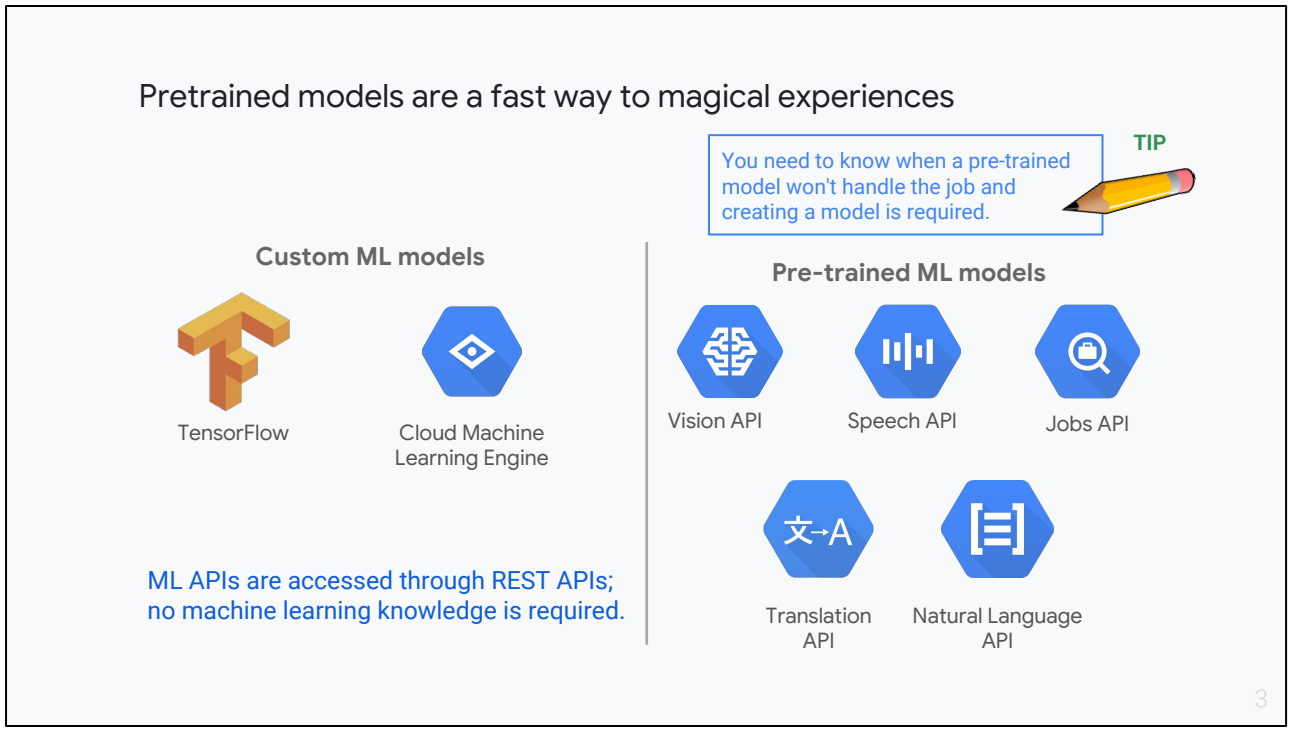
Analyzing and Modeling

Analyzing data and enabling machine learning



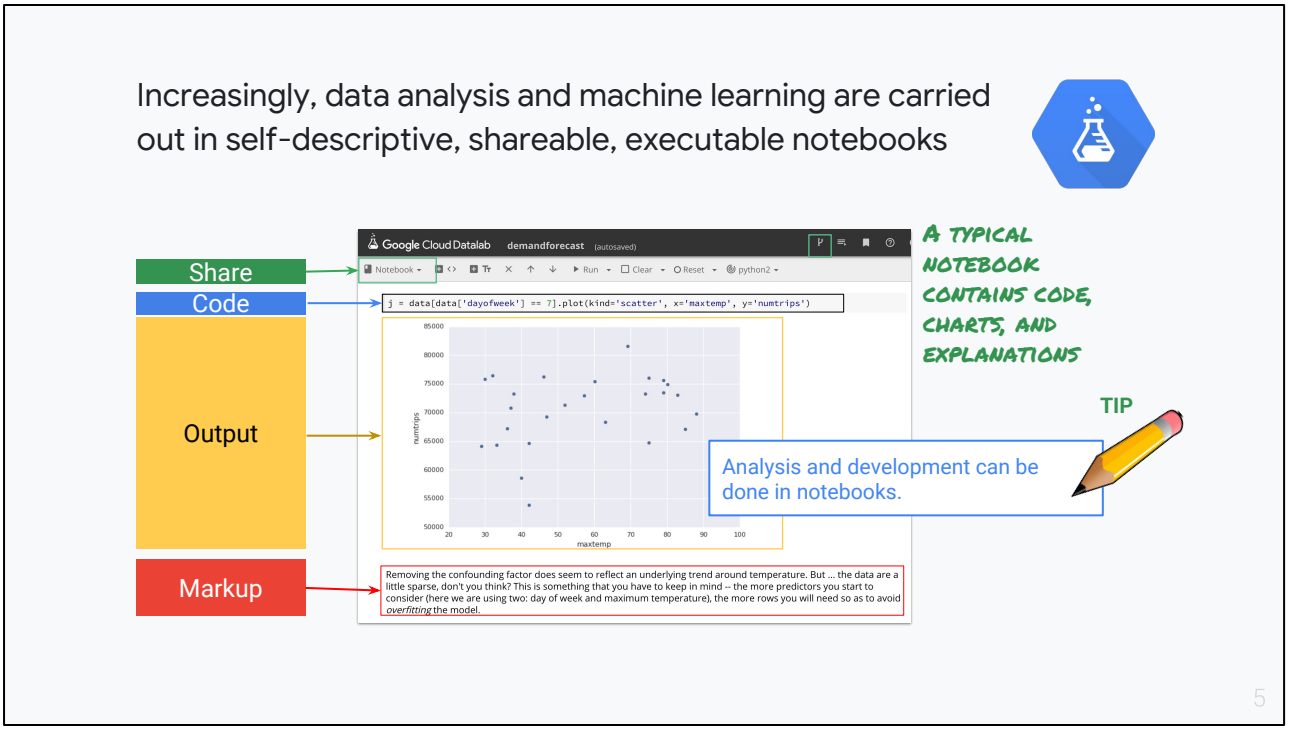
TIP: Make sure you are familiar with each pre-trained ML model.

This page has a list and description of all the ML and AI technologies.
Google Cloud Machine Learning and AI Products:
https://cloud.google.com/products/ai/

Example: What are the three modes of the Natural Language API?

Answer: Sentiment, Entities, and Syntax.
Would Natural Language API be an appropriate tool for identifying all of the people and characters in a document?

Analyzing data

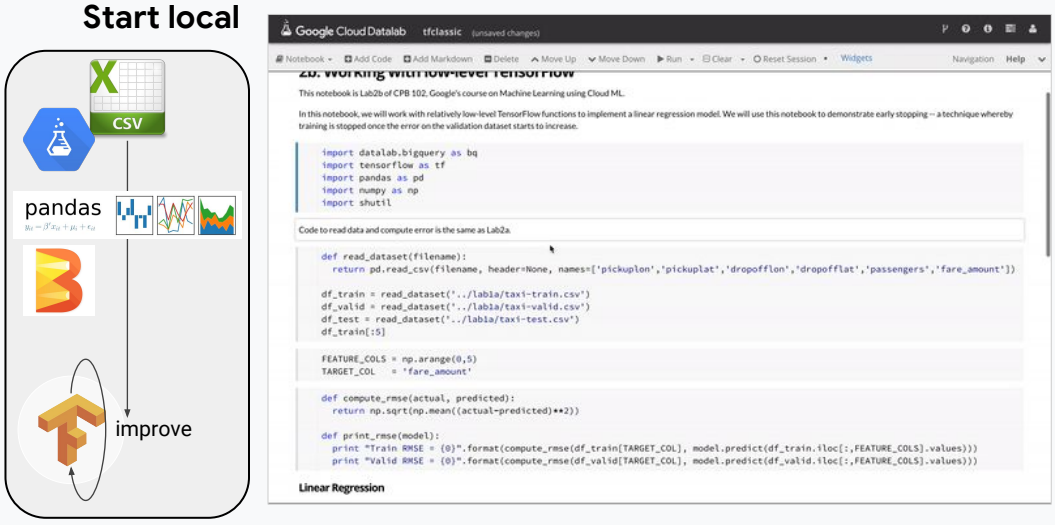


TIP: Notebooks are a self-contained development environment and are often used in modern data processing and machine learning development because they combine code management, source code, control, visualization, and step-by-step execution for gradual development and debugging.

Do you use IPython or Jupyter notebooks today? Increasingly, data scientists work in self-descriptive, shareable, executable notebooks whenever they want to do data analysis or machine learning.
Cloud Datalab is based on Jupyter and it's open source.

Combines code, documentation, results, and visualizations together in an intuitive notebook format
Cloud Datalab is built on Jupyter (formerly IPython)
Python, SQL, and JavaScript
Supports Google Charting or matplotlib for easy visualizations
How to launch:
From Cloud Shell: `datalab create <vm-name>`
Or Cloud Dataproc Initialization action

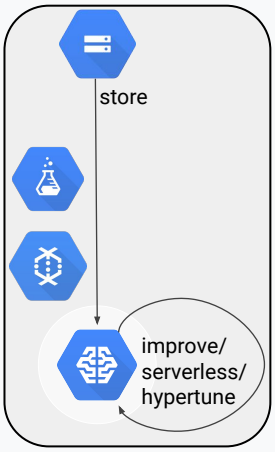
In Cloud Datalab, start locally on sampled dataset

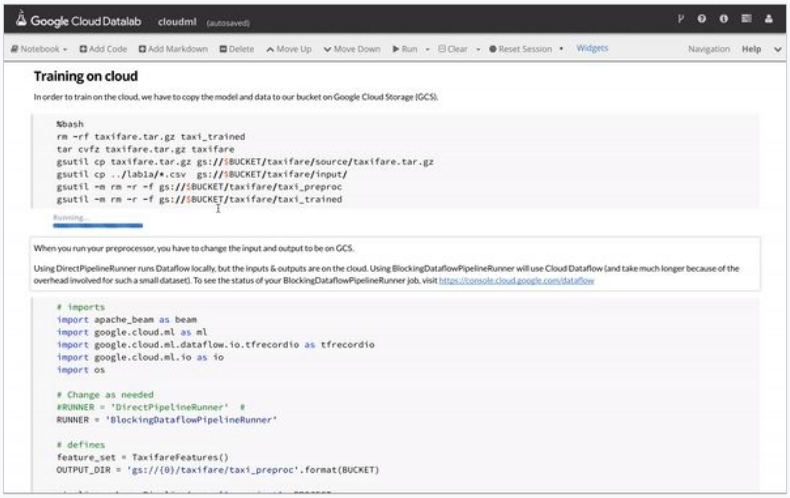


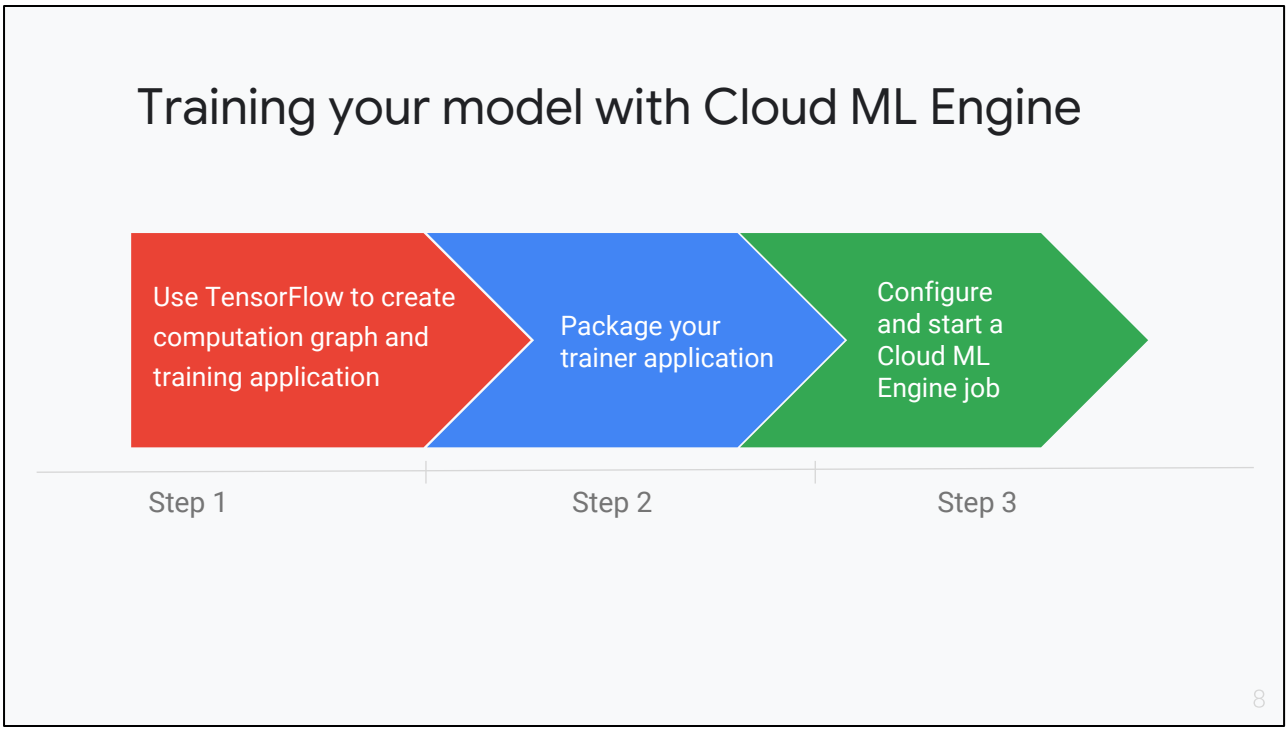
The fail fast tool to analyze and learn from your data
Interactively explore data
- Define features with rich visualization support
- Launch training and evaluation
- ML lifecycle support
- Combine code, results, visualizations, and documentation in notebook format
- Share results with your team
- Pick from a rich set of tutorials and samples to learn and get started with your project

Then, scale it out to GCP using serverless technology

Grow serverless







Preparation: (1) gather and prepare training data (clean, split, engineer features, preprocess features), and (2) store training data in an online source that Cloud ML Engine can access, such as Cloud Storage.

Machine learning

In reality, machine learning is...

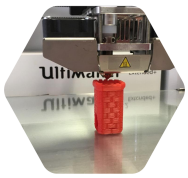
TIP

Which step or steps is the question asking about?

Collect data

Organize data

Create model



Use machines to flesh out the model from data

Deploy fleshed out model

TIP: In questions about machine learning, make sure you identify the step. Some steps involve similar or related actions

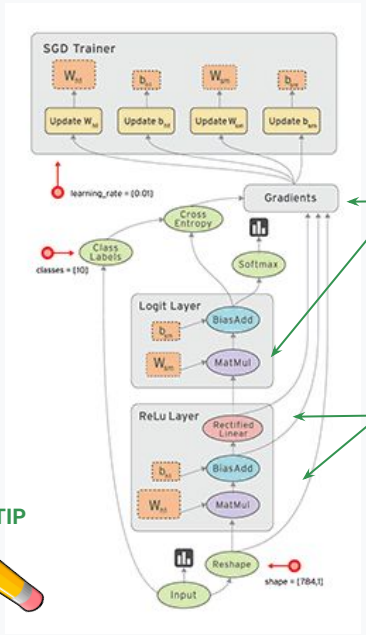
On GCP, we can use:
Logging APIs, Cloud Pub/Sub, etc. and other real-time streaming to collect the data.
BigQuery, Cloud Dataflow and ML preprocessing SDK to organize the data [different types of organization].
TensorFlow to create the model.
Cloud ML Engine to train and deploy the model.

https://pixabay.com/en/ant-brown-carrying-egg-white-44588/ (cc0)
https://pixabay.com/en/tile-organization-exterior-materials-846016/ (cc0)
https://pixabay.com/en/deer-dream-animal-fantasy-1333814/ (cc0), cropped
https://pixabay.com/en/printer-3d-pressure-3d-printing-1455169/ (cc0)
https://pixabay.com/en/deer-statue-silhouette-animal-stag-732122/ (cc0)

TensorFlow is an
open-source
high-performance library
for numerical
computation
that uses directed graphs

Directed Graphs are mentioned in multiple technologies in Google Cloud.

TIP



11

TensorFlow is an open-source high-performance library for numerical computation. Not just about machine learning. Any numeric computation. In fact, people have used TensorFlow for all kinds of GPU computing; for example, you can use TensorFlow to solve partial differential equations -- these are useful in domains like fluid dynamics. TensorFlow as a numeric programming library is appealing because you can write your computation code in a high-level language -- Python -- and have it be executed in a fast way.

The way TensorFlow works is that you create a directed graph (a DG) to represent your computation. In this schematic, the nodes represent mathematical operations -- things like adding, subtracting, and multiplying. Also more complex functions. Here, for example, softmax, and matrix-multiplication, etc. are mathematical operations that are part of the directed graph.

Connecting the nodes are the edges. The input and output of the mathematical operations. The edges represent arrays of data. Essentially, the result of computing the cross-entropy is one of the three inputs to the BiasAdd operation, and the output of the BiasAdd operation is sent along to the matrix-multiplication operation (MatMul). The other input to MatMul -- you need two inputs to a matrix multiplication -- the other input is a variable, the weight.

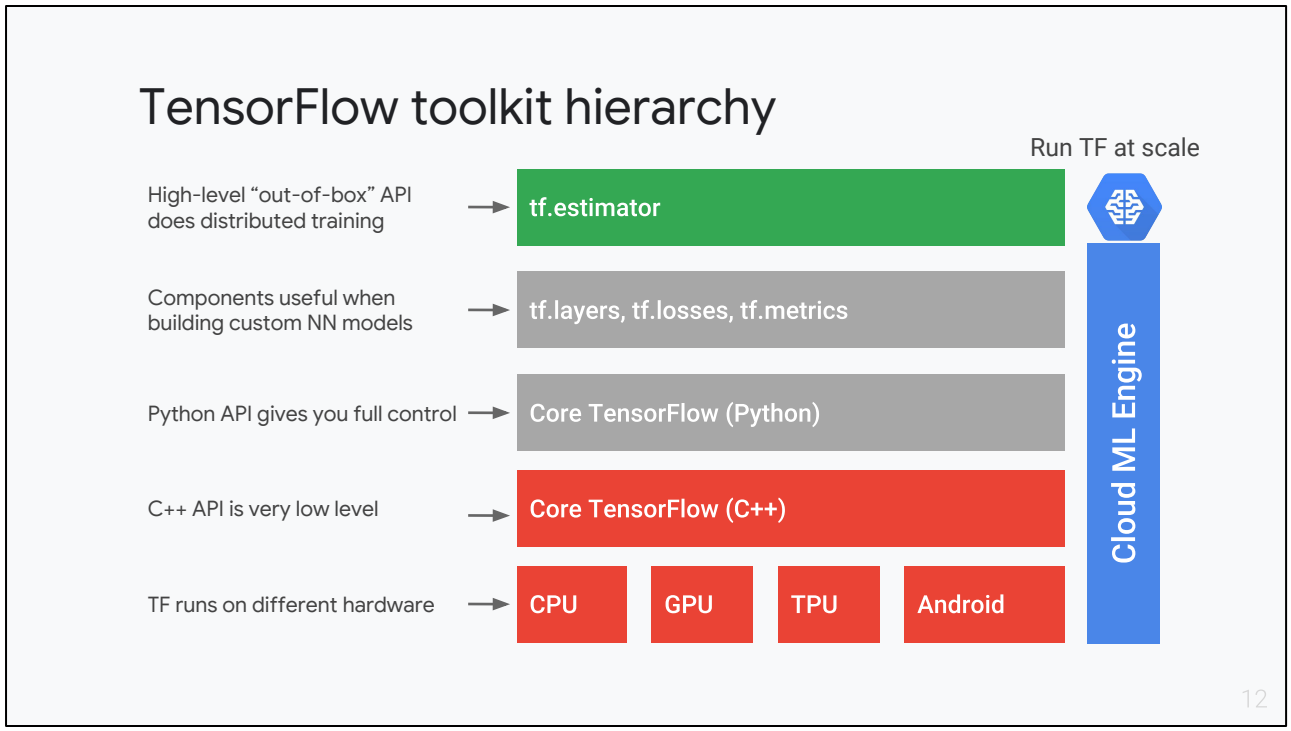
Portable across GPUs, CPUs, mobile ...

Developed at Google

Uses data flow graphs: neural network training and evaluation can be represented as data flow graphs

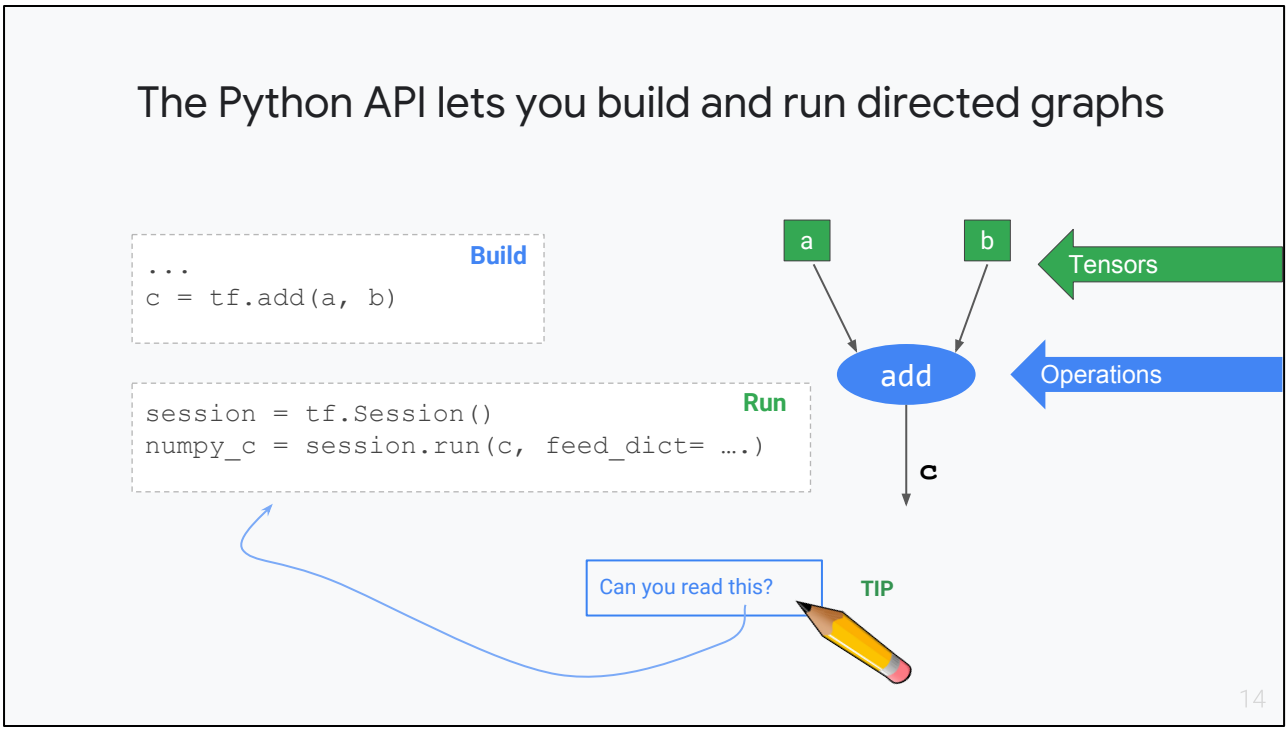
http://www.tensorflow.org/

GIF: https://www.tensorflow.org/images/tensors_flowing.gif



TIP: Be familiar with all the layers of TensorFlow.

Similar to how Java programs run on many different types of hardware because of the JVM. TensorFlow C++ plays the role of the JVM here, providing hardware instruction sets.



TIP: Be able to read a TensorFlow program and understand generally what it is doing. Know the major objects and methods.

So, let's look at the code on this slide. At first glance, this looks just like, say, numpy.
You want to add two tensors, a and b.
So, you write tf.add(a, b)
It returns a tensor c.

Unlike typical Python code, though, running the tf.add() doesn't execute it. It only builds the DG.
In the DG -- in the directed graph -- a, b and c are tensors. Add is an operation.
In order to run this code, in order to execute the DG, you need to run it. And you run it as part of what is called a Session. So, you say you want the value of c and you ask the session to evaluate c for you.
That's what runs the DG and you get back a traditional numeric array in Python that contains the values for c.

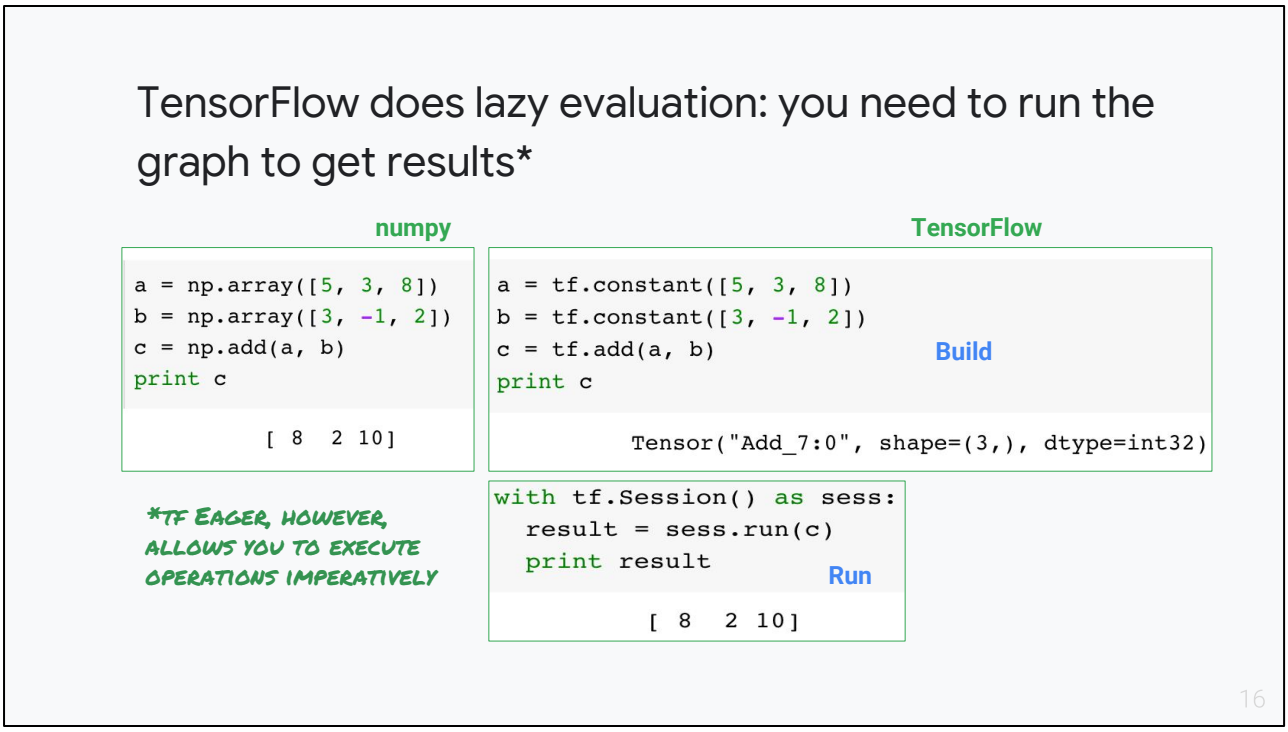
Programming TensorFlow involves programming a DG. There are two steps.
First step: create a graph.
Second step: run it.

The graph definition is separate from the training loop because this is a lazy evaluation model. It minimizes the Python/C++ context switches and enables the computation to be very efficient. Conceptually, this is like writing a program, compiling

it, and then running it on some data. Don't take the analogy too far, though.There is no explicit compile phase here.

Note that c after you call tf.add is not the actual values -- you have to evaluate c in the context of a TensorFlow session to get a numpy array of values ('numpy_c').

Note: We use the term **Directed Graph (DG)** instead of **Directed Acyclic Graph (DAG)** because TensorFlow supports cyclic graphs.
For more on this topic:
https://stackoverflow.com/questions/37551389/cyclic-computational-graphs-with-tensorflow-or-theano/37551496#37551496
http://kias.dyndns.org/comath/33.html



TIP: Do you know what numpy is?

So, to reiterate, TensorFlow does lazy evaluation. You write a DG. Then you run the DG in the context of a session to get results. Now, there is a different mode in which you can run TensorFlow, called tf.eager, where the evaluation is immediate and it is not lazy. But eager mode is typically not used in production programs; it's only for development. We'll look at tf.eager a little bit later in this course, but for the most part, we'll focus on the lazy evaluation paradigm. And almost all the code we create and run in production will be in lazy evaluation mode.

The majority of Python numeric software is written using the numpy library. Numpy is very fast because it has been implemented using the C-language. The difference between numpy and TensorFlow in this example is lazy evaluation in TensorFlow. In the numpy example, a and b are numpy arrays. When you call np.add(), that add gets done in C-language, so it is evaluated immediately. Then the CPU runs the line of code np.add(a, b) and the numpy array c gets populated with the sums. So, when you print "c," you get the 8, 2, 10. 8 is the sum of 5 and 3. Add 3 and -1 to get 2. etc. The point is the np.add() is evaluated immediately.

Unlike with numpy, in TensorFlow c is not the actual values. Instead, c is a tensor -- you have to evaluate c in the context of a TensorFlow session to get a numpy array of values ('result').
So, when the CPU or GPU or other whatever hardware evaluates tf.add(a, b), a Tensor gets created in the directed graph, in the DG. But the addition is not carried

out until the sess.run() gets called.
So, if we call print c, what gets printed out in the first box is the “debug” output of the Tensor class. It includes a system-assigned unique name for the node in the DG (“ADD_7”) and the shape and data type of the value that will appear when the DG is run.

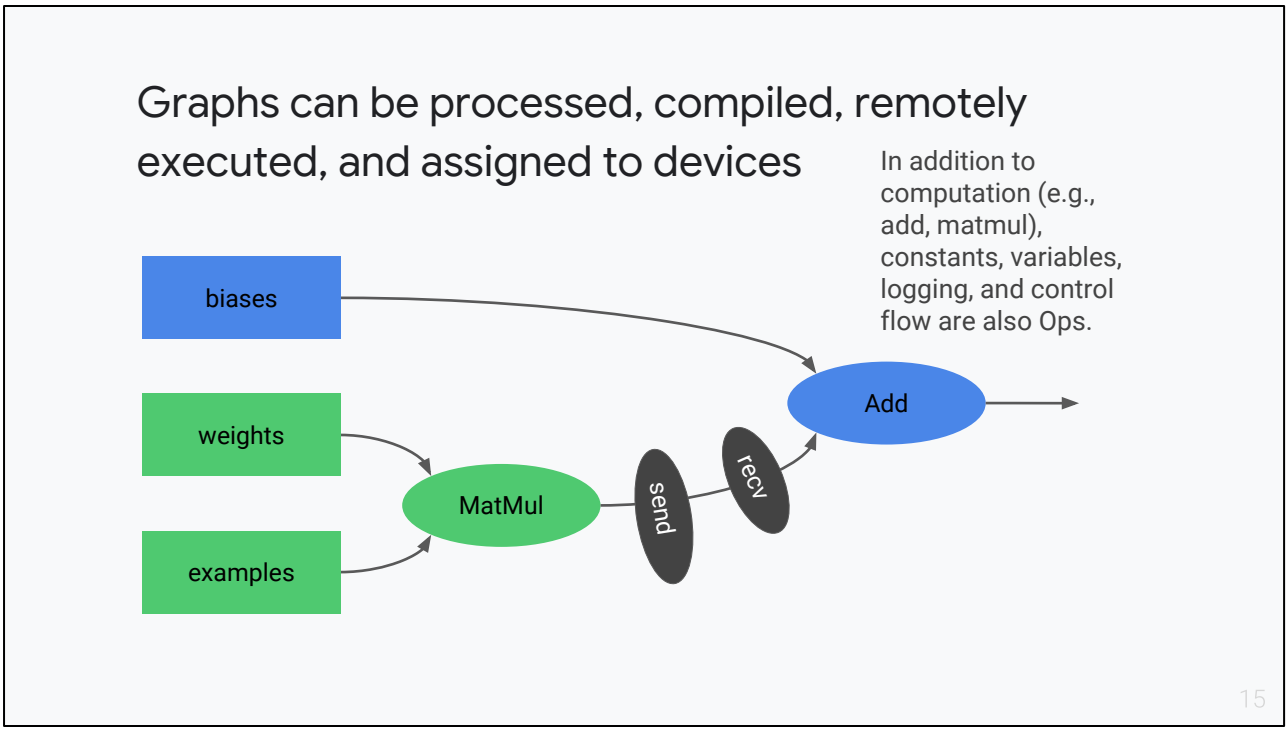
After the session is run and c is evaluated in the context of the session, we can print the result and we’ll get [8, 2, 10]. So, two stages: a build stage and a run stage.

But why? Why does TensorFlow do lazy evaluation?

https://github.com/tensorflow/tensorflow/tree/master/tensorflow/python/eager

https://github.com/tensorflow/tensorflow/tree/master/tensorflow/python/eager

TIP: Don't get confused. TensorFlow uses lazy evaluation. The Eager Execution module is a frontend to TensorFlow that is used for interactive learning of TensorFlow and for experimentation/prototyping. It enables imperative commands from Python that are executed immediately.
https://github.com/tensorflow/tensorflow/blob/master/tensorflow/contrib/eager/README.md



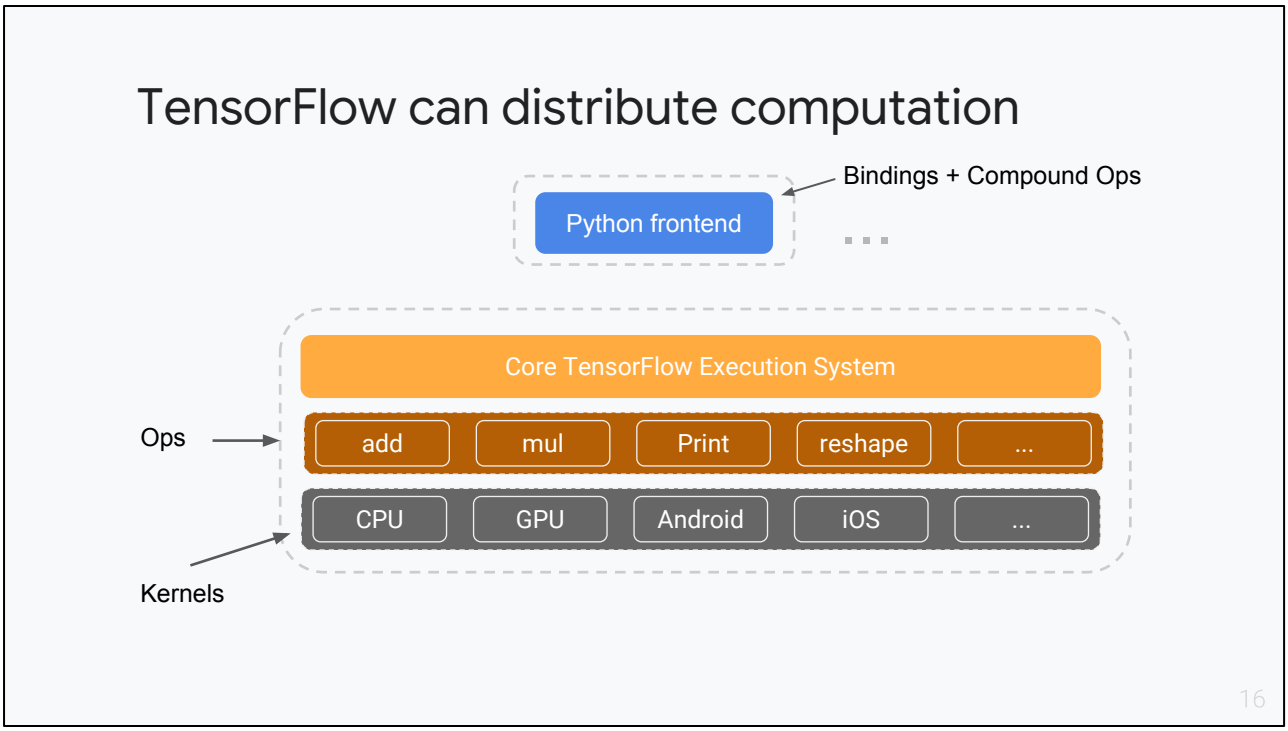
TIP: Because developing ML models are so processing-intensive, it is important to get the model right before scaling up; otherwise the models can become expensive!

Example of each mentioned use case:

Processed: add quantization or data types, add debug nodes; create summaries to write values out so that Tensorboard can read ...

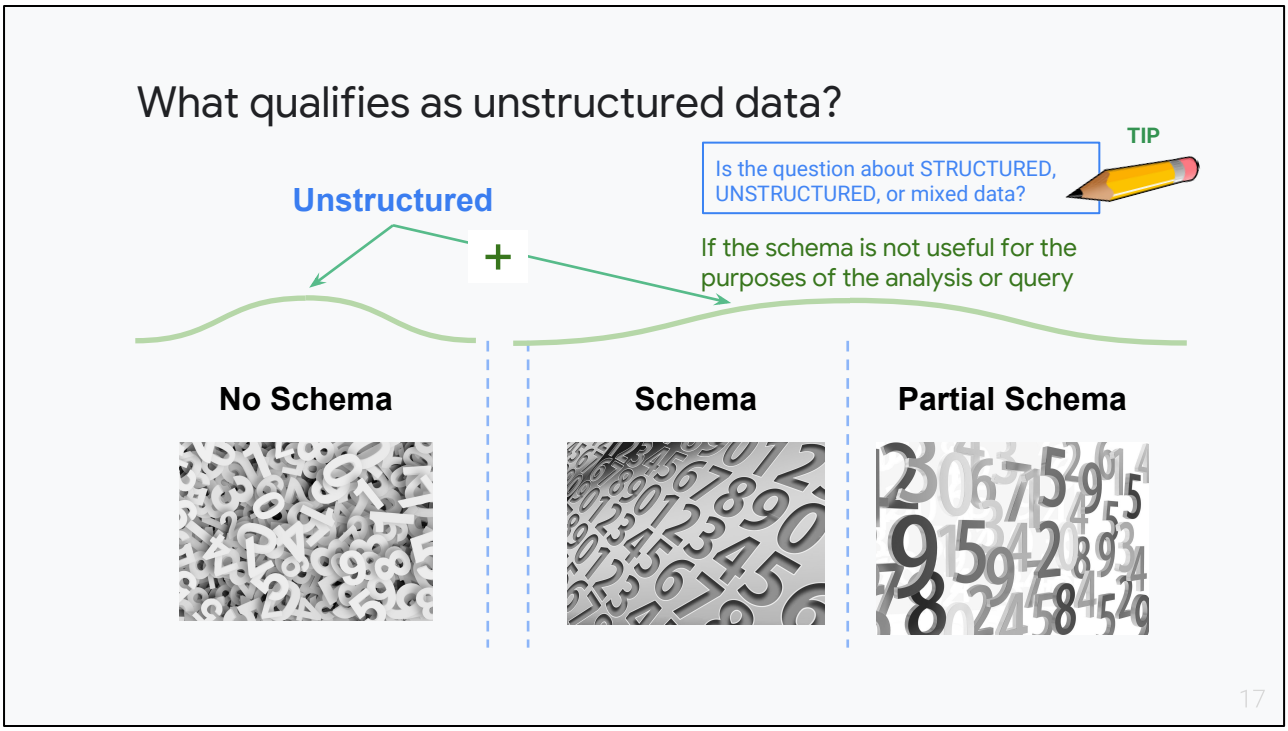
Compiled: fuse ops to improve performance. For example, two consecutive add nodes can be fused into a single one.

Remotely executed, assigned to devices: **note colors**, several parts of the graph can be on different devices, doesn't matter whether GPU or several computers. Automatically insert send/recv nodes.



TIP: One benefit of Google's ML Platform is the ability to scale up to production level.

One key benefit of this model -- to be able to distribute computation across many machines, and many types of machines. No need to assign a Print op to a GPU.



TIP: Even data that has a schema might still be unstructured if it is not useful for your intended purpose.

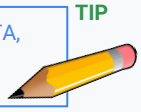
Example: A survey form contains answers to questions including a text field for comment and some multiple choice fields. The form itself is structured and would easily be stored in a relational database table with a schema. However, what if you were interested in counting the number of surveys that contained positive comments? For this purpose the data is unstructured. Machine Learning (Natural Language Processing Sentiment Analysis) could label or tag the comment field with a sentiment value making the data useful for your purpose.

Data without a schema is unstructured. However, if data has a schema or partial schema but it is not helpful to the purposes of analysis or query, that data is considered unstructured also.

https://pixabay.com/en/pay-digit-number-fill-count-mass-1036472/
https://pixabay.com/en/pay-numbers-digits-mathematics-2662758/
https://pixabay.com/en/pay-digit-number-fill-count-mass-1036469/

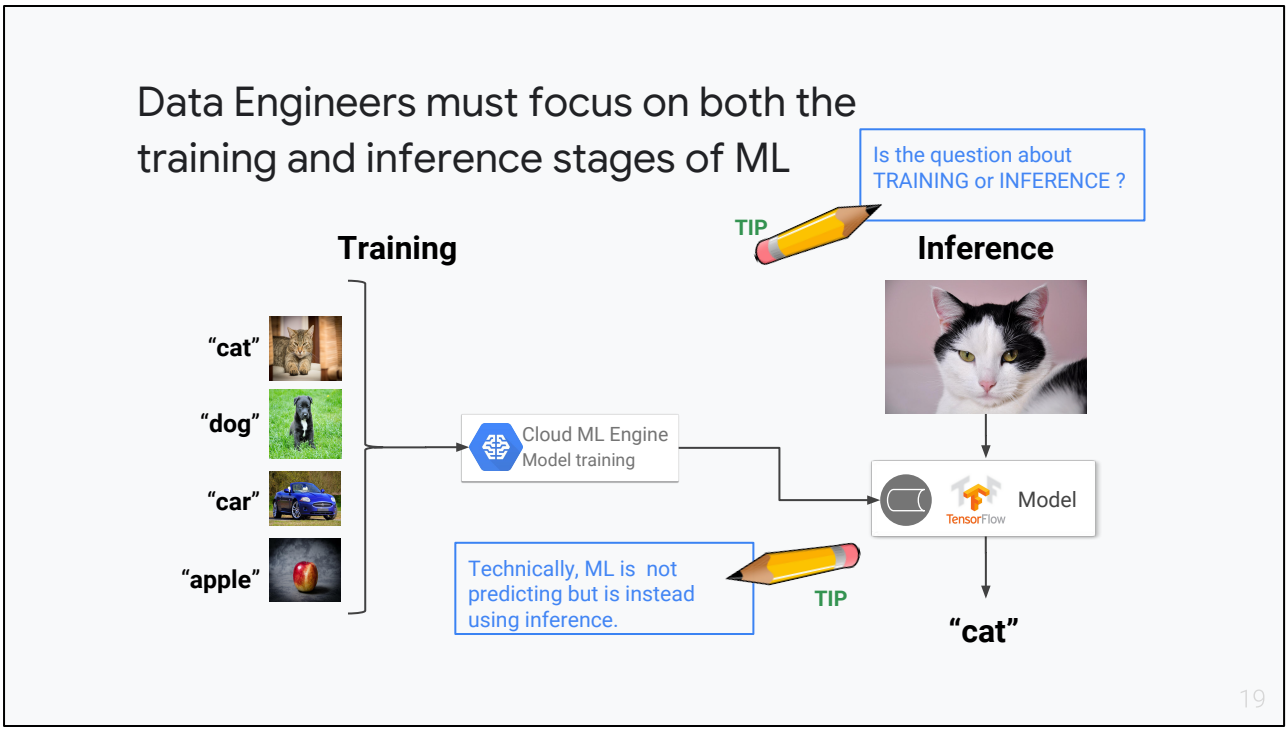
Working with unstructured data

Is the problem suited for BIG DATA, MACHINE LEARNING, or human decision-making?



Human Reasoning	Easy counting problems = big data	Harder counting problems = ML
Real-time insight into supply chain operations. Which partner is causing issues?	Did error rates decrease after the bug fix was applied?	Are programmers checking in low-quality code?
Drive product decisions. How do people really use feature X?	Which stores are experiencing long delays in payment processing?	Which stores are experiencing a lack of parking space?

TIP: Distinguish between one-off (reasoning) problems that are best solved by humans, Big Data problems, and Machine Learning problems.

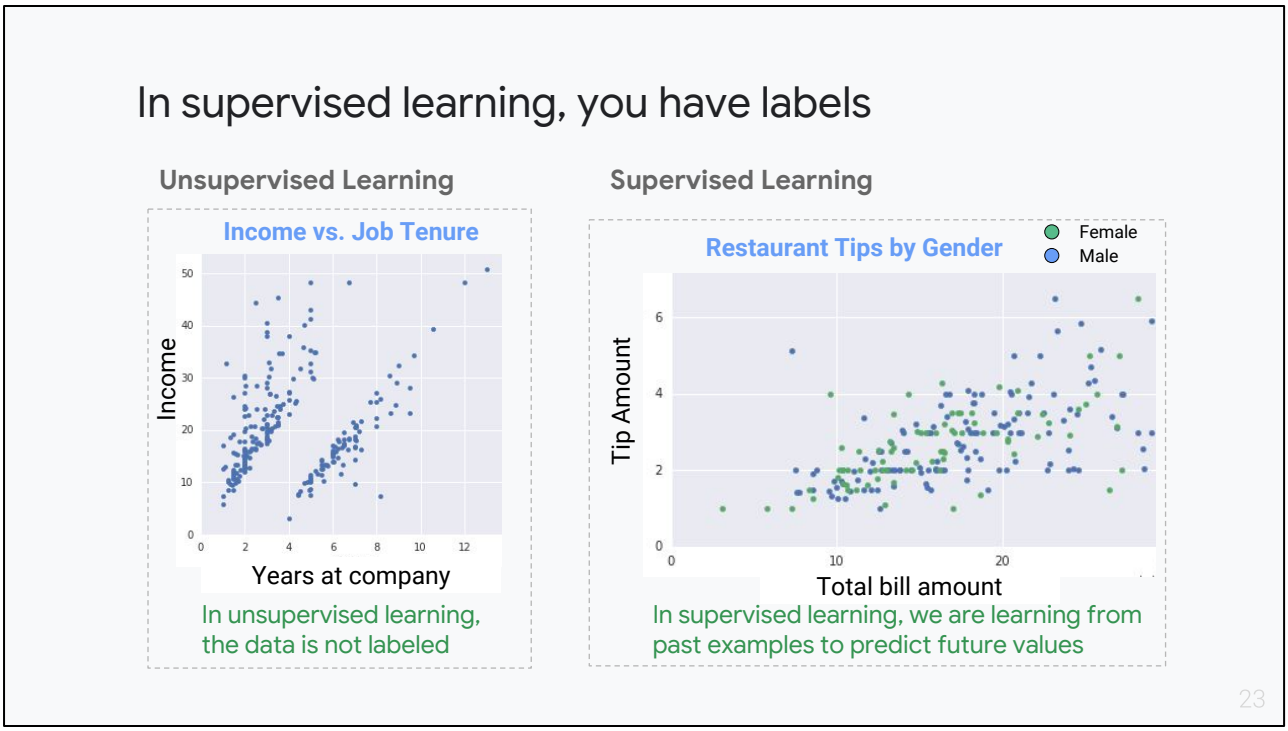


It is important to realize that machine learning has two stages: training and inference. Sometimes, people refer to "prediction" as "inference" because "prediction" seems to imply a future state and in the case of images like this, we are not really "predicting" that it is a cat, just "inferring" that it is a cat based on the pixel data.

It can be tempting, as a Data Engineer, to focus all your energy, on the first stage. But this is not enough.
You need to be able to operationalize the model, put the model into production so that you can run inferences.

Look at many books on machine learning, blog posts, university courses ... They tend to ignore this second stage of ML. But in the real-world, what is the use of training a ML model if you can not use it?
In this specialization, we will be careful to show you machine learning, end-to-end. And by end-to-end, we mean putting ML models into production.

https://pixabay.com/en/cat-cute-cat-baby-kitten-pet-1992140/(cc0)
https://pixabay.com/en/mouse-rodent-cute-mammal-nager-1708347/



TIP: THEREFORE… if you have an ML Question that refers to LABELS it is a question about SUPERVISED LEARNING.

Example mode: Clustering
Is this employee on the "fast-track" or not?

Two of the most common classes of machine learning models are supervised and unsupervised ML models. The key difference is that with supervised models, we have labels, or in other words, the correct answers to whatever it is that we want to learn to predict.

In unsupervised learning, the data does not have labels.

This graph is an example of the sort of problem that an unsupervised model might try to solve. Here, we want to look at tenure and income, and then group or cluster employees, to see whether someone is on the fast track. Critically, there is no ground truth here; management doesn't, as far as we know, have a big table of people they are going to promote fast, and those they are not going to promote. Consequently, unsupervised problems are all about discovery, about looking at the raw data, and seeing if it naturally falls into groups. At first look, it seems that there are two distinct clusters or groups that I could separate nicely with a line.

In this course though, we'll be focused on supervised machine learning problems, like this one. The critical difference is that with supervised learning, we have some notion

of a “label,” or one characteristic of each data point that we care about a lot.

Typically, this is something we know about in historical data, but we don’t know in real time. We know other things, which we call predictors, and we want to use those predictors to predict the thing we don’t know.

For example, let’s say you are the waiter in a restaurant. You have historical data of the bill amount and how much different people tipped. Now, you are looking at the group sitting at the corner table. You know what their total bill is, but, you don’t know what their tip is going to be. In the historical data, the tip is a label. You create a model to predict the tip from the bill amount. Then, you try to predict the tip, in real time, based on the historical data and the values that you know for the specific table.

Regression and classification are supervised ML model types

	total_bill	tip	sex	smoker	day	time
2	16.99	1.01	Female	No	Sun	Dinner
3	10.34	1.66	Male	No	Sun	Dinner
4	21.01	3.5	Male	No	Sun	Dinner
5	23.68	3.31	Male	No	Sun	Dinner
6	24.59	3.61	Female	No	Sun	Dinner
7	25.29	4.71	Male	No	Sun	Dinner
8	8.77	2	Male	No	Sun	Dinner
9	26.88	3.12	Male	No	Sun	Dinner

Option 1
Regression Model
Predict the tip amount

Option 2
Classification Model
Predict the sex of the customer

21

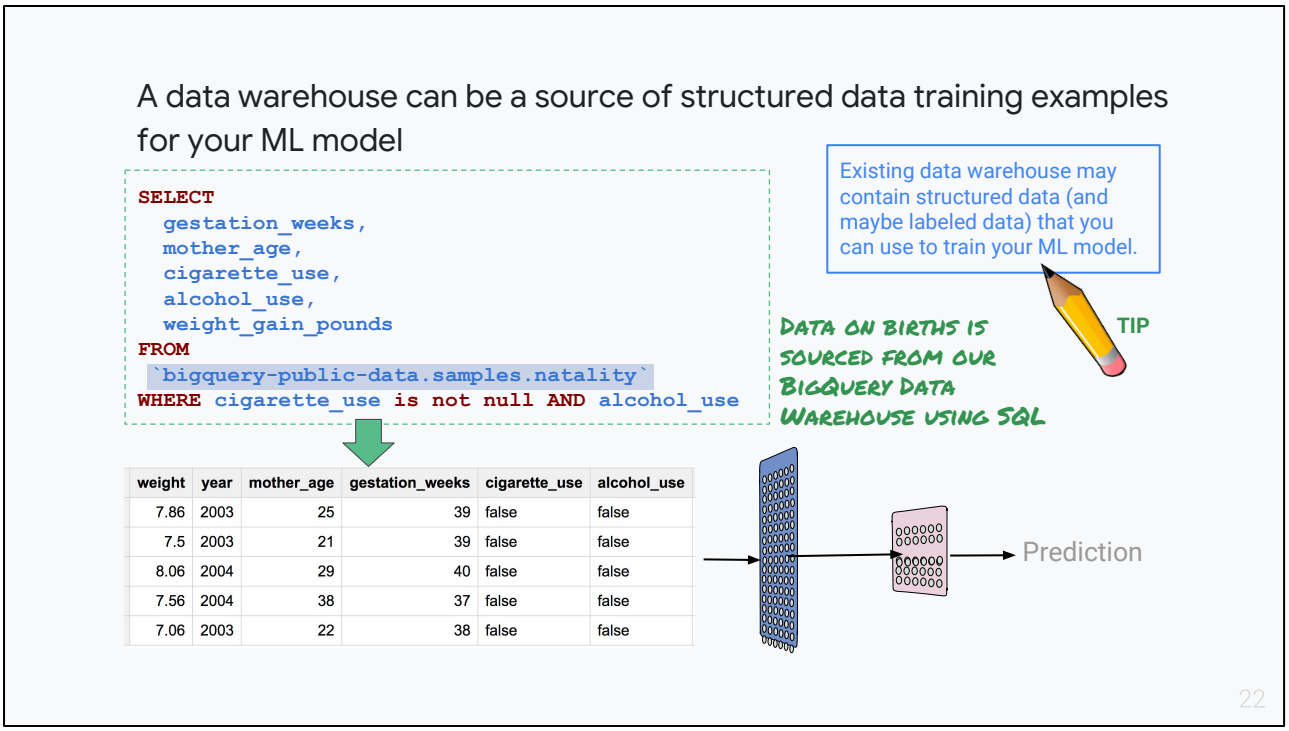
TIP: THEREFORE… if the question is about REGRESSION or CLASSIFICATION it is using SUPERVISED ML.

Within supervised ML there are two types of problems: regression and classification. To explain them, let's dive a little deeper into this data.

In this dataset of tips, an example dataset that comes with the Python package seaborn, each row has many characteristics, such as total bill, tip, and sex. In machine learning, we call each row an "example." We'll choose one of the columns as the characteristic we want to predict, called the "label," and we'll choose a set of the other columns, which are called the "features."

In model option 1, we want to predict the tip amount, therefore the column tip is my label. I can use one, all, or any number of the other columns as my features to predict the tip. This will be a regression model because tip is a continuous label.

In model option 2, we want to predict the sex of the customer, therefore the column sex is the label. Once again, I will use some set of the rest of the columns as my features, to try and predict the customer's sex. This will be a classification model because our label sex has a discrete number of values or classes.



A very common source of structured data for machine learning is your data warehouse. Unstructured data is things like pictures, audio, or video.

Shown is a natality dataset, a public dataset of medical information in BigQuery.

The goal is to predict when a baby will be born. (Gestation weeks).

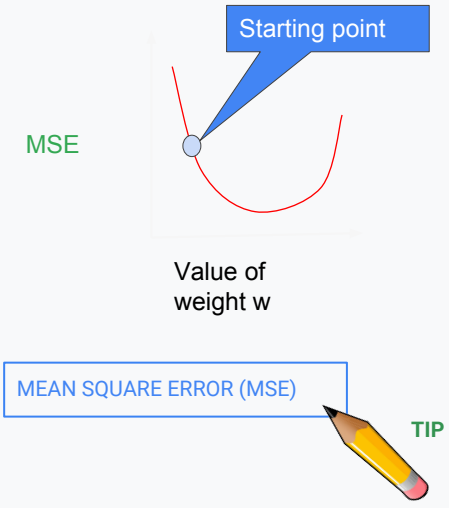
You can do a SQL SELECT statement in BigQuery to create an ML dataset -- choosing input features to the model, things like mother's age and weight gain in pounds, and the label, gestation weeks.

Because gestation weeks is a continuous number, this is a regression problem.

Mean squared error is a measure of loss

Y-cap is the model estimate
Y is the labeled value
Mean squared error (MSE) is:

$$\frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$



23

TIP: If you don't define a metric or measure for how well your model works, how will you know it is working sufficiently to be useful for your business purpose?

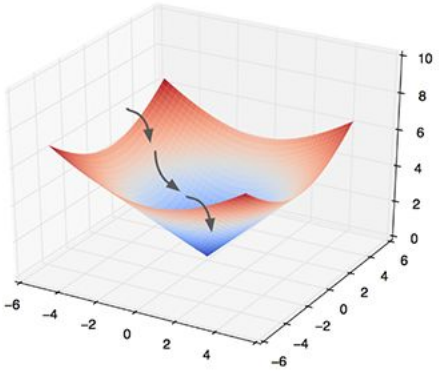
Mathematically speaking, we prefer differentiable error measures so that we can do gradient descent. Mean-square error is differentiable.

Mean Square Error or Mean squared error?
https://stats.stackexchange.com/questions/30816/mean-square-error-or-mean-squared-error

In most cases you would use Mean Square Error because the RMSE (Root of the Mean Square Error) is consistently used without the 'D'.

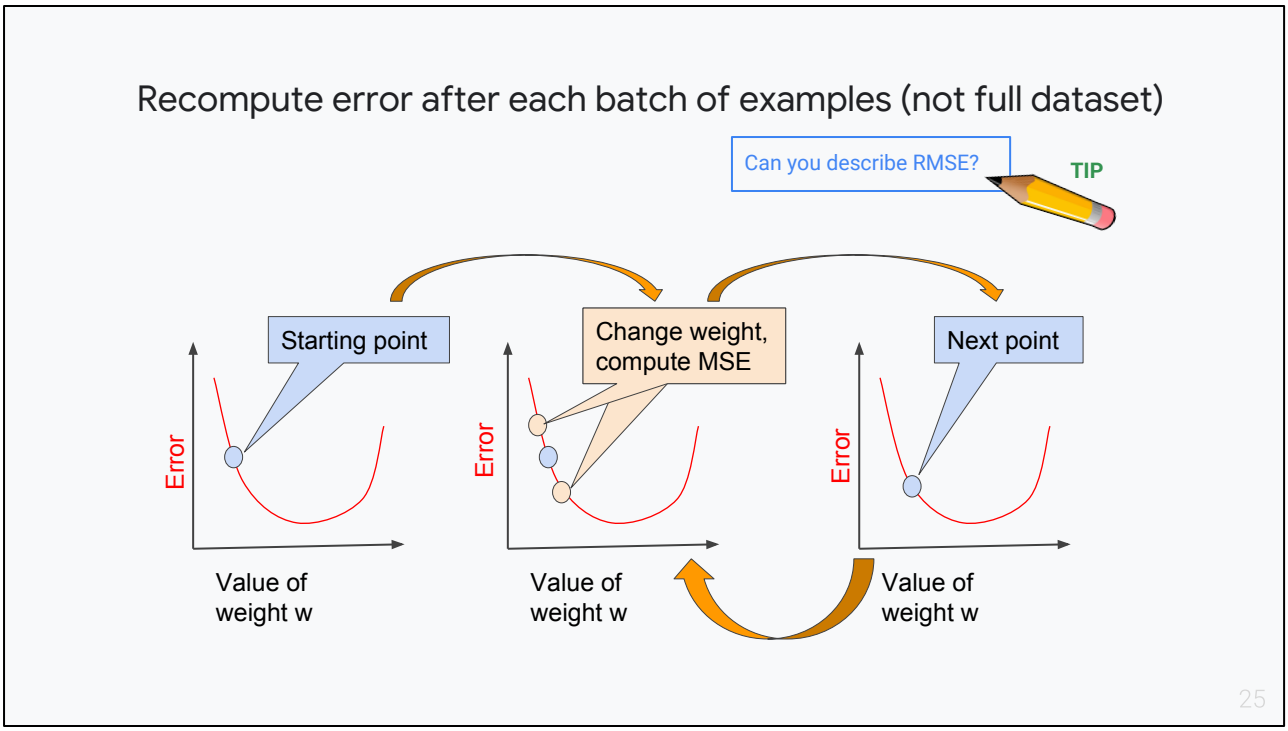
However, this slide conforms to the Google training style, so "sqaure" appears with the 'd', because "Mean square error" is a more popular search result than "Mean squared error".
Also, the second and third terms are in lower case to conform to Google training style.

Gradient descent is used to find the best parameters



	-5	-4	-3	-2	-1	0	1	2	3	4	5
5	50	41	34	29	26	25	26	29	34	41	50
4	41	32	25	20	17	16	17	20	25	32	41
3	34	25	18	13	10	9	10	13	18	25	34
2	29	20	13	8	5	4	5	8	13	20	29
1	26	17	10	5	2	1	2	5	10	17	26
0	25	16	9		1	0	1	4	9	16	25
-1	26	17	10	5	2	1	2	5	10	17	26
-2	29	20	13	8	5	4	5	8	13	20	29
-3	34	25	18	13	10	9	10	13	18	25	34
-4	41	32	25	20	17	16	17	20	25	32	41
-5	50	41	34	29	26	25	26	29	34	41	50

TIP: Gradient descent is an important method to understand. It is *HOW* an ML problem is turned into a search problem.



TIP: MSE and RMSE are measures of how well the model fits reality, how well the model works to categorize or predict.

The Root of the Mean Square Error

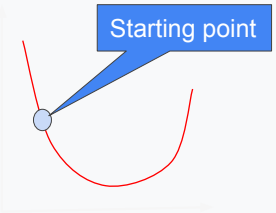
One reason for using the Root of the Mean Square Error rather than the Mean Square Error, is because the RMSE is in the units of the measurement, making it easier to read and understand the significance of the value.

https://en.wikipedia.org/wiki/Root-mean-square_deviation

Typical batch size = 100-500 samples.

For classification problems, we use cross entropy

For classification problems, the most commonly used error measure is *cross entropy*—because it is differentiable:



$$-\frac{1}{N}\sum_{n=1}^{N}\left[y_n \log \hat{y}_n + (1-y_n)\log(1-\hat{y}_n)\right]$$

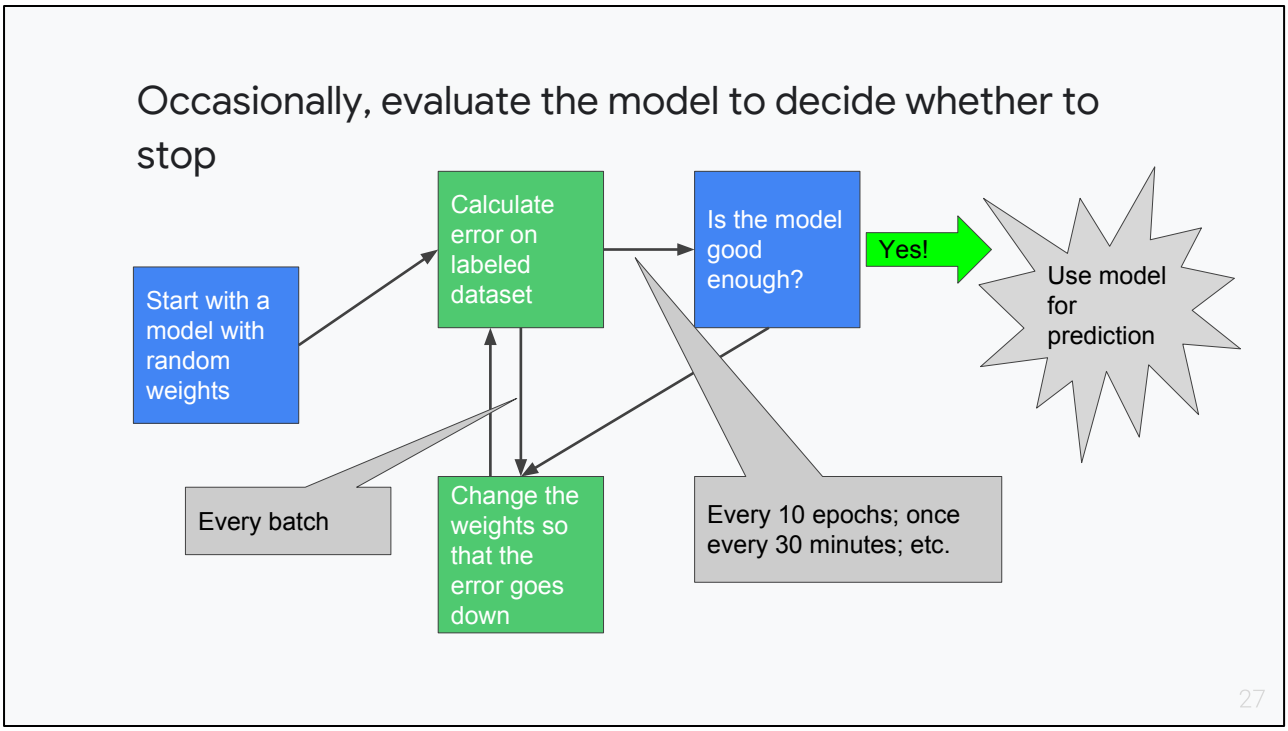
TIP: Categorizing (discrete values) and regression (continuous values) use different methods. If the question describes cross-entropy… it is a classification ML problem.

What is cross entropy?: https://en.wikipedia.org/wiki/Cross_entropy

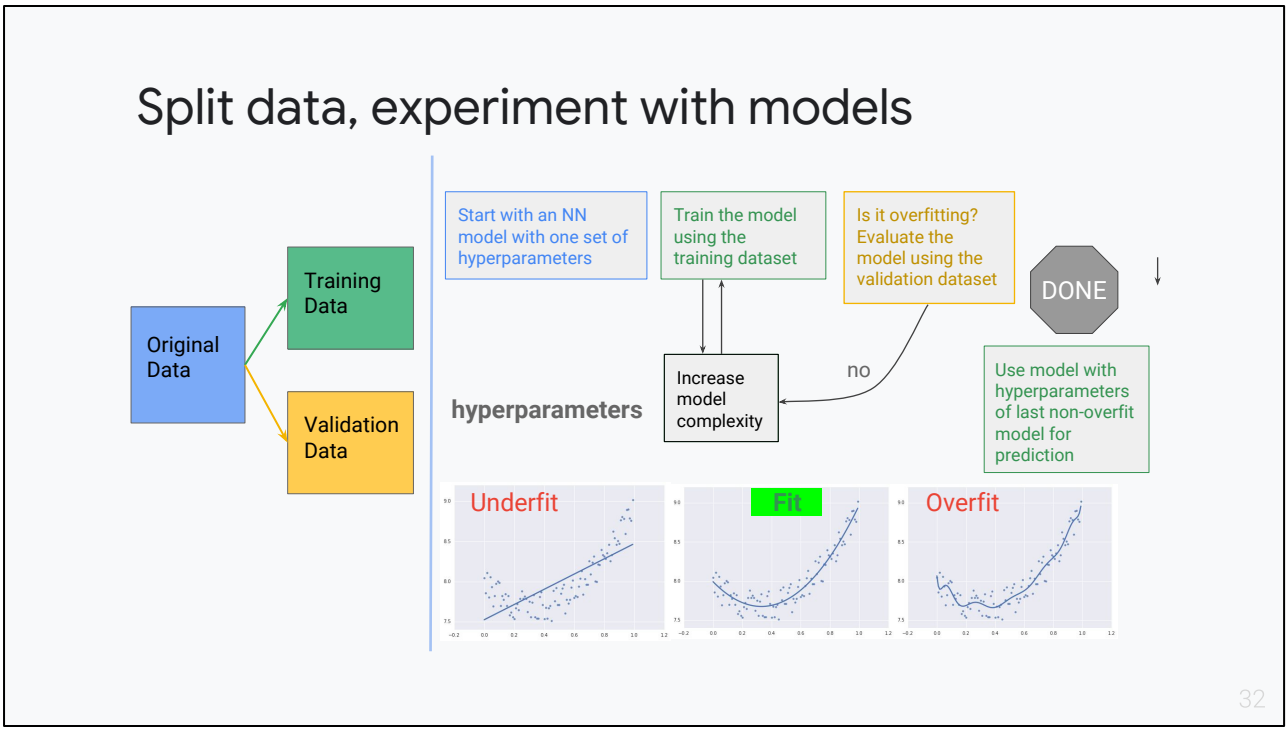
Mathematically speaking, we prefer differentiable error measures so that we can do gradient descent. So, for classification, we'll use cross entropy.

Why cross entropy for categorical variables? See: https://en.wikipedia.org/wiki/Cross_entropy#Cross-entropy_error_function_and_logistic_regression.

In non-mathematical terms: You use a neural network whose output node is a soft-max (so that outputs are restricted to [0,1]) and whose objective function is cross entropy. The result of the NN can be treated as a probability. So if you train using data on whether a customer bought your product or did not buy the product and you train on that data using the right NN structure, the trained NN can be used to estimate the probability that a visitor to your website will buy your product! This is hugely useful.



TIP: This is the implementation of the principal "Practical, not perfect." If this is missing, you could have run-away costs, poor performance, or a model that doesn't work sufficiently and is misleading.
Note that after we calculate error on batch, we can either keep going or we can evaluate the model.
Evaluating the model needs to happen on full dataset, not just a small batch!



TIP: If you have one pool of data, then you will need training data and validation data. You can't use it all in both places or you won't get measurable error.
Training and evaluating an ML model is an experiment with finding the right generalizable model that fits your training dataset but doesn't memorize it. As you see here, we have an overly simplistic linear model that doesn't fit the relationships in the data. You'll be able to see how bad this is immediately by looking at your loss metric during training (and visually on this graph here as there are quite a few points outside the shape of trend line). This is called underfitting.

On the opposite end of the spectrum is overfitting, as shown on the right extreme. Here we greatly increased the complexity of our linear model and turned it into an n-th order polynomial which seems to model the training dataset really well -- almost too well. This is where the evaluation dataset comes in -- you can use the evaluation dataset to determine if the model parameters are leading to overfitting. Overfitting or memorizing your training dataset can be far worse than having a model that only adequately fits your data.

Somewhere in between an underfit where the loss metric is not low enough and an overfit whether the model doesn't generalize is the right level of model complexity.

In addition to helping you choose between two completely different ML models (like regression or neural networks), you can also use your validation dataset to help fine-tune the hyperparameters of a single model which, if you recall, are set before training. This tuning process is accomplished through successive training runs and

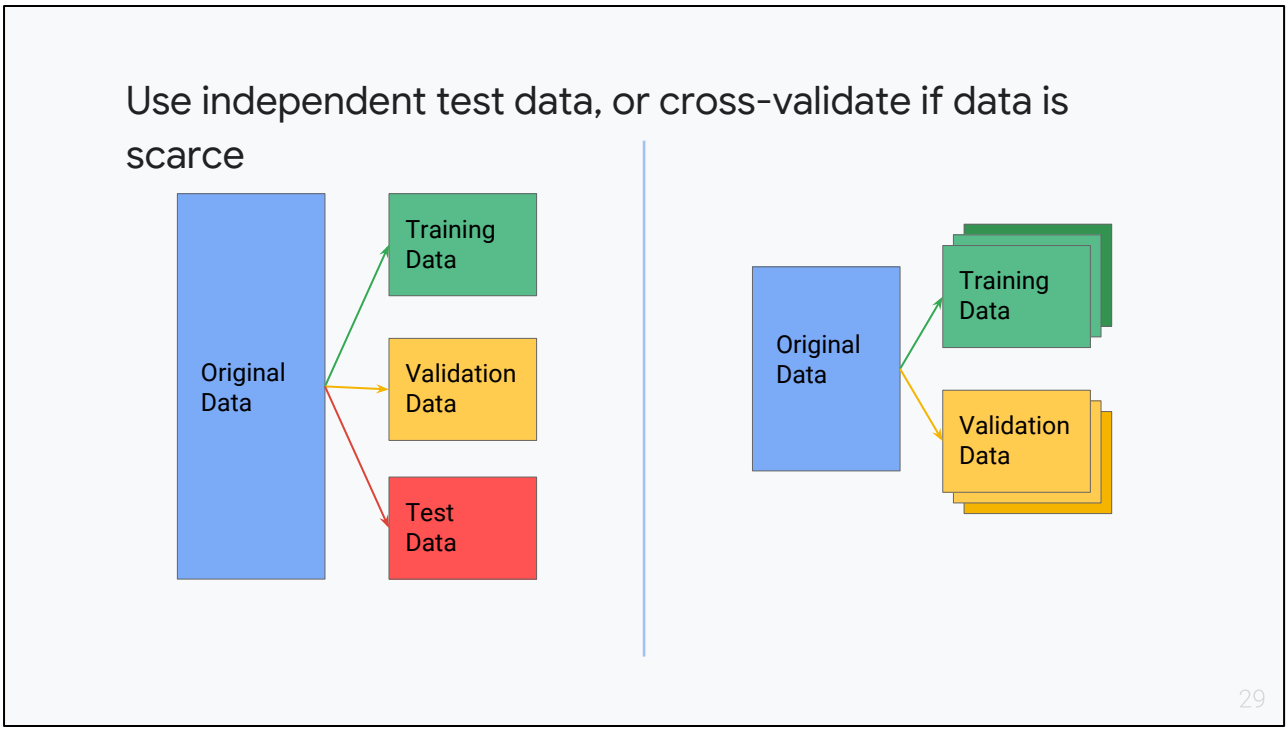
comparisons against your validation dataset to check for overfitting.

So here's how your validation dataset will actually be used after your model training. As you saw when we covered Optimization, training the model is where we start with random weights, calculate a loss metric on a batch of data, adjust the weights to minimize the loss metric, and repeat.

Periodically, we want to assess the performance of our model against data it has not yet seen in training which is where we use our validation dataset after a completed training run with the initial hyperparameters we set.

If there is not significant divergence between the loss metrics from the training run and the loss metric for the validation dataset then the model could still be optimized and tuned. We can go back to our hyperparameters that are set before training happens and try another training model run.
Perhaps we add one more layer to our neural network.

You can use a loop similar to this to also figure out model parameters like what we just did for hyperparameters, for example the number of layers or nodes to use in a neural network. Essentially, you will train with one configuration and evaluate on the validation dataset. Then, you will try out a different configuration that has more or fewer nodes and evaluate on the validation dataset. You will choose the model configuration that results in the lower loss on the validation dataset, not the model configuration that results in lower loss on the training one.



TIP: Read these slides backwards. If the question says "data is scarce"... then you should be thinking "independent test data" or "cross-validate" are candidate answers. Be familiar with the various methods of cross validation, including training, validation and test, and cross validation.

Cannot just use the original validation dataset. That is no longer truly independent.
Cross-validation could be done in both cases on the ice cream cone example. If we do random sampling, it's just a matter of choosing a different seed. If we divide by days/month, it's just a matter of varying the cutoff day (it was the first 25 days in our example, but we could cycle through and take the middle 25 days). This also ensures that we don't throw away any data completely due to leakage.

Included in Option 2 is the idea of going ahead and using all of the data and allowing the passage of time to provide independent test data.

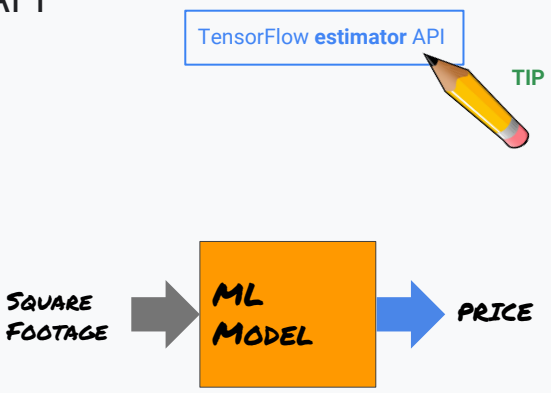
Working with estimator API

Set up machine learning model

1. Regression or classification?
2. What is the label?
3. What are the features?

Carry out ML steps

1. Train the model.
2. Evaluate the model.
3. Predict with the model.



TIP: Expect to know the basics of TensorFlow and key methods. They are covered in the Data Engineering courses.

Two parts to it. One is static -- how to set the ML problem up.

The second is the ML steps you carry out.

Imagine that you want to create a ML model to predict cost of a house given the sq footage. Answer the first three questions.

Quickly and inexpensively develop an application that sorts product reviews by most favorable to least favorable.

A. Train an entity classification model with TensorFlow. Deploy the model using Cloud Machine Learning Engine. Use the entity to sort the reviews.
B. Build an application that performs entity analysis using the Natural Language API. Use the entity to sort the reviews.
C. Build an application that performs sentiment analysis using the Natural Language API. Use the score and magnitude to sort the reviews.
D. Train a sentiment regression model with TensorFlow. Deploy the model using Cloud Machine Learning Engine. Use the magnitude to sort the reviews.

Quickly and inexpensively develop an application that sorts product reviews by most favorable to least favorable.

A. Train an entity classification model with TensorFlow. Deploy the model using Cloud Machine Learning Engine. Use the entity to sort the reviews.
B. Build an application that performs entity analysis using the Natural Language API. Use the entity to sort the reviews.
C. Build an application that performs sentiment analysis using the Natural Language API. Use the score and magnitude to sort the reviews. ✓
D. Train a sentiment regression model with TensorFlow. Deploy the model using Cloud Machine Learning Engine. Use the magnitude to sort the reviews.

Solution

C is correct. Use a pre-trained model whenever possible. In this case the Natural Language API with sentiment analysis returns score and magnitude of sentiment.

A and B are incorrect because they require creating a model instead of using an existing one.

D is incorrect because entity analysis will not determine sentiment: it recognizes objects, not opinions.

https://cloud.google.com/natural-language/docs/sentiment-tutorial
https://cloud.google.com/natural-language/docs/analyzing-entities
https://cloud.google.com/natural-language/docs/analyzing-sentiment

Maximize speed and minimize cost of deploying a TensorFlow machine-learning model on GCP.

A. Export your trained model to a SavedModel format. Deploy and run your model on Cloud ML Engine.
B. Export your trained model to a SavedModel format. Deploy and run your model from a Kubernetes Engine cluster.
C. Export 2 copies of your trained model to a SavedModel format. Store artifacts in Cloud Storage. Run 1 version on CPUs and another version on GPUs.
D. Export 2 copies of your trained model to a SavedModel format. Store artifacts in Cloud ML Engine. Run 1 version on CPUs and another version on GPUs.

DEPE

Maximize speed and minimize cost of deploying a TensorFlow machine-learning model on GCP.

A. Export your trained model to a SavedModel format. Deploy and run your model on Cloud ML Engine. ✓
B. Export your trained model to a SavedModel format. Deploy and run your model from a Kubernetes Engine cluster.
C. Export 2 copies of your trained model to a SavedModel format. Store artifacts in Cloud Storage. Run 1 version on CPUs and another version on GPUs.
D. Export 2 copies of your trained model to a SavedModel format. Store artifacts in Cloud ML Engine. Run 1 version on CPUs and another version on GPUs.

DEPE

Solution

A is correct because of Google recommended practices; that is, "just deploy it."

B is not correct because you should not run your model from Kubernetes Engine.

C and D are not correct because you should not export 2 copies of your trained model, etc., for this scenario.

https://cloud.google.com/ml-engine/docs/deploying-models
https://cloud.google.com/ml-engine/docs/prediction-overview

DEPE

Operationalizing Machine Learning Models

Exam Guide Review

There are many "Machine Learning Algorithm Cheat Sheets". Here is one:
https://github.com/soulmachine/machine-learning-cheat-sheet/blob/master/machine-learning-cheat-sheet.pdf
Google Machine Learning uses relatively general algorithms that are applicable to a wide variety of datasets, to simplify algorithm selection.

Linear algorithms:
https://towardsdatascience.com/introduction-to-machine-learning-algorithms-linear-regression-14c4e325882a
DNN (Deep Neural Network) algorithms:
https://en.wikipedia.org/wiki/Deep_learning#Deep_neural_networks
Classifier: The goal of a classifier is to identify a discrete category or class. Unordered, discrete, label or category.
Regressor: The goal of a regressor is to identify a value or numerical representation. Ordered, continuous, variable.
https://math.stackexchange.com/questions/141381/regression-vs-classification

Debugging a model: Clean datasets are consistent. Messy datasets are complicated in their own unique ways.

Using Pre-built Machine Learning Models

Leveraging pre-built ML as a service.	ML APIs Customizing ML APIs Conversational experiences

Tip: There are a few elements here. First, is building systems that use these services. Second is using additional services to augment, improve, or enhance the base functionality.

Sometimes called Machine Learning application "building blocks".

Study these:

- Cloud Vision API
- Cloud Text-to-speech API
- Cloud Speech-to-text API
- Cloud AutoML Vision
- Cloud AutoML Natural Language
- Cloud AutoML Translation
- Dialogflow

Deploying Existing Machine Learning Models

Deploying an ML pipeline.	Ingesting appropriate data Retraining of machine learning models Continuous evaluation

39

Tip: You need to know how to deploy existing models to Cloud ML Engine and to maintain them which might involve retraining.

Tip: Continuous evaluation… that is setting up continuous evaluation of the Machine Learning model so that steps can be taken to improve it.

Study these:

- Kubeflow
- Cloud Machine Learning Engine
- Spark ML
- BigQuery ML

Machine Learning Infrastructure

Choosing the appropriate training and serving infrastructure.	Distributed versus single machine Use of edge compute Hardware accelerators

Tip: Edge computing is the design of distributing processing in a strategic way so that model processing is pushed closer to the inputs. For example, in IoT, doing Machine Learning processing closer to the IoT sensors, by performing work in nearby datacenters or regions, is 'edge computing'.

Study these:

- GPU
- TPU

Maintaining Machine Learning

Measuring, monitoring, and troubleshooting machine learning models.	Machine learning terminology Impact of dependencies Common sources of error

Tip: One common source of error is accidental inclusion of biased data in the data being used for model training or validation.

Do you know these terms in a Machine Learning context?

- Features
- Labels
- Models
- Regression
- Classification
- Recommendation
- Supervised and unsupervised learning
- Evaluation
- Metrics
- Assumptions about data

Data Engineer Case Study 02:
A customer had this interesting business requirement...

Capture data reading and updates events to know who, what, when, and where.

Separation of who manages the data and who can read the data.

Allocate costs appropriately; costs to read/process vs. costs to store.

Prevent exfiltration of data to other GCP projects and to external systems.

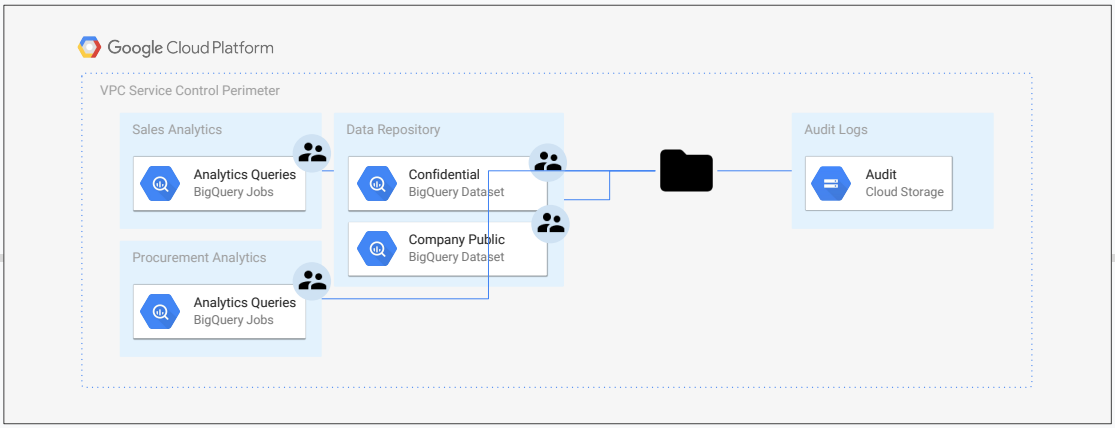
We mapped that to technical requirements like this...

All access to data should be captured in audit logs.

All access to data should be managed via IAM.

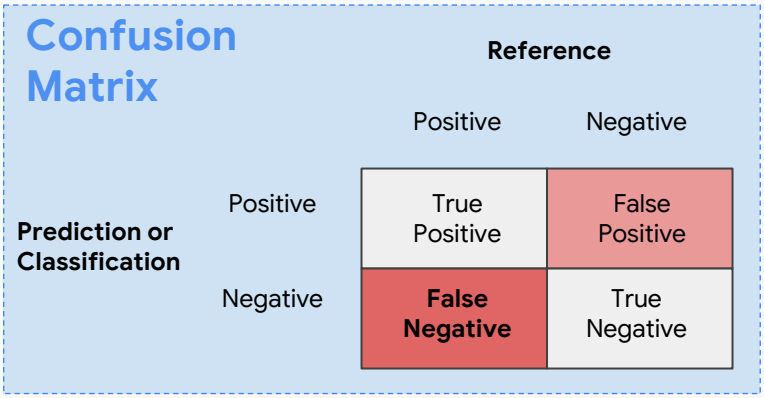
Configure service perimeters with VPC service controls.

And this is how we implemented that technical requirement.



Modeling business processes for analysis and optimization

What happens if your model is wrong?



Recall = True Positive rate

Accuracy = Fraction correct
Accuracy fails if dataset is unbalanced

51

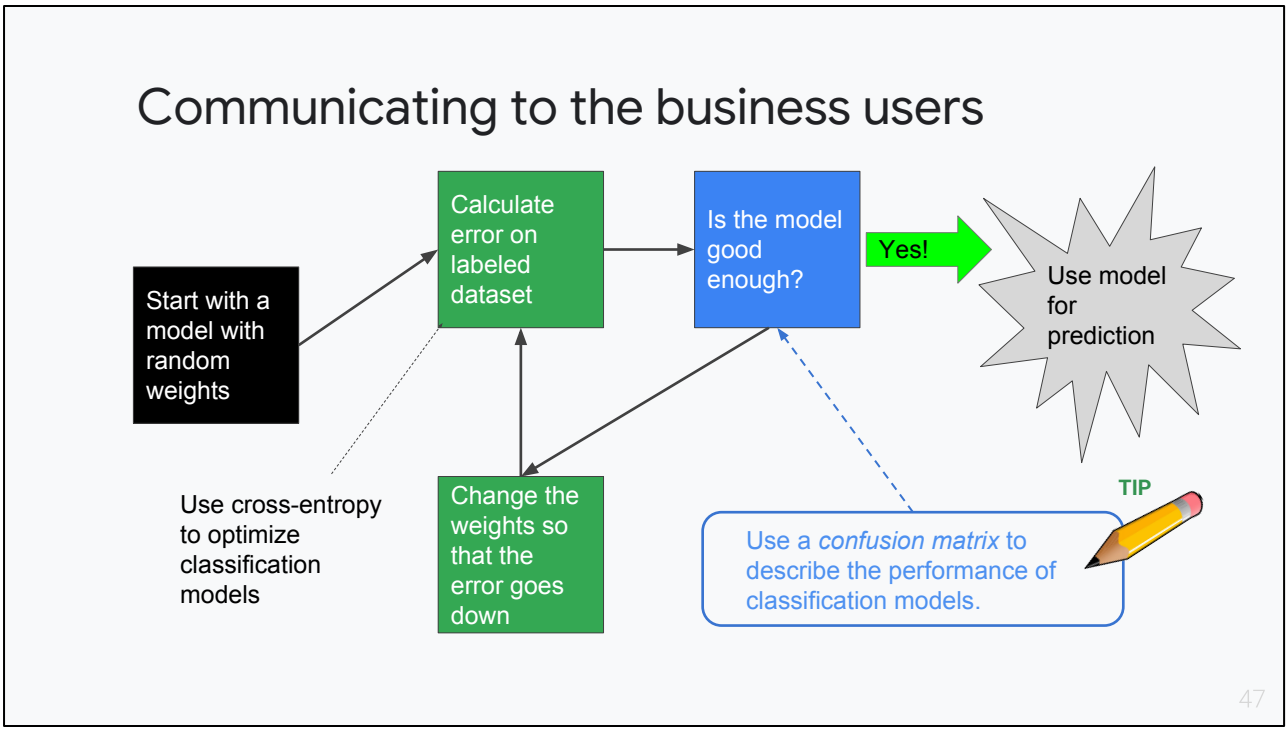
TIP: There are two kinds of mistakes your model can make. The consequences are not the same.

Example. Imagine you are predicting a dangerous condition in an automobile part. Positive means that the part is hazardous and dangerous. Negative means the part is safe. A False Positive means that your model predicted that the part was dangerous when it wasn't. So you removed a part that you didn't need to eliminate. A False Negative means that your model predicted that the part was safe when it was actually dangerous. So the part was used in an automobile because it was thought to be safe, and as a result, accidents occurred.

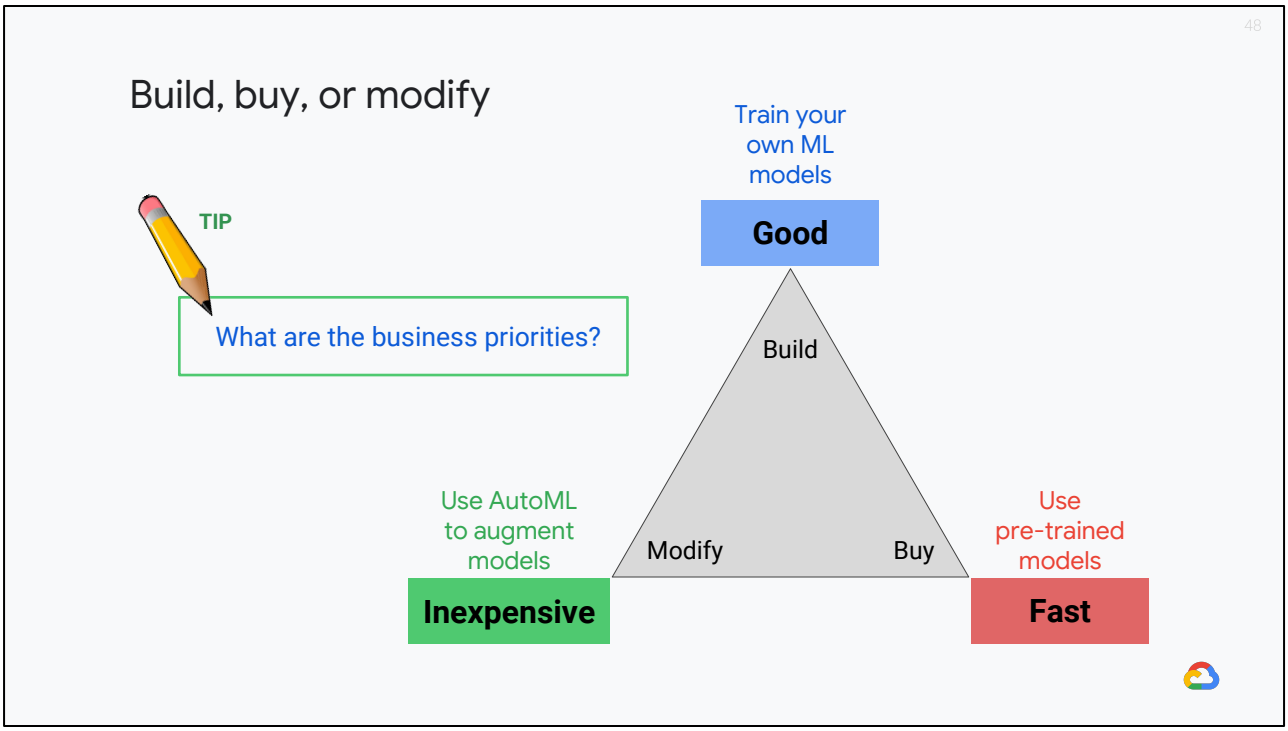
The logic could be reversed. If you were predicting that a part was safe, the False Positive would be that it is actually dangerous. So you have to think through this kind of logic problem to understand what business decision, procedure, or action should be taken as a result of the ML model.

Confusion Matrix: https://en.wikipedia.org/wiki/Confusion_matrix
Severity of ERROR

	POSITIVE	NEGATIVE
TRUE	TRUE POSITIVE	TRUE NEGATIVE
FALSE	FALSE POSITIVE	FALSE NEGATIVE



TIP: You may have to plan to explain your reasoning and design. In this example, a confusion matrix is used to make the data engineering choices understandable to the business users.



TIP: What is the priority or the blend of priorities -- Good, Fast, Cheap? And what are the qualities, time, budget by which those priorities will be measured?

Good-Fast-Cheap is one way to think about the decision to a) build a solution from scratch yourself, b) purchase a solution off-the-shelf or from a vendor, or c) customize existing products and services to trade-off speed for features.

"Good" in this case means control. If you build it yourself you get absolute control over what the solution does and how it works. But DIY (do-it-yourself) is rarely the fastest way to get a solution and it certainly is not cheap.

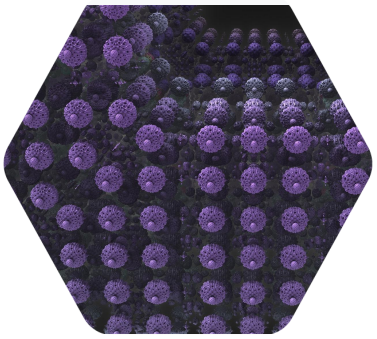
"Fast" means outsourcing. If you go to the experts who already have handled similar problems and have experience creating and implementing similar solutions, it is often the fastest way to get the matter resolved. But -- you have to give up control. And it might not be everything you had hoped for -- because you are basically giving the contractor/vendor the right to make tradeoffs for you. When you are short on time -- and the problem is imminent -- sometimes this is the most expedient way forwards.

"Cheap" means adapting what you already have (which might be partially depreciated, or already paid for). You might not get all the features you want. In some cases you might have to make up for deficiencies in the solution with manual methods -- transforming output of one part to provide input to another part through manual processes, for example.

Ways to build effective ML

Big Data

Feature Engineering



Model Architectures

TIP: One of the themes in ML is to start simply and build to production. Shown is the general progression of building an ML solution; start with Big Data. Go through Feature Engineering. Then create the Model and deploy it.

https://pixabay.com/en/large-data-dataset-word-895563/ (cc0)
https://pixabay.com/en/fractal-complexity-render-3d-1232494/ (cc0)
https://pixabay.com/en/robot-artificial-intelligence-woman-507811/ (cc0)

To read sharded CSV files, create a TextLineDataset that has a function to decode the CSV into features, labels

```
CSV_COLUMNS =
['amount','pickuplon','pickuplat','dropofflon','dropofflat','passengers']
LABEL_COLUMN = 'amount'
DEFAULTS = [[0.0], [-74.0], [40.0], [-74.0], [40.7], [1.0]]

def read_dataset(filename, mode, batch_size=512):
    def decode_csv(value_column):
      columns = tf.decode_csv(value_column, record_defaults=DEFAULTS)
      features = dict(zip(CSV_COLUMNS, columns))
      label = features.pop(LABEL_COLUMN)
      return features, label

    dataset = tf.data.TextLineDataset(filename).map(decode_csv)

    ...
    return ...
```

Consider using data in place instead of ETL.

TIP: As in this example, sometimes it is most efficient to use the data in its current location and format rather than ingesting and ETL.

Code for reading a CSV file
Directly supports a list of filenames if you have sharded files

Repeat the data and send it along in chunks

```
def read_dataset(filename, mode, batch_size=512):
    ...

    dataset = tf.data.TextLineDataset(filename).map(decode_csv)
    if mode == tf.estimator.ModeKeys.TRAIN:
        num_epochs = None # indefinitely
    else:
        num_epochs = 1 # end-of-input after this
    dataset = dataset.repeat(num_epochs).batch(batch_size)

    return dataset.make_one_shot_iterator().get_next()
```

Grouping the work can be efficient and give additional control over the processing of the data.

We need to make our ML pipeline more robust

Estimator examples:

1. Ran the training_op for num_steps or num_epochs iterations.
2. Saved checkpoints during training.
3. Used final checkpoint as model.

Realistic, real-world ML models need:

1. Use a fault-tolerant distributed training framework.
2. Choose model based on validation dataset.
3. Monitor training, especially if it will take days.
4. Resume training if necessary.

52

TIP: Don't get confused between initial proof-of-concept and production activities. In general, the cases will refer to real-world situations.

https://pixabay.com/en/hang-out-plush-toys-kermit-1521663/ (cc0)
https://pixabay.com/en/london-england-hdr-boats-ships-123778/ (cc0)

Our tf.learn examples have been a toy and is okay if the only thing we are going to be handling are plush-toys. We can't just hang a clothesline and call it a bridge. In the real world, we need to make things more robust.

Monitoring: Experiment will export summaries, so that we can see them in TensorBoard.

Feature engineering is a complicated process

Choosing good features

1. Is the feature you are considering related to the result you are trying to predict?

2. Is the predictive value known?

3. Is the feature a numeric value with meaningful magnitude?

4. Are there enough examples?

Feature engineering process

Pre-processing
Feature creation
Hyperparameter tuning

Good features bring human insight to a problem

Other important concepts and considerations

Feature crosses

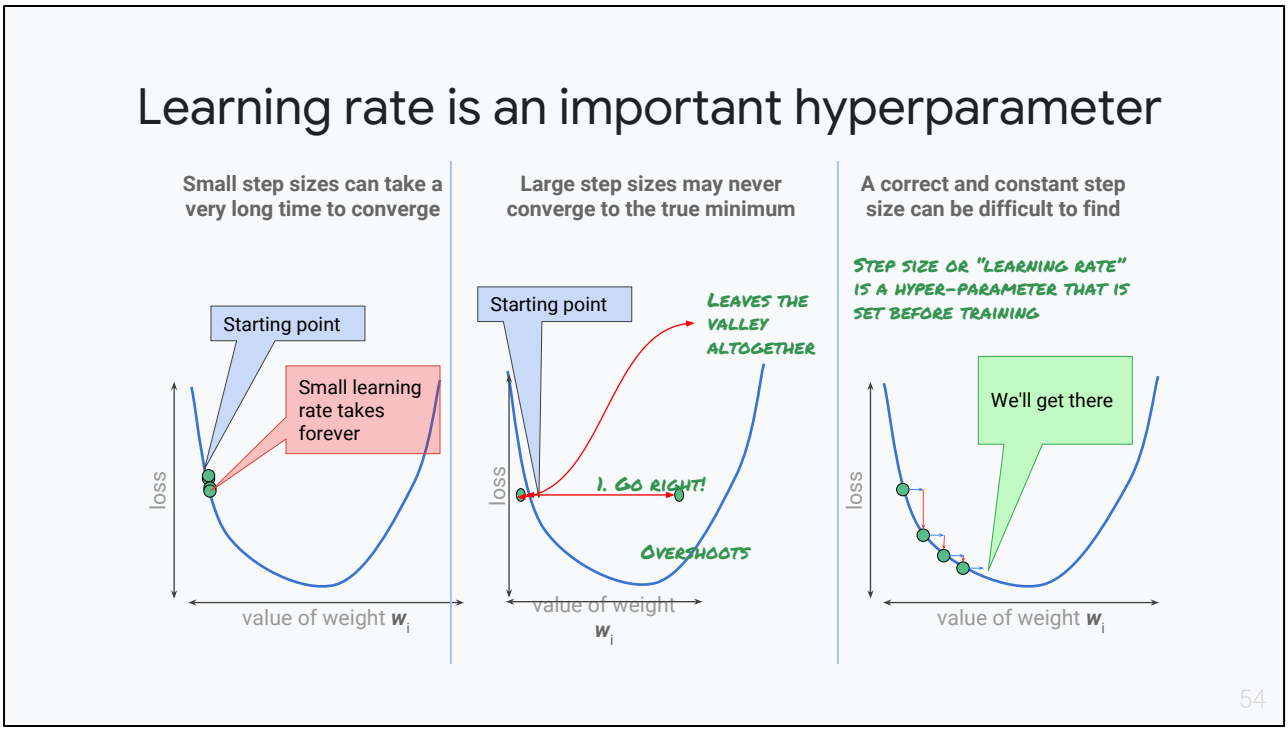
Discretize floats that are not meaningful

Bucketize features

Dense and sparse features

DNNs for dense, highly correlated
Linear for sparse, independent

https://pixabay.com/en/book-glasses-read-education-1176256/



TIP: Hyperparameters can determine whether your model converges on the truth quickly or not at all!

Small step sizes: Putting aside direction for the moment, if your step size is too small, your training might take forever. You are guaranteed to find the minimum though (so long as there is only one minimum like this linear regression loss curve here -- remember our two minimum loss surface we just showed previously? We'll cover how to deal with those efficiently later).

Larger step sizes: If your step size is too big, you might either bounce from wall to wall or bounce out of your valley entirely, and into an entirely new part of the losos surface. Because of this, when the step size is too big, the process is not guaranteed to converge.

Correct step size: If your step size is just right …. Well, then, you're set. But whatever this value is, it's unlikely to be just as good on a different problem.

BigQuery performance

Input data and data sources (I/O): How many bytes does your query read?

Communication between nodes (shuffling): How many bytes does your query pass to the next stage? How many bytes does your query pass to each slot?

Computation: How much CPU work does your query require?

Outputs (materialization): How many bytes does your query write?

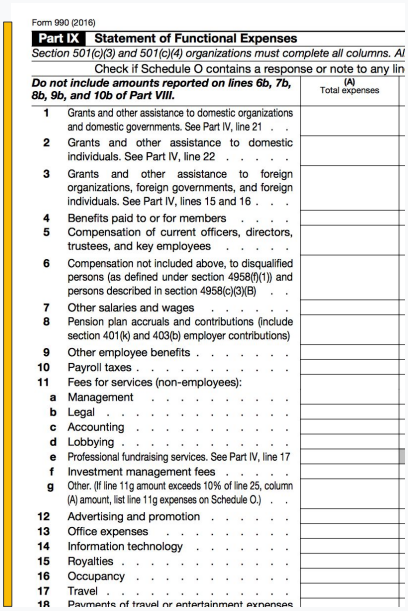
Query anti-patterns: Are your queries following SQL best practices?

Performance is critical to practical solutions.

https://cloud.google.com/bigquery/docs/best-practices-performance-overview

Tax schema from the perspective of a data architect

Each of these data fields needs to be stored in a structured way.



Form 990 (2016)

Part IX **Statement of Functional Expenses**

Section 501(c)(3) and 501(c)(4) organizations must complete all columns. Al

Check if Schedule O contains a response or note to any lin

	Do not include amounts reported on lines 6b, 7b, 8b, 9b, and 10b of Part VIII.	(A) Total expenses
1	Grants and other assistance to domestic organizations and domestic governments. See Part IV, line 21	
2	Grants and other assistance to domestic individuals. See Part IV, line 22	
3	Grants and other assistance to foreign organizations, foreign governments, and foreign individuals. See Part IV, lines 15 and 16	
4	Benefits paid to or for members	
5	Compensation of current officers, directors, trustees, and key employees	
6	Compensation not included above, to disqualified persons (as defined under section 4958(f)(1)) and persons described in section 4958(c)(3)(B)	
7	Other salaries and wages	
8	Pension plan accruals and contributions (include section 401(k) and 403(b) employer contributions)	
9	Other employee benefits	
10	Payroll taxes	
11	Fees for services (non-employees):	
a	Management	
b	Legal	
c	Accounting	
d	Lobbying	
e	Professional fundraising services. See Part IV, line 17	
f	Investment management fees	
g	Other. (If line 11g amount exceeds 10% of line 25, column (A) amount, list line 11g expenses on Schedule O.)	
12	Advertising and promotion	
13	Office expenses	
14	Information technology	
15	Royalties	
16	Occupancy	
17	Travel	
18	Payments of travel or entertainment expenses	

Page 10 of the IRS Form 990 lists 24 different expense types

What happens if I need to add another expense type? I need to change the schema and provide NULLs historically? Ugh..

Adding each expense field as a New Column

Table Details: irs_990_2015

Schema | Details | Preview

/eebenef	payrolltx	feesforsrvcmgmt	legalfees	accntingfees	feesforsrvclobby	profndraising	feesforsrvcinvstmgmt	feesforsrvcothr	advrtpromo	officexpns	infotech	royaltsexpns	occupancy	travel
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
816623	847695	0	0	28654	0	0	0	27770	0	155715	43796	0	1156758	0
524396	539651	127071	34165	44264	0	0	0	0	567392	732920	875416	0	887599	33446
177305	209707	0	120	22000	0	0	0	11551	0	165306	11391	0	231092	0
1289799	543608	7415	14888	33514	0	0	0	0	1273856	2101383	217216	0	604569	40173
512540	170264	0	24000	64500	0	0	0	96660	344208	2128823	0	0	540746	0
217097	115324	0	0	0	0	0	0	0	0	0	0	0	0	0

… results in a really WIDE table that is **not scalable**...

Page 10 of the IRS Form 990 lists 24 different expense types as of 2017

What happens if I need to add another expense type? I need to change the schema and provide NULLs historically? Ugh..

Break out expenses into another Lookup Table

Organization Details

Company ID	Company Name
161218560	NY Association Inc.

Historical Transactions

Company ID	Expense Code	Amount
161218560	1	$10,000

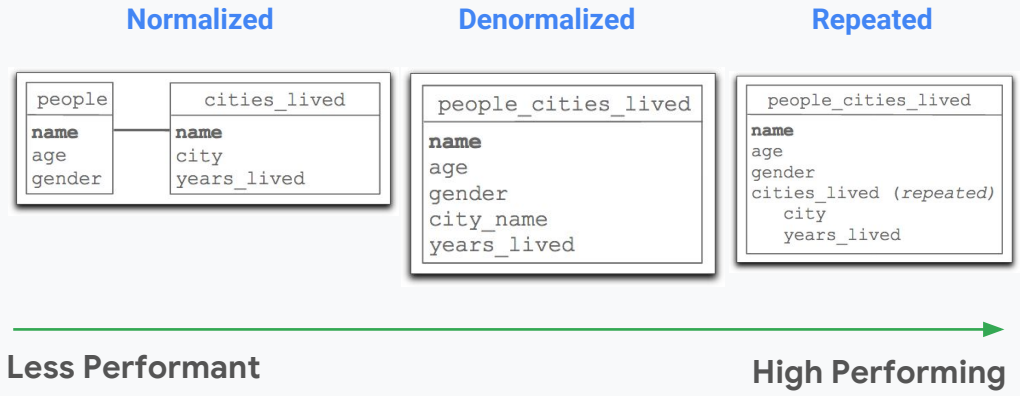
Code Lookup Tables

Expense Code	Expense Type
1	Lobbying
2	Legal
3	Insurance

… this breaking-apart process is called **normalization** ...

This process is called normalization

BigQuery architecture introduces repeated fields



Normalization represent relations between tables.
Denormalized data presents information in a flat format.
Repeated fields enables related data to be handled in a loop, making it more efficient to process.

Normalize for efficiency

Original data

ID	Name	Transaction		
1	Bob			
2	Doug			
		OrderID	Date	Quantity
		11537	10/29/2018	11
		78244	11/05/2018	14
		32367	09/09/2018	18
3	Tom			
		OrderID	Date	Quantity
		13323	04/09/2018	6
		45452	11/10/2018	13
		17671	08/20/2018	15

Normalized data

Customer Table

ID	Name
1	Bob
2	Doug
3	Tom

Transaction Table

ID	OrderID	Date	Quantity
2	11537	10/29/2018	11
2	78244	11/05/2018	14
2	32367	09/09/2018	18
3	13323	04/09/2018	6
3	45452	11/10/2018	13
3	17671	08/20/2018	15

The trade-off is performance versus efficiency. Normalized is more efficient. Denormalized is more performant.

The original data is organized visually, but if you had to write an algorithm to process the data, how might you approach it? Could be by rows, by columns, by rows-then-fields. And the different approaches would perform differently based on the query. Also, your method might not be parallelizable.

The original data can be interpreted and stored in many ways in a database. Normalizing the data means turning it into a relational system. This stores the data efficiently and makes query processing a clear and direct task. Normalizing increases the orderliness of the data.

Denormalize for performance

Normalized data

Customer Table

ID	Name
1	Bob
2	Doug
3	Tom

Transaction Table

ID	OrderID	Date	Quantity
2	11537	10/29/2018	11
2	78244	11/05/2018	14
2	32367	09/09/2018	18
3	13323	04/09/2018	6
3	45452	11/10/2018	13
3	17671	08/20/2018	15

Denormalized data

Denormalized Table

ID	Name	OrderID	Date	Quantity
2	Doug	11537	10/29/2018	11
2	Doug	78244	11/05/2018	14
2	Doug	32367	09/09/2018	18
3	Tom	13323	04/09/2018	6
3	Tom	45452	11/10/2018	13
3	Tom	17671	08/20/2018	15

REPEATED FIELD

61

Denormalizing is the strategy of accepting repeated fields in the data to gain processing performance.

Data must first be normalized before it can be denormalized. Denormalization is another increase in the orderliness of the data. Because of the repeated fields (in the example, the Name field is repeated), the denormalized form takes more storage. However, because it is no longer relational, queries can be processed more efficiently and in parallel using columnar processing.

BigQuery can use nested schemas for highly scalable queries

Organization Details with Nested Historical Transactions

	Company ID	Company Name	Transactions.Amount	Code.Expense
	161218560	NY Association Inc.	$10.000	Lobbying
NESTED			$5,000	Legal
			$1,000	Insurance
	123435560	ACME Co.	$7,000	Travel

62

https://discourse.looker.com/t/why-nesting-is-so-cool/4182

- Avoids joins by already having the data matched in a single table, just UNNEST it
- Scans on Unnested data still fast COUNT(*) on Company ID

Four key elements of work

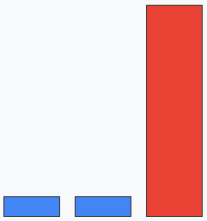
- **I/O:** How many bytes did you read?
- **Shuffle:** How many bytes did you pass to the next stage?
  - Grouping: How many bytes do you pass to each group?
- **Materialization:** How many bytes did you write to storage?
- **CPU work:** User-defined functions (UDFs), functions

Avoid input/output wastefulness

- Do not SELECT *, use only the columns you need
- Filter using WHERE as early as possible in your queries

Image (filter) cc0: https://unsplash.com/search/photos/filter?photo=1pZbNwlGzNY

Shuffle wisely: Be aware of data skew in your dataset



Skewed data creates an imbalance between BigQuery worker slots (uneven data partition sizes)

- **Filter your dataset** as early as possible (this avoids overloading workers on JOINs).
- Hint: Use the Query Explanation map and compare the Max vs. the Avg times to highlight skew.
- BigQuery will automatically attempt to reshuffle workers that are overloaded with data.

https://cloud.google.com/bigquery/docs/best-practices-performance-patterns

Walk through example using the Query Explanation map:
https://cloud.google.com/bigquery/query-plan-explanation#data_skew_in_stage_1

Careful use of GROUP BY

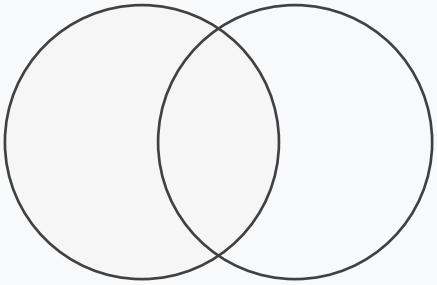
- Best when the number of distinct groups is small (fewer shuffles of data).
- Grouping by a high-cardinality unique ID is a bad idea.

Row	contributor_id	LogEdits
1	2221364	4
2	104574	4
3	73576	4
4	311307	4
5	291919	4
6	140178	4
7	181636	4
8	3661553	4
9	3600820	4
10	4737290	4
11	938404	4
12	295955	4
13	183812	4
14	1811786	4
15	8918196	4
16	561624	4
17	5338406	4

← Do not Group on an ID

Joins and unions

- Know your join keys and whether they're unique: no accidental cross joins.
- LIMIT Wildcard UNIONS with _TABLE_SUFFIX filter.



Limit UDFs to reduce computational load

- Use native SQL functions whenever possible.
- Concurrent rate limits:
  - for non-UDF queries: 50
  - for UDF-queries: **6**

```
CREATE TEMP FUNCTION SumFieldsNamedFoo(json_row STRING)
  RETURNS FLOAT64
  LANGUAGE js AS """
function SumFoo(obj) {
  var sum = 0;
  for (var field in obj) {
    if (obj.hasOwnProperty(field) && obj[field] != null) {
      if (typeof obj[field] == "object") {
        sum += SumFoo(obj[field]);
      } else if (field == "foo") {
        sum += obj[field];
      }
    }
  }
  return sum;
}
var row = JSON.parse(json_row);
return SumFoo(row);
""";
```

Diagnose performance issues with Query Explanation map

Diagnose performance issues with the Query Explanation map

The following ratios are also available for each stage in the query plan.

API JSON Name	Web UI*	Ratio Numerator **
`waitRatioAvg`	AVG	Time the average worker spent waiting to be scheduled.
`waitRatioMax`	MAX	Time the slowest worker spent waiting to be scheduled.
`readRatioAvg`	AVG	Time the average worker spent reading input data.
`readRatioMax`	MAX	Time the slowest worker spent reading input data.
`computeRatioAvg`	AVG	Time the average worker spent CPU-bound.
`computeRatioMax`	MAX	Time the slowest worker spent CPU-bound.
`writeRatioAvg`	AVG	Time the average worker spent writing output data.
`writeRatioMax`	MAX	Time the slowest worker spent writing output data.

* The labels 'AVG' and 'MAX' are for illustration only and do not appear in the web UI.

** All of the ratios share a common denominator that represents the longest time spent by any worker in any segment.

Table partitioning

Time-partitioned tables are a cost-effective way to manage data.

Queries spanning time periods are straightforward to write.

When you create tables with time-based partitions, BigQuery automatically loads data in the correct partition.

To declare the table as partitioned at creation time, use this flag:

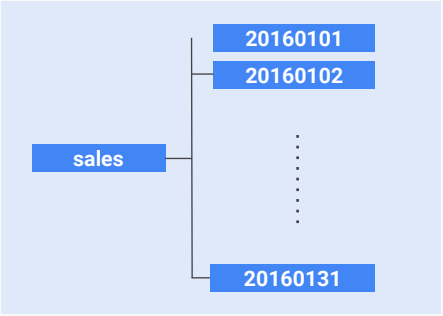
`--time_partitioning_type`

To create a partitioned table with expiration time for data, use this flag:

`--time_partitioning_expiration`

Example

```
SELECT ...
FROM `sales`
WHERE _PARTITIONTIME
BETWEEN TIMESTAMP("20160101")
    AND TIMESTAMP("20160131")
```



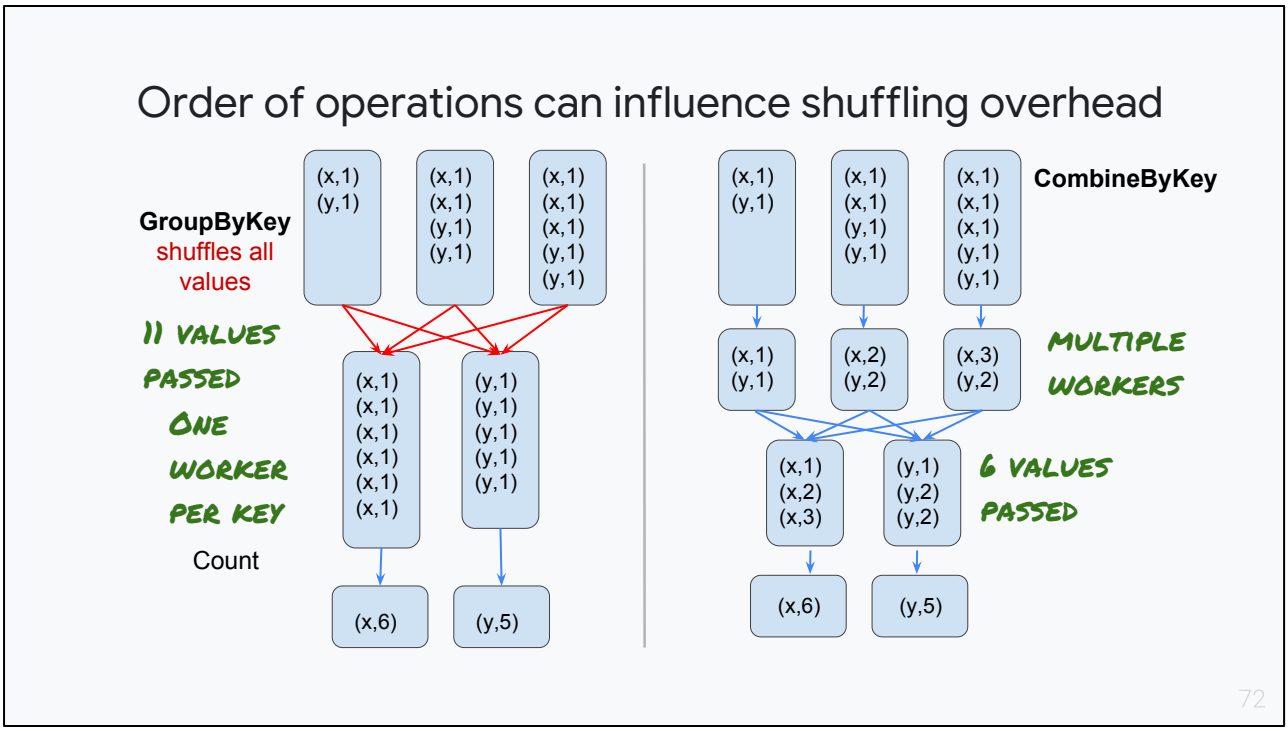
71

TIP: Partitioning by time is often a way to evenly distribute work and avoid hot-spots in processing data.

Partitioned tables include a pseudo column named _PARTITIONTIME that contains a date-based timestamp for data loaded into the table. The timestamp is based on UTC time and represents the number of microseconds since the unix epoch. For example, if data is appended to a table on April 15, 2016, all of the rows of data appended on that day contain the value TIMESTAMP("2016-04-15") in the _PARTITIONTIME column.

For information on table partitioning best practices, see: https://cloud.google.com/bigquery/docs/partitioned-tables#best_practices.

Comparing table partitioning in BigQuery with ML training and test data. Note that when you identify data for training ML, be careful to randomize rather than organize by time, because, you might train the model on the first part (summer) and test using the second part (winter).



The way that **GroupByKey** works, Dataflow can use no more than one worker per key. In this example, **GroupByKey** causes all the values to be shuffled so they are all transmitted over the network. And then there is one worker for the 'x' key and one worker for the 'y' key.

Combine allows Dataflow to distribute a key to multiple workers and process it in parallel. In this example, **CombineByKey** first aggregates values and then processes the aggregates with multiple workers. Also, only 6 aggregate values need to be passed over the network.

Combine is a Java interface that tells Dataflow that the combine operation (like Count) is both commutative and associative. This allows Dataflow to shard within a key vs. having to group each key first. As a developer, you can create your own custom **Combine** class for any operation that has commutative and associative properties.

Can also group by time (Windowing)

For batch inputs, explicitly emit a timestamp in your pipeline:

- Instead of `c.output()`

```
c.outputWithTimestamp(f, Instant.parse(fields[2]));
```

Then use windows to aggregate by time:

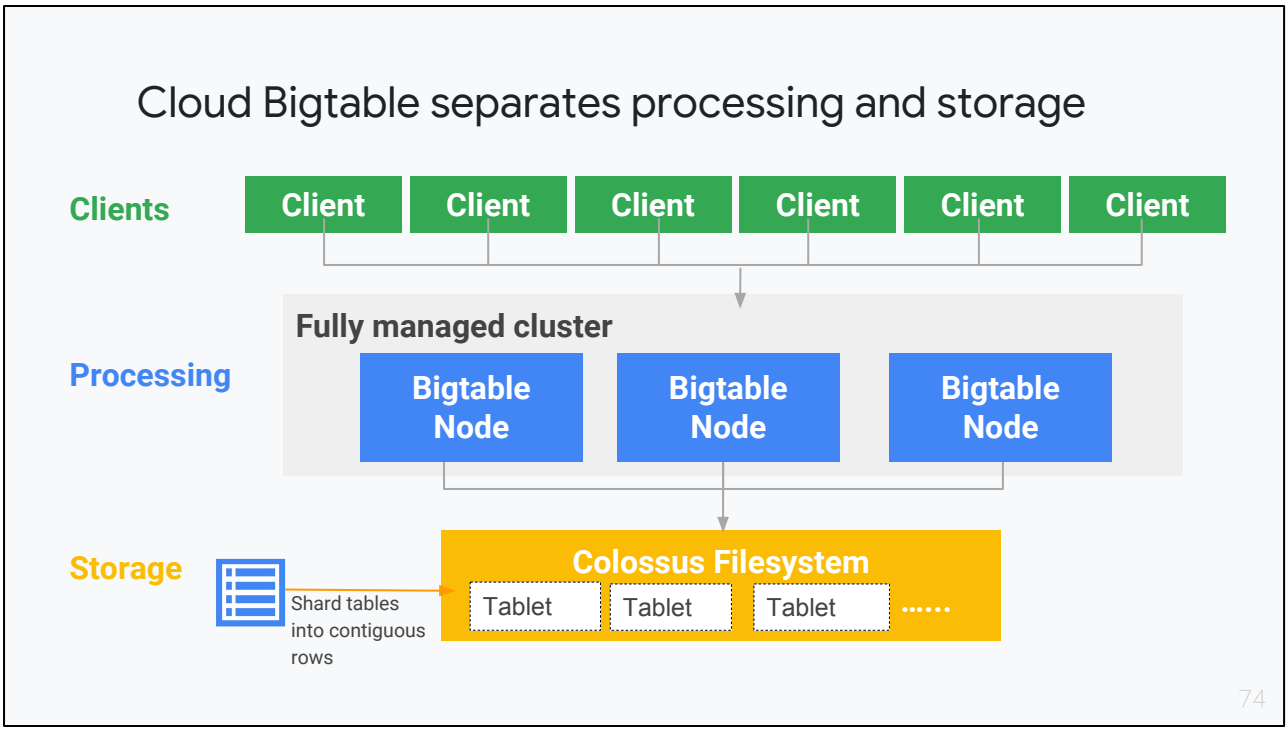
```
PCollection<KV<String, Integer>> scores = input
    .apply(Window.into(FixedWindows.of(Minutes(2)))
    .apply(Sum.integersPerKey());
```

SUBSEQUENT GROUPS, AGGREGATIONS, ETC. ARE COMPUTED ONLY WITHIN TIME WINDOW

73

TIP: Consider when data is unbounded / streaming, that using windows to divide the data into groups can make the processing of the data much more manageable. Of course, then you have to consider the size of the window and whether the windows are overlapping.

This timestamp will be the time at which the element was published to PubSub. If you want to use a custom timestamp, it must be published as a PubSub attribute, and you tell Dataflow about it using the `timestampLabel` setter.



Notes:

A Cloud Bigtable table is sharded into blocks of contiguous rows, called tablets, to help balance the workload of queries. (Tablets are similar to HBase regions.) Tablets are stored on Colossus, Google's file system, in SSTable format. An SSTable provides a persistent, ordered immutable map from keys to values, where both keys and values are arbitrary byte strings. Each tablet is associated with a specific Cloud Bigtable node.
Importantly, data is never stored in Cloud Bigtable nodes themselves; each node has pointers to a set of tablets that are stored on Colossus. As a result:
Rebalancing tablets from one node to another is very fast, because the actual data is not copied. Cloud Bigtable simply updates the pointers for each node.
Recovery from the failure of a Cloud Bigtable node is very fast, because only metadata needs to be migrated to the replacement node.
When a Cloud Bigtable node fails, no data is lost.

Bigtable has efficiency when it comes to fast streaming ingestion..just storing data….whereas bigquery is efficient for queries with SQL support

A table can have only one index (the row key)

	user_information				
Row Key	followers	birthdate	age	gender	messageCount
andrew	34	06_04_1986	34	F	1
brianna	31	07_24_1993	23	F	12
caitlyn	55	03_22_1952	51	F	15

ROWS ARE STORED IN ASCENDING ORDER OF THE ROW KEY

ENTRIES IN THE BIRTHDATE COLUMN CANNOT BE INDEXED

75

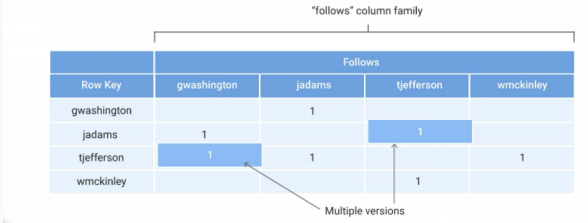
Notes:

Each table has only one index, the row key. There are no secondary indices.

Rows are sorted lexicographically by row key, from the lowest to the highest byte string. Row keys are sorted in big-endian, or network, byte order, the binary equivalent of alphabetical order. This is important to keep in mind given how tables are automatically sharded.

All operations are atomic at the row level. For example, if you update two rows in a table, it's possible that one row will be updated successfully and the other update will fail. Avoid schema designs that require atomicity across rows.

Wide Table



Sparse Table

Row Key	Column Data
gwashington	Follows:jadams:1
jadams	Follows:gwashington:1, Follows:tjefferson:1
tjefferson	Follows:gwashington:1, Follows:jadams:1, Follows:wmckinley:1
wmckinley	Follows:tjefferson:1

Row key design

GOOD:
highway-MILEMARKER-RTS
(I35-347-123456789)

MAKE QUERY PARAMETER THE ROW KEY
E.G. highway-MILEMARKER
(I35-347)

ADD REVERSE TIMESTAMP TO THE ROW KEY
E.G. highway-MILEMARKER-RTS
(I35-347-123456789)

BAD:
TS-highway-MILEMARKER
(20170531T102354-I35-347)
MILEMARKER-highway-RTS
(347-I35-1234567)

77

Notes:

Q1: assume that we want to show data on a traffic map. The most common query will involve highway and range of mile markers (e.g. I35 between exits 332 and 363).

Q2: If you store in original timestamp order, then new data will at the *bottom* of the table. We want table scans within a tablet to return the first few rows if possible. Hence, reverse timestamp.

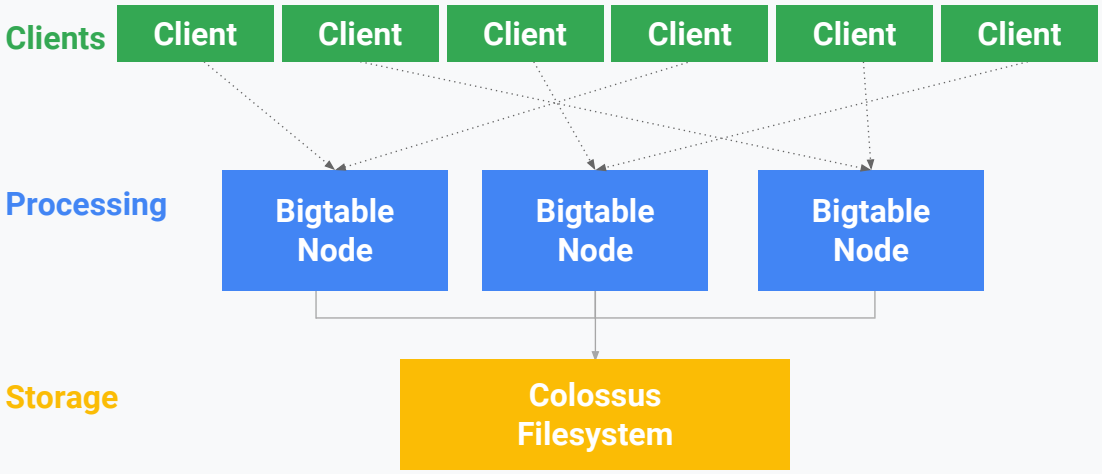
Q3: The problem with the first bad example is that writes won't be distributed. The problem with the second bad example is that the reads won't be of consecutive blocks. Remember that rows are stored consecutively. So, if all your keys at a certain time start with the same string, they will all hit the same tablet. That's the problem with starting with the timestamp (same problem occurs for reverse timestamp since the number of seconds left till 2028 is pretty much the same now as it is 10 minutes from now; it's just easier to see with real timestamp). Starting the key with the milemarker makes block reads hard, since queries will be for specific highways (think about an application that displays traffic on a map).

Years of engineering to...

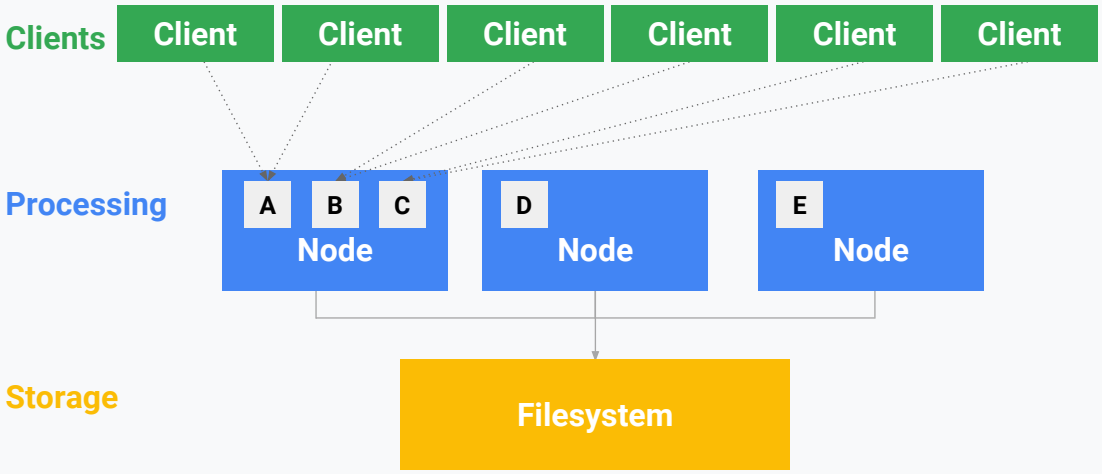
Cloud Bigtable

- Teach Cloud Bigtable to configure itself.
- Isolate performance from “noisy neighbors.”
- React automatically to new patterns, splitting, and balancing.

Cloud Bigtable looks at access patterns and improves itself



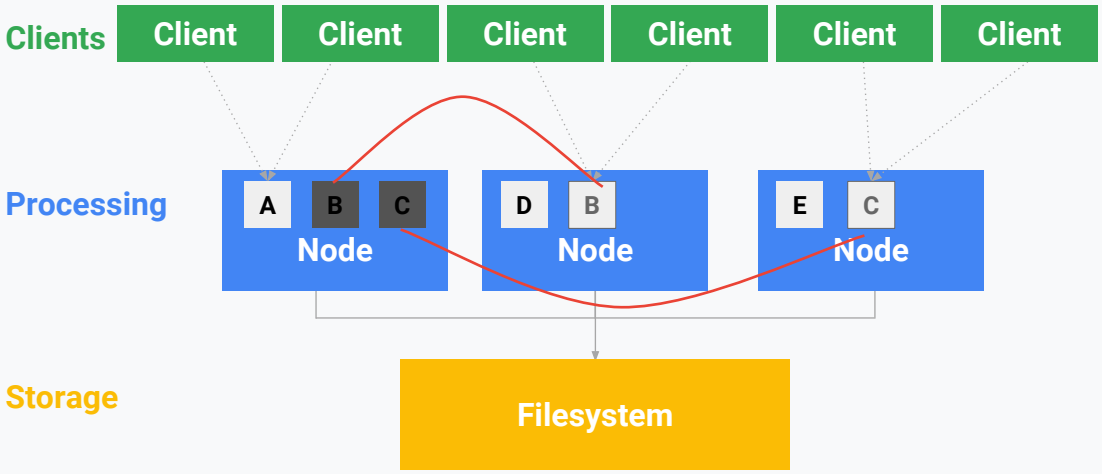
Cloud Bigtable learns access patterns...



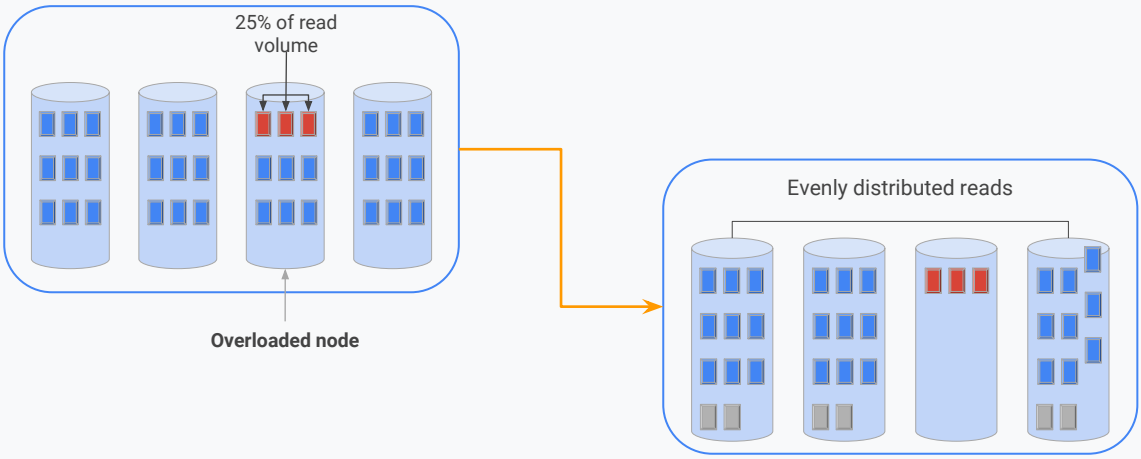
Notes:

A,B,C,D,E are not data but rather pointers/references and cache….which is why rebalancing is not time consuming...we're just moving pointers. Actual data is in tablets in Colossus FS.

...and rebalances data accordingly



Rebalance strategy: distribute reads



Growing a Cloud Bigtable cluster

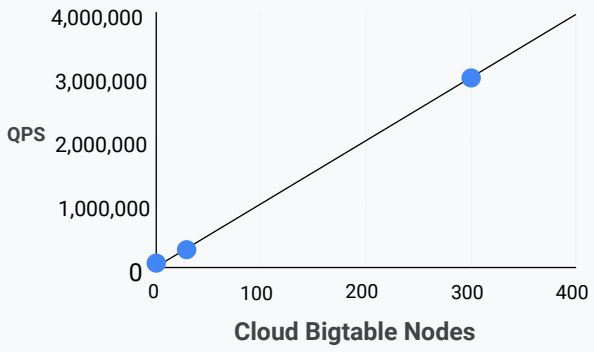
Make sure clients and Cloud Bigtable are in same zone.

Change schema to minimize data skew.

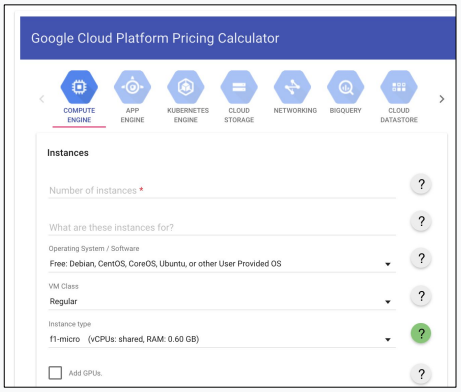
Performance increases linearly with the number of nodes.

Disk speed: SSD faster than HDD

Takes some time after scaling up nodes for performance improvement to be seen.



Pricing Calculator



TIP

Performance and cost are related.

https://cloud.google.com/products/calculator/

TIP: The pricing calculator can be used with BigQuery to estimate the cost of a query before you submit it.

Three categories of BigQuery pricing

Avoid SELECT *
Query only the columns that you need.

TIP

Storage

- Amount of data in table
- Ingest rate of streaming data
- Automatic discount for old data

Processing

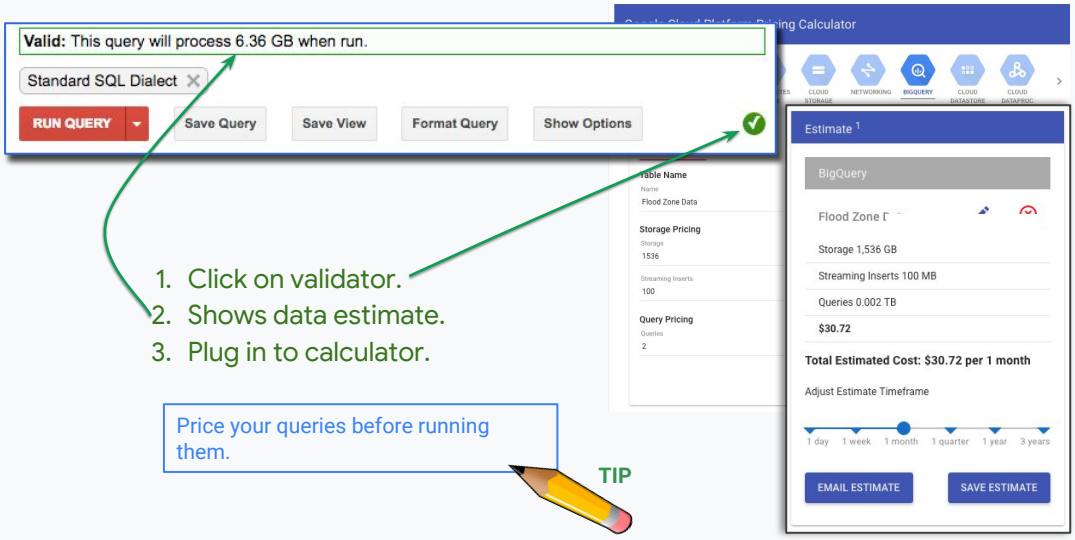
- On-demand OR Flat-rate plans
- On-demand based on amount of data processed
- 1 TB/month free
- Have to opt in to run high-compute queries

Free

- Loading
- Exporting
- Queries on metadata
- Cached queries
- Queries with errors

https://pixabay.com/en/storage-papers-office-cabinet-1209059/ (cc0)
https://pixabay.com/en/jet-engine-turbine-jet-airplane-371412/ (cc0)
https://pixabay.com/en/skydiving-jump-falling-parachuting-678168/ (cc0)

Use query validator with Pricing Calculator for estimates



https://cloud.google.com/products/calculator/
https://cloud.google.com/bigquery/docs/estimate-costs

Data Engineer Case Study 03:
A customer had this interesting business requirement...

The overall business requirement was to migrate to Cloud a on-premise reporting solution aimed to produce daily reports to regulators.

The on-premises solution was coded using a 'SQL-like language' and it was run on a Hadoop cluster using Spark/MapReduce (leveraging proprietary third-party software).

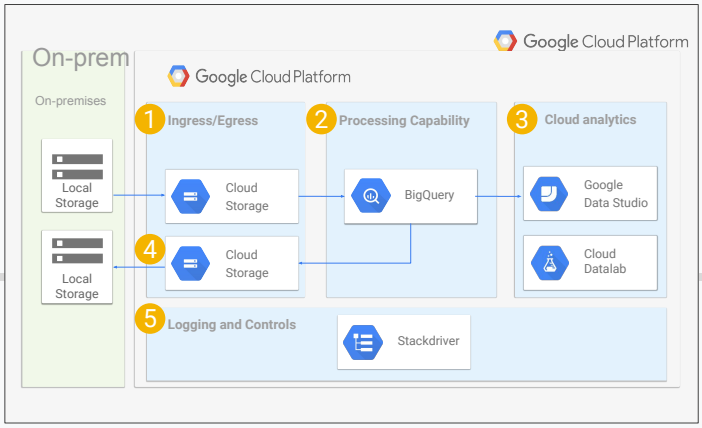
The client wanted to optimize the target cloud solution to:

- Minimize changes to their processes and codebase.
- Leverage PaaS and native cloud solution (i.e., avoid third-party software).
- Significantly improve performance (from hours to minutes).

We mapped that to technical requirements like this...

Business Requirement	Technical Requirement
Minimum changes to their processes and codebase	• Programmatically convert 'SQL-like' into ANSI SQL • No changes to source/target input structures • Automated 'black box' regression testing
Leverage PaaS and native Cloud Solution (i.e., avoid third-party software)	• Minimize number of systems/interfaces, aim for full execution in BigQuery (i.e., remove the need for a Hadoop cluster)
Significantly improve performance (from hours to minutes)	• Analyze and (manually) optimize the SQL code in BigQuery for performance

And this is how we implemented that technical requirement.



Source data is ingested in Cloud Storage (no changes to sourcing)

Financial reports are generated entirely in BigQuery

Cloud Analytics allow users to review the reports

Data is egressed on-premises (no changes to target structures → no changes downstream)

Logs and controls out of the box in Stackdriver

Challenge Lab 02

PDE Prep—Cloud Dataproc cluster operations and maintenance: Challenge Lab

A Challenge Lab has minimal instructions. It explains the circumstance and the expected results -- you have to figure out how to implement them.
This is a timed lab.
The lab will expire after 60 minutes.
The lab *can* be completed in 45 minutes.

https://pixabay.com/en/hourglass-timer-rainbow-1895102/


Google Cloud Training and Google Cloud