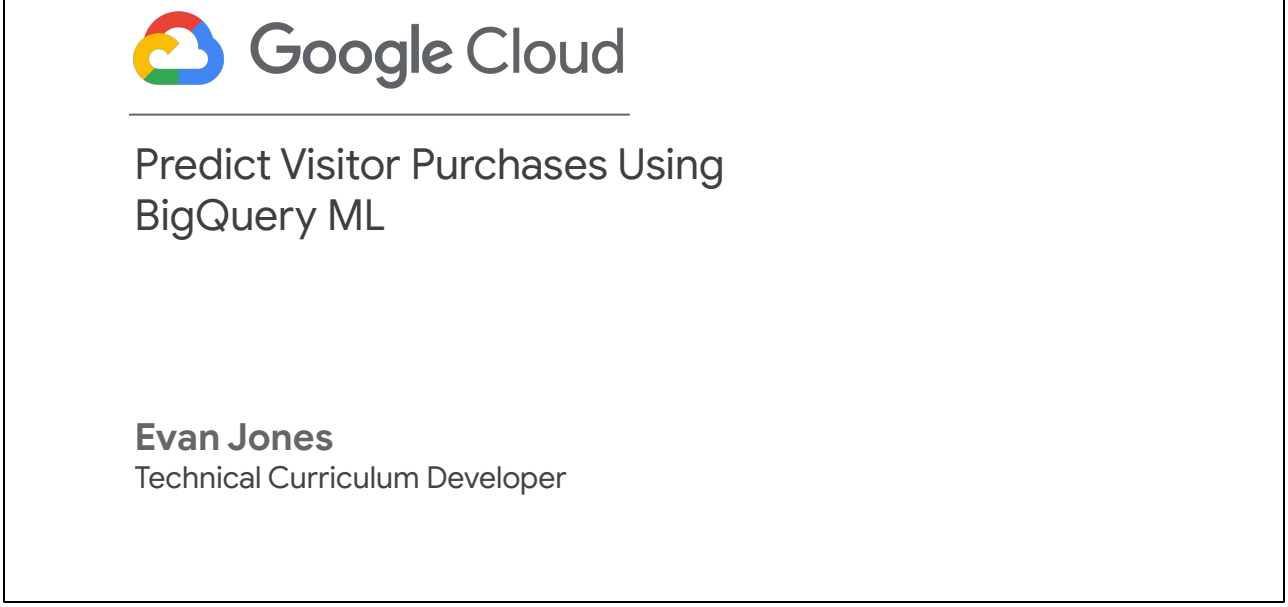


Hi, Evan here and in this module we will explore in great detail some of the cool things you can do with SQL in BigQuery. Now I know what some of you may be thinking -- SQL? Isn't that just used for selecting and returning records from my database?

Well - as the title shown here has already spoiled it, you can now build machine learning models too. It's literally my favorite feature in Google Cloud Platform so I'll spend ample time walking you through how you can create and analyze the performance of your models right within BigQuery.

Agenda

Introduction to BigQuery

- Fast SQL Query Engine
- Managed Storage for Datasets

Insights from Geographic data

Machine Learning on Structured Data

- Choosing the right model type
- Scenario: Predicting Customer Lifetime Value

Creating ML models with SQL

- Introduction to BigQuery ML
- ML project phases
- Key features walkthrough

Before we jump into ML, we first will introduce you to BigQuery as a service which allows you to have a petabyte-scale analytics data warehouse.

You'll soon learn that BigQuery is actually two services in one: a fast SQL engine and fully managed storage.

Agenda

Introduction to BigQuery

- Fast SQL Query Engine
- Managed Storage for Datasets

Insights from Geographic data

Machine Learning on Structured Data

- Choosing the right model type
- Scenario: Predicting Customer Lifetime Value

Creating ML models with SQL

- Introduction to BigQuery ML
- ML project phases
- Key features walkthrough

Then we'll show you some of the other cool features built-in like using GIS functions on your geographic data and how you can even visualize insights on maps.

Agenda

Introduction to BigQuery

- Fast SQL Query Engine
- Managed Storage for Datasets

Insights from Geographic data

Machine Learning on Structured Data

- Choosing the right model type
- Scenario: Predicting Customer Lifetime Value

Creating ML models with SQL

- Introduction to BigQuery ML
- ML project phases
- Key features walkthrough

Next, we'll expand on our course theme of applying ML to your datasets by looking at how to choose the right model type to use for your structured data.

Agenda

Introduction to BigQuery

- Fast SQL Query Engine
- Managed Storage for Datasets

Insights from Geographic data

Machine Learning on Structured Data

- Choosing the right model type
- Scenario: Predicting Customer Lifetime Value

Creating ML models with SQL

- Introduction to BigQuery ML
- ML project phases
- Key features walkthrough

Lastly, we'll build a custom model using just SQL with BigQuery ML. I'll also show you how I commonly organize my ML projects and some advanced BigQuery ML features that let you do things like see which fields the model thinks are the most important in it's predictions.

Sound good? Let's get started

BigQuery is a petabyte-scale fully-managed data warehouse

1. It's serverless
2. Flexible pricing model
3. Data encryption and security
4. Geospatial data types & functions
5. Foundation for BI and AI

BigQuery is designed to be an easy to use data warehouse where you can focus on writing SQL statements on small or large datasets without worrying about infrastructure. If you've never written SQL before I'll also provide resources and labs to get you up to speed as we go.

BigQuery is a petabyte-scale fully-managed data warehouse

1. It's serverless
2. Flexible pricing model
3. Data encryption and security
4. Geospatial data types & functions
5. Foundation for BI and AI

That's point number one, it's serverless.

BigQuery is a petabyte-scale fully-managed data warehouse

1. It's serverless
2. Flexible pricing model
3. Data encryption and security
4. Geospatial data types & functions
5. Foundation for BI and AI

BigQuery's common pricing model is pay-as-you-go where you pay for the number of bytes of data your query processes and for any permanent table storage. If you want to have a set bill every month, you can subscribe to flat tier pricing where you get a special reserved amount of resources for use.

BigQuery is a petabyte-scale fully-managed data warehouse

1. It's serverless
2. Flexible pricing model
3. Data encryption and security
4. Geospatial data types & functions
5. Foundation for BI and AI

Data in BigQuery is encrypted at rest by default.

BigQuery is a petabyte-scale fully-managed data warehouse

1. It's serverless
2. Flexible pricing model
3. Data encryption and security
4. Geospatial data types & functions
5. Foundation for BI and AI

You can also specify the geographic locality of your data if you need to meet regulatory requirements. Controlling access to your data can be as granular as specific columns (say any column tagged with PII or Personally Identifiable Information) or specific rows (like if marketing only needs to see certain published rows in a table). BigQuery works in tandem with Cloud IAM to set these roles and permissions at a project level which you will see more of later.

SQL as a language has been around since the 1970s and just watching the functionality added to the language over time has been awe-inspiring. You can now perform GIS functions (like distances between lat/long points, what map points are in this geographic region) and more. Your data most likely has some kind of geographic component (like city, state, zip code) so now it's time to unlock additional insights.

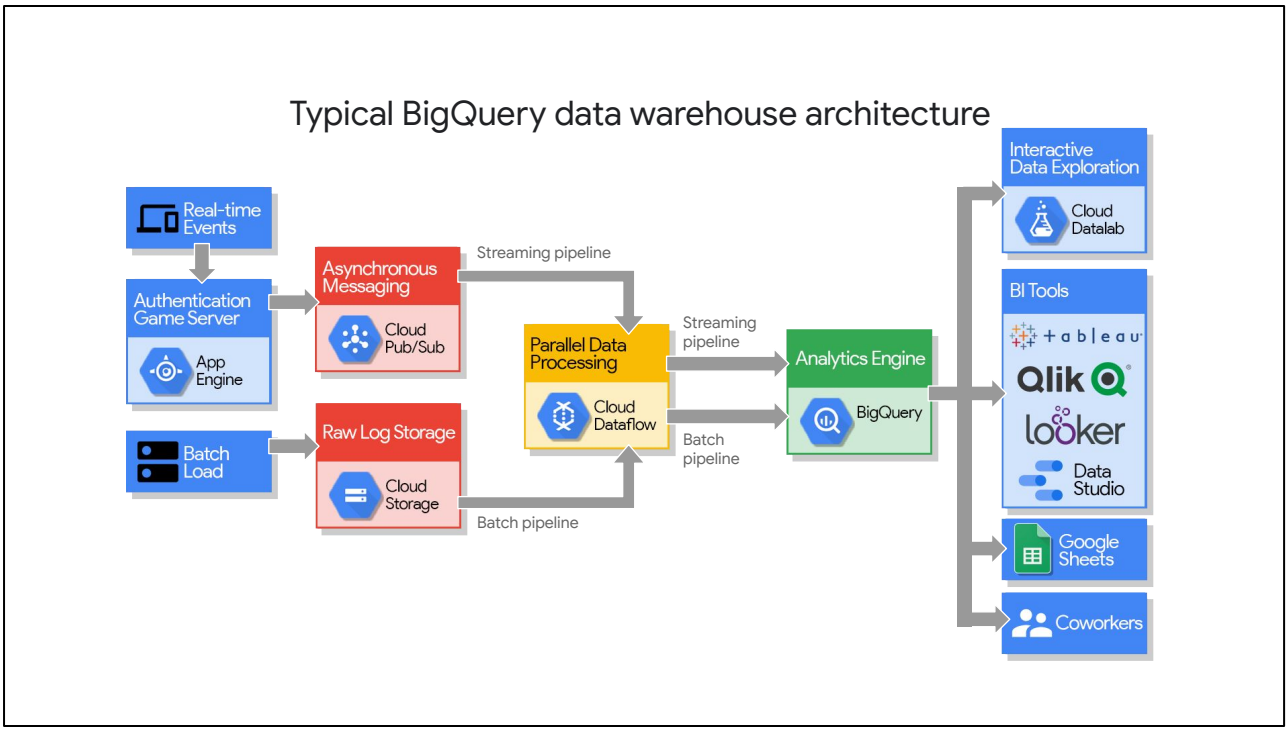
BigQuery is a petabyte-scale fully-managed data warehouse

1. It's serverless
2. Flexible pricing model
3. Data encryption and security
4. Geospatial data types & functions
5. Foundation for BI and AI

Lastly, BigQuery as both a data warehouse and an advanced query engine is foundational for your AI and ML workloads. It's common for analysts, engineers, and data scientists to use BigQuery to store, transform, and feed large datasets directly into their models. This is a huge leap over training ML models on just small samples of data locally -- you can now train on all the data you have available. That's the elastic data warehouse component for ML datasets.

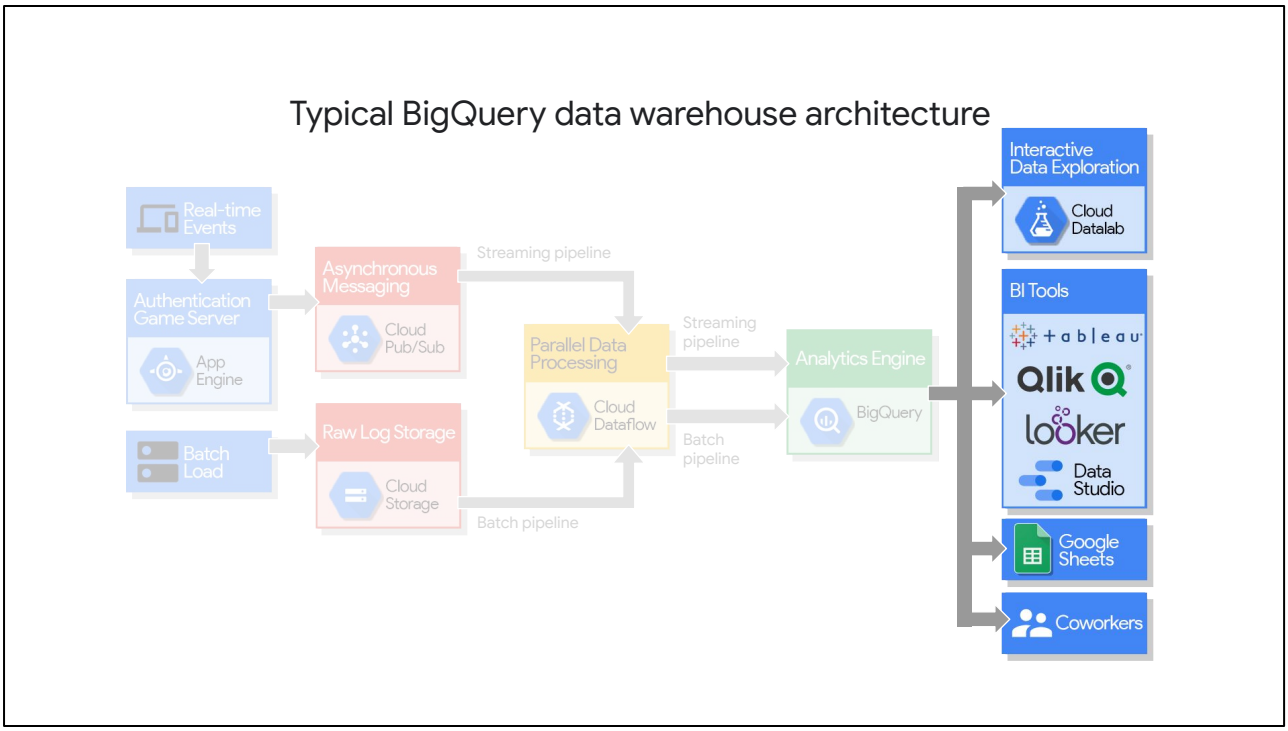
Beyond that, as you've seen in the demo, you can now write ML models directly in BigQuery using SQL. This is a great start for model prototyping as you are engineering what features to use.

https://cloud.google.com/bigquery/

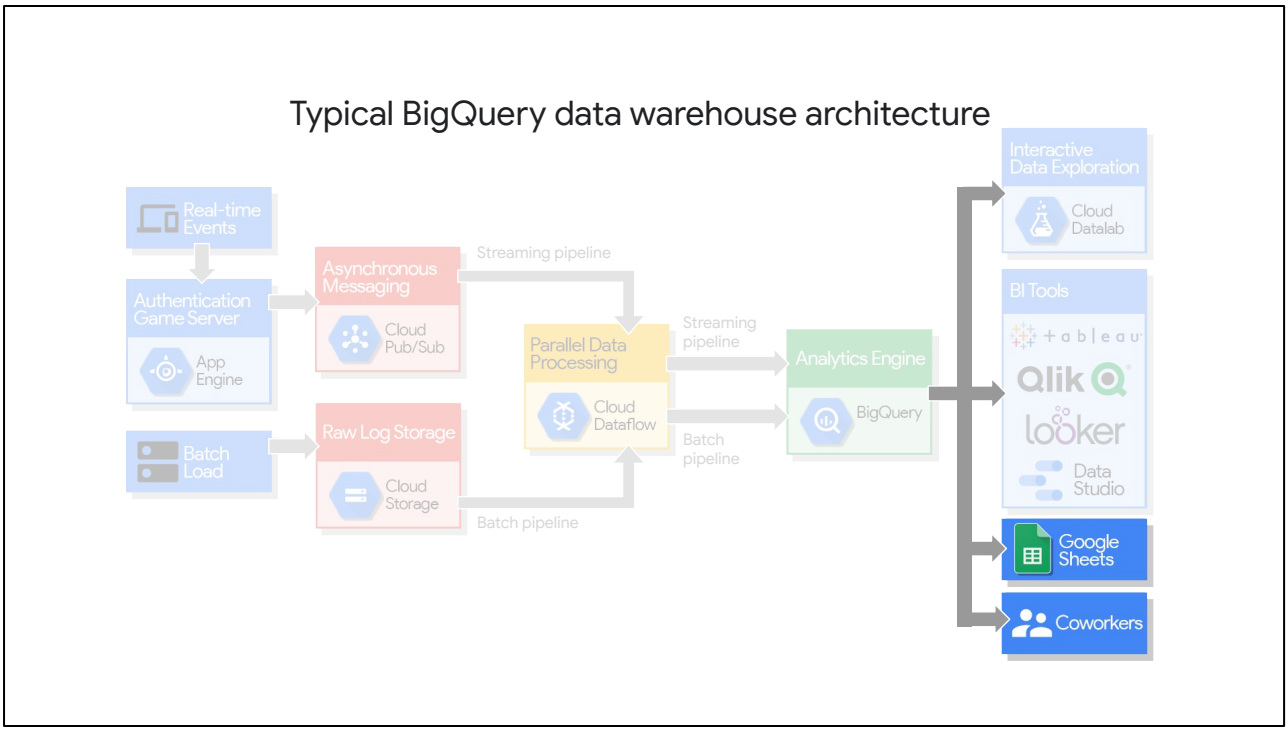


So what does a typical data warehouse solution architecture look like?

Take a look at the green box. BigQuery is the analytics engine that sits at the end of the data pipeline. It stores all the incoming data from the left and then allows you to do your analysis and model building.

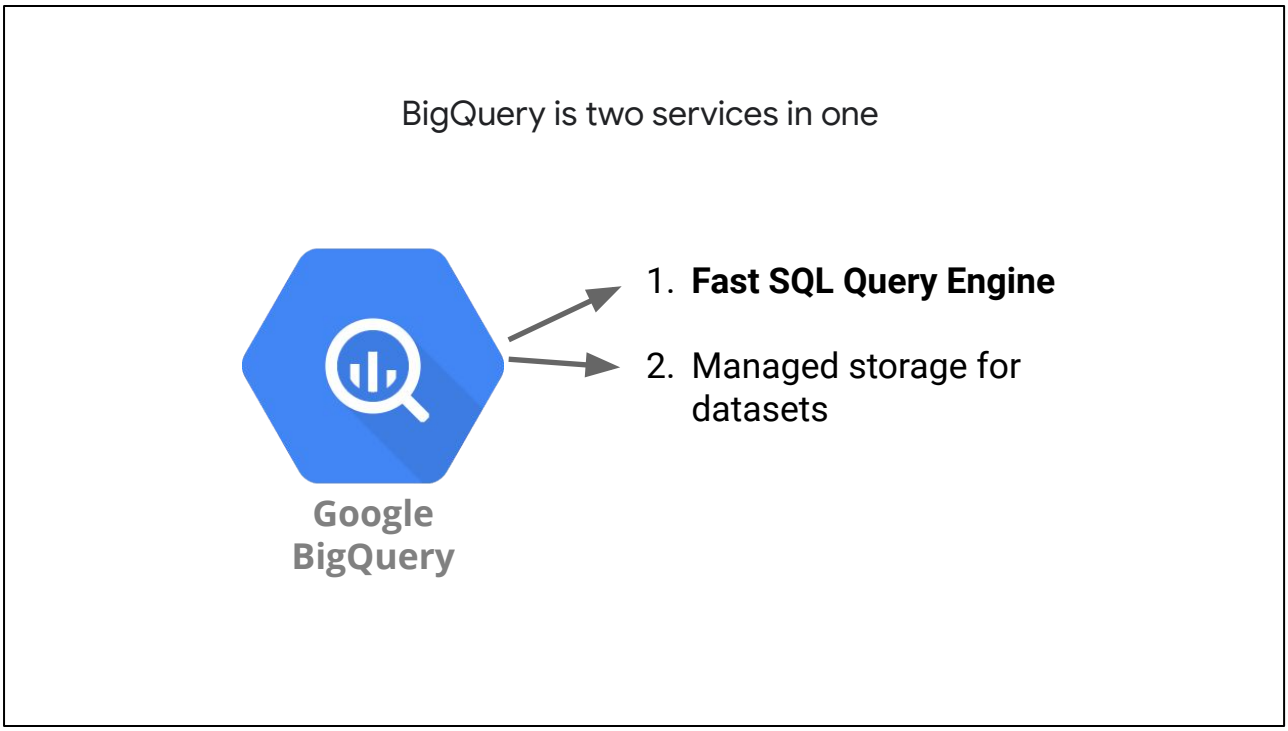


On the far right you can see the myriad of visualization and analysis tools that you can connect to BigQuery. For ML engineers, once your dataset is in BigQuery you can easily call it from your iPython ML notebook in the cloud with just a few commands. If you're a business intelligence analyst, you can connect to visualization tools like Data Studio, Tableau, Looker, and more.



Lastly, and worth mentioning here, if you have a team of analysts who prefers to work in spreadsheets you can now query your small or huge BigQuery datasets directly from Google Sheets and perform common operations like Pivot tables and more.

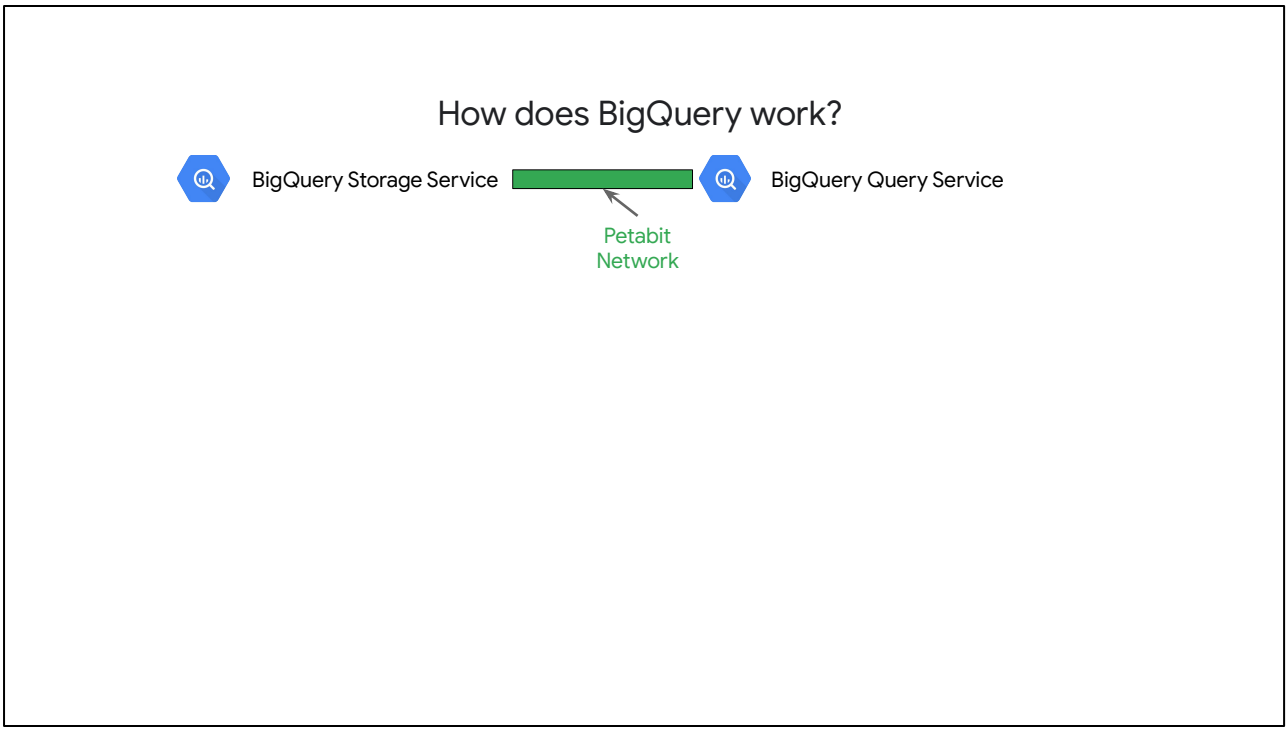
The key takeaway is that BigQuery is a common sink or staging area for data analytics workloads. Once your data is there, analysts, BI developers, and ML engineers can be granted access your data for their own insights.



It's time to get a little more technical. So how does BigQuery actually work? Well it's actually two services in one as we hinted at earlier.

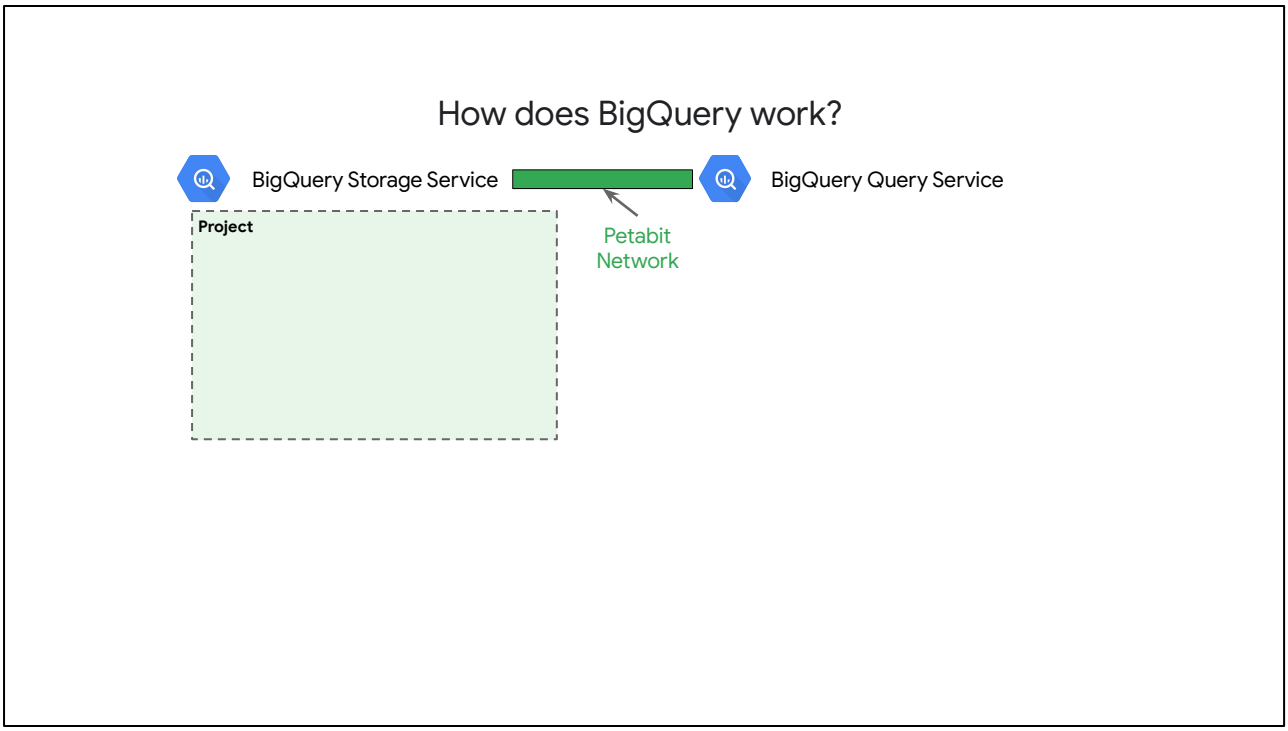
It's both a fast SQL query engine and also a fully-managed storage layer for loading and storing your datasets.

Keep in mind that it's a "serverless" service, meaning that it is fully managed. Let's take a look at what Google Cloud is managing for you as part of the BigQuery service.



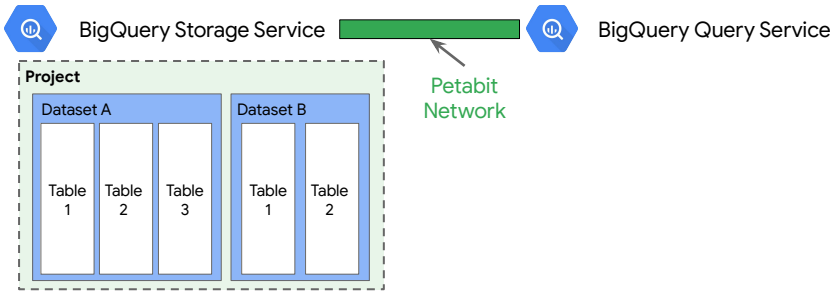
The two services are connected by Google's high speed internal network. Recall that this super fast network enables us to separate compute and storage.

So what does each service do? The short answer is that the services are fully-managed so users like you and me don't have to worry about how BigQuery stores data on disk or autoscales machines for big queries. That said, it's a great learning exercise to understand how the services perform their magic so we can better understand what is happening.

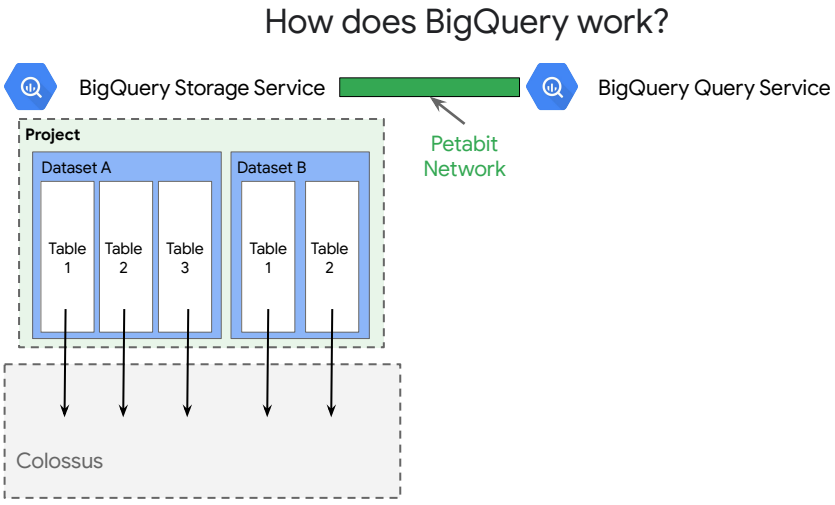


First up is the BigQuery storage service. The storage service automatically manages the data that you ingest into the platform.

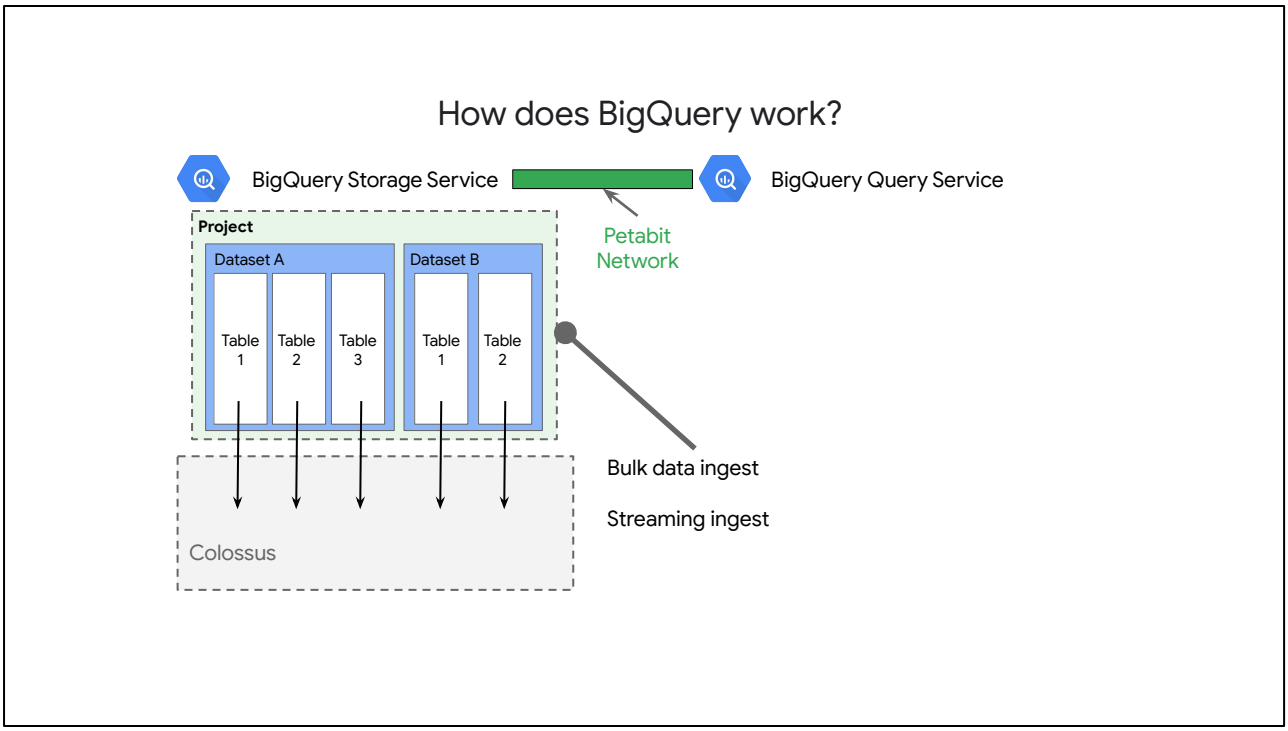
How does BigQuery work?



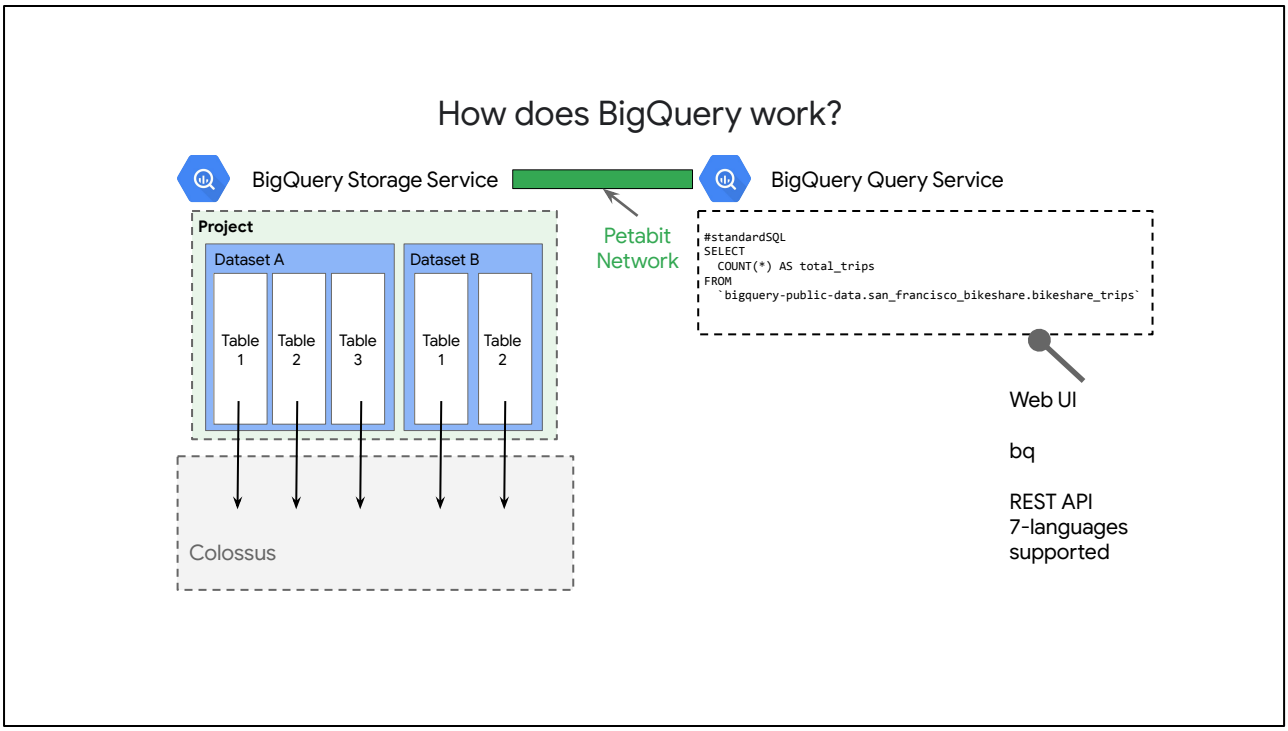
Data is contained within a project in datasets in tables.



The tables are stored as highly compressed columns in Google's Colossus file system which provides durability and availability.

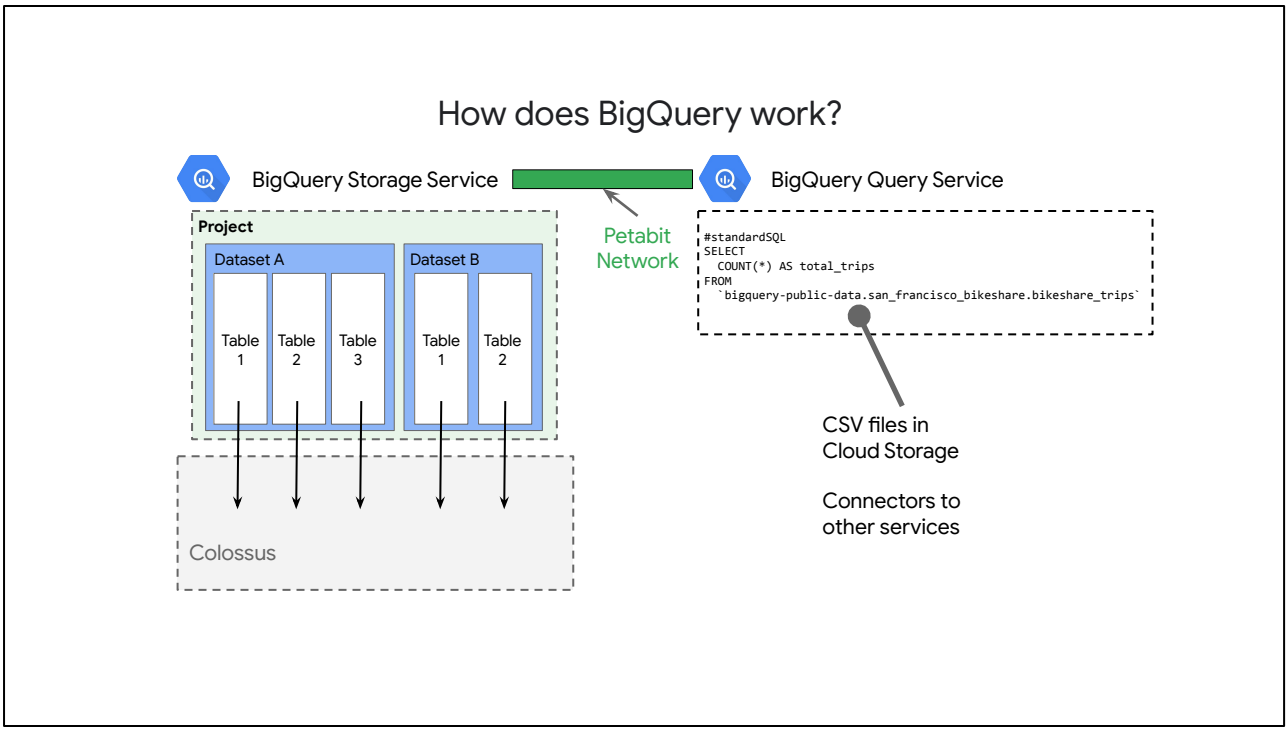


The storage service supports **bulk data ingest** and streaming ingest. So it can work with huge amounts of data and also real-time data streams.



The query service runs interactive or batch queries that are submitted through console, the BigQuery web UI, the bq command line tool, or via REST API. The REST API is supported for seven programming languages.

There are connectors to other services such as Cloud Dataproc which simplify creating complex workflows between BigQuery and other GCP data processing services.



The query service can also run query jobs on data contained in other locations, such as **tables in CSV files hosted in Cloud Storage**.

Before you get too excited about running queries on your data in Google Sheets you should know that BigQuery is most efficient when working with data contained in its own storage service.

The storage service and the query service work together to internally organize the data to make queries efficient over huge datasets of Terabytes and Petabytes in size. They even optimize your SQL statement syntax after you submit it.

One of the most important controls you have over resource consumption and costs is controlling the amount of data processed. In general, you only want to select the columns of data you actually want to process and return. A good rule of thumb is to start broad when you're exploring the dataset and then hone in on those critical fields and rows that you need.

BigQuery supports standard SQL queries for analysis

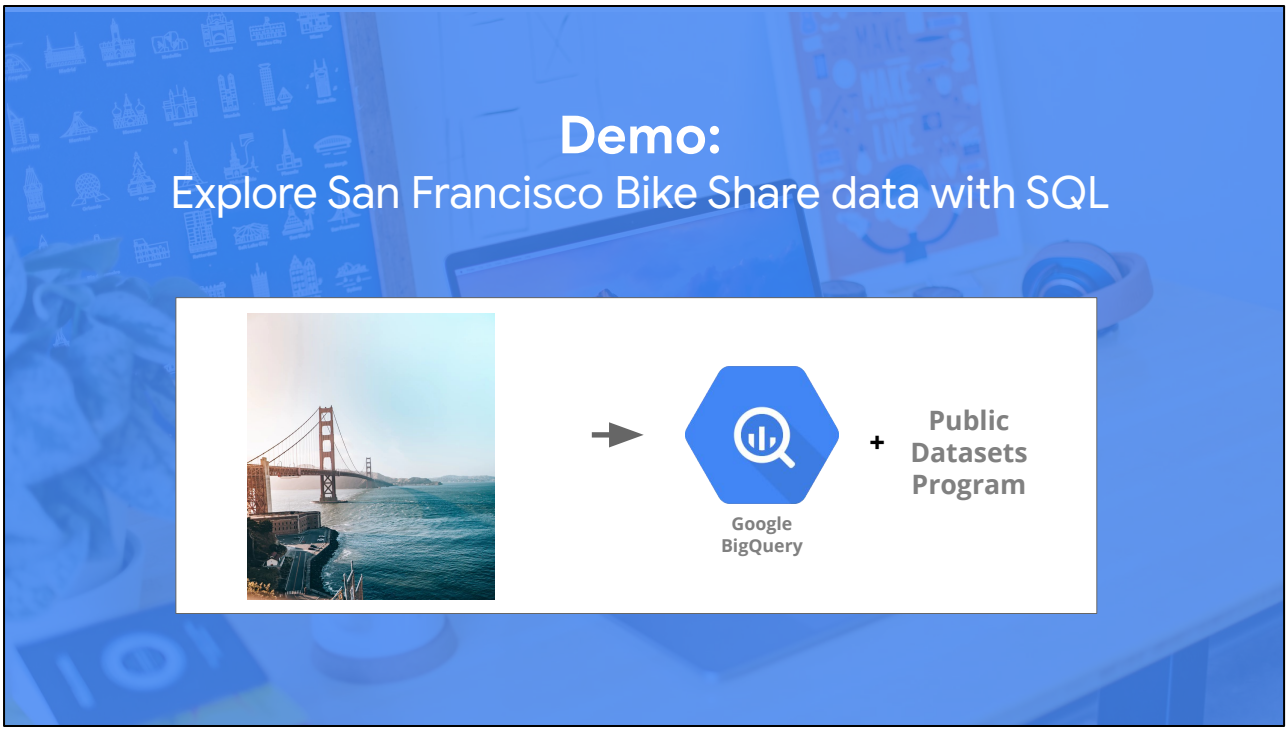
```
#standardSQL
SELECT
  COUNT(*) AS total_trips
FROM
  `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips`
```

Row	total_trips
1	1947419

In the last lesson we showed this SQL statement ran using the query engine. As any data analyst will tell you, exploring your dataset with SQL is your first step in uncovering those hidden insights.

Here we're counting the total number of trips taken on the public dataset for San Francisco Bike Share trips. Let's explore this public dataset in greater detail with a demo.

Example SF Bike Share query
https://console.cloud.google.com/bigquery?sq=1057666841514:6e9100c9fbaf421bac62a862efbe934c&_ga=2.65081326.-156181847.1534452292



[Demo]
Pick from common sample queries and showcase common SQL functions:
https://console.cloud.google.com/marketplace/details/san-francisco-public-data/sf-bike-share?filter=solution-type%3Adataset&filter=category%3Atransportation&id=3ff66be9-b024-4500-b61a-385851b5be96

Updated hourly
Exploration SQL: https://github.com/marycboardman/Bay-Area-Bike-Share
https://medium.com/google-cloud/predicting-san-francisco-bikeshare-availability-with-tensorflow-and-lstms-a3ced14d13dc

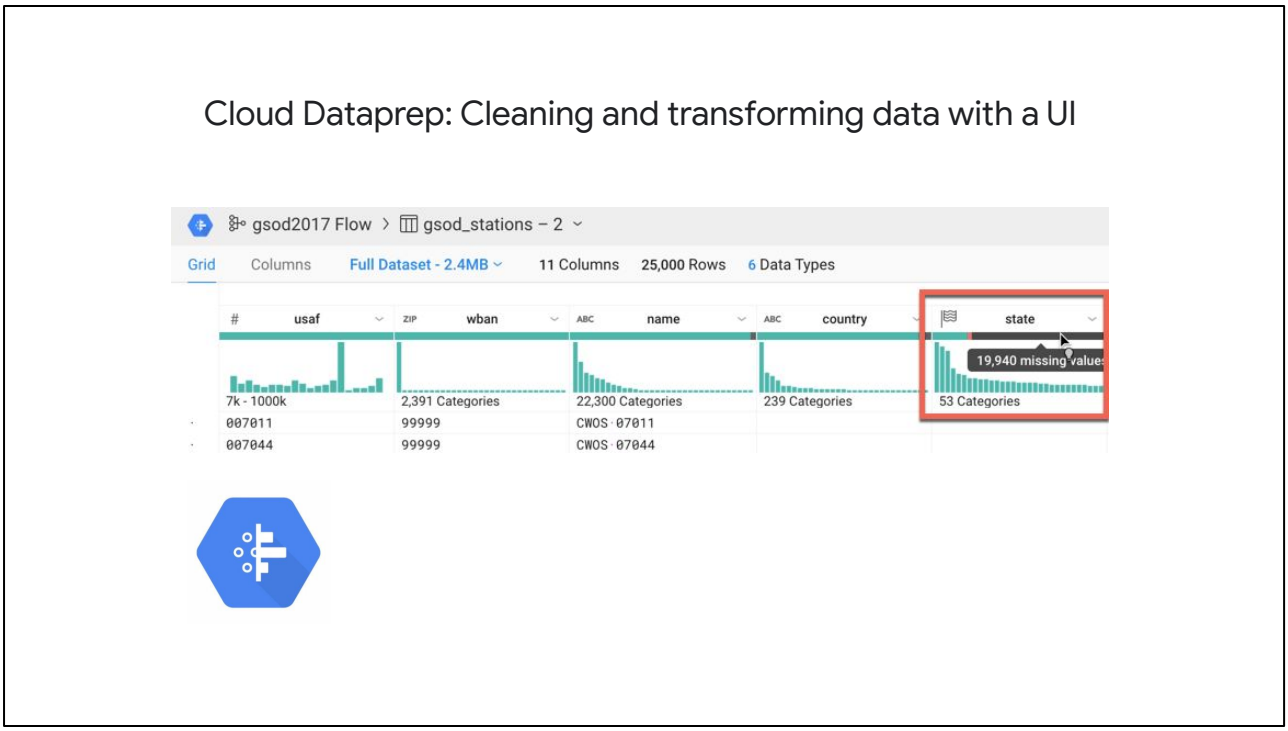
https://gist.github.com/jonesevan/ed55d2dacd7371f2972229b9c6f28d1c

Common SQL operations include deduplication and cleansing

```
# Top 5 stations for casual bike share users
SELECT
  start_station_name,
  COUNT(*) AS total_trips
FROM
  `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips`
WHERE c_subscription_type = 'Customer' # casual user
AND member_birth_year IS NOT NULL
GROUP BY start_station_name
ORDER BY total_trips DESC
LIMIT 5
```

As we saw in the demo, a lot of the time you aren't sure of the quality of the data in your dataset so you can use SQL to explore and filter for anomalies. Here we filtered out customer records who did not provide a birth date by setting member_birth_year IS NOT NULL as a filter. I'll also provide a SQL syntax reference that you can bookmark and refer to. While I have been working with SQL awhile I certainly don't have all the functions and clauses committed to memory yet and am constantly on Google searching for syntax.

[resource]
https://www.fordgobike.com/system-data

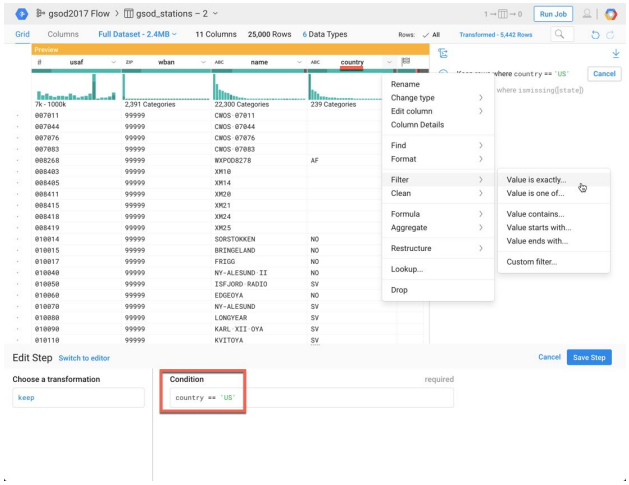


If you prefer to use a UI to inspect the quality of your datasets consider using Cloud Dataprep which is a GCP product offered in partnership with Trifacta. Using Dataprep, you can load a sample of your BigQuery datasets into an exploration view which will provide you with column histogram charts as you see here. This specific dataset is the location of weather stations around the world. The histograms in the State column on the right highlight the data is skewed towards 3 or so values -- which we could infer means that there are some states that have many weather stations. Maybe large States like California -- you can hover over the bars to see the actual values.

What I've highlighted in the horizontal bar graph is how well the values in a column of data map to the expected data type. Dataprep knows what a U.S. state allowable value is and is telling us there are 19,940 missing or NULL values for State. Why might that be?

As you might have guessed, these weather station readings came from all across the world and State is an optional field.

Common SQL operations include deduplication and cleansing



In Dataprep, if I wanted to create a pipeline to filter all incoming records to only include weather station readings where the country field is U.S. you can use the UI to quickly setup those transformation steps.

After your transformation recipe is complete, when you run a Cloud Dataprep job it farms out the work to Cloud Dataflow which handles the processing of your data pipeline at scale. The main advantage to Cloud Dataprep is for teams who want a UI for data exploration and want to spend minimal time coding to build pipelines.

Lastly, with Dataprep you can schedule your pipeline to run at regular intervals.

Schedule your queries to run with @run_time

```
SELECT @run_time AS time,
  title,
  author,
  text
FROM `bigquery-public-data.hacker_news.stories`
LIMIT
  1000
```

If you'd prefer to do all your SQL and exploration work inside of BigQuery, you can even use SQL to setup scheduled queries by using the @run_time parameter or the query scheduler in the BigQuery UI. I'll provide a link to how you can setup a scheduled query in the course resources.

[resource]
https://cloud.google.com/bigquery/docs/scheduling-queries
https://cloud.google.com/bigquery/docs/scheduling-queries#setting_up_a_scheduled_query

IAM **Project Roles** and Inherited BigQuery Permissions

When a project is created, BigQuery grants the `Owner` role to the user who created the project.

Primitive role	Capabilities
`Viewer`	• Can start a job in the project. Additional dataset roles are required depending on the job type. • Can list and get all jobs, and update jobs that they started for the project • If you create a dataset in a project that contains any viewers, BigQuery grants those users the bigquery.dataViewer predefined role for the new dataset.

`Editor`	• Same as **`Viewer`**, plus:
	  • Can create a new dataset in the project
	  • If you create a dataset in a project that contains any editors, BigQuery grants those users the bigquery.dataEditor predefined role for the new dataset.

`Owner`	• Same as **`Editor`**, plus:
	  • Can list all datasets in the project
	  • Can delete any dataset in the project
	  • Can list and get all jobs run on the project, including jobs run by other project users
	  • If you create a dataset, BigQuery grants all project owners the bigquery.dataOwner predefined role for the new dataset.

Now that you're familiar with some of the great insights you can glean from your data, it's time to talk data security so your insights are only shared with those who you want.

BigQuery inherits data security roles that you have setup in Cloud IAM as you see here in this table. For example, if you are an overall Project Viewer in Cloud IAM you can start BigQuery jobs but you cannot create new Datasets yourself. If you're an Editor you can create datasets. And if you're and Owner, you can also delete datasets.

Keep in mind that default access to datasets **can be overridden on a per-dataset basis**.

https://cloud.google.com/bigquery/docs/access-control

- Dataset users should have the minimum permissions needed for their role
- Use separate projects or datasets for different environments (e.g. DEV, QA, PRD)
- Audit roles periodically

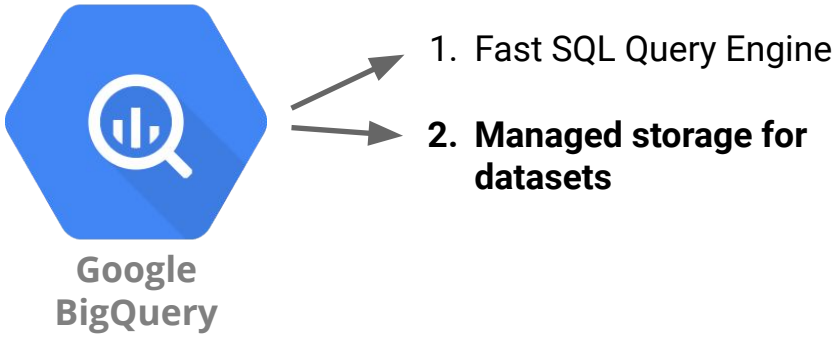
Data security is more than simply applying the right project permissions and roles during your dataset creation. You should have a written and circulated Data Access Policy for your organization that specifies how and when data can be shared and with whom.

One first step to ensure healthy data access controls is to periodically audit the users and groups associated with your GCP project and individual datasets. Do these users need to be Owners or Admins or would a more restrictive role suffice for their job duties?

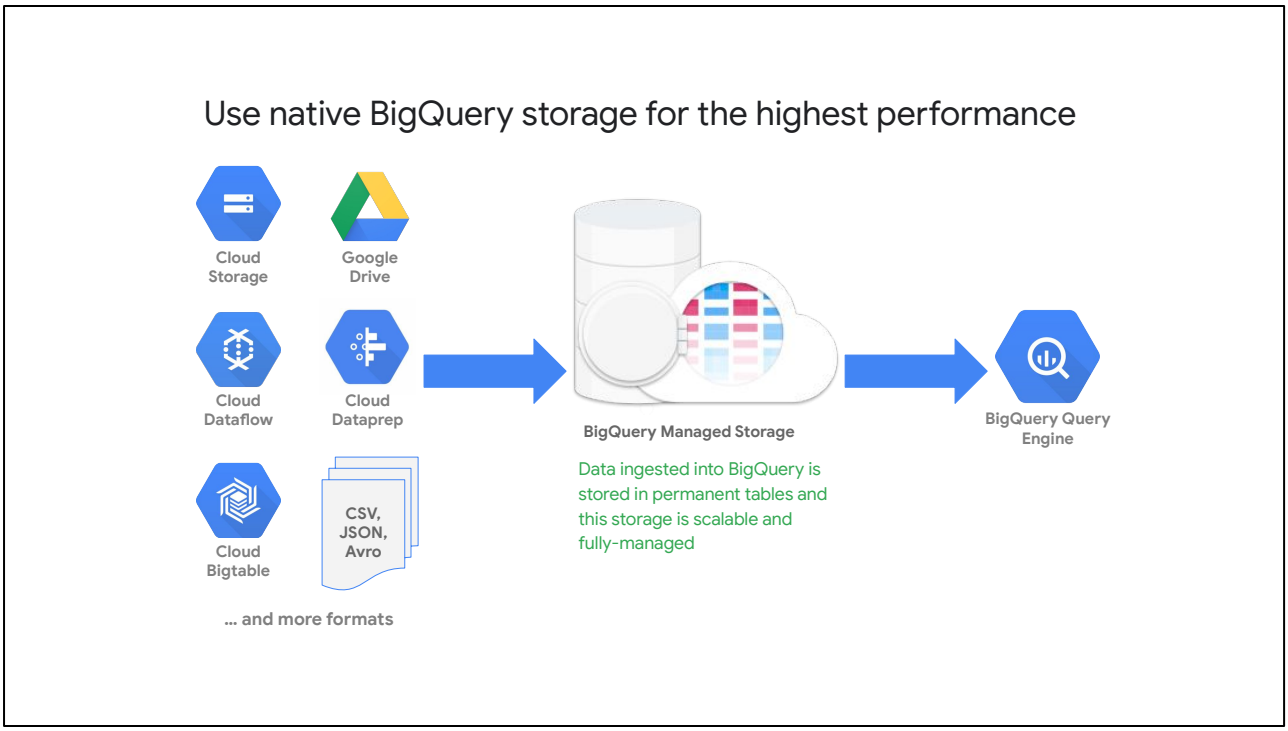
A second pitfall is working, testing, and editing datasets in production. It is often wise to have a completely separate GCP project or dataset for different testing environments to prevent unintentional data loss.

https://cloud.google.com/bigquery/docs/access-control

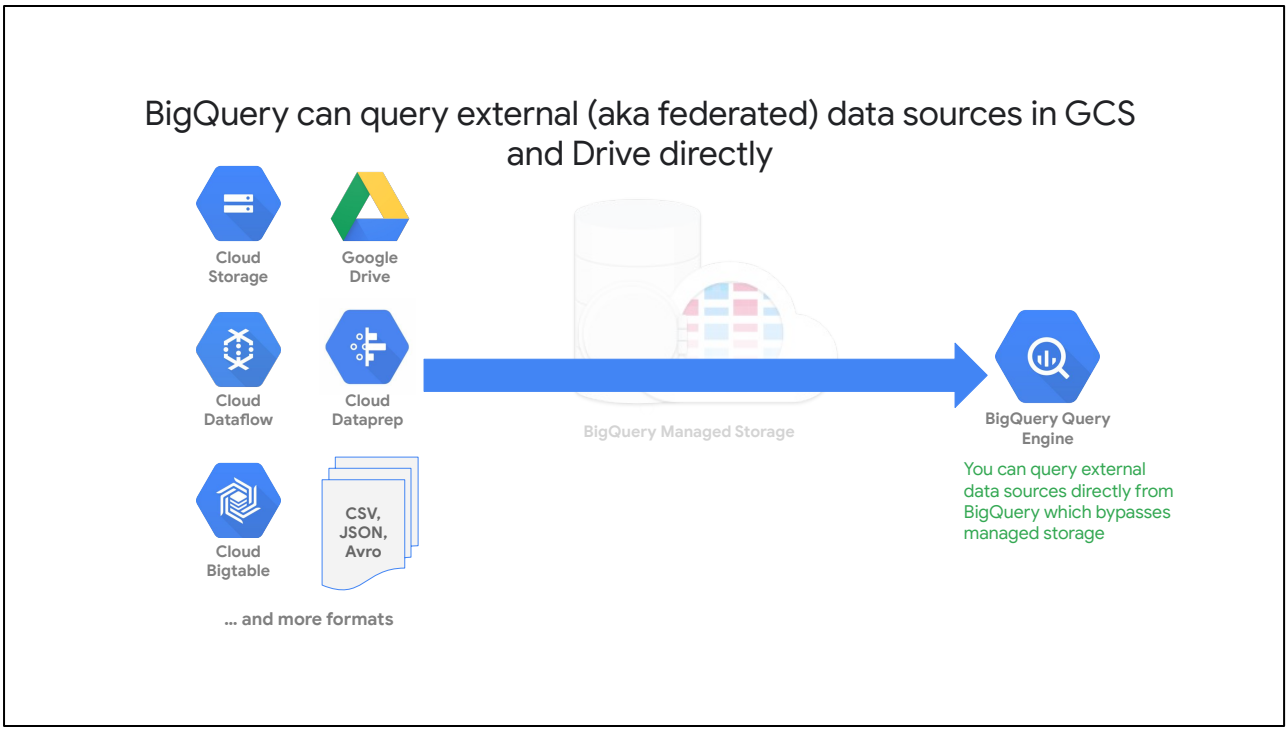
BigQuery is two services in one



In addition to super fast query execution times, BigQuery also manages the storage and metadata for your datasets.



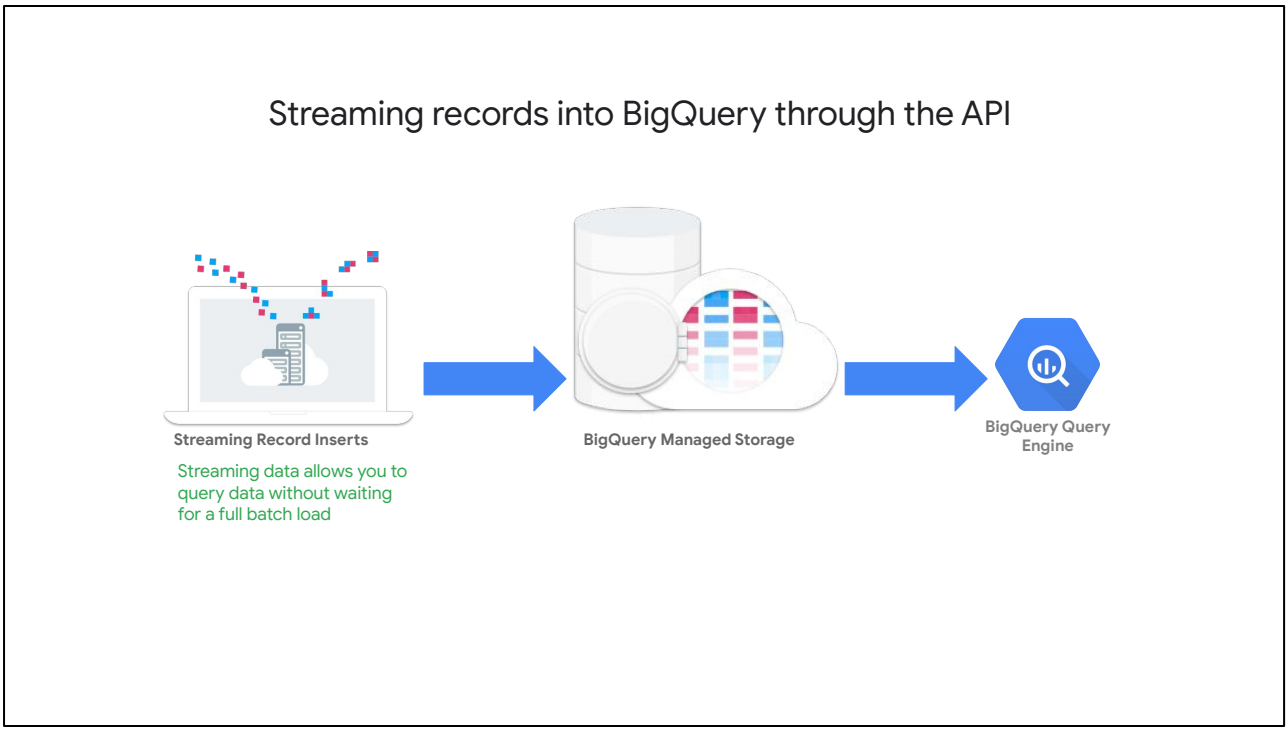
As we mentioned earlier, BigQuery can ingest datasets from a variety of different formats. Once inside BigQuery native storage, it is fully managed by the BigQuery team here at Google and is automatically replicated, backed up, and setup to autoscale. You can even recover recently deleted dataset tables within a certain period too.



You also have the option of querying external data sources directly and bypassing BigQuery managed storage. That means if you have a raw CSV file in GCS or a Google Sheet you can write a query against it without having to ingest the data into BigQuery first.

A common use case for this is having an external data source that is small and constantly changes like say a price-list for commodities that another team maintains.

But there are a few reasons why should you not do this. The first is that data consistency is not guaranteed. If the data source changes mid-flight, BigQuery doesn't guarantee that those updates were captured. If you're concerned about that, consider building a streaming data pipeline into BigQuery with Cloud Dataflow which we will cover in the next module.

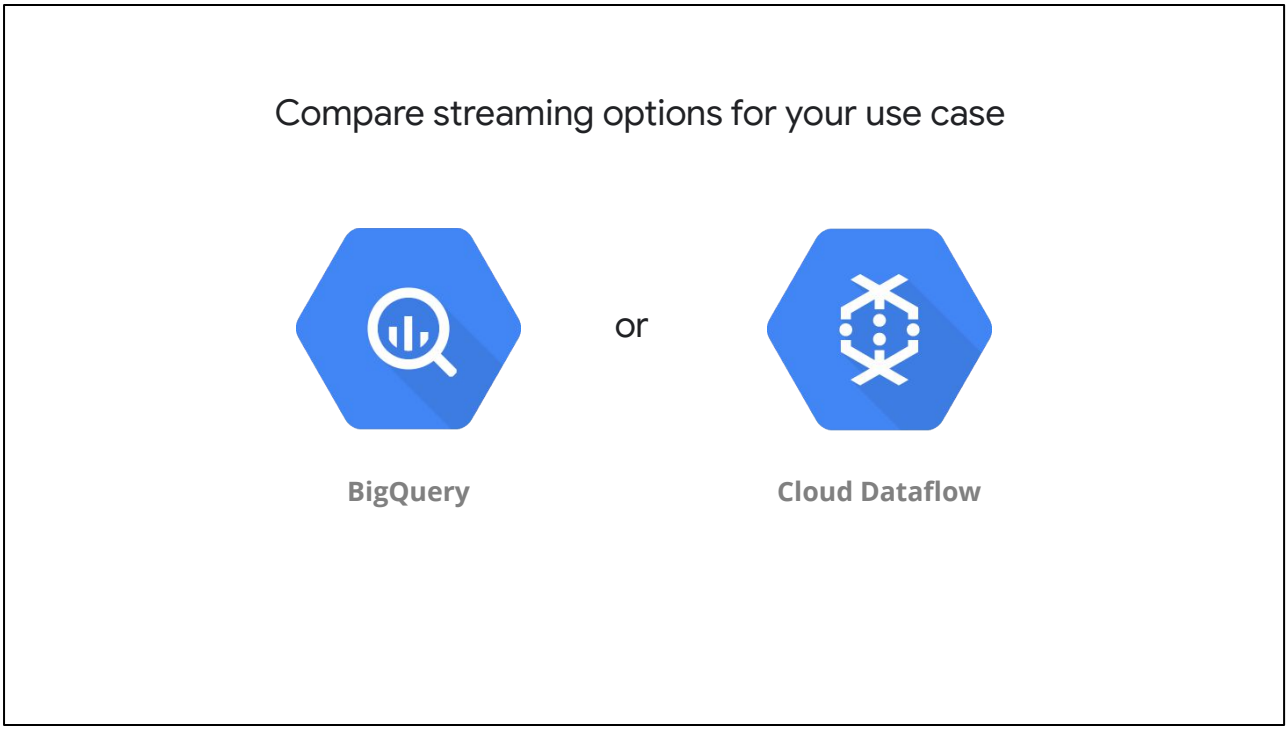


In addition to ingesting datasets as a batch, like uploading a CSV, you can also stream records into BigQuery using the API.

Note that there are a few quota restrictions that you should be aware of. The max row size is 1MB and the maximum throughput is 100,000 records per second per project.

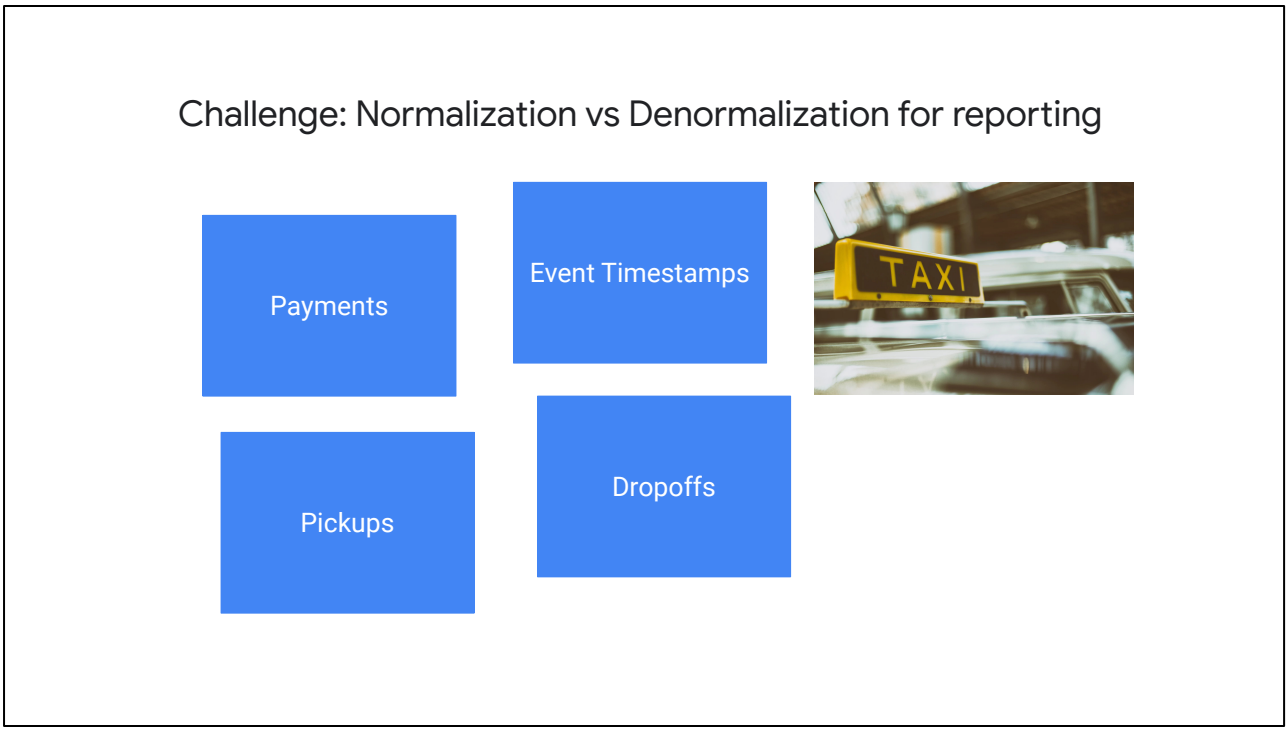
If you need higher throughput (on the order of millions of records per second) for use cases like application logging or real-time event consider using Cloud BigTable instead.

https://cloud.google.com/bigquery/streaming-data-into-bigquery



Before you start streaming thousands of records into BigQuery using the API, consider the other options you have for streaming solutions.

If you have data that needs to be transformed or aggregated mid-flight into table and row format or even joined against other data sources mid-stream or if you want to take just a window or segment of that data you should consider using Cloud Dataflow for your streaming data pipeline. We'll cover working with that solution in the very next module.



Lastly, if you're familiar with database design you'd likely heard of the concept of normalization which is the act of breaking up one huge parent table into component child tables so you're storing one fact in one place and not repeating yourself across records.

Take a taxi company for example. You could track payments, timestamps of events, and geographic lat/longs of pickups and dropoffs all in separate tables as you see here. What would you need to do to bring all these data sources together for reporting? You'd write one big join and specify all the tables to bring in.

While breaking apart database tables into silos is common practice for relational databases like mySQL or SQL Server, for reporting let me show you a cool new way to structure your reporting tables.

Use STRUCTs and ARRAYS for consolidated reporting

Job information | Results | JSON | Execution details

Row	order_id	service_type	payment_method	event.status	event.time	pickup.latitude	pickup.longitude	destination.latitude	destination.longitude	total_distance_km
1	CR-6098	GO_CAR	CASH	DRIVER_FOUND	2018-12-31 01:45:15.667 UTC	-6.9228116	107.5930599	-6.912966	107.609604	2.9749999046325684
				COMPLETED	2018-12-31 02:06:56.894 UTC					
				PICKED_UP	2018-12-31 02:05:40.075 UTC					
				CREATED	2018-12-31 01:44:59.239 UTC					

BigQuery natively supports
ARRAYs as data types

Take a look at the table at the top. What do you notice that you haven't seen before?

First you have just one row of data but it looks like you have four. What's going on? Well, event.time is actually an array data type. You can have multiple data values (in this case timestamps) for a single taxi booking row. Likewise with the corresponding array of status values.

So already in one table you can get the high level total number of orders but also the number of completed orders without having to do any joins.

The second insight which is a bit more hidden is that some of the column names have almost like a family name. EVENT.status, EVENT.time or DESTINATION.latitude, DESTINATION.longitude. These grouped fields are part of a SQL data type known as a STRUCT.

Use STRUCTs and ARRAYS for consolidated reporting

Job information | Results | JSON | Execution details

Row	order_id	service_type	payment_method	event.status	event.time	pickup.latitude	pickup.longitude	destination.latitude	destination.longitude	total_distance_km
1	CR-6098	GO_CAR	CASH	DRIVER_FOUND	2018-12-31 01:45:15.667 UTC	-6.9228116	107.5930599	-6.912966	107.609604	2.9749999046325684
				COMPLETED	2018-12-31 02:06:56.894 UTC					
				PICKED_UP	2018-12-31 02:05:40.075 UTC					
				CREATED	2018-12-31 01:44:59.239 UTC					

Schema | Details | Preview

Field name	Type	Mode
order_id	STRING	NULLABLE
service_type	STRING	NULLABLE
payment_method	STRING	NULLABLE
event	RECORD	REPEATED
event. **status**	STRING	NULLABLE
event. **time**	TIMESTAMP	NULLABLE
pickup	RECORD	NULLABLE
pickup. **latitude**	FLOAT	NULLABLE
pickup. **longitude**	FLOAT	NULLABLE

BigQuery natively supports ARRAYs as data types (and STRUCTs too)

If you were to inspect the schema for this table you'd notice that for the EVENT field the type is a RECORD which means it's a STRUCT. You can think of a STRUCT as essentially a collection of other fields. From a reporting standpoint you can have many nested STRUCTs within a single table which conceptually is like having many other tables pre-joined into one big table.

A big benefit of doing it this way is that you have a single table which has all the fields in a single place for you to analyze. You don't need to worry about join keys or differing levels of table granularity anymore.

[resource]
https://medium.freecodecamp.org/exploring-a-powerful-sql-pattern-array-agg-struct-and-unnest-b7dcc6263e36

Agenda

Introduction to BigQuery

- Fast SQL Query Engine
- Managed Storage for Datasets

Insights from Geographic data

Machine Learning on Structured Data

- Choosing the right model type
- Scenario: Predicting Customer Lifetime Value

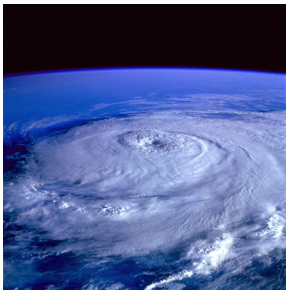
Creating ML models with SQL

- Introduction to BigQuery ML
- ML project phases
- Key features walkthrough

BigQuery natively supports GIS (or Geographic Information System) functions for gleaning insights from your geographic datapoints like latitude and longitude. Let's examine how you can put these into practice.

Unlock your geographic insights with BigQuery GIS functions

```
#standardSQL
SELECT
  ST_GeogPoint(longitude, latitude) AS point,
  name,
  iso_time,
  dist2land,
  usa_wind,
  usa_pressure,
  usa_sshs,
  (usa_r34_ne + usa_r34_nw + usa_r34_se + usa_r34_sw)/4 AS radius_34kt,
  (usa_r50_ne + usa_r50_nw + usa_r50_se + usa_r50_sw)/4 AS radius_50kt
FROM
  `bigquery-public-data.noaa_hurricanes.hurricanes`
WHERE
  name LIKE '%MARIA%'
  AND season = '2017'
  AND ST_DWithin(ST_GeogFromText('POLYGON((-179 26, -179 48, -10 48, -10 26,
-100 -10.1, -179 26))'),
    ST_GeogPoint(longitude, latitude), 10)
ORDER BY
  iso_time ASC
```



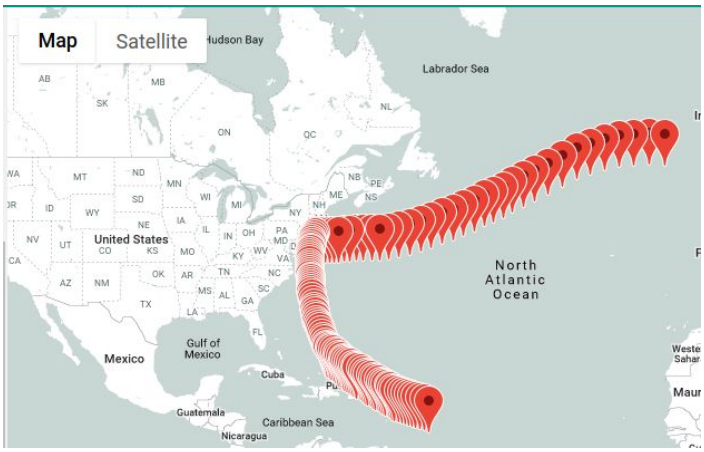
In this SQL query we are plotting the path of a hurricane using SQL and GIS functions.

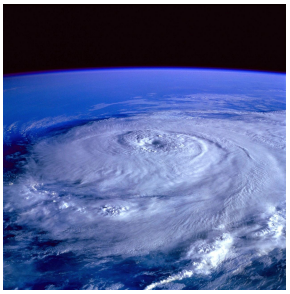
We first create a Geographic Point based on lat/long data. We also bring in other useful fields like wind speed, distance to land fall, and the radius of the hurricane.

We pull all this raw data from the BigQuery public dataset from NOAA on hurricanes and filter for one hurricane in particular with the WHERE clause -- hurricane Maria in 2017. Then we bound the points we care about with a GIS WITHIN function to ensure it will fit the map we're going to visualize these points on next.

https://cloud.google.com/bigquery/docs/gis-tutorial-hurricane#the_global_hurricane_tracks_ibtracs_dataset

Unlock your geographic insights with BigQuery GIS functions





Lastly, we explore our points using Geo Viz which is a web tool for visualization of geospatial data in BigQuery using Google Maps APIs.

Here you can see the hurricane making landfall in the U.S.

https://cloud.google.com/bigquery/docs/gis-tutorial-hurricane#the_global_hurricane_tracks_ibtracs_dataset

BigQuery has over 130 Public Datasets to explore

- Advertising (7)
- Analytics (6)
- Big data (27)
- Climate (20)
- Databases (1)
- Developer tools (20)
- Economics (27)
- Encyclopedic (29)
- Finance (3)
- Genomics (3)
- Health (8)
- Machine learning (1)
- Maps (1)
- Public safety (13)
- Science & research (47)
- Social (3)
- Transportation (1)
- Other (11)

GitHub

One of the best ways to get better data analysis skills is by practicing on a variety of datasets. The BigQuery Public Datasets program partners with companies and organizations to host their datasets in BigQuery analysis by the public. Currently there are well over 100 different datasets for you to explore right now and you can find them all in the BigQuery Web UI under Explore Data just above your own datasets.

https://console.cloud.google.com/marketplace/browse?filter=solution-type:dataset&project=qwiklabs-gcp-220e4ec0b4fa40bc&folder&organizationId

Activity: Practice Exploring BigQuery Public Datasets

Navigate to the BigQuery Public Dataset page and find a dataset that interests you

1. What's the name of the dataset?
2. How many records are in the tables?
3. Are there any data quality concerns?
4. After analyzing the schema, what types of insights do you think you could find?
5. Are there any other datasets that you could join this one against for additional insights?

So let's do a quick activity. Explore a BigQuery public dataset that interests you and answer the following questions:

- What's the name of the dataset?
- How many records are in the tables?
- Are there any data quality concerns?
- After analyzing the schema, what types of insights do you think you could find?
- Are there any other datasets that you could join this one against for additional insights?

Take a moment to find and explore your dataset and then we will bring in the head of the BigQuery Public Datasets Program, Shane Glass, to explore one of his favorites in a demo.

Demo:
Analyzing lightning strikes with BigQuery GIS

Google BigQuery + Public Datasets Program

https://cloud.google.com/marketplace/
https://cloud.google.com/bigquery/docs/gis-intro

Agenda

Introduction to BigQuery

- Fast SQL Query Engine
- Managed Storage for Datasets

Insights from Geographic data

Machine Learning on Structured Data

- Choosing the right model type
- Scenario: Predicting Customer Lifetime Value

Creating ML models with SQL

- Introduction to BigQuery ML
- ML project phases
- Key features walkthrough

It's time to revisit the exciting topic of machine learning. Previously in the course, your data science team had build a recommendation model using SparkML that you ran on Cloud Dataproc.

Later in this course, you're going to be building custom models yourself with just SQL using BigQuery ML. But before we jump into the code, we need to expand our machine learning foundation and cover the models and key terminology you need to know before you set off to build your models.

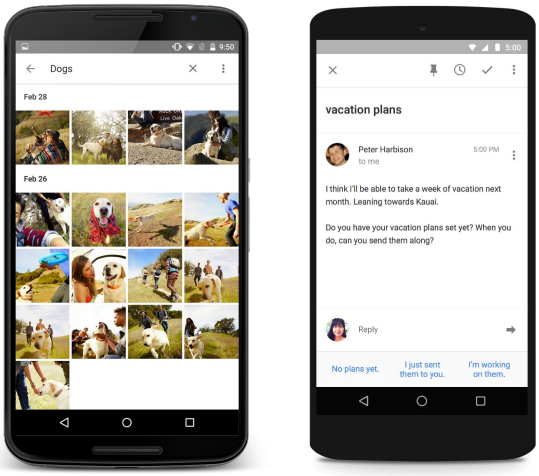
When you hear “AI or ML,” you probably think of:

Image models

Sequence models

CNNs

RNNs



When people think of AI or Machine Learning they generally think of the advanced models like those you saw earlier for Google Photos video stabilization and Smart Reply in Gmail. And yes, later in this course you built image models on unstructured datasets, but did you know that at Google …

ML on structured data drives value

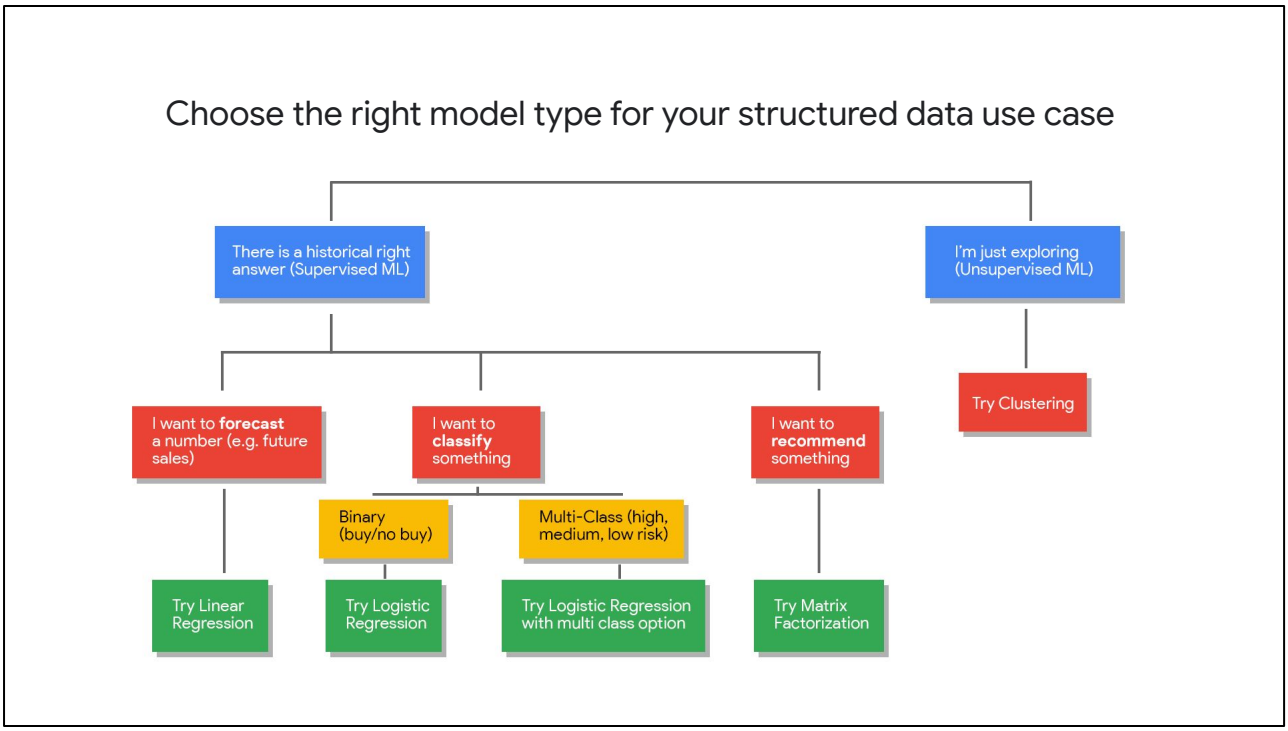
The most common ML models at Google are those that operate on structured data

Type of network	# of network layers	# of weights	% of deployed models
MLP0	5	20M	61%
MLP1	4	5M	
LSTM0	58	52M	29%
LSTM1	56	34M	
CNN0	16	8M	5%
CNN1	89	100M	

Source (2017): https://cloud.google.com/blog/big-data/2017/05/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu

... the majority of models deployed are models that operate on structured data? These aren't your 50+ layer deep neural networks that play Starcraft or Chess. They're models built on rows and columns of data just like you've seen in BigQuery.

Source: https://cloud.google.com/blog/big-data/2017/05/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu

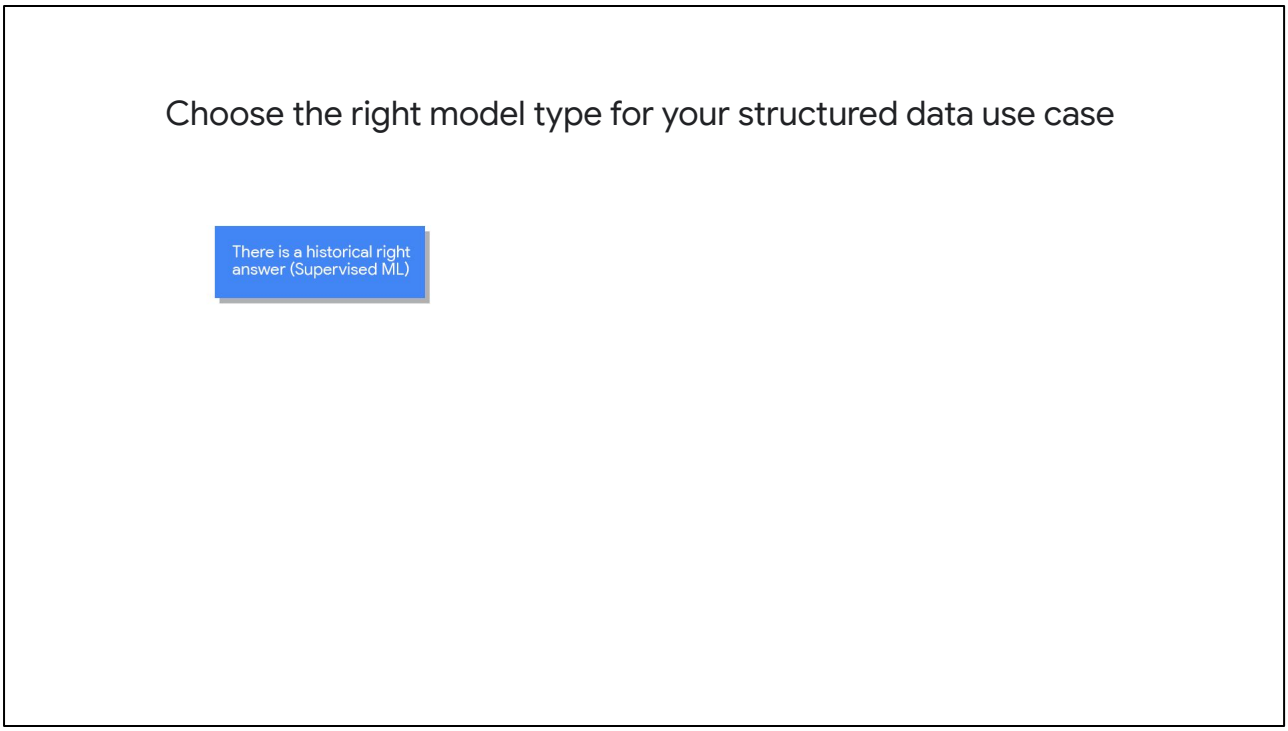


So, if you have a structured dataset that you think is a good use case for machine learning the next step is to find a model type that is appropriate for your use case.

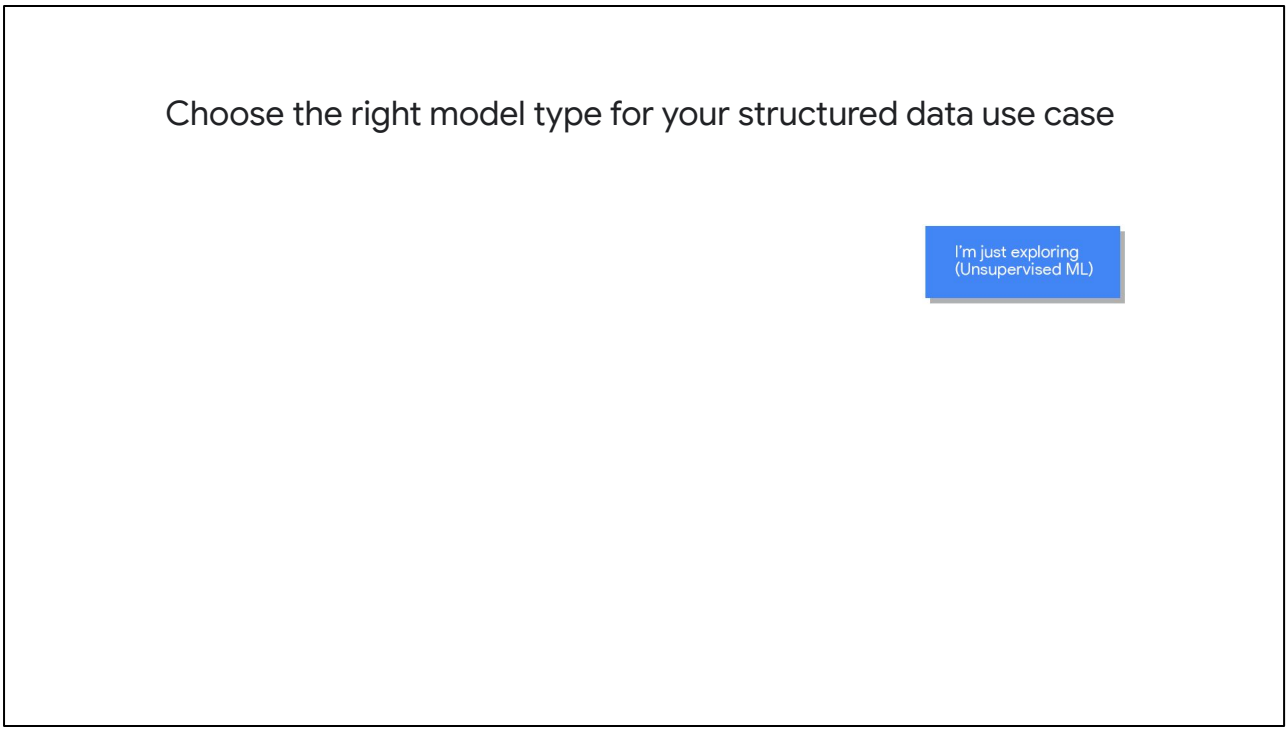
Out of all the models out there, what's a good place for you to start for simple prototyping?

Here's a decision tree to help guide us. We'll walk through each of the branches.

The first question is what kind of activity you're engaging in. Is there a right answer or ground truth that exists in your historical data that you want to model?



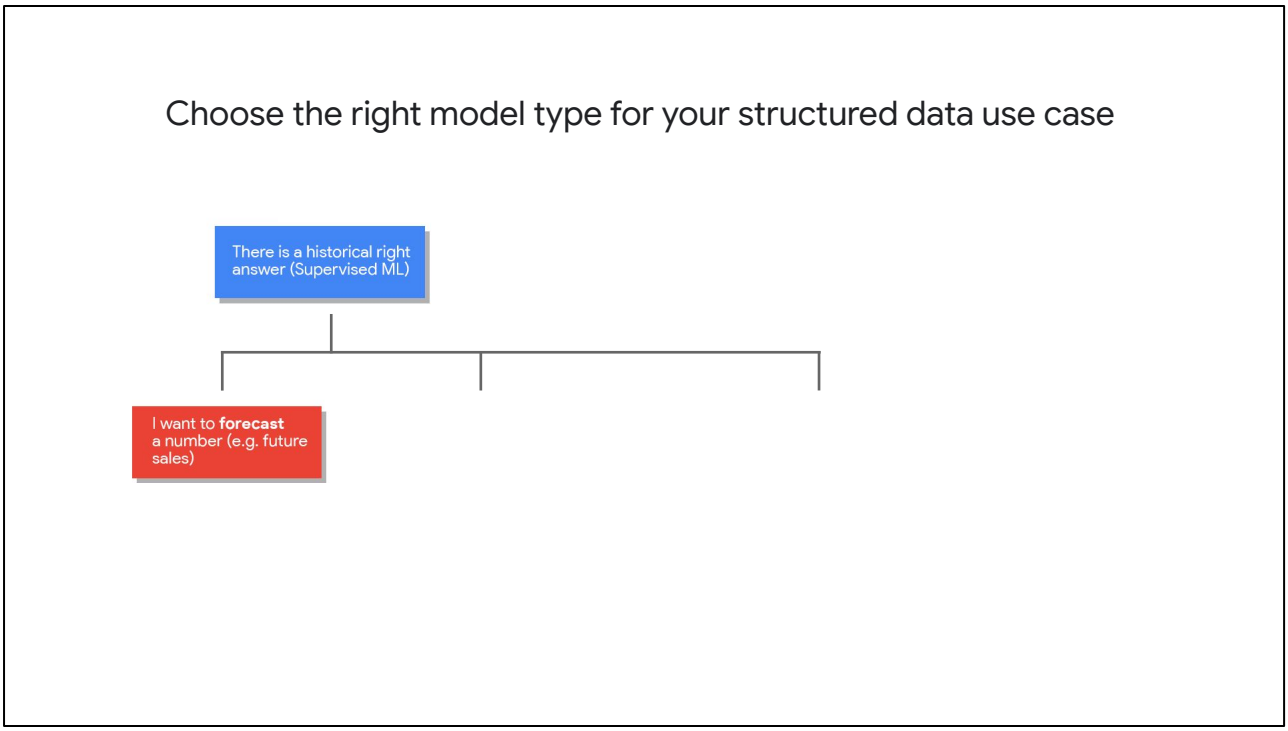
If so, you'll want to start with supervised learning. Alternatively, if you are interested in exploring your data for unknown relationships try unsupervised learning with a clustering model to start.



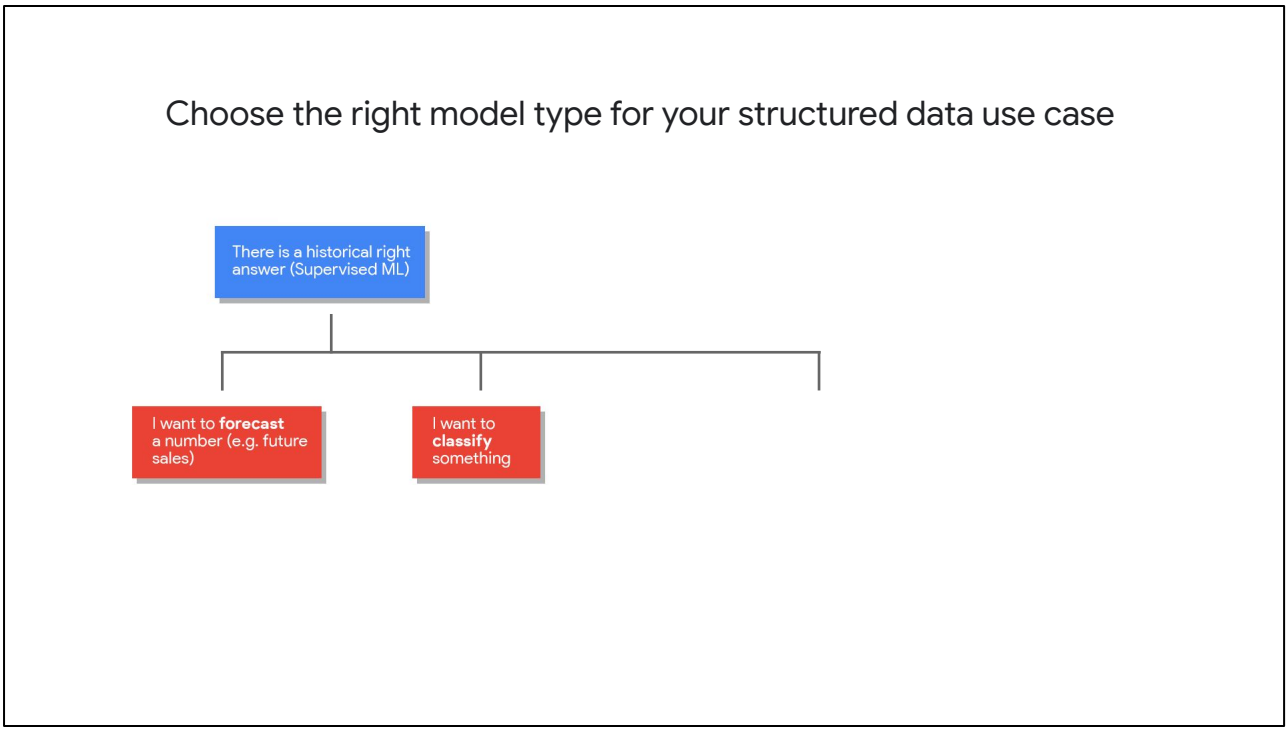
Unsupervised learning is outside the scope of this course but I'll link you a few resources to show you how it can be done quickly with BigQuery ML.

Choose the right model type for your structured data use case

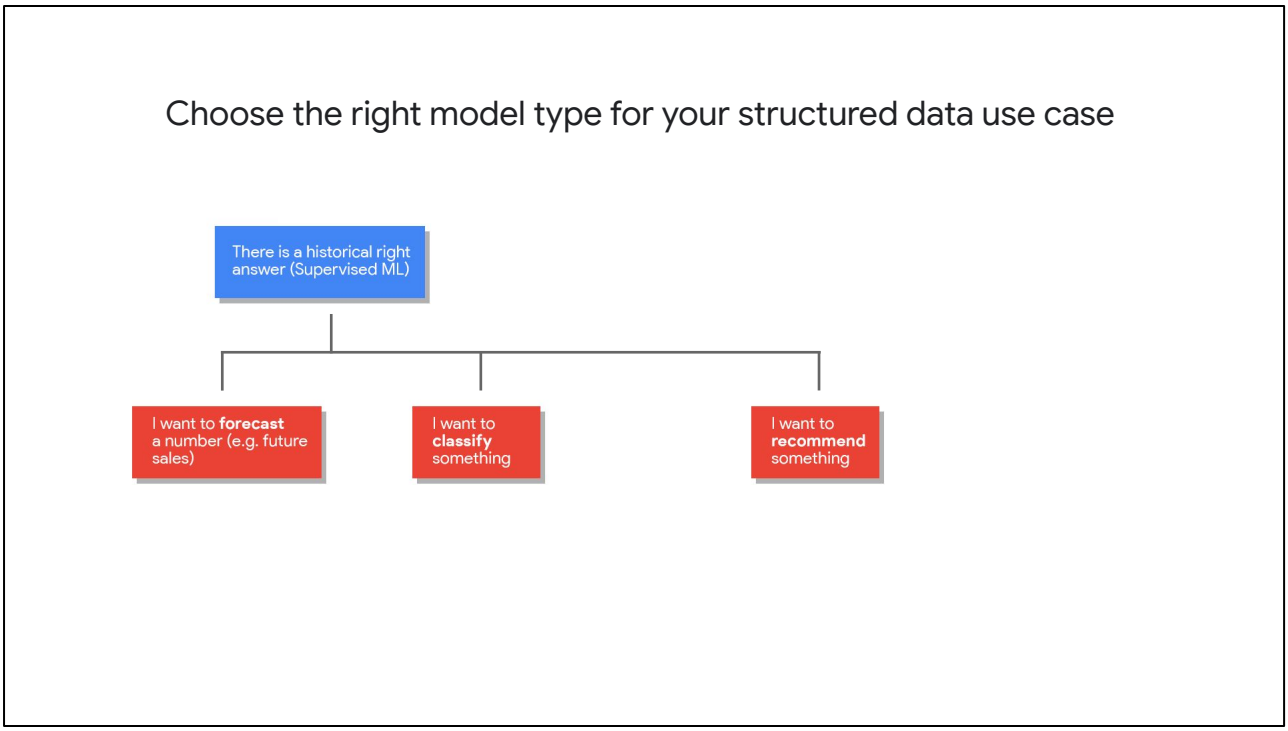
The majority of the problems we're going to tackle here are these three:



Forecasting - like predicting next month’s sales figures or demand for your product.



Classifying - like high, medium, low risk or buy/no buy or



Recommending something.

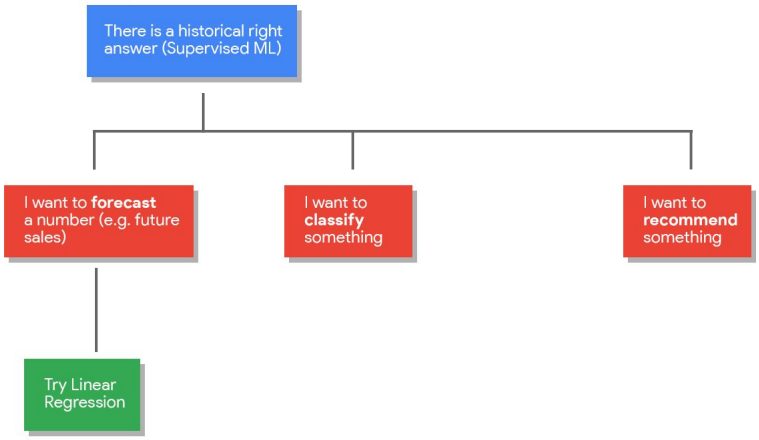
An easy way to tell if you're forecasting or classifying is to look at the type of label (column) of data you are predicting:

- Generally, If it's a numeric data type (like units sold or profits earned), you're doing forecasting
- If it's a string value you're doing classification (this row is either this in class or this other class)
- ... and If you have more than two classes (like high, low, or medium) you are doing multi-class classification

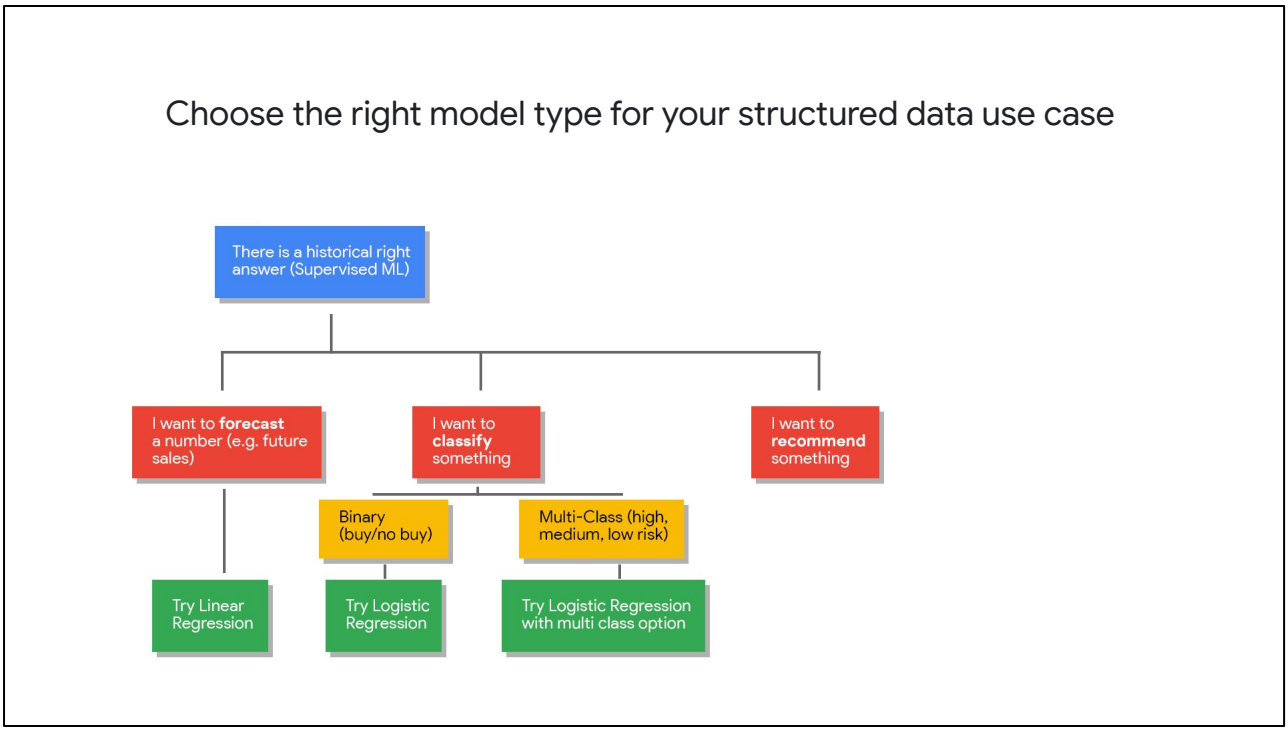
Once you have your problem outlined, it’s time to go shopping for models which are tools to help to achieve your goal.

Now there are many different model types you can choose from for these problems. We’re recommending you start with these simpler ones (which can still be highly accurate) to see if they meet your benchmark. By the way, your ML benchmark is the performance threshold you’re willing to accept from your model before you allow it to be used in production. It’s critical that you set your benchmark before you train a model so you can be objective.

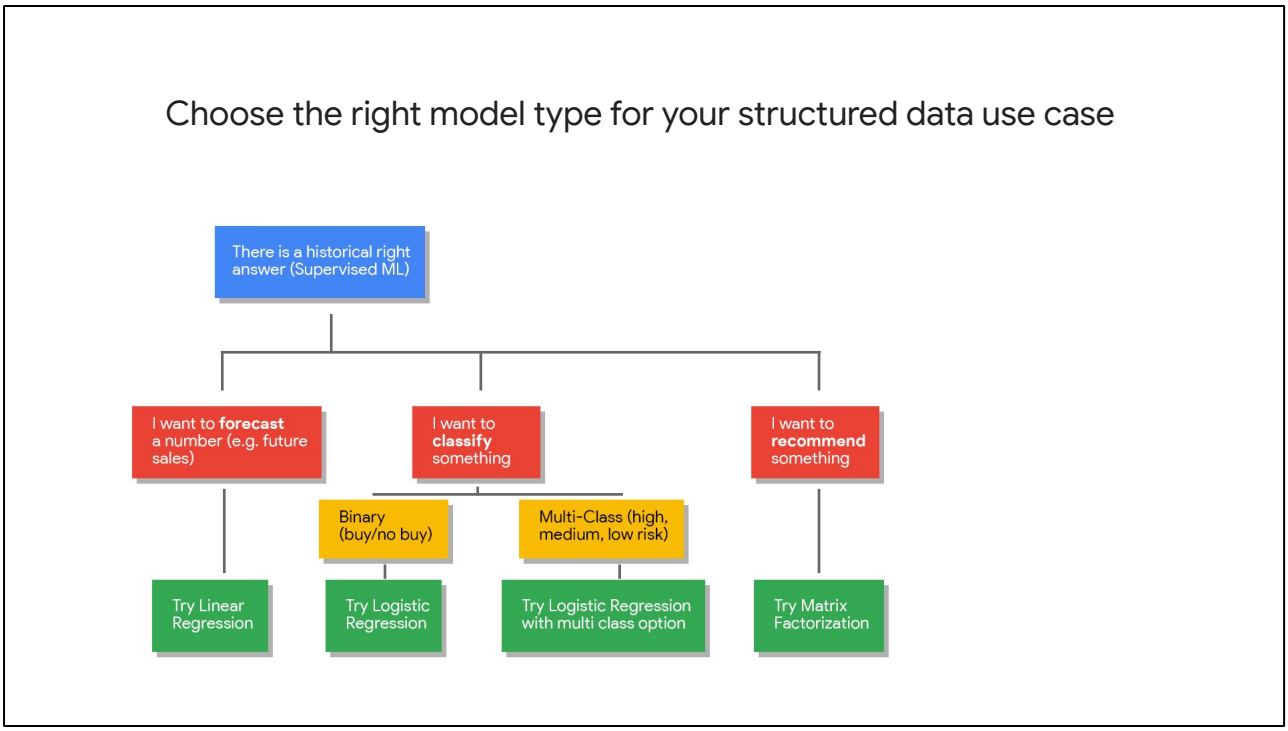
Choose the right model type for your structured data use case



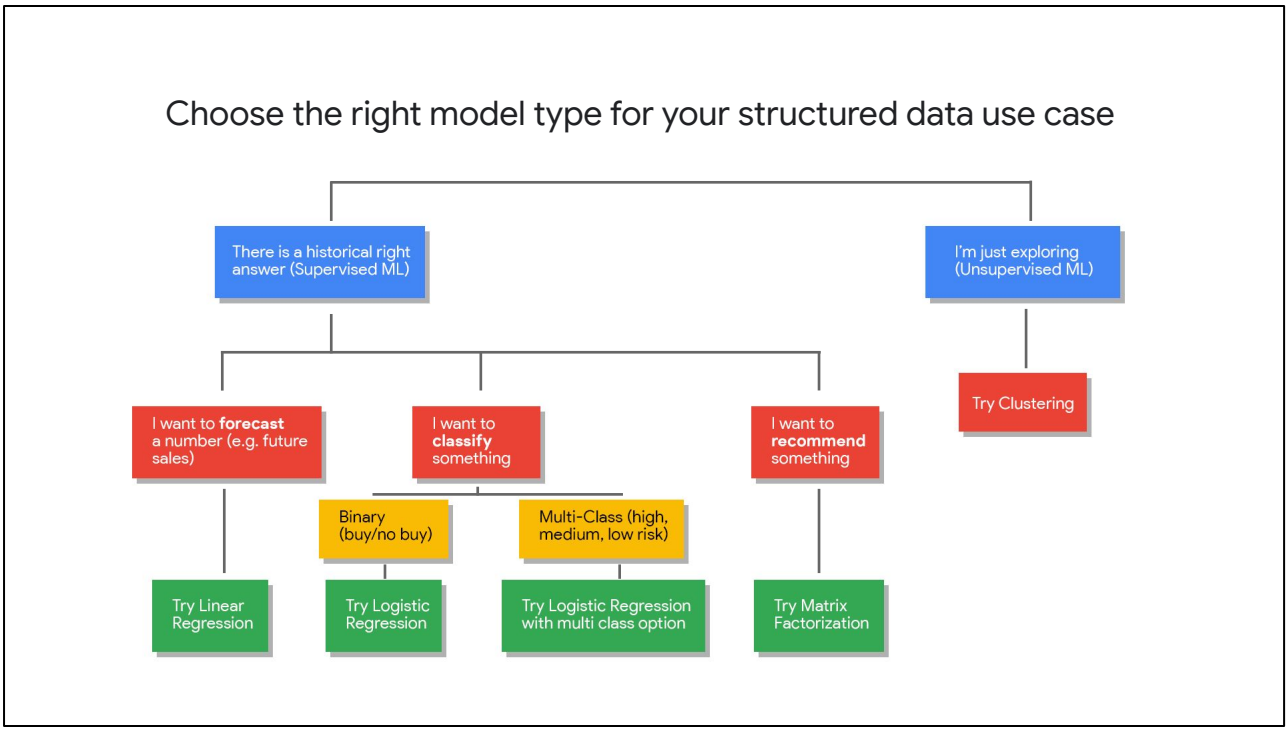
Now onto types of models. For forecasting, try linear regression



For classification, try logistic regression. It's called binary logistic regression if you have just two classes or buckets that an observation can fall into or multi-class if it's more than two.



For recommendations, try matrix factorization which is a commonly used algorithm for problems involving a matrix of users and items.



Here's the complete picture again. You'll see later in BigQuery ML that you can just specify model type equal 'linear regression' and BigQuery handles the rest for you.

What didn't you see here that you may have heard of?

There are many different types of models out there that you may not see on this chart. More complex models like deep neural networks, decision trees, random forests are also available for modelling. You'll even build a custom model using Neural Architecture Search to build a Deep Neural Network later in this course without using any code with AutoML. It's my recommendation that even if you know how to build advanced models that you start with simpler ones first because they often train faster and give you an indication of whether or not ML is a viable solution for your problem.

Agenda

Introduction to BigQuery

- Fast SQL Query Engine
- Managed Storage for Datasets

Insights from Geographic data

Machine Learning on Structured Data

- Choosing the right model type
- Scenario: Predicting Customer Lifetime Value

Creating ML models with SQL

- Introduction to BigQuery ML
- ML project phases
- Key features walkthrough

Now that you are familiar with the types of models to choose from, we need to feed them high quality training data for them to learn. That's the learn part of machine learning. Machine is the tool or algorithm like linear regression and learning is the insights it has into the relationships between known and unknown data.

The best way I find to learn the key concepts of machine learning on structured datasets is with an example.

A quick example

Predicting customer LTV **with a ML model**

Here our quick scenario will be predicting customer lifetime value with a model. Lifetime value or LTV is a common metric in marketing which is where we'll estimate how much revenue or profit we can expect from a customer given their history and customers with similar patterns.

ML for Customer LTV

ABC hits_product_v2ProductName
177 Categories
Google · Men's · Vintage · Badge · Tee · Black
Google · Women's · 1/4 · Zip · Performance · Pullover · Two-Tone · Blue
Google · Men's · Lightweight · Microfleece · Jacket · Black
Google · Men's · Lightweight · Microfleece · Jacket · Black
Google · Women's · Quilted · Insulated · Vest · Black
Google · Women's · Lightweight · Microfleece · Jacket
Google · Women's · Lightweight · Microfleece · Jacket
Women's · Weatherblock · Shell · Jacket · Black
Men's · Weatherblock · Shell · Jacket · Black

The dataset that we'll use is a Google Analytics ecommerce dataset on Google's own merchandise store that sells swag like T-Shirts and other products you see listed here.

Our goal is to target high value customers to our ecommerce throughout their customer lifecycle with special promotions and incentives.

Predict Lifetime Value (LTV) of a Customer

Results | Details

Row	fullVisitorId	distinct_days_visited	ltv_pageviews	ltv_visits	ltv_avg_time_on_site_s	ltv_revenue	ltv_transactions
1	7813149961404844386	79	1395	138	479.63	6245720000	67
2	7713012430069756739	2	514	6	1954.33	181940000	35
3	6760732402251466726	30	868	41	723.55	4812820000	34
4	5526675926038480325	1	466	1	7013.0	87960000	25
5	1957458976293878100	148	4303	284	796.46	77113430000	22
6	4983264713224875783	2	366	4	3807.5	74850000	21
7	2402527199731150932	28	559	31	906.61	3270100000	19

After exploring the available data, we came up with a few fields that we thought might be useful to the model in determining whether a customer is high value based on their behavior on our website. These fields include how many lifetime pageviews, how many total visits, what their average time on site was, the total revenue brought in, and the count of ecommerce transactions on our site.

Now that we have some data (and note that we'd need a lot more than just 7 records, it would be more like tens of thousands) we can get ready to feed it into our model.

But, before we do, we first need to define our data and columns in the language that data scientists and other ML professionals use.

Query:
https://bigquery.cloud.google.com/savedquery/133415875420:a620c7a9058c4552b38d3b3e08037e48

An instance (or observation) is a row of data

Results | Details

Row	fullVisitorId	distinct_days_visited	ltv_pageviews	ltv_visits	ltv_avg_time_on_site_s	ltv_revenue	ltv_transactions	avg_session_quality	first_visit	last_visit	ltv_days
1	6007196403211981721	8	147	11	772.5	null	null	7.5	2016-08-04	2017-07-15	345
2	7587138749751940102	9	94	9	312.33	24380000	1	1.0	2016-08-03	2017-07-14	345
3	0720311197761340948	114	148	146	2118.0	null	null	1.0	2016-08-05	2017-07-15	344
4	9557989866096732580	3	18	3	356.5	null	null	1.0	2016-08-03	2017-07-13	344
5	0824839726118485274	127	3153	282	1520.0	null	null	26.0	2016-08-01	2017-07-10	343
6	2742641486650042668	17	113	20	266.28	387000000	2	23.0	2016-08-02	2017-07-11	343
7	1957458976293878100	148	4303	284	796.46	77113430000	22	1.5	2016-08-04	2017-07-12	342
8	1950585318332186454	6	19	7	51.4	null	null	1.5	2016-08-05	2017-07-11	340

Taking the ecommerce example we had in the previous lesson, a record or row is called an instance or observation. In the screenshot you see here we have 8 instances.

Query:
https://bigquery.cloud.google.com/savedquery/133415875420:a620c7a9058c4552b38d3b3e08037e48

A label is the correct answer

Results | Details

Row	fullVisitorId	distinct_days_visited	ltv_pageviews	ltv_visits	ltv_avg_time_on_site_s	ltv_revenue	ltv_transactions	avg_session_quality	first_visit	last_visit	ltv_days
1	6007196403211981721	8	147	11	772.5	null	null	7.5	2016-08-04	2017-07-15	345
2	7587138749751940102	9	94	9	312.33	24380000	1	1.0	2016-08-03	2017-07-14	345
3	0720311197761340948	114	148	146	2118.0	null	null	1.0	2016-08-05	2017-07-15	344
4	9557989866096732580	3	18	3	356.5	null	null	1.0	2016-08-03	2017-07-13	344
5	0824839726118485274	127	3153	282	1520.0	null	null	26.0	2016-08-01	2017-07-10	343
6	2742641486650042668	17	113	20	266.28	387000000	2	23.0	2016-08-02	2017-07-11	343
7	1957458976293878100	148	4303	284	796.46	77113430000	22	1.5	2016-08-04	2017-07-12	342
8	1950585318332186454	6	19	7	51.4	null	null	1.5	2016-08-05	2017-07-11	340

A label is the correct answer that you know historically (like how much this particular customer spent) and it is what you are looking to train the model to predict on future data. For example, if we know that a customer who has made transactions in the past and spends a lot of time on our website often turns out to have high LTV Revenue we could predict the same for newer customers on that same spending trajectory. Here we're forecasting a number so we'd use something like linear regression as a starting point to model.

Notice how I didn't say that IF transactions > 10 THEN expect Revenue to be greater than 1,000. Remember in ML we feed in columns of data and let the model figure out the relationship to best predict the label. It may even turn out that some of the columns were not useful at all to the model in predicting the outcome -- I'll show you how you can find this out later.

Query:
https://bigquery.cloud.google.com/savedquery/133415875420:a620c7a9058c4552b38d3b3e08037e48

A label is the correct answer

Row	fullVisitorId	distinct_days_visited	ltv_pageviews	ltv_visits	ltv_avg_time_on_site_s	ltv_revenue	ltv_transactions	avg_session_quality	first_visit	last_visit	ltv_days	label
1	7587138749751940102	9	94	9	312.33	24380000	1	1.0	2016-08-03	2017-07-14	345	High Value Customer
2	6007196403211981721	8	147	11	772.5	null	null	7.5	2016-08-04	2017-07-15	345	
3	9557989866096732580	3	18	3	356.5	null	null	1.0	2016-08-03	2017-07-13	344	
4	0720311197761340948	114	148	146	2118.0	null	null	1.0	2016-08-05	2017-07-15	344	
5	2742641486650042668	17	113	20	266.28	387000000	2	23.0	2016-08-02	2017-07-11	343	High Value Customer
6	0824839726118485274	127	3153	282	1520.0	null	null	26.0	2016-08-01	2017-07-10	343	
7	1957458976293878100	148	4303	284	796.46	77113430000	22	1.5	2016-08-04	2017-07-12	342	High Value Customer
8	9801276214964695322	79	462	106	219.44	null	null	1.5	2016-08-01	2017-07-07	340	
9	1950585318332186454	6	19	7	51.4	null	null	1.5	2016-08-05	2017-07-11	340	
10	0084834161383601528	7	97	7	258.0	69260000	2	2.0	2016-08-04	2017-07-10	340	High Value Customer
11	9283984083989251 52	40	553	43	285.37	462190000	2	2.0	2016-08-02	2017-07-07	339	High Value Customer
12	3512777258200611	20	60	20	221.33	null	null	1.0	2016-08-05	2017-07-10	339	
13	4143624098732715494	6	13	7	52.5	null	null	1.0	2016-08-03	2017-07-08	339	
14	1927175312147751345	13	180	14	427.21	44970000	1	2.0	2016-08-03	2017-07-08	339	High Value Customer
15	1315777378660606104	28	272	26	340.3	279320000	3	21.25	2016-08-09	2017-07-14	339	High Value Customer

Labels could also be things like binary values like “High Value Customer” or not as shown here. Knowing what you are trying to predict (a class, a number etc) will greatly influence the type of model you will use.

Query:
https://bigquery.cloud.google.com/savedquery/133415875420:a620c7a9058c4552b38d3b3e08037e48

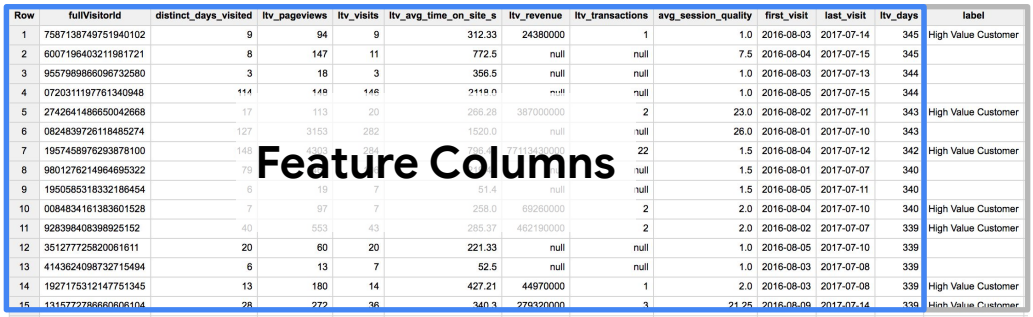
What about the other columns?

Row	fullVisitorId	distinct_days_visited	ltv_pageviews	ltv_visits	ltv_avg_time_on_site_s	ltv_revenue	ltv_transactions	avg_session_quality	first_visit	last_visit	ltv_days	label
1	7587138749751940102	9	94	9	312.33	24380000	1	1.0	2016-08-03	2017-07-14	345	High Value Customer
2	6007196403211981721	8	147	11	772.5	null	null	7.5	2016-08-04	2017-07-15	345	
3	9557989866096732580	3	18	3	356.5	null	null	1.0	2016-08-03	2017-07-13	344	
4	0720311197761340948	114	148	146	2118.0	null	null	1.0	2016-08-05	2017-07-15	344	
5	2742641486650042668	17	113	20	266.28	387000000	2	23.0	2016-08-02	2017-07-11	343	High Value Customer
6	0824839726118485274	127	3153	282	1520.0	null	null	26.0	2016-08-01	2017-07-10	343	
7	1957458976293878100	148	4303	284	796.46	77113430000	22	1.5	2016-08-04	2017-07-12	342	High Value Customer
8	9801276214964695322	79	462	106	219.44	null	null	1.5	2016-08-01	2017-07-07	340	
9	1950585318332186454	6	19	7	51.4	null	null	1.5	2016-08-05	2017-07-11	340	
10	0084834161383601528	7	97	7	258.0	69260000	2	2.0	2016-08-04	2017-07-10	340	High Value Customer
11	928398408398925152	40	553	43	285.37	462190000	2	2.0	2016-08-02	2017-07-07	339	High Value Customer
12	351277725820061611	20	60	20	221.33	null	null	1.0	2016-08-05	2017-07-10	339	
13	4143624098732715494	6	13	7	52.5	null	null	1.0	2016-08-03	2017-07-08	339	
14	1927175312147751345	13	180	14	427.21	44970000	1	2.0	2016-08-03	2017-07-08	339	High Value Customer
15	1315772786660606104	28	272	36	340.3	279320000	3	21.25	2016-08-09	2017-07-14	339	High Value Customer

So we've got the label field there on the right as High Value customer or not, what do we call all these other data columns in our data table?

Query:
https://bigquery.cloud.google.com/savedquery/133415875420:a620c7a9058c4552b38d3b3e08037e48

What about the other columns?



Row	fullVisitorId	distinct_days_visited	ltv_pageviews	ltv_visits	ltv_avg_time_on_site_s	ltv_revenue	ltv_transactions	avg_session_quality	first_visit	last_visit	ltv_days	label
1	7587138749751940102	9	94	9	312.33	24380000	1	1.0	2016-08-03	2017-07-14	345	High Value Customer
2	6007196403211981721	8	147	11	772.5	null	null	7.5	2016-08-04	2017-07-15	345	
3	9557989866096732580	3	18	3	356.5	null	null	1.0	2016-08-03	2017-07-13	344	
4	0720311197761340948	114	149	146	2118.0	null	null	1.0	2016-08-05	2017-07-15	344	
5	2742641486650042668	17	113	20	266.28	387000000	2	23.0	2016-08-02	2017-07-11	343	High Value Customer
6	0824839726118485274	127	3153	282	1520.0	null	null	26.0	2016-08-01	2017-07-10	343	
7	1957458976293878100	148	4393	284	796.5	77113430000	22	1.5	2016-08-04	2017-07-12	342	High Value Customer
8	9801276214964695322	79	1	1	0.0	null	null	1.5	2016-08-01	2017-07-07	340	
9	1950585318332186454	6	19	7	51.4	null	null	1.5	2016-08-05	2017-07-11	340	
10	0084834161383601528	7	97	7	258.0	69260000	2	2.0	2016-08-04	2017-07-10	340	High Value Customer
11	928398408398925152	40	553	43	285.37	462190000	2	2.0	2016-08-02	2017-07-07	339	High Value Customer
12	351277725820061611	20	60	20	221.33	null	null	1.0	2016-08-05	2017-07-10	339	
13	4143624098732715494	6	13	7	52.5	null	null	1.0	2016-08-03	2017-07-08	339	
14	1927175312147751345	13	180	14	427.21	44970000	1	2.0	2016-08-03	2017-07-08	339	High Value Customer
15	1315772786660606104	28	272	36	340.3	279320000	3	21.25	2016-08-09	2017-07-14	339	High Value Customer

Those columns are called features -- or potential features at least. Each column of data is like a cooking ingredient you can use from the kitchen pantry. As my young daughter often finds out, cooking with too many ingredients can spoil the dinner.

This process of sifting through your data is where you can expect to spend the majority of you time as a data analyst, engineer, or data scientist. Understanding the quality of the data in each column and working with teams to get more features or more history is often the hardest part of any ML project. You can even combine or transform these feature columns in a process called Feature Engineering. Sounds fancy right? Well - if you've ever created calculated fields in SQL (like combining columns) you've just done the basics of feature engineering. Also, BigQuery ML does a lot of the hard work for you like automatic one-hot encoding of categorical variables and splitting your dataset into training and evaluation automatically too.

Query:
https://bigquery.cloud.google.com/savedquery/133415875420:a620c7a9058c4552b38d3b3e08037e48

What if I don't know where a new customer will fit?

Historical Training Data (Known LTV)

Row	fullVisitorId	distinct_days_visited	ltv_pageviews	ltv_visits	ltv_avg_time_on_site_s	ltv_revenue	ltv_transactions	avg_session_quality	first_visit	last_visit	ltv_days	label
1	7587138749751940102	9	94	9	312.33	24380000	1	1.0	2016-08-03	2017-07-14	345	High Value Customer
2	6007196403211981721	8	147	11	772.5	null	null	7.5	2016-08-04	2017-07-15	345	
3	9557989866096732580	3	18	3	356.5	null	null	1.0	2016-08-03	2017-07-13	344	
4	0720311197761340948	114	148	146	2118.0	null	null	1.0	2016-08-05	2017-07-15	344	
5	2742641486650042668	17	113	20	266.28	387000000	2	23.0	2016-08-02	2017-07-11	343	High Value Customer
6	0824839726118485274	127	3153	282	1520.0	null	null	26.0	2016-08-01	2017-07-10	343	
7	1957458976293878100	148	4303	284	796.46	77113430000	22	1.5	2016-08-04	2017-07-12	342	High Value Customer
8	9801276214964695322	79	462	106	219.44	null	null	1.5	2016-08-01	2017-07-07	340	
9	1950585318332186454	6	19	7	51.4	null	null	1.5	2016-08-05	2017-07-11	340	
10	0084834161383601528	7	97	7	258.0	69260000	2	2.0	2016-08-04	2017-07-10	340	High Value Customer
11	928398408398925152	40	553	43	285.37	462190000	2	2.0	2016-08-02	2017-07-07	339	High Value Customer
12	351277725820061611	20	60	20	221.33	null	null	1.0	2016-08-05	2017-07-10	339	
13	4143624098732715494	6	13	7	52.5	null	null	1.0	2016-08-03	2017-07-08	339	
14	1927175312147751345	13	180	14	427.21	44970000	1	2.0	2016-08-03	2017-07-08	339	High Value Customer
15	1315772786660606104	28	272	36	340.3	279320000	3	21.25	2016-08-09	2017-07-14	339	High Value Customer

Future Data (Unknown LTV)

17	7904807859681747547	3	42	3	1162.0	null	null	1.0	2016-08-05	2017-07-09	338	??????????????????
18	4405445121320750966	51	358	62	517.36	null	null	1.0	2016-08-08	2017-07-12	338	??????????????????
19	1419607020881916790	5	22	5	711.0	null	null	1.0	2016-08-12	2017-07-15	337	??????????????????
20	3862335714593915688	13	92	16	154.23	238000000	1	2.0	2016-08-09	2017-07-12	337	??????????????????

Let's talk now about predicting on future data.

Say some new data comes in that you don't have a label for -- you don't know whether they will be a high value customer or not. But you do have a rich history of labeled examples for you to train a model on. So we train a model on the known data up top and then, once trained and we're happy with the performance, we can use it to predict on that bottom set of data.

Query:
https://bigquery.cloud.google.com/savedquery/133415875420:a620c7a9058c4552b38d3b3e08037e48

Agenda

Introduction to BigQuery

- Fast SQL Query Engine
- Managed Storage for Datasets

Insights from Geographic data

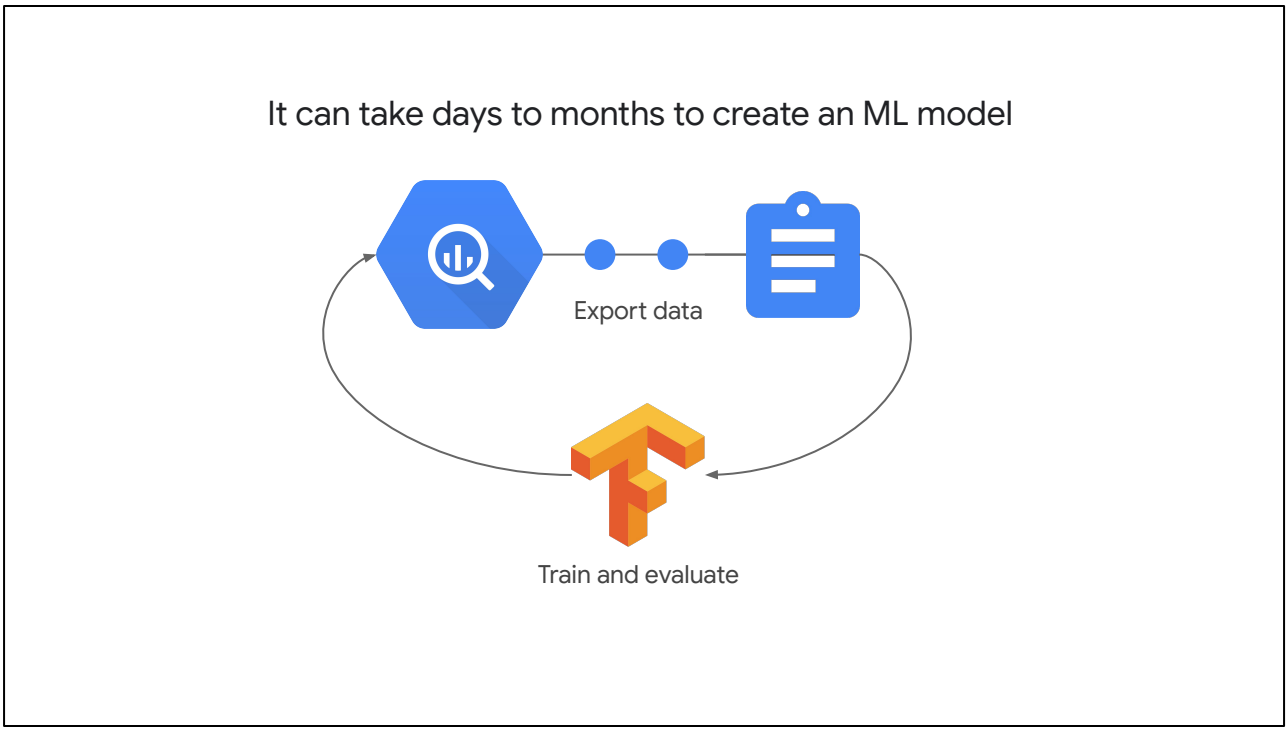
Machine Learning on Structured Data

- Choosing the right model type
- Scenario: Predicting Customer Lifetime Value

Creating ML models with SQL

- Introduction to BigQuery ML
- ML project phases
- Key features walkthrough

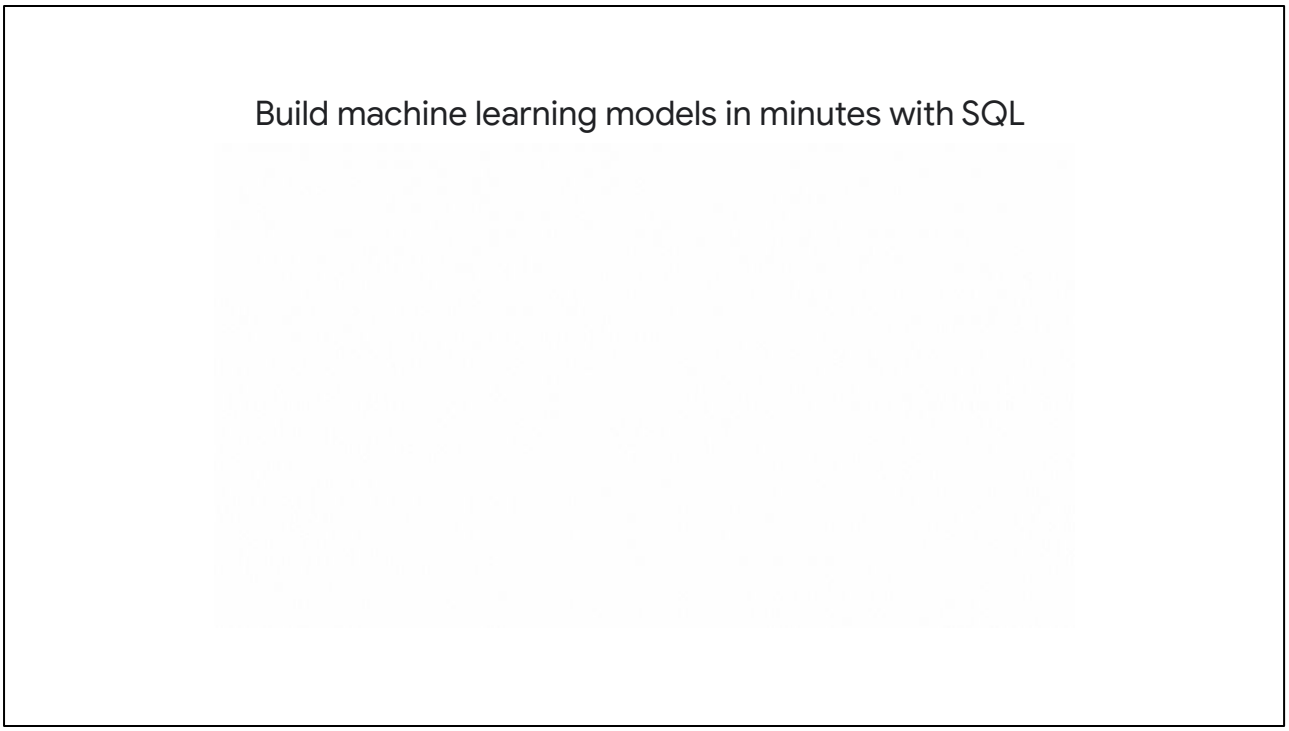
It's time to cover how you can create these models with SQL in BigQuery and what the project phases you can expect along the way.



If you've worked with ML models before, you know that building and training them can be very time intensive. You first must export small amount of data from your data store into your iPython notebook and into data handling frameworks like Pandas in Python. If you're building custom model, you first need to transform the data and perform all your feature engineering steps before you can feed the model data.

Then finally you build the model in Tensorflow or a similar library and train it locally on your laptop or on a VM. Doing that with a small model then requires that you go back and get more data to create new features, and improve performance. Repeat. It's hard, so you stop after a few iterations.

Also, I mentioned iPython notebooks and Python -- in the past if you weren't familiar with these technologies, ML was left to the data scientists on your team and out of your reach.



Now you can do ML on your structured datasets in BigQuery using SQL in just a few minutes.

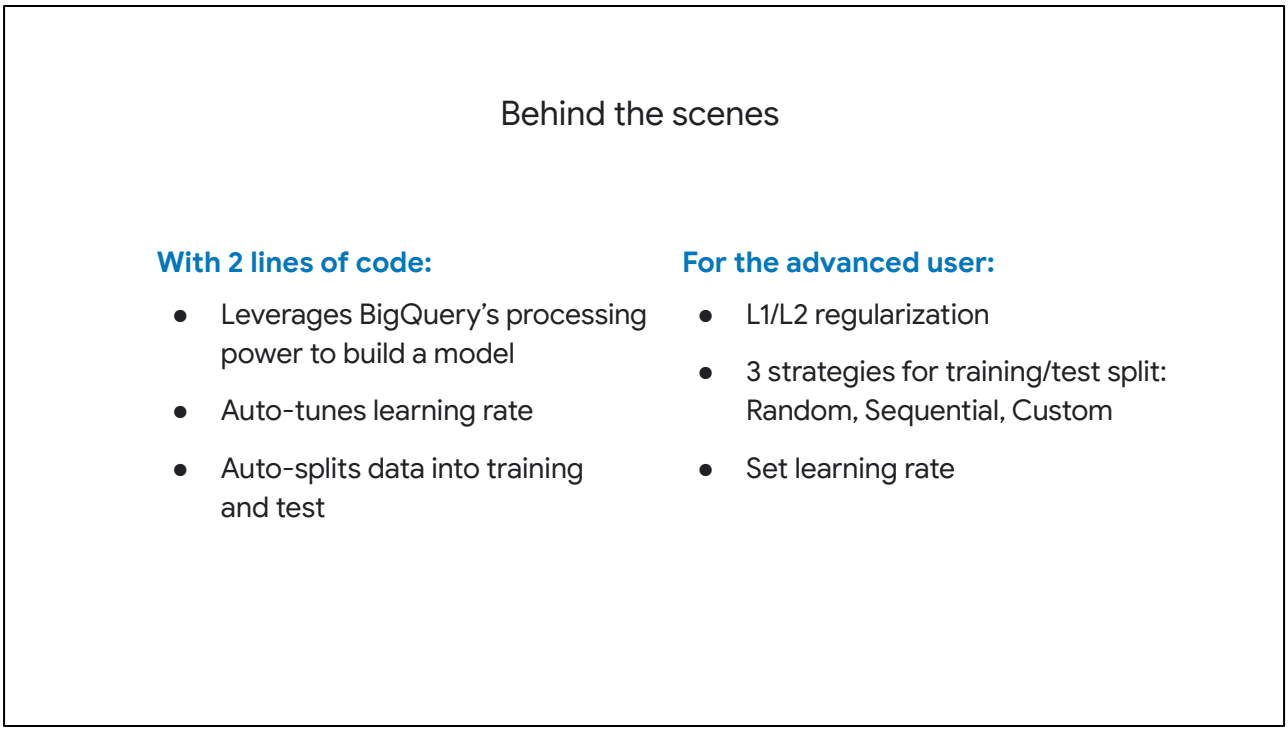
Take a look at how you can perform ML in BigQuery with two steps

Step 1: Create a model with just a SQL statement. Here we'll use a bikeshare dataset

Step 2: Write a SQL prediction query and invoke ml.Predict

Step 3: Profit -- that's it, you've got a model and can view the results.

In reality there are more than 3 steps (like evaluating the model) but if you know basic SQL you can now do ML which is pretty cool.



BigQuery ML was designed with simplicity in mind. To that end, you don't have to define the ML hyperparameters (a fancy way of saying knobs that are set on the model before the training starts) like the learning rate or even the training and test set split. BigQuery ML does that for you.

In addition, with OPTIONS if you wanted to you can also set regularization or different strategies for creating your training and test sets, and manually setting the learning rate.

Supported features

- StandardSQL and UDFs within the ML queries
- Linear Regression (Forecasting)
- Binary and Multi-Class Logistic Regression (Classification)
- Model evaluation functions for standard metrics, including ROC and precision-recall curves
- Model weight inspection
- Feature distribution analysis through standard functions

So what do you get out of the box?

Supported features

- StandardSQL and UDFs within the ML queries
- Linear Regression (Forecasting)
- Binary and Multi-Class Logistic Regression (Classification)
- Model evaluation functions for standard metrics, including ROC and precision-recall curves
- Model weight inspection
- Feature distribution analysis through standard functions

First, BigQuery ML runs on StandardSQL and you can use normal SQL syntax like UDFs, sub-queries, and joins to create your training datasets.

Supported features

- StandardSQL and UDFs within the ML queries
- Linear Regression (Forecasting)
- Binary and Multi-Class Logistic Regression (Classification)
- Model evaluation functions for standard metrics, including ROC and precision-recall curves
- Model weight inspection
- Feature distribution analysis through standard functions

For model types, you can either choose from linear regression for forecasting

Supported features

- StandardSQL and UDFs within the ML queries
- Linear Regression (Forecasting)
- Binary and Multi-Class Logistic Regression (Classification)
- Model evaluation functions for standard metrics, including ROC and precision-recall curves
- Model weight inspection
- Feature distribution analysis through standard functions

or binary logistic regression for classification.

Supported features

- StandardSQL and UDFs within the ML queries
- Linear Regression (Forecasting)
- Binary and Multi-Class Logistic Regression (Classification)
- Model evaluation functions for standard metrics, including ROC and precision-recall curves
- Model weight inspection
- Feature distribution analysis through standard functions

As part of your model evaluation, you will get access to fields like the ROC curve as well as accuracy, precision, and recall that you can simply SELECT from with SQL after your model is trained.

Supported features

- StandardSQL and UDFs within the ML queries
- Linear Regression (Forecasting)
- Binary and Multi-Class Logistic Regression (Classification)
- Model evaluation functions for standard metrics, including ROC and precision-recall curves
- Model weight inspection
- Feature distribution analysis through standard functions

You can also inspect the weights of the model

Supported features

- StandardSQL and UDFs within the ML queries
- Linear Regression (Forecasting)
- Binary and Multi-Class Logistic Regression (Classification)
- Model evaluation functions for standard metrics, including ROC and precision-recall curves
- Model weight inspection
- Feature distribution analysis through standard functions

and perform feature distribution analysis.

Next we'll walkthrough the phases of a typical BigQuery ML project.

Agenda

Introduction to BigQuery

- Fast SQL Query Engine
- Managed Storage for Datasets

Insights from Geographic data

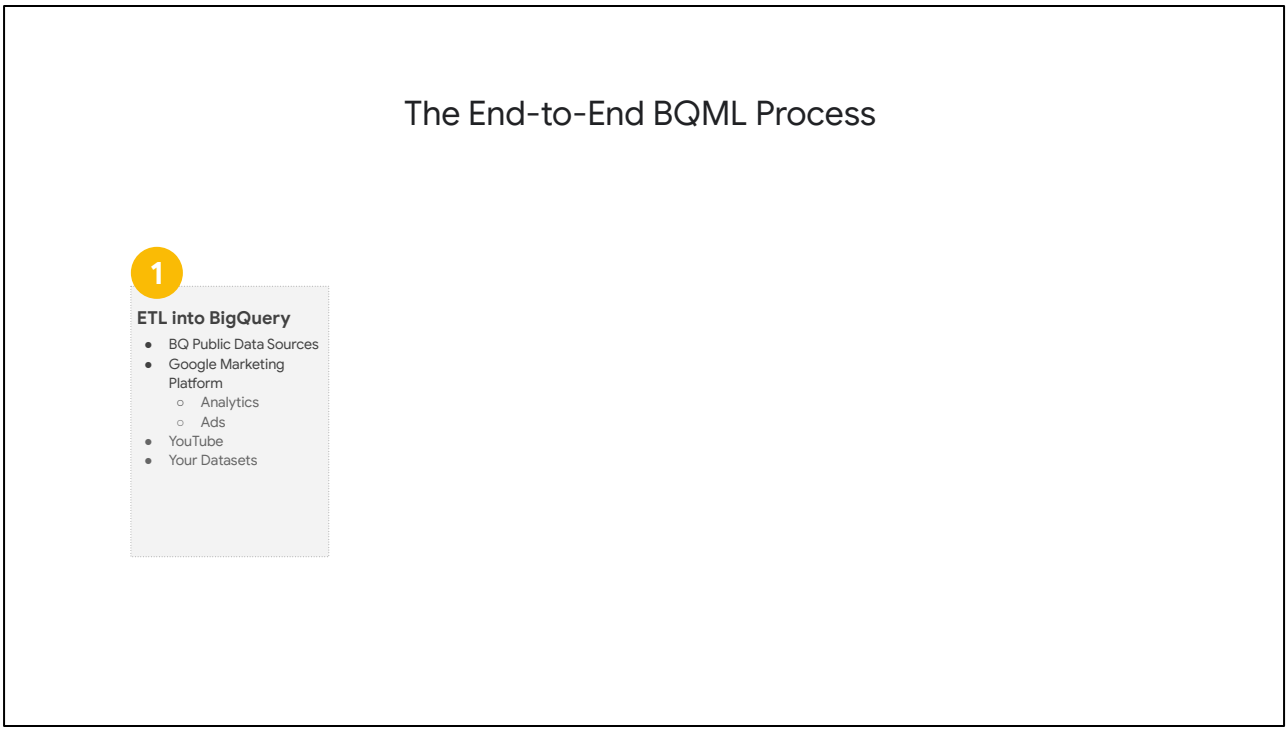
Machine Learning on Structured Data

- Choosing the right model type
- Scenario: Predicting Customer Lifetime Value

Creating ML models with SQL

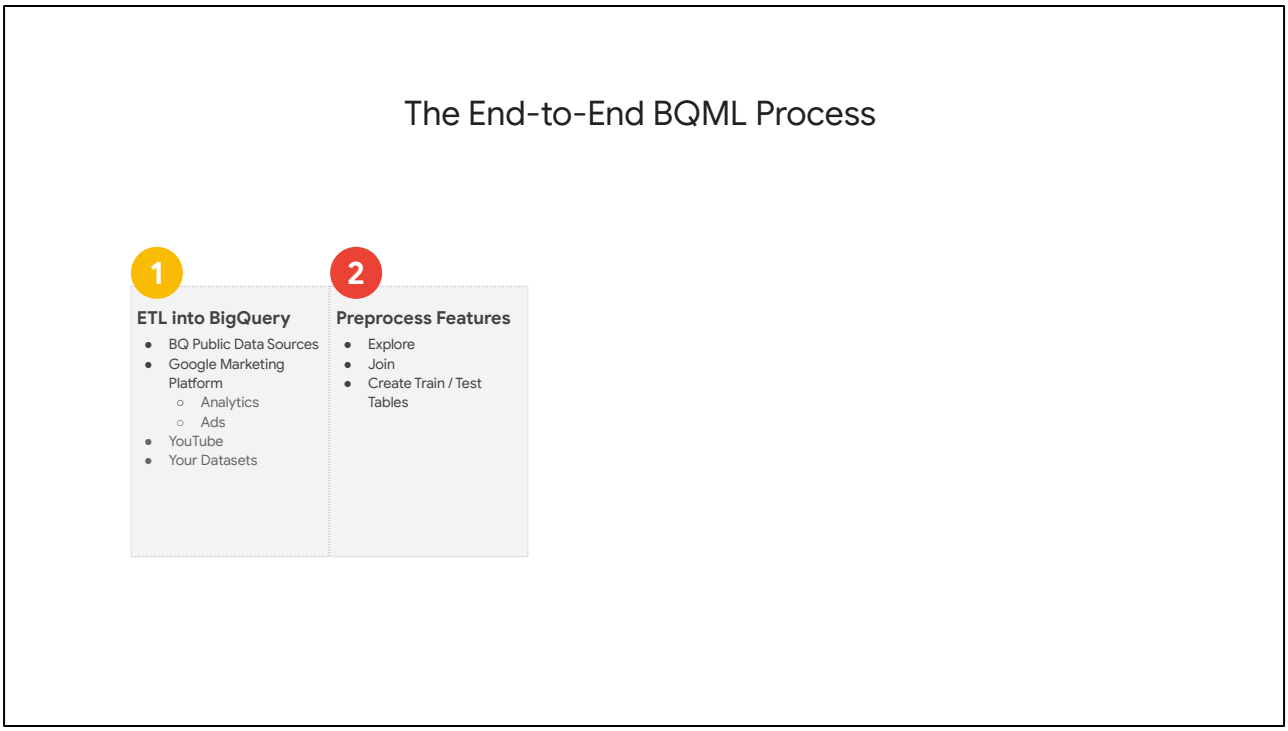
- Introduction to BigQuery ML
- ML project phases
- Key features walkthrough

Let's walkthrough the key phases of your ML project. What you'll find is that writing the code to create the actual model will be the easy part -- getting all the data for the model is the hard part.



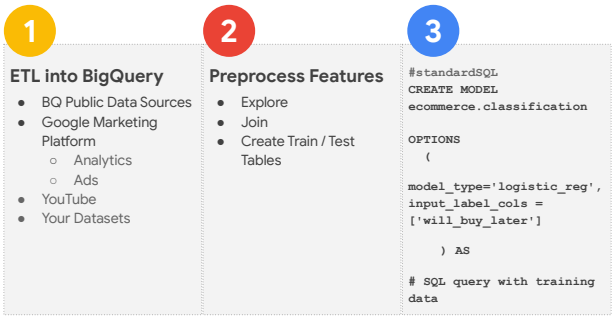
The entire process will look like this. First we need to bring our data into BigQuery if it isn't there already. Here, again, you can enrich your existing data warehouse with other data sources by simply using SQL joins.

If you're already using other Google products like AdWords or YouTube -- look out for easy connectors to get that data into BigQuery before you build your own pipeline.

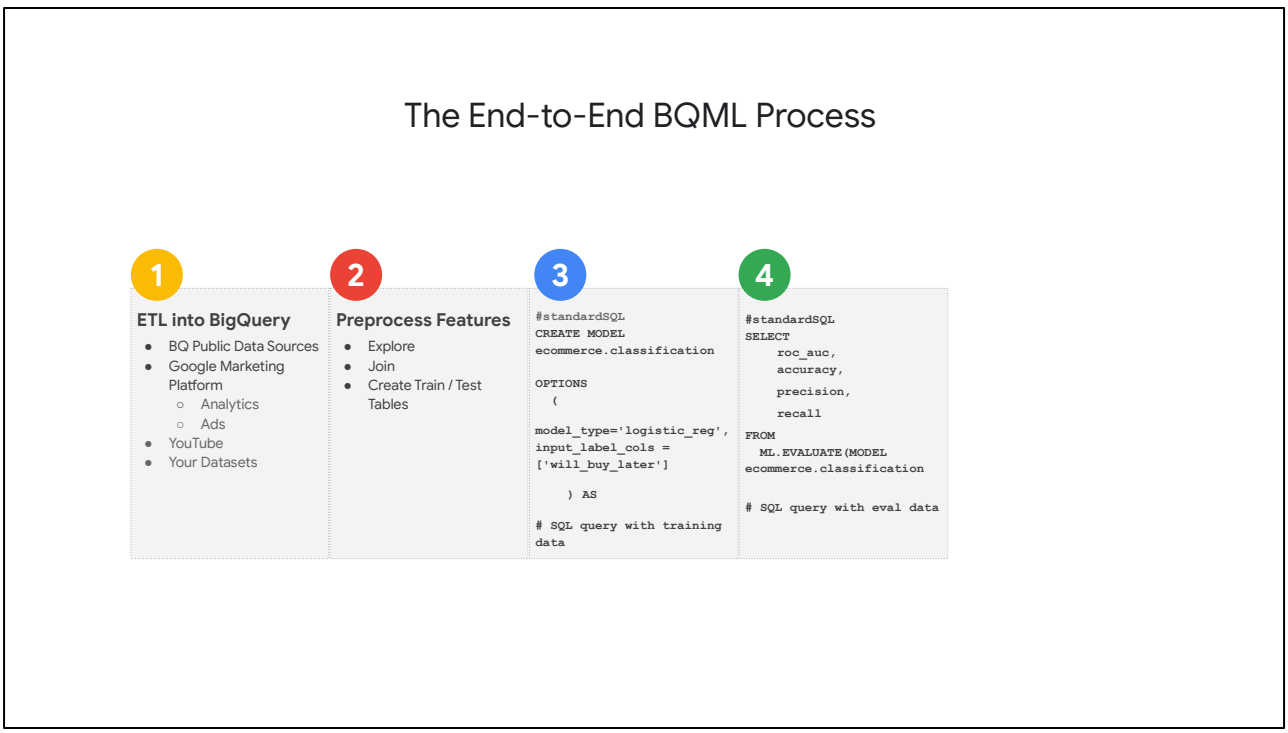


Next is the feature selection and preprocessing step which is similar to what you have been exploring so far as part of this specialization. Here is where you can put all your good SQL skills to the test in creating a great training dataset for your model to learn from. Keep in mind BigQuery ML does some of the preprocessing for you like one-hot encoding of your categorical variables for you.

The End-to-End BQML Process

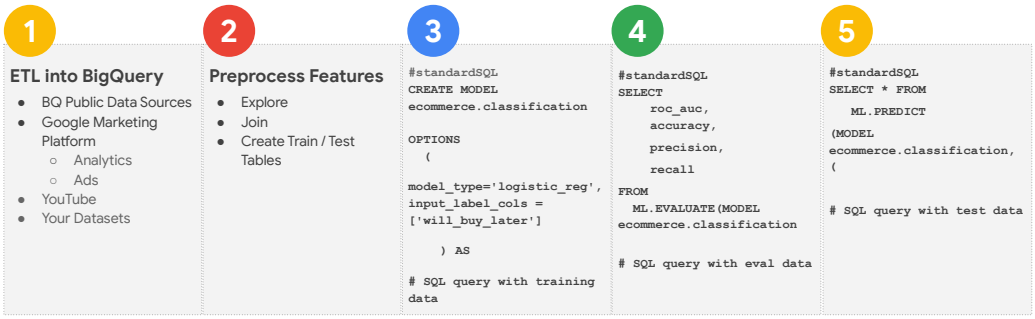


After that, here is the actual syntax for creating a model inside of BigQuery. It's short enough that I could fit it in just this one box of code. You simply say "CREATE MODEL", give it a name, specify mandatory options like model type and pass in your SQL query with your training dataset and hit Run Query and watch your model run.



After your model is trained you'll see it as a new dataset object in BigQuery and then you can execute a ML.EVALUATE query to evaluate the performance of your trained model on your evaluation dataset. Here you can analyze loss metrics like Root Mean Squared Error for forecasting models and area-under-the-curve, accuracy, precision, and recall, for classification models like the one you see here.

The End-to-End BQML Process



Once you're happy with your model performance, you can then predict with it with this even shorter query. Just invoke the ml.PREDICT command on your newly trained model and get back predictions as well as the model's confidence in those predictions. You notice a new field in the results when you run this query where you'll see your label field with "predicted" added to the field name which is your model's prediction for that label.

Agenda

Introduction to BigQuery

- Fast SQL Query Engine
- Managed Storage for Datasets

Insights from Geographic data

Machine Learning on Structured Data

- Choosing the right model type
- Scenario: Predicting Customer Lifetime Value

Creating ML models with SQL

- Introduction to BigQuery ML
- ML project phases
- Key features walkthrough

Now that you're familiar with the key phases of your ML project, it's time to walk you through some of my favorite advanced features of BigQuery ML.

Choosing the right model options

```
CREATE OR REPLACE MODEL
  `mydataset.mymodel`
OPTIONS
  ( model_type='linear_reg',
  input_label_cols= 'sales'
  ls_init_learn_rate=.15,
  l1_reg=1,
  max_iterations=5 ) AS
```

Recall that you can create a model with just CREATE MODEL. If you wanted to overwrite an existing model you would write CREATE OR REPLACE MODEL.

Models take OPTIONS which you can specify. The most important and the only required one is the model type. Linear Regression for Forecasting, Logistic Regression for classification and more types coming soon.

I'll add a link in the resources where you can view all the available model options like the model learning rate for how fast it learns and even parameters for regularization to prevent overfitting.

[resource]
Refer to the BigQuery ML documentation guide for a list of all [available model options](https://cloud.google.com/bigquery/docs/reference/standard-sql/bigqueryml-syntax-create#model_option_list) and settings. In our case we already have a field named "label" so we avoid having to specify our label column by using the model option: input_label_cols.

Inspect what the model learned with ML.WEIGHTS

```
SELECT
  category,
  weight
FROM
  UNNEST((
    SELECT
      category_weights
    FROM
      ML.WEIGHTS(MODEL
`bracketology.ncaa_model`)
    WHERE
      processed_input = 'seed')) # try other
features like 'school_ncaa'
      ORDER BY weight DESC
```

You can inspect the importance the model placed on each feature by looking at the WEIGHTS it learned. You do this by using ML.WEIGHTS and filtering on a given input column.

In the output, each feature column will have a weight from -1 to 1. The closer the number is to -1 or 1 means the more useful that field is in predicting the value for the label.

Use ML.EVALUATE to see model performance

```
SELECT
  *
FROM
  ML.EVALUATE(MODEL
`bracketology.ncaa_model`)
```

To evaluate the model's performance you can run a simple ML.EVALUATE against a trained model. You'll get different performance metrics that depend on the model type you chose. You can also look at a model's performance in the UI by clicking on the model object and looking at the metadata available.

Make batch predictions with ML.PREDICT

```
CREATE OR REPLACE TABLE `bracketology.predictions`
AS (

SELECT * FROM ML.PREDICT(MODEL
`bracketology.ncaa_model`,

# predicting for 2018 tournament games (2017
season)
(SELECT * FROM
`data-to-insights.ncaa.2018_tournament_results`)
)
)
```

Making predictions is as simple as calling ML.PREDICT on a trained model and passing through the dataset you want to predict on.

BigQuery ML Cheatsheet

- **Label** = alias a column as 'label' or specify column in OPTIONS using input_label_cols

Now let's do a final review in one big cheatsheet.

First in BigQuery ML, you need to have a field in your training dataset titled LABEL or you need to specify which field or fields are your labels using the input_label_cols in your model OPTIONS.

https://cloud.google.com/bigquery/docs/reference/standard-sql/bigqueryml-syntax-create

BigQuery ML Cheatsheet

- **Label** = alias a column as 'label' or specify column in OPTIONS using input_label_cols
- **Feature** = passed through to the model as part of your SQL SELECT statement
  `SELECT * FROM ML.FEATURE_INFO(MODEL `mydataset.mymodel`)`

Second, your model features are simply the data columns that are part of your SELECT statement after your CREATE MODEL statement. After a model is trained, you can use ML.FEATURE_INFO to get statistics and metrics about that column for additional analysis.

BigQuery ML Cheatsheet

- **Label** = alias a column as 'label' or specify column in OPTIONS using input_label_cols
- **Feature** = passed through to the model as part of your SQL SELECT statement
  `SELECT * FROM ML.FEATURE_INFO(MODEL `mydataset.mymodel`)`
- **Model** = an object created in BigQuery that resides in your BigQuery dataset

Next is the model object itself. You train many different models which will all be objects stored under your BigQuery dataset -- much like your tables and views. Try clicking on a model object to view information about when it was last updated or how many training runs it completed.

BigQuery ML Cheatsheet

- **Label** = alias a column as 'label' or specify column in OPTIONS using input_label_cols
- **Feature** = passed through to the model as part of your SQL SELECT statement
  ```
  SELECT * FROM ML.FEATURE_INFO(MODEL `mydataset.mymodel`)
  ```
- **Model** = an object created in BigQuery that resides in your BigQuery dataset
- **Model Types** = Linear Regression, Logistic Regression
  ```
  CREATE OR REPLACE MODEL <dataset>.<name>
  OPTIONS(model_type='<type>') AS
  <training dataset>
  ```

Creating a new model is as easy as writing CREATE MODEL, choosing a type, and passing in a training dataset. Again, if you're predicting on a numeric field (like sales next year) consider linear regression for forecasting. If its a discrete class like high, medium, low or spam / not-spam consider using logistic regression for classification.

BigQuery ML Cheatsheet

- **Label** = alias a column as 'label' or specify column in OPTIONS using input_label_cols
- **Feature** = passed through to the model as part of your SQL SELECT statement
  `SELECT * FROM ML.FEATURE_INFO(MODEL `mydataset.mymodel`)`
- **Model** = an object created in BigQuery that resides in your BigQuery dataset
- **Model Types** = Linear Regression, Logistic Regression
  ```
  CREATE OR REPLACE MODEL <dataset>.<name>
  OPTIONS(model_type='<type>') AS
  <training dataset>
  ```
- **Training Progress** = `SELECT * FROM ML.TRAINING_INFO(MODEL `mydataset.mymodel`)`

While the model is running (and even after it's complete) you can view training progress with ML.TRAINING_INFO.

BigQuery ML Cheatsheet

- **Label** = alias a column as 'label' or specify column in OPTIONS using input_label_cols
- **Feature** = passed through to the model as part of your SQL SELECT statement
  `SELECT * FROM ML.FEATURE_INFO(MODEL `mydataset.mymodel`)`
- **Model** = an object created in BigQuery that resides in your BigQuery dataset
- **Model Types** = Linear Regression, Logistic Regression
  ```
  CREATE OR REPLACE MODEL <dataset>.<name>
  OPTIONS(model_type='<type>') AS
  <training dataset>
  ```
- **Training Progress** = `SELECT * FROM ML.TRAINING_INFO(MODEL `mydataset.mymodel`)`
- **Inspect Weights** = `SELECT * FROM ML.WEIGHTS(MODEL `mydataset.mymodel`, (<query>))`

As we mentioned previously, you can see what the model learned about the importance of each feature as it relates to the label you're predicting. Those are your model weights.

BigQuery ML Cheatsheet

- **Label** = alias a column as 'label' or specify column in OPTIONS using input_label_cols
- **Feature** = passed through to the model as part of your SQL SELECT statement
  `SELECT * FROM ML.FEATURE_INFO(MODEL `mydataset.mymodel`)`
- **Model** = an object created in BigQuery that resides in your BigQuery dataset
- **Model Types** = Linear Regression, Logistic Regression
  ```
  CREATE OR REPLACE MODEL <dataset>.<name>
  OPTIONS(model_type='<type>') AS
  <training dataset>
  ```
- **Training Progress** = `SELECT * FROM ML.TRAINING_INFO(MODEL `mydataset.mymodel`)`
- **Inspect Weights** = `SELECT * FROM ML.WEIGHTS(MODEL `mydataset.mymodel`, (<query>))`
- **Evaluation** = `SELECT * FROM ML.EVALUATE(MODEL `mydataset.mymodel`)`

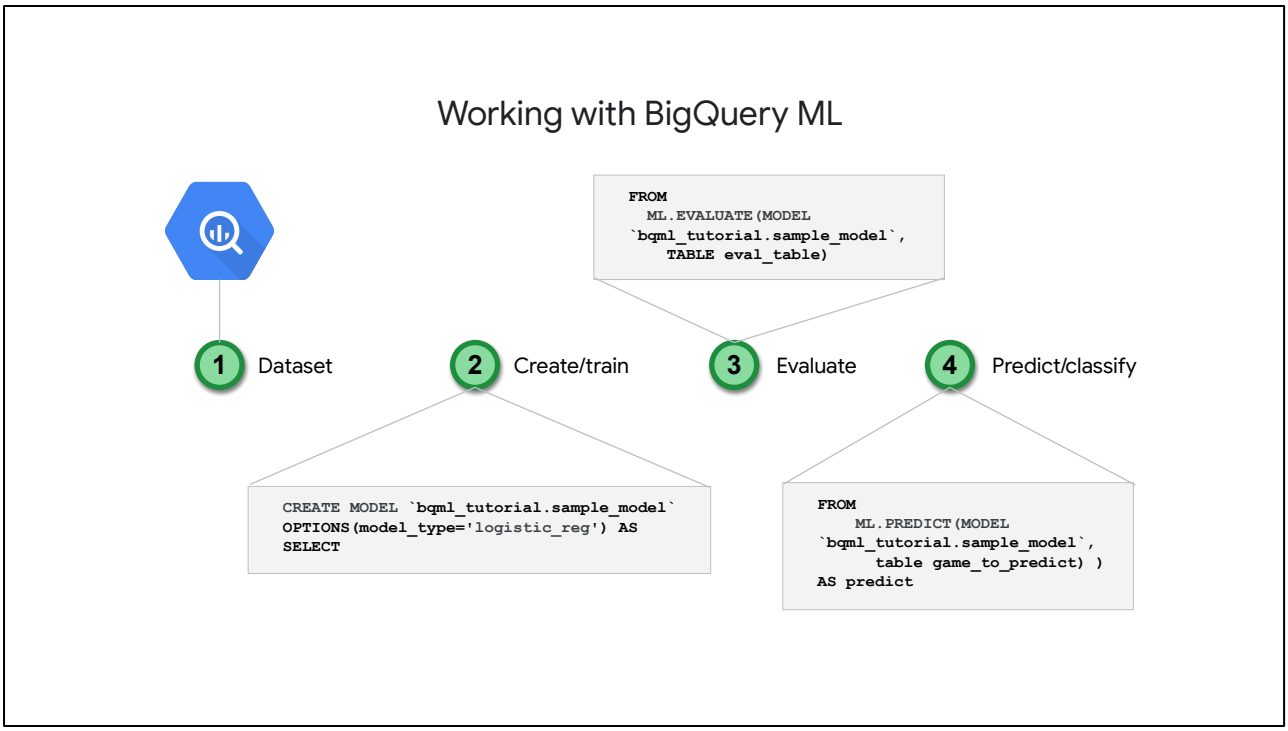
You can see how well the model did against it's evaluation dataset by using ML.EVALUATE

BigQuery ML Cheatsheet

- **Label** = alias a column as 'label' or specify column in OPTIONS using input_label_cols
- **Feature** = passed through to the model as part of your SQL SELECT statement
  `SELECT * FROM ML.FEATURE_INFO(MODEL `mydataset.mymodel`)`
- **Model** = an object created in BigQuery that resides in your BigQuery dataset
- **Model Types** = Linear Regression, Logistic Regression
  ```
  CREATE OR REPLACE MODEL <dataset>.<name>
  OPTIONS(model_type='<type>') AS
  <training dataset>
  ```
- **Training Progress** = `SELECT * FROM ML.TRAINING_INFO(MODEL `mydataset.mymodel`)`
- **Inspect Weights** = `SELECT * FROM ML.WEIGHTS(MODEL `mydataset.mymodel`, (<query>))`
- **Evaluation** = `SELECT * FROM ML.EVALUATE(MODEL `mydataset.mymodel`)`
- **Prediction** = `SELECT * FROM ML.PREDICT(MODEL `mydataset.mymodel`, (<query>))`

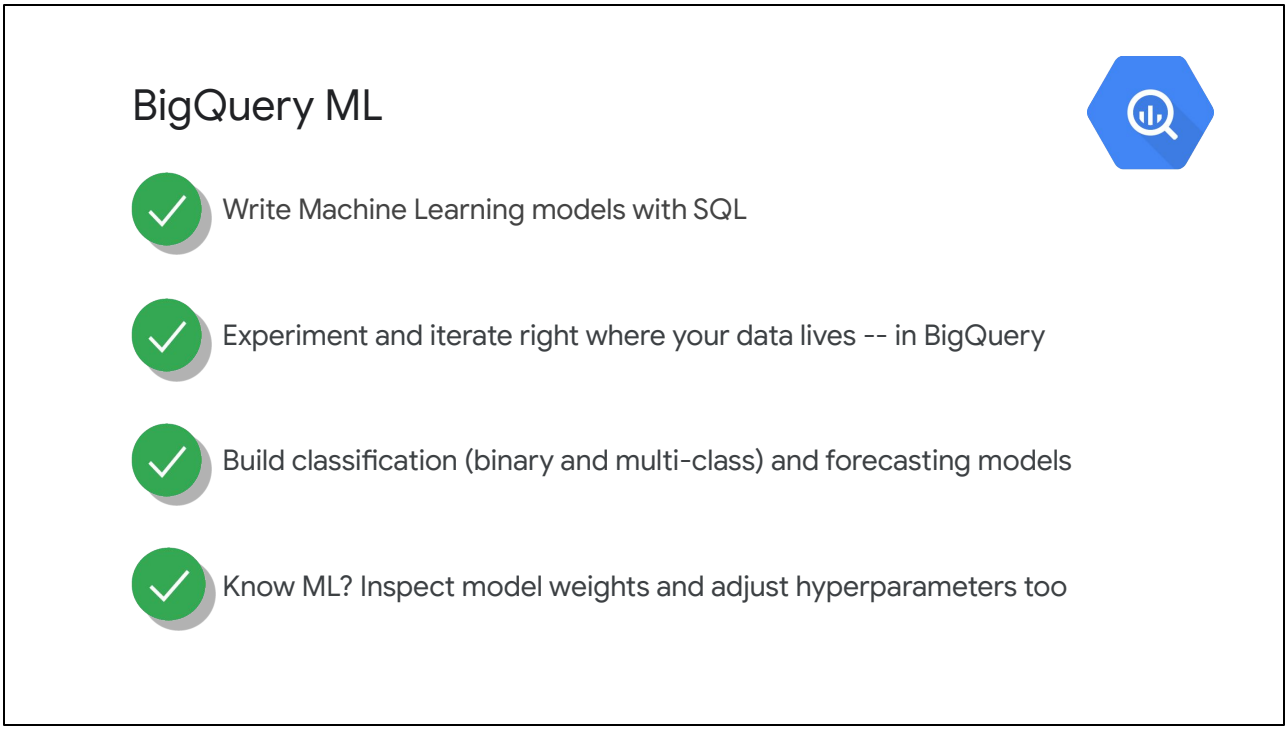
Lastly, it's as simple as writing ML.PREDICT and referencing your model and prediction dataset to get predictions.

An important note here is that when using ML.PREDICT and passing in a new dataset with an unknown label you can add in other columns that you didn't train on initially. The model is not being re-trained during prediction. Note that if you happen to REMOVE or RENAME columns from your prediction dataset that the model was expecting then you will be given an error.



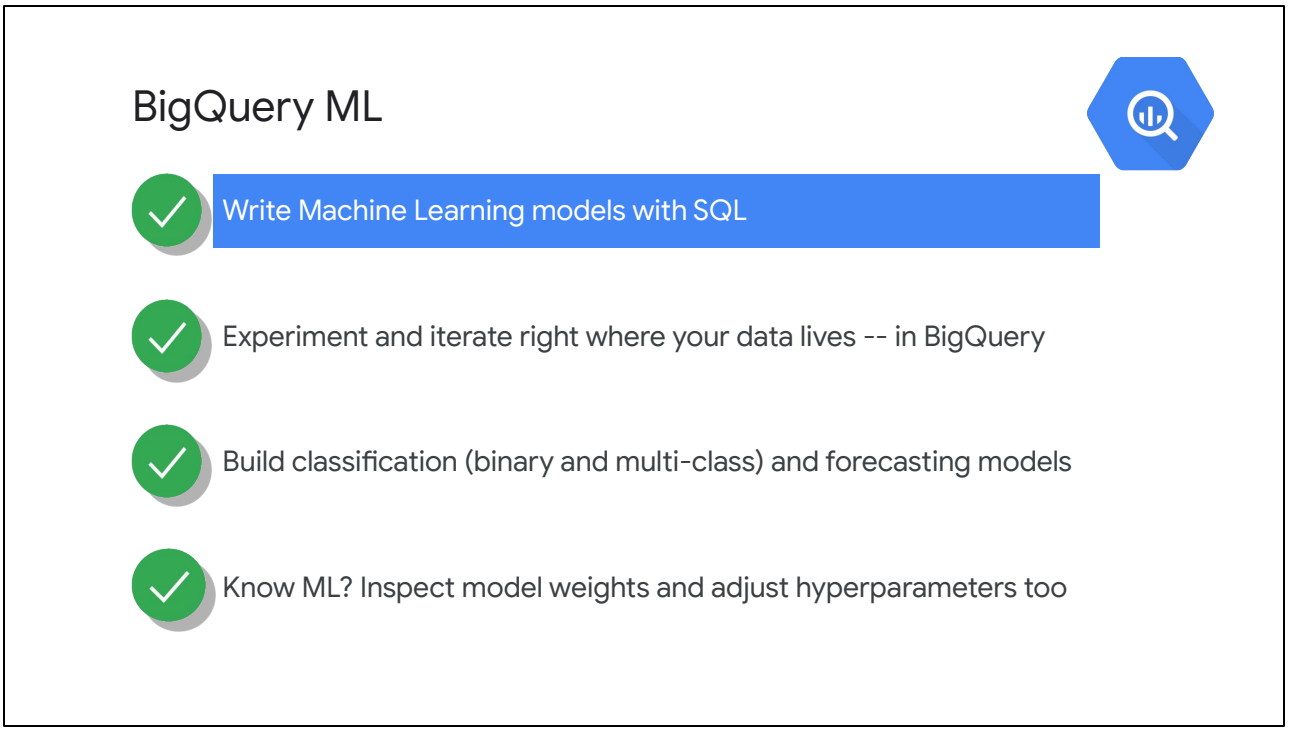
In four major steps it looks like this:

1. Write SQL query to extract training data from BigQuery
2. Create a model, specifying model type
3. Evaluate model and verify that it meets requirements
4. Predict using model on data extracted from BigQuery

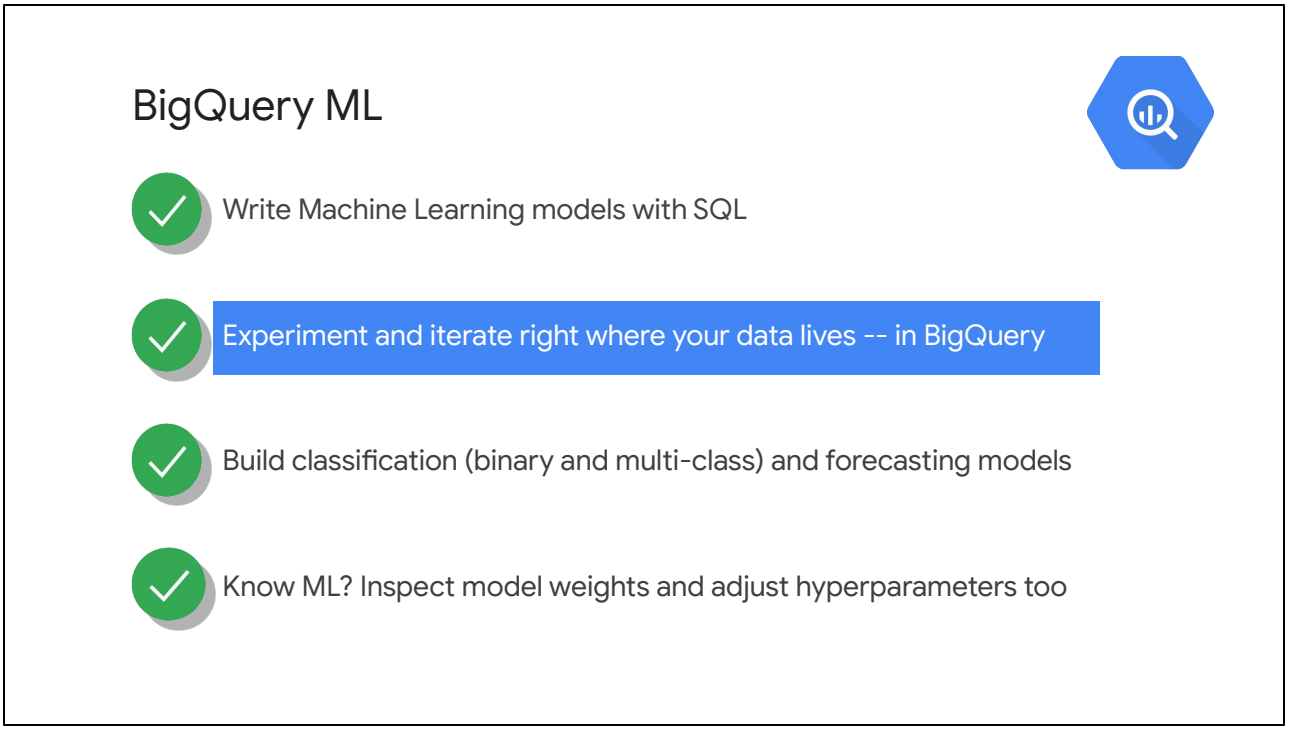


And if you're explaining BigQuery ML to others, I often just list these main points. To recap

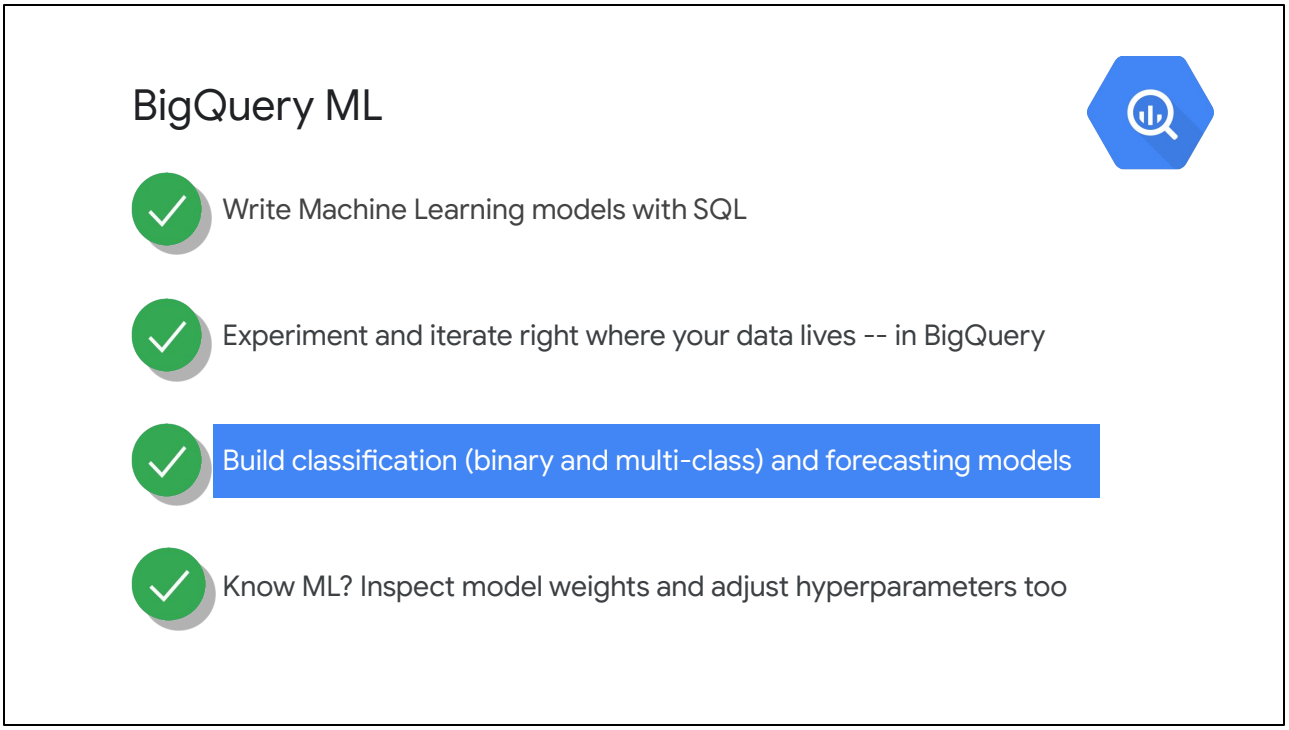
BigQuery ML allows you to:



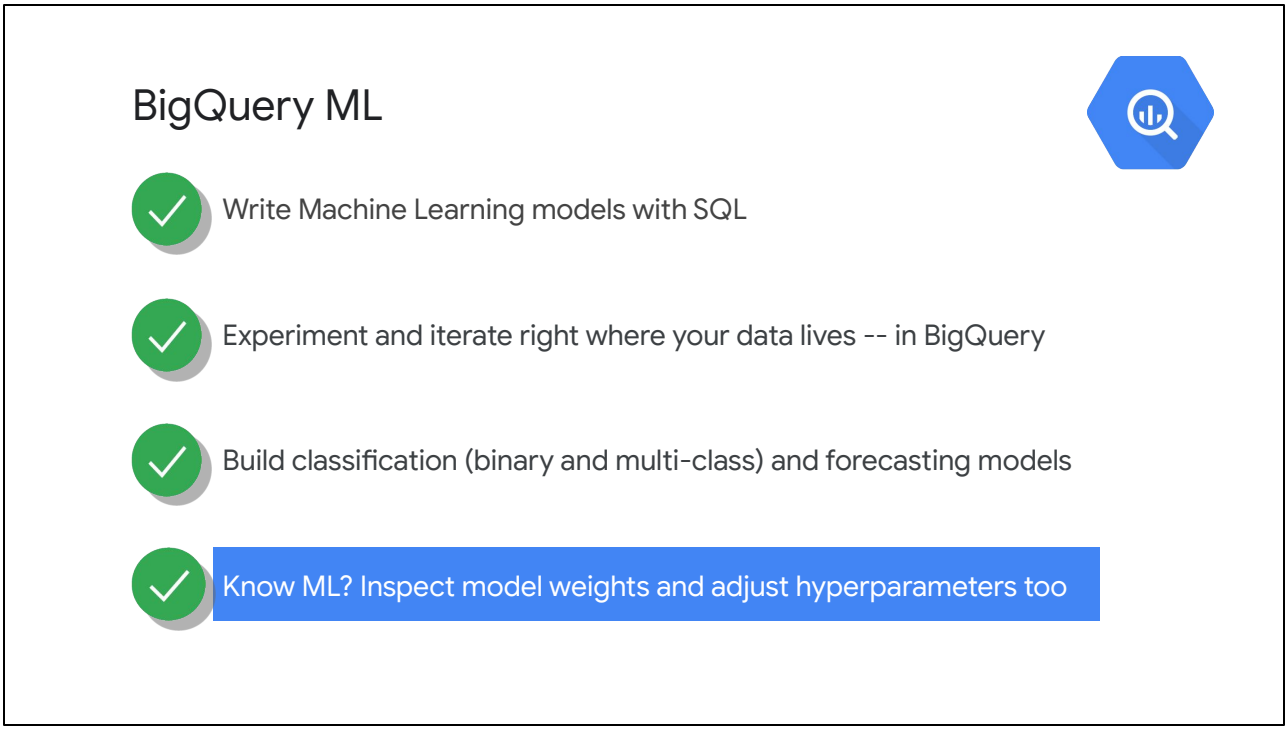
Write Machine Learning models with SQL



Experiment and iterate right where your data lives -- in BigQuery



Build classification (binary and multi-class) and forecasting models and more model types coming soon



And if you know machine learning already, you can get into the details of your model options and weights very easily. Common adjustments you can make above what BigQuery ML defaults to everyone include the learning rate, regularization, the training/eval dataset split, pre-defined weights for classes and much more. Check out the BigQuery ML documentation link and bookmark it as a reference -- especially as new features get added.

Lab: Demand Forecasting using BigQuery ML

Which bike share stations should I focus on keeping stocked to meet demand?

It's now time to put your skills to the test with model building.

- Your company rents bikes to users all across New York city. You are the CTO and you've been tasked with making sure each bike station has enough bikes to meet demand given recent trends
- Your hardware team has equipped each station with a GPS so you get lat/long information on each trip for each bike
- Your data science team is small and wants to quickly assess the feasibility of machine learning before investing too much time in model building

Objectives

- Explore the San Fran bikes dataset
- Create a ML training dataset
- Select a model to train
- Train, evaluate, and predict
- Improve ML model performance
- Visualize results

In this lab you will:

Objectives

- Explore the San Fran bikes dataset
- Create a ML training dataset
- Select a model to train
- Train, evaluate, and predict
- Improve ML model performance
- Visualize results

Explore the San Francisco bikes dataset

Objectives

- Explore the San Fran bikes dataset
- Create a ML training dataset
- Select a model to train
- Train, evaluate, and predict
- Improve ML model performance
- Visualize results

Create a ML training dataset based on the features you want included in your model

Objectives

- Explore the San Fran bikes dataset
- Create a ML training dataset
- Select a model to train
- Train, evaluate, and predict
- Improve ML model performance
- Visualize results

Select the right model type to train

Objectives

- Explore the San Fran bikes dataset
- Create a ML training dataset
- Select a model to train
- Train, evaluate, and predict
- Improve ML model performance
- Visualize results

Train your model, evaluate performance, and then finally predict using your model

Objectives

- Explore the San Fran bikes dataset
- Create a ML training dataset
- Select a model to train
- Train, evaluate, and predict
- Improve ML model performance
- Visualize results

After your initial model you will look at ways to improve your ML model's performance

Objectives

- Explore the San Fran bikes dataset
- Create a ML training dataset
- Select a model to train
- Train, evaluate, and predict
- Improve ML model performance
- Visualize results

and finally Visualize results with Data Studio.

Keep in mind you have multiple attempts at this upcoming Qwiklab so you can come back and practice more if you wish.