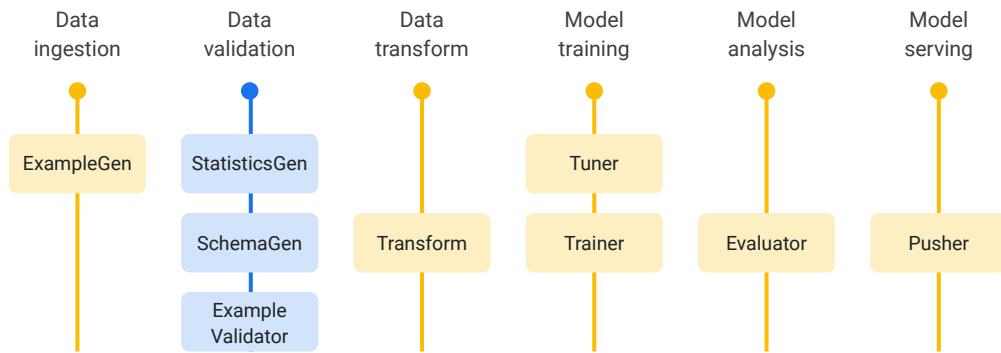
Components of TensorFlow Data Validation

Module 02

Designing adaptable ML systems

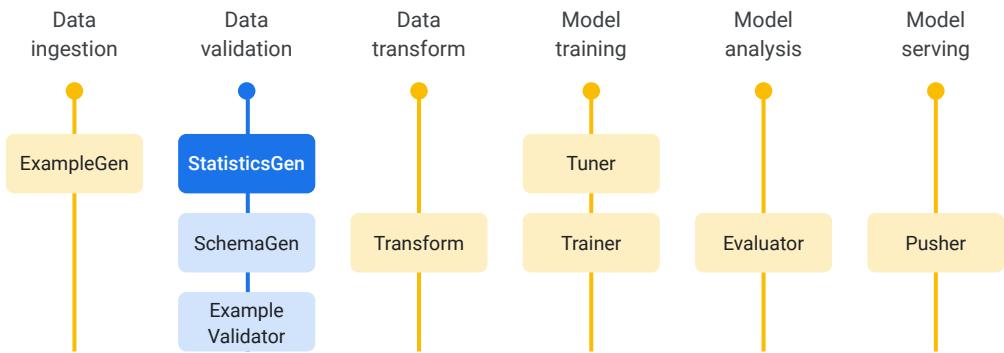


TensorFlow Data Validation components



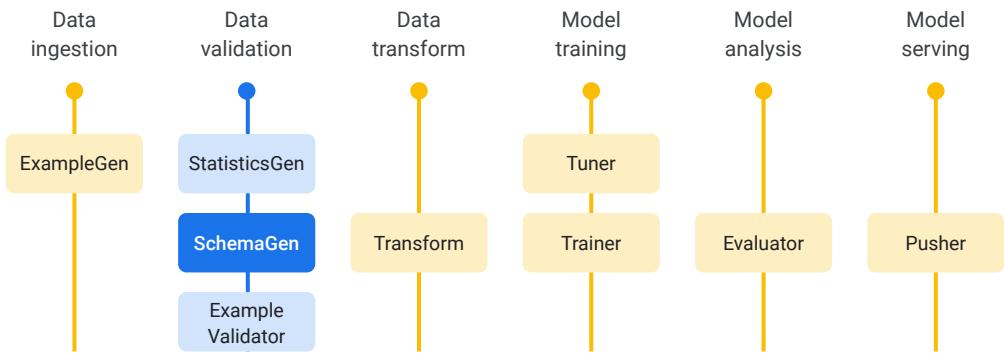
TensorFlow Data Validation is a library for analyzing and validating machine learning data, for which there are three components:

TensorFlow Data Validation components



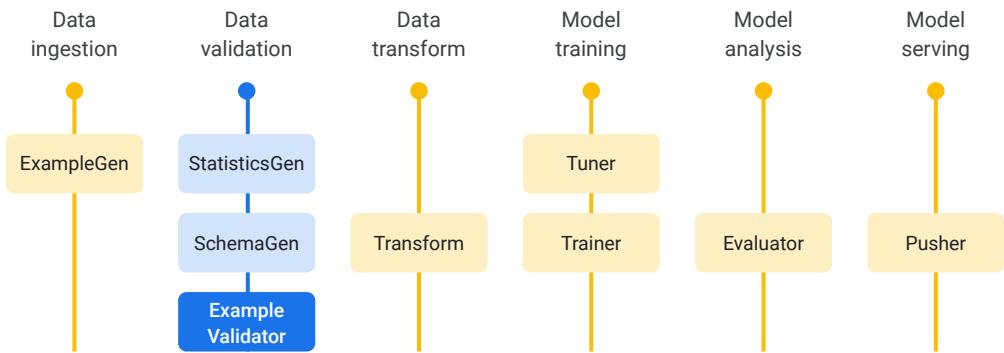
The Statistics Generation component,

TensorFlow Data Validation components



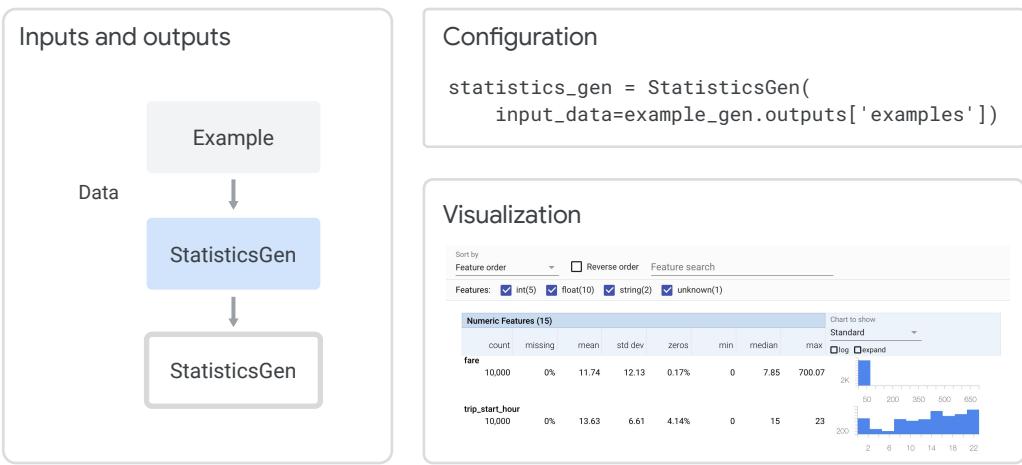
the Schema Generation component,

TensorFlow Data Validation components



and the Example Validator component.

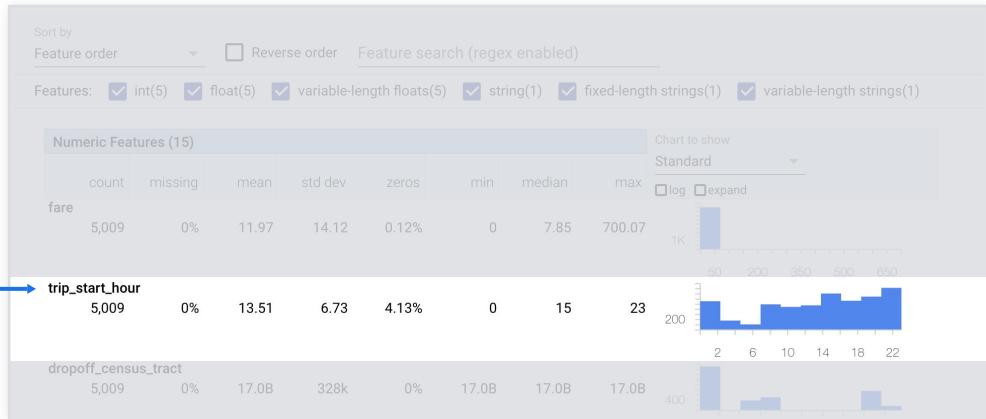
StatisticsGen



The StatisticsGen component generates features statistics and random samples over training data, which can be used for visualization and validation. It requires minimal configuration.

For example, StatisticsGen takes as input a dataset ingested using ExampleGen. After StatisticsGen finishes running, you can visualize the outputted statistics.

Analyzing data with TensorFlow Data Validation



In an example using taxi data, the StatisticsGen component has generated data statistics for the numeric features. For the `trip_start_hour` feature, it appears there is not that much data in the early morning hours.

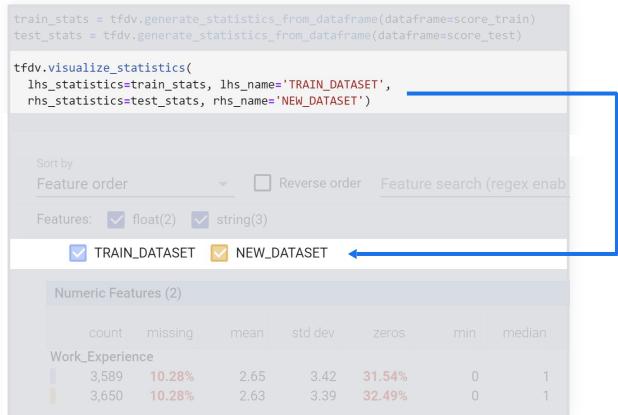
It appears that the `trip_start_hour` column, where the time window is between 2:00am to 6:00am, has data missing.

This helps determine the area we need to focus on to fix any data-related problems. We'll need to get more data, otherwise, the prediction for 4:00am data will be overgeneralized.

Generate and visualize data

How big is the difference between **TRAIN_DATASET** and **NEW_DATASET**?

Does this difference matter?



Let's look at another example, this time using a consumer spending score dataset.

Here we are generating statistics for both a training dataset, that may have arrived on day one, and a new dataset, that may have arrived on day two. These statistics are being generated from a Pandas DataFrame. You can also generate statistics from a CSV file or a TF.Record formatted file.

By comparing both datasets, you can analyze how big of a difference there is between the two, and then determine if that difference matters.

Numeric and categorical features

Numeric Features (2)				Categorical Features (3)							
	count	missing	mean	std dev		count	missing	unique	top	freq top	avg str len
Work_Experience					Graduated						
	3,589	10.28%	2.65	3.42		3,964	0.9%	2	Yes	2,433	2.61
	3,650	10.28%	2.63	3.39		4,026	1.03%	2	Yes	2,535	2.63
Family_Size					Profession						
	3,831	4.23%	2.84	1.51		3,944	1.4%	9	Artist	1,218	8.14
	3,902	4.08%	2.86	1.55		4,000	1.67%	9	Artist	1,298	8.09
					Spending_Score						
						4,000	0%	3	Low	2,448	4.11
						4,068	0%	3	Low	2,430	4.14

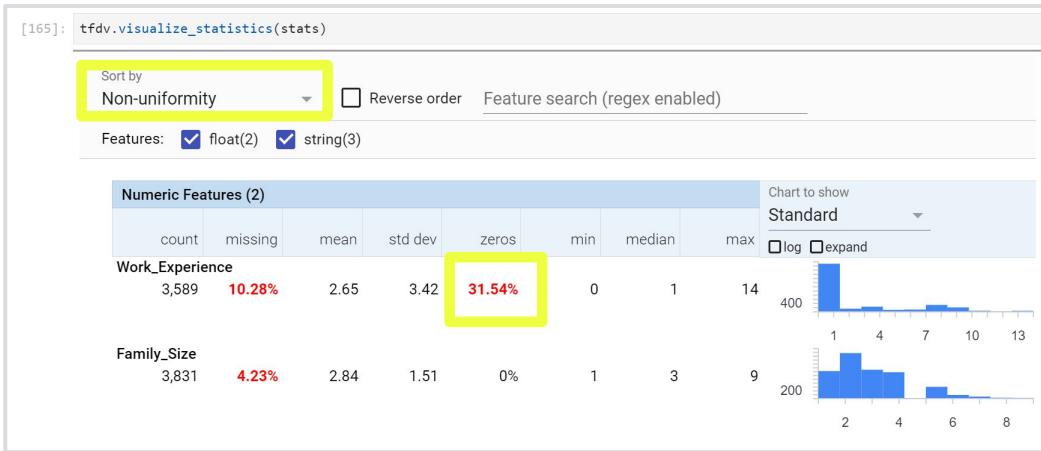


StatisticsGen generates both numeric and categorical features.

In this example, our dataset has two numerical features, `Work_Experience` and `Family_Size`.

Our dataset also has three categorical features: `Graduated`, `Profession`, and `Spending_Score`. Notice that for our categorical features, in addition to seeing the number of missing values, we also see the number of unique values.

Identify unbalanced data distributions



TensorFlow Data Validation can also help you identify unbalanced data distributions. For example, if you have a dataset you are using for a classification problem and you see that one feature has a lower percentage of values than the other, you can use TensorFlow Data Validation to detect this “unbalance”.

The most unbalanced features will be listed at the top of each feature-type list. For example, the following screenshot shows `Work_Experience` with zero has a data value - which means that 31.54% of the values in this feature are zeroes.

Data validation checks

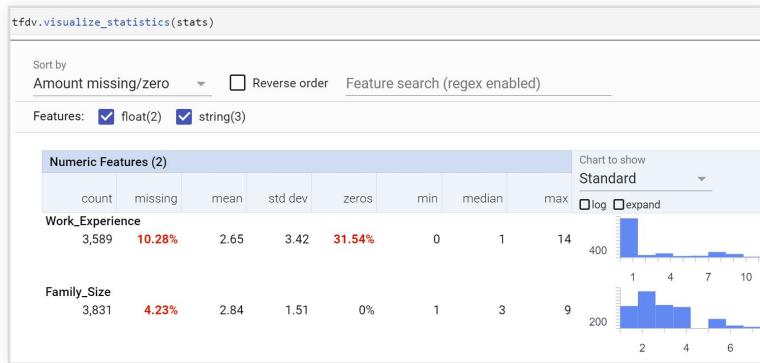
Feature min, max, mean, mode, median

Feature correlations

Class imbalance

Check to see missing values

Histograms of features (for both numerical and categorical)



There are a number of StatisticsGen data validation checks that you should be aware of. These include:

Feature min, max, mean, mode, and median

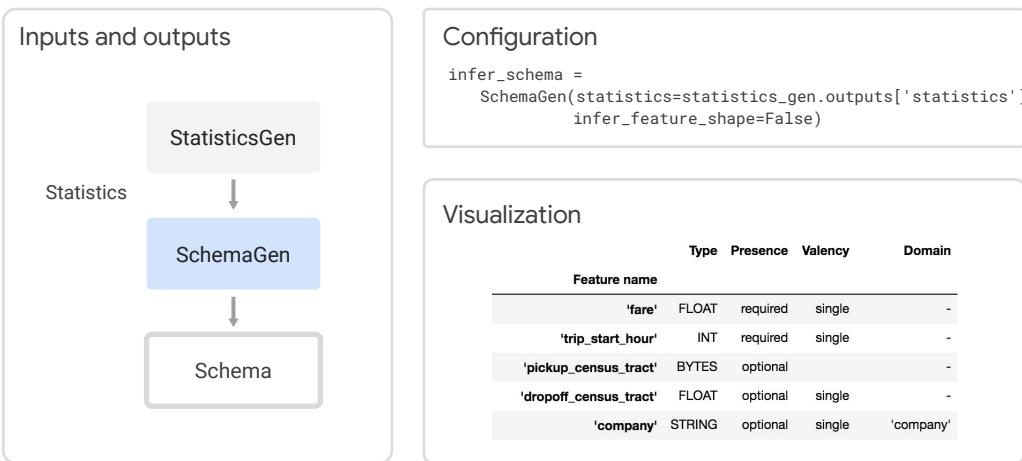
Feature correlations

Class imbalance

Check to see missing values

and histograms of features (for both numerical and categorical).

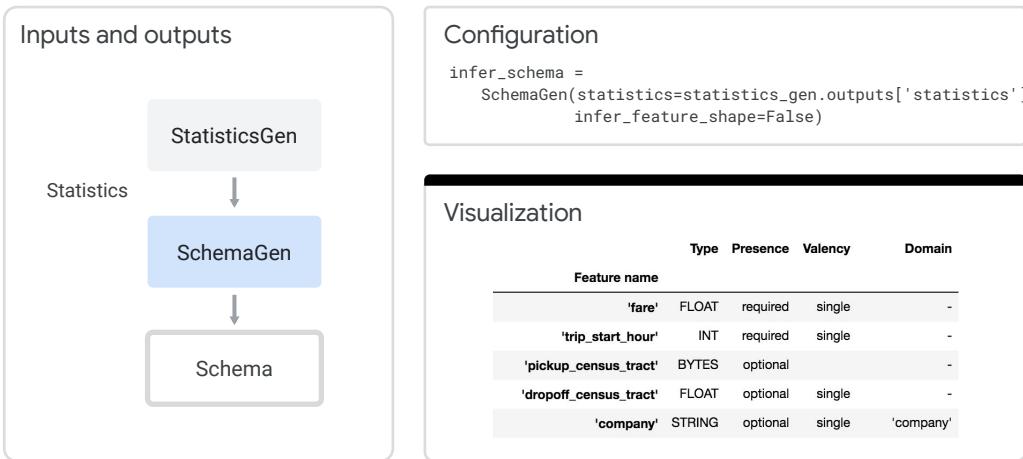
SchemaGen



The SchemaGen TFX pipeline component can specify data types for feature values, whether a feature has to be present in all examples, allowed value ranges, and other properties. A SchemaGen pipeline component will automatically generate a schema by inferring types, categories, and ranges from the training data.

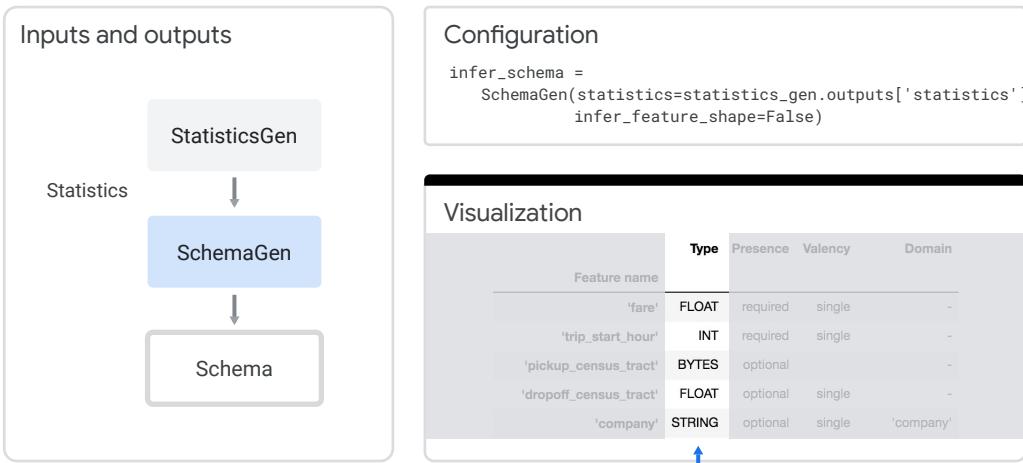
In essence, SchemaGen is looking at the data type of the input, is it an int, float, categorical, etc. If it is categorical then what are the valid values? It also comes with a visualization tool to review the inferred schema and fix any issues.

SchemaGen



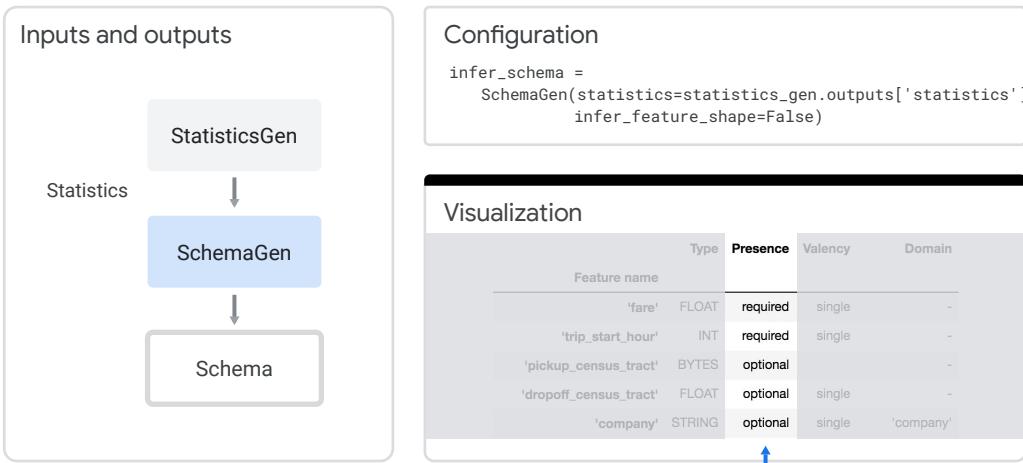
In this example visualization:

SchemaGen



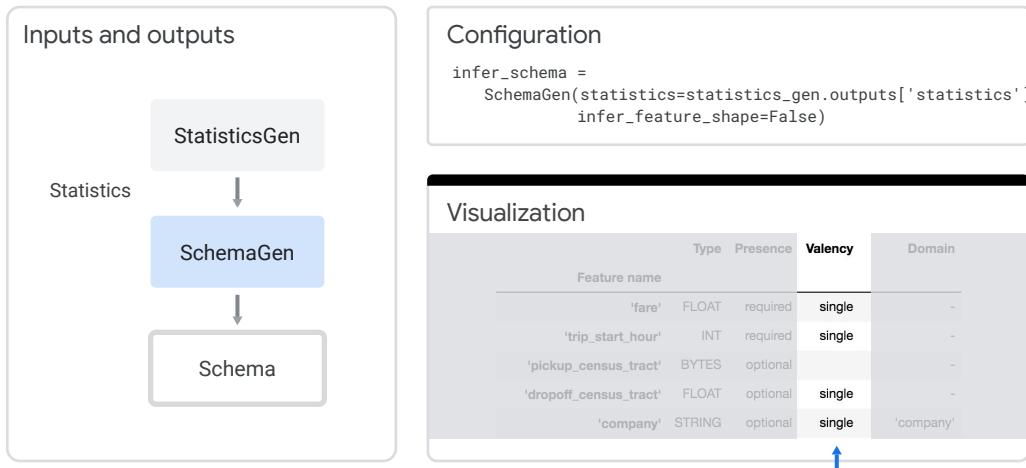
Type indicates the feature datatype.

SchemaGen



Presence indicates whether the feature must be present in 100% of examples (required) or not (optional).

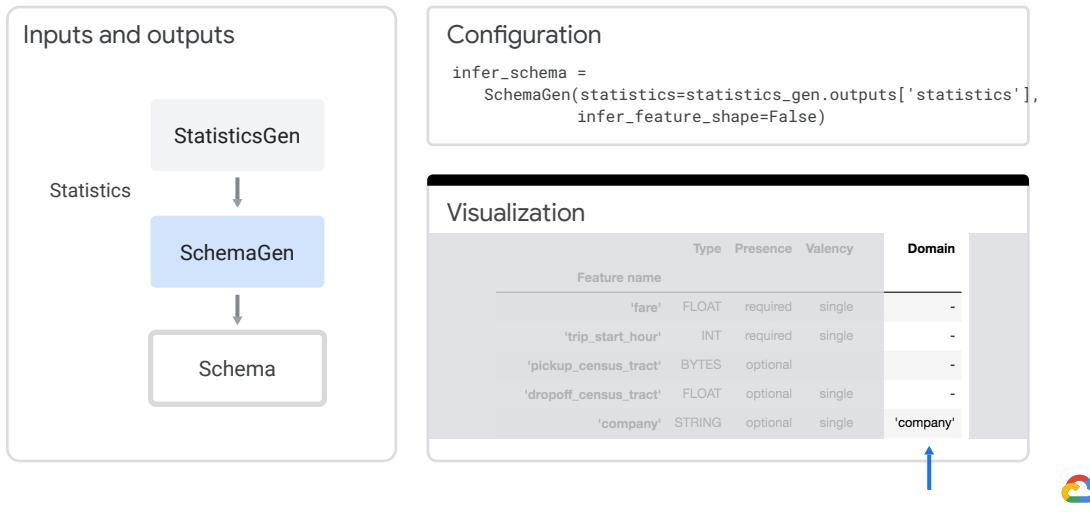
SchemaGen



Valency indicates the number of values required per training example.



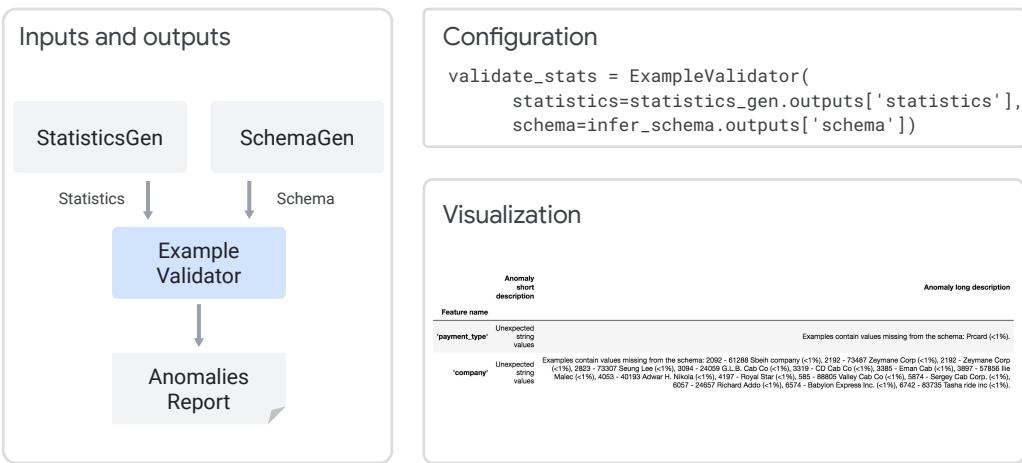
SchemaGen



Domain and Values indicates the feature domain and its values.

In the case of categorical features, single indicates that each training example must have exactly one category for the feature.

ExampleValidator

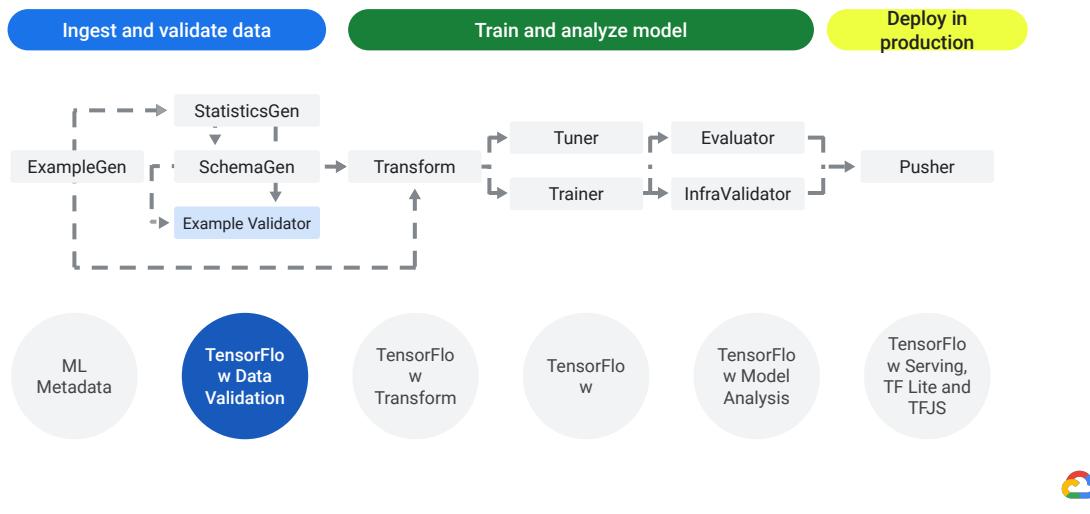


The ExampleValidator pipeline component identifies anomalies in training and serving data. It can detect different classes of anomalies in the data and emit validation results.

The ExampleValidator pipeline component identifies any anomalies in the example data by comparing data statistics computed by the StatisticsGen pipeline component against a schema.

It takes the inputs and looks for problems in the data, like missing values, and reports any anomalies.

TensorFlow Data Validation (TFDV)



As we've explored, TensorFlow Data Validation is a component of TensorFlow Extended and it helps you to analyze and validate your data. Data validation checks include identifying feature correlations, checking for missing values, and identifying class imbalances.

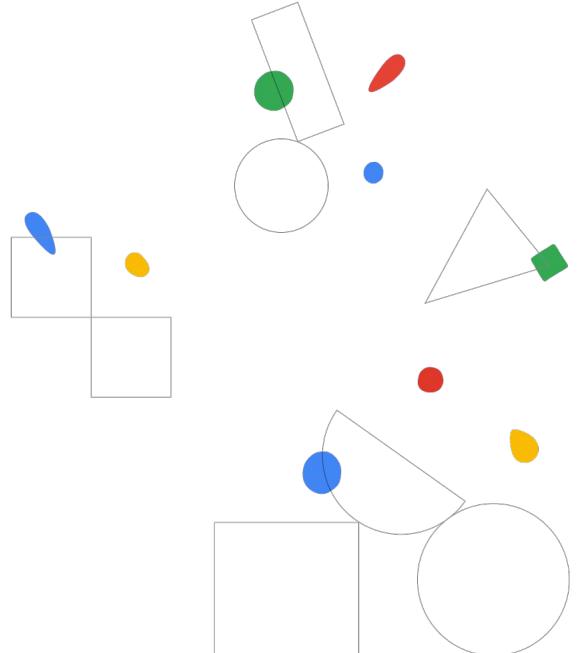


Lab

Introduction to TensorFlow Data Validation

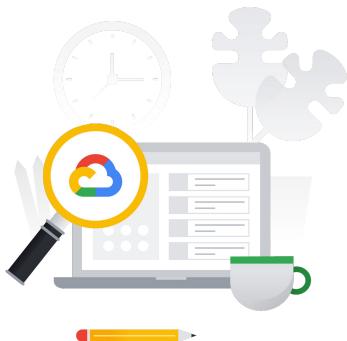
Module 02

Designing adaptable ML systems



This lab provides a hands-on introduction to TensorFlow Data Validation.

Lab objectives

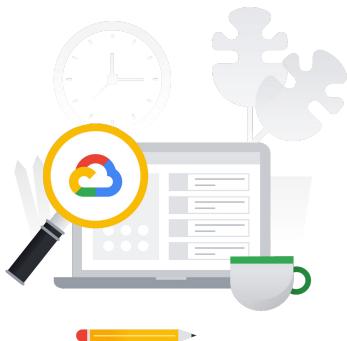


Review TFDV methods.



You'll begin by reviewing the different TensorFlow Data Validation methods,

Lab objectives

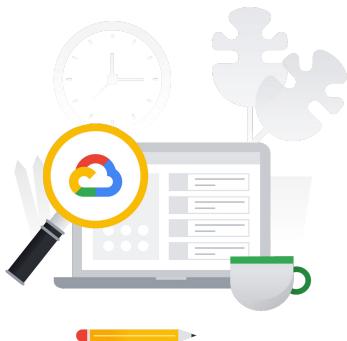


- Review HBV methods.
- Generate statistics.



then continue on to generate statistics,

Lab objectives

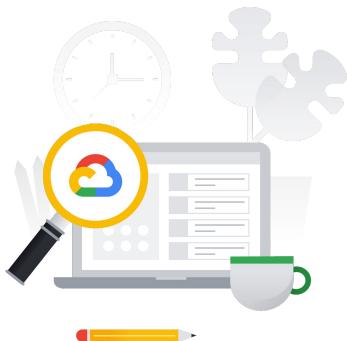


- Review TFDV methods.
- Generate statistics.
- Visualize statistics.



visualize statistics,

Lab objectives

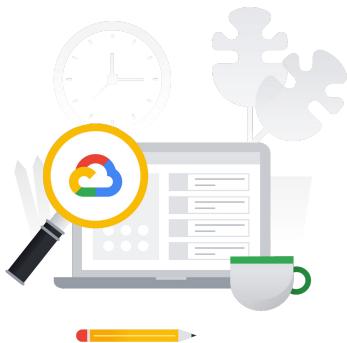


- Review TFDV methods.
- Generate statistics.
- Visualize statistics.
- Infer a schema.



infer a schema,

Lab objectives



- Review TFDV methods.
- Generate statistics.
- Visualize statistics.
- Infer a schema.
- Update a schema.



and, finally, update a schema.

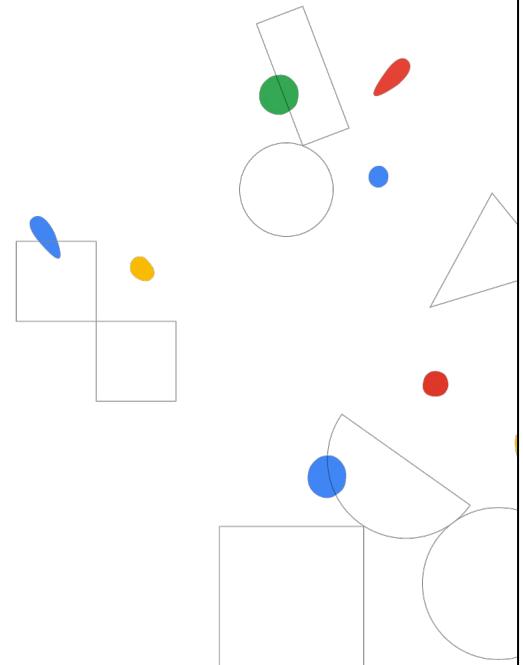


Lab

Advanced Visualizations with TensorFlow Data Validation

Module 02

Designing adaptable ML systems

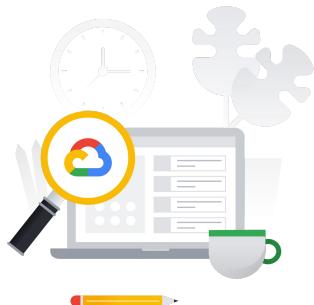


This lab demonstrates how TensorFlow Data Validation can be used to investigate and visualize a dataset.

Lab objectives



Install TensorFlow Data Validation.



You'll begin with steps to install TensorFlow Data Validation,

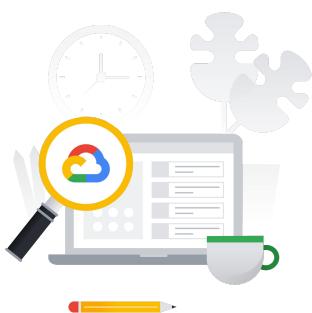
Lab objectives



Install TensorFlow Data Validation.

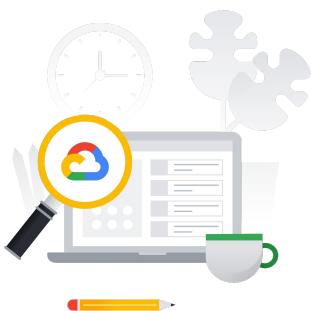


Compute and visualize statistics.



then continue on to compute and visualize statistics,

Lab objectives

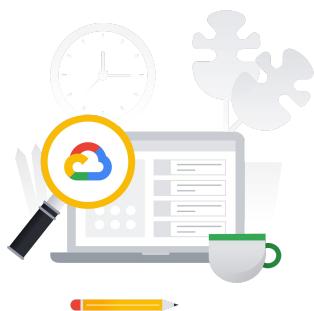


- Install TensorFlow Data Validation.
- Compute and visualize statistics.
- Infer a schema.



infer a schema,

Lab objectives

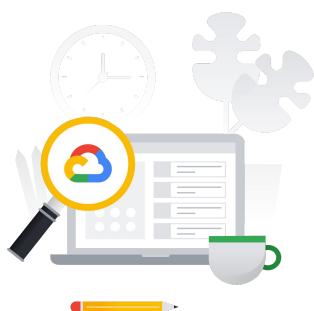


- Install TensorFlow Data Validation.
- Compute and visualize statistics.
- Infer a schema.
- Check evaluation data for errors.



check evaluation data for errors,

Lab objectives

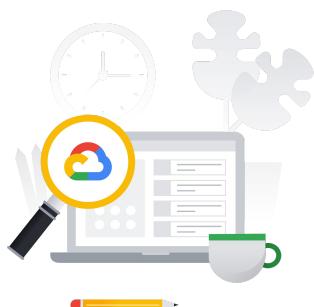


- Install TensorFlow Data Validation.
- Compute and visualize statistics.
- Infer a schema.
- Check evaluation data for errors.
- Check for and fix evaluation anomalies.



check for and fix evaluation anomalies,

Lab objectives

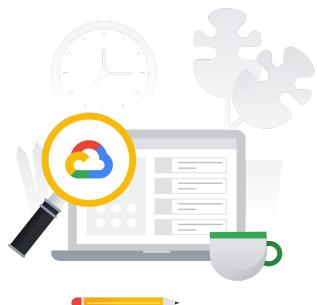


- Install TensorFlow Data Validation.
- Compute and visualize statistics.
- Infer a schema.
- Check evaluation data for errors.
- Check for and fix evaluation anomalies.
- Check for drift and skew.



check for drift and skew,

Lab objectives



- Install TensorFlow Data Validation.
- Compute and visualize statistics.
- Infer a schema.
- Check evaluation data for errors.
- Check for and fix evaluation anomalies.
- Check for drift and skew.
- Freeze a schema.

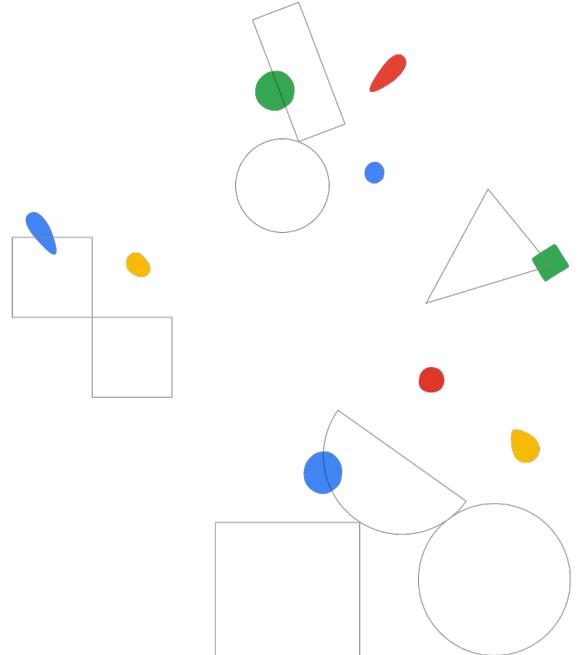


and finally, freeze a schema.

Mitigating Training-Serving Skew Through Design

Module 02

Designing adaptable ML systems



Training/serving skew



We've talked about training/serving skew a number of times in previous videos, but always at a high level.

Let's look at it now in a little more detail.

Training/serving skew

The differences in performance
that occur as a function of
differences in environment



Training/Serving skew refers to differences in performance that occur as a function of differences in environment.

Training/serving skew

The differences in performance
that occur as a function of
differences in environment



Specifically, training/serving skew refers to differences caused by one of three things:

Training/serving skew

The differences in performance
that occur as a function of
differences in environment

A discrepancy between how you
handle data in the training and
serving pipelines



A discrepancy between how you handle data in the training and serving pipelines.

Training/serving skew

The differences in performance
that occur as a function of
differences in environment

A discrepancy between how you
handle data in the training and
serving pipelines

A change in the data between
when you train and when you
serve



A change in the data between when you train and when you serve, or

Training/serving skew

The differences in performance that occur as a function of differences in environment

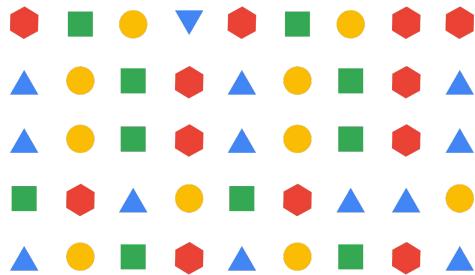
A discrepancy between how you handle data in the training and serving pipelines

A change in the data between when you train and when you serve

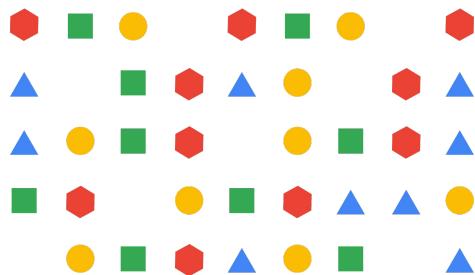
A feedback loop between your model and your algorithm



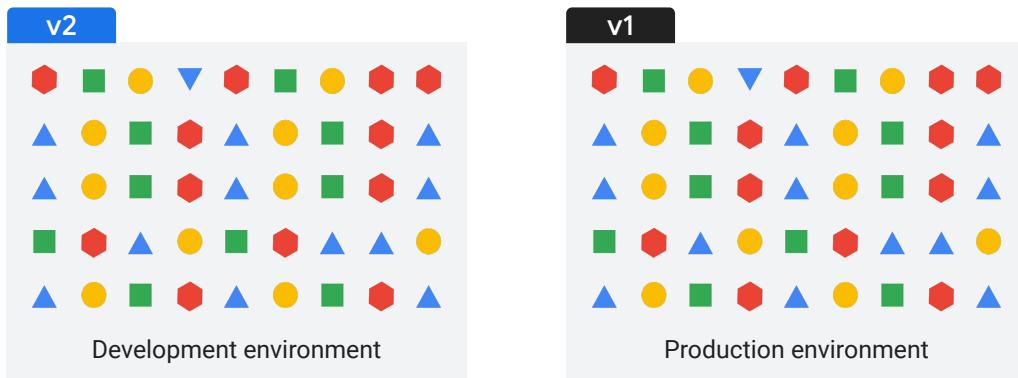
A feedback loop between your model and your algorithm.



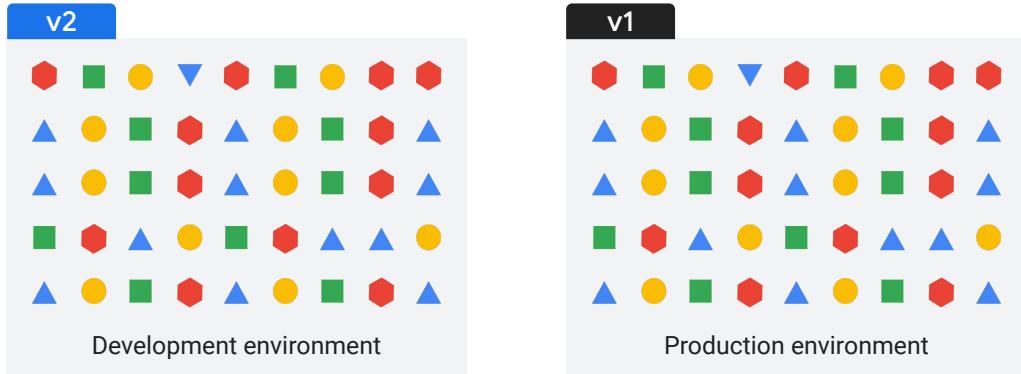
Up until now, we've focused on the data aspect of training-serving skew



but it's also possible to have inconsistencies that arise after the data have been introduced.



Say, for example, that in your development environment, you have version 2 of a library, but in production you have version 1.

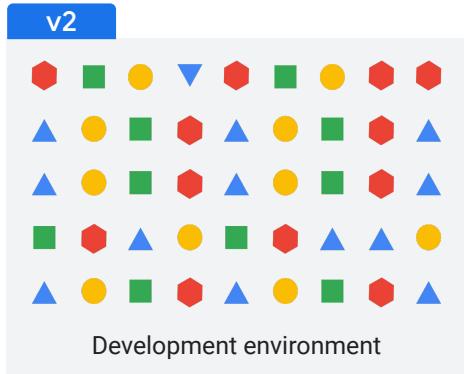


Highly optimized

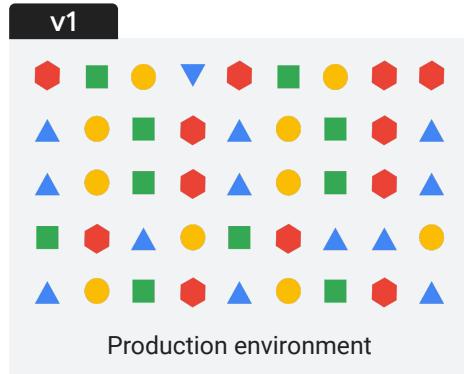


The libraries may be functionally equivalent but version 2 is highly optimized and version 1 isn't.

Consequently, predictions might be significantly slower or consume more memory in production than they did in development.



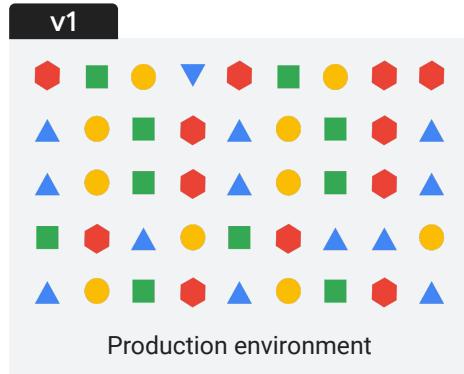
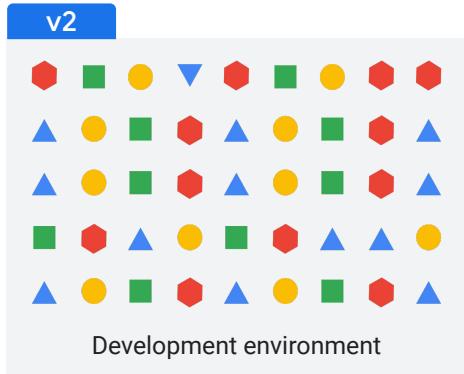
Highly optimized



Contains a bug



Alternately, it's possible that version 1 and version 2 are functionally different, perhaps because of a bug.



Different code



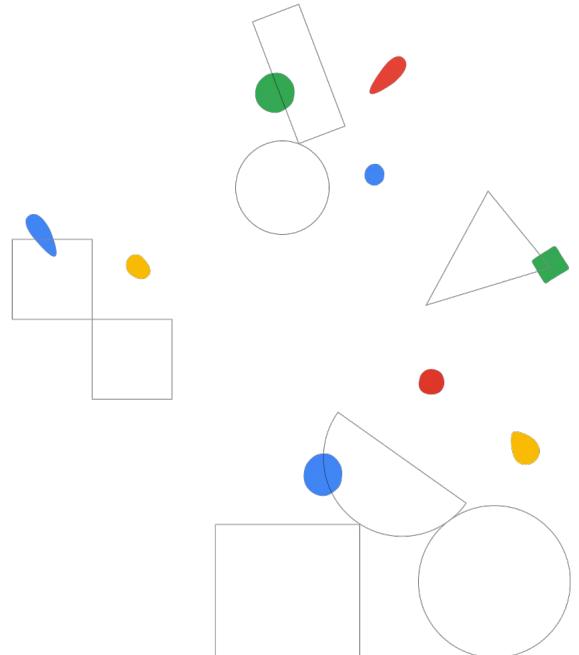
Finally, it's also possible that different code is used in production vs. development, perhaps because of recognition of one of the other issues, but though the intent was to create equivalent code, the results were imperfect.



Lab

Serve ML Predictions in Batch and Real Time

Module 02
Designing adaptable ML systems



This lab provides hands-on practice serving machine learning predictions in batch and real time.

Lab objectives

Create a prediction service to call a trained model that has been deployed in Google Cloud.



You'll begin by creating a prediction service to call a trained model that has been deployed in Google Cloud.

Lab objectives

Create a prediction service to call a trained model that has been deployed in Google Cloud.

Run a Dataflow job where the prediction service reads in batches from a CSV file.



From there, you'll practice running a Dataflow job where the prediction service reads in batches from a CSV file.

Lab objectives

Create a prediction service to call a trained model that has been deployed in Google Cloud.

Run a Dataflow job where the prediction service reads in batches from a CSV file.

Run a streaming Dataflow pipeline to read requests in real time from Pub/Sub.



And, finally, you'll learn how to run a streaming Dataflow pipeline to read requests in real time from Cloud Pub/Sub,

Lab objectives

Create a prediction service to call a trained model that has been deployed in Google Cloud.

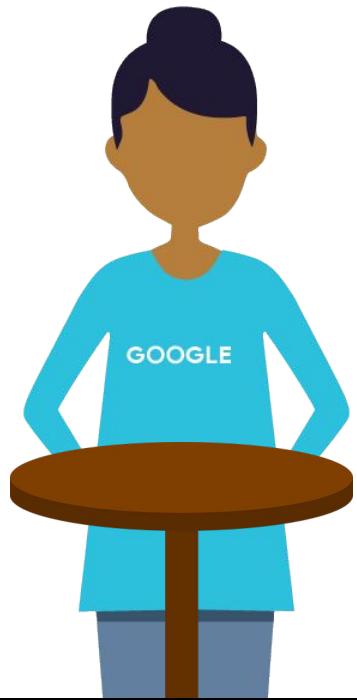
Run a Dataflow job where the prediction service reads in batches from a CSV file.

Run a streaming Dataflow pipeline to read requests in real time from Pub/Sub.

Write predictions to a BigQuery table.



and also write predictions to a BigQuery table.



LAB SOLUTION RECORDING