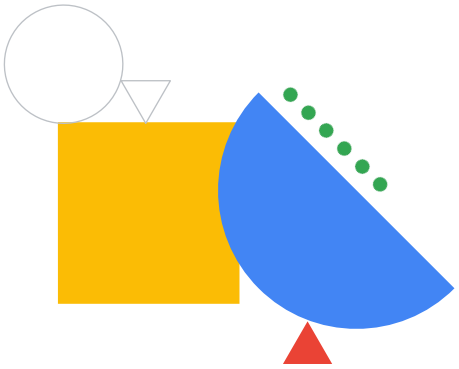
Google Cloud

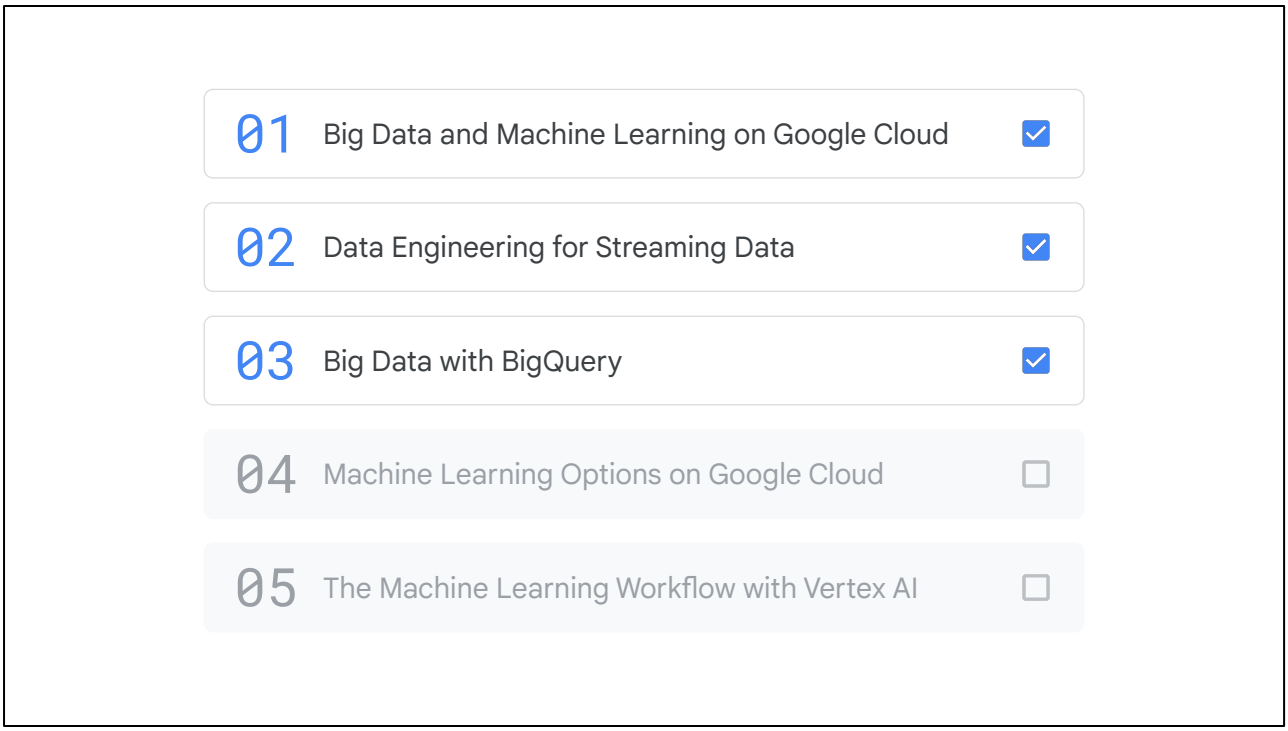
Big Data with BigQuery



Module 3

Google Cloud Big Data and Machine Learning Fundamentals

Introduction



In the previous module of this course, you learned how to build a streaming dataflow using Pub/sub, Dataflow, and Looker vs. Data Studio. Now let's switch our focus to a popular data warehouse product on Google, BigQuery.

Agenda

BigQuery

- Storage and analytics
- BigQuery demo

BigQuery ML

- BigQuery ML
- BigQuery ML project phases
- BigQuery ML key commands
- Hands-on lab

You'll begin by exploring BigQuery's two main services, storage and analytics, and then get a demonstration of BigQuery in use.

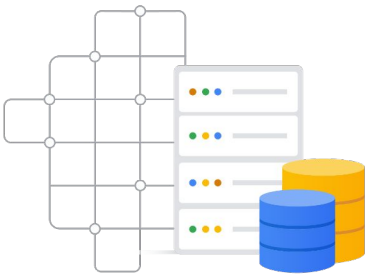
After that, you'll see how BigQuery ML provides a data-to-AI lifecycle all within one place. You'll also learn about BigQuery ML project phases, as well as key commands.

Finally, you'll get hands-on practice using BigQuery ML to build a custom ML model.

Let's get started!

BigQuery: a fully-managed data warehouse

"Fully managed" means that BigQuery takes care of the underlying infrastructure



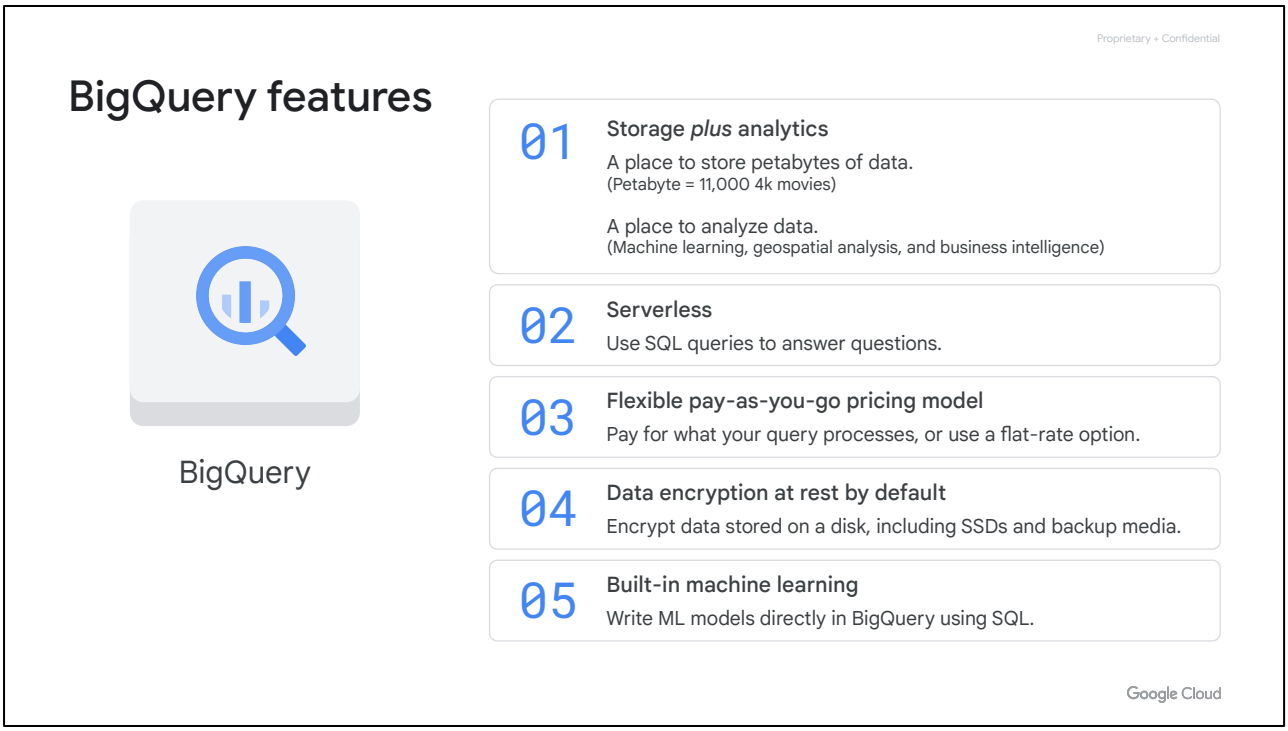
Terabytes and petabytes of data gathered from a wide range of sources

BigQuery is a fully-managed data warehouse.

A data warehouse is a large store, containing **terabytes** and **petabytes** of data gathered from a wide range of sources within an organization, that's used to guide management decisions.

Being fully managed means that BigQuery takes care of the underlying infrastructure, so you can focus on using SQL queries to answer business questions–without worrying about deployment, scalability, and security.

At this point, it's useful to consider what the main difference is between a data warehouse and a data lake. A data lake is just a pool of raw, unorganized, and unclassified data, which has no specified purpose. A data warehouse on the other hand, contains structured and organized data, which can be used for advanced querying.



Let's look at some of the key features of BigQuery.

- BigQuery provides two services in one: **storage plus analytics**. It's a place to store petabytes of data. For reference, 1 petabyte is equivalent to 11,000 movies at 4k quality. BigQuery is also a place to analyze data, with built-in features like machine learning, geospatial analysis, and business intelligence, which we'll explore a bit later on.

- BigQuery is a fully managed **serverless** solution, meaning that you use SQL queries to answer your organization's biggest questions in the frontend without worrying about infrastructure in the backend. If you've never written SQL before, don't worry. This course provides resources and labs to help.

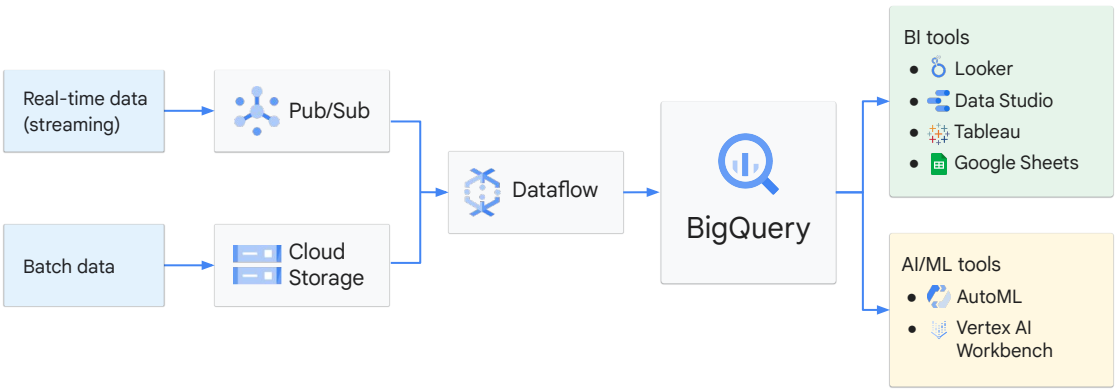
- BigQuery has a **flexible pay-as-you-go pricing model** where you pay for the number of bytes of data your query processes and for any permanent table storage. If you prefer to have a fixed bill every month, you can also subscribe to flat-rate pricing where you have a reserved amount of resources for use.

- Data in BigQuery is **encrypted** at rest by default without any action required from a customer. By encryption at rest, we mean encryption used to protect data that is stored on a disk, including solid-state drives, or backup media.

- BigQuery has **built-in machine learning** features so you can write ML models directly in BigQuery using SQL. Also, if you decide to use other professional

- tools—such as Vertex AI from Google Cloud—to train your ML models, you can export datasets from BigQuery directly into Vertex AI for a seamless integration across the data-to-AI lifecycle.

Google data warehouse solution architecture



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

The input data can be either real-time or batch data. If you think back to the last module of the course, you'll recall that there are four challenges of big data in modern organizations. They are that data can be any format (variety), any size (volume), any speed (velocity), and possibly inaccurate (veracity).

If it's streaming data, which can be either structured or unstructured, high speed, and large volume, Pub/Sub is needed to digest the data. If it's batch data, it can be directly uploaded to Cloud Storage.

After that, both pipelines lead to Dataflow to process the data. Dataflow is where we ETL – extract, transform, and load – the data if needed.

BigQuery sits in the middle to link data processes using Dataflow and data access through analytics, AI, and ML tools.

The job of the analytics engine of BigQuery at the end of a data pipeline is to ingest all the processed data after ETL, store and analyze it, and possibly output it for further use such as data visualization and machine learning.

BigQuery outputs usually feed into two buckets: business intelligence tools and AI/ML tools.

If you’re a business analyst or data analyst, you can connect to visualization tools like Looker, Data Studio, Tableau, and other BI tools.

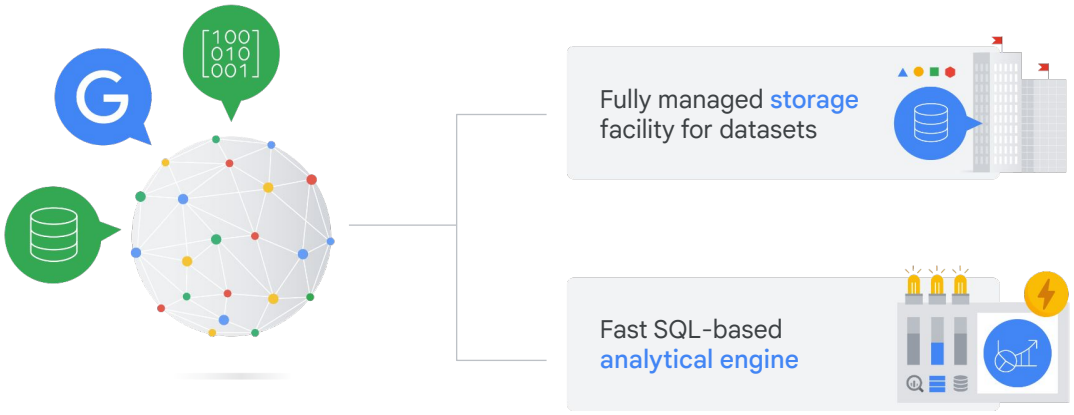
If you prefer to work in spreadsheets, you can query both small or large BigQuery datasets directly from Google Sheets and even perform common operations like pivot tables.

Alternatively if you’re a data scientist or machine learning engineer, you can directly call the data from BigQuery through AutoML or Workbench. These AI/ML tools are part of Vertex AI, Google's unified ML platform.

BigQuery is like a common staging area for data analytics workloads. When your data is there, business analysts, BI developers, data scientists, and machine learning engineers can be granted access to your data for their own insights.

Storage and analytics

BigQuery: Storage + analytics



BigQuery provides two services in one. It's both a fully managed storage facility to load and store datasets, and also a fast SQL-based analytical engine.

The two services are connected by Google's high-speed internal network. It's this super-fast network that allows BigQuery to scale both storage and compute independently, based on demand.

Let's look at how BigQuery manages the storage and metadata for datasets.

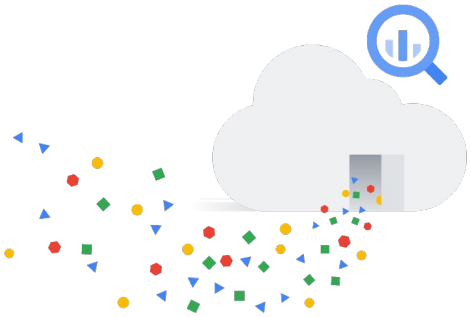
BigQuery can ingest datasets from various sources

Internal data: Data saved in BigQuery

External data:
- Google Cloud storage (e.g. Cloud Storage)
- Google Cloud database (e.g., Spanner or Cloud SQL)

Multi-cloud data: such as AWS and Azure

Public datasets



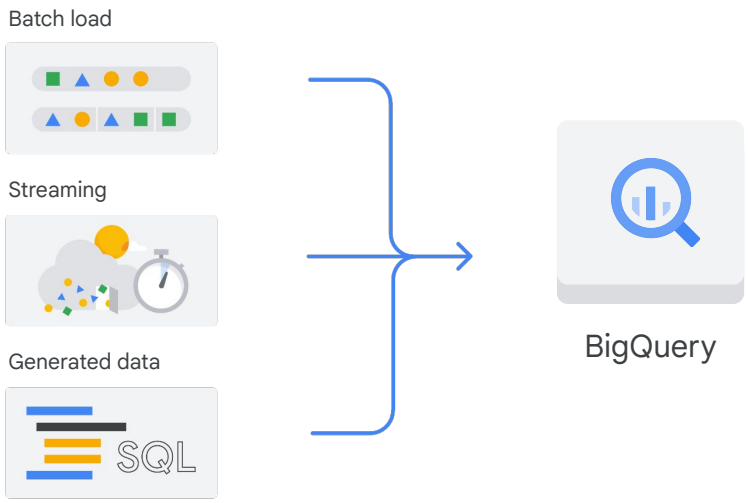
Replicated Backed up Autoscaled

BigQuery can ingest datasets from various sources, including:

- internal data, which is data saved directly in BigQuery,
- external data, BigQuery also offers the option to query external data sources–like data stored in other Google Cloud storage services such as Cloud Storage, or in other Google Cloud database services, such as Spanner or Cloud SQL–and bypass BigQuery managed storage. This means a raw CSV file in Cloud Storage or a Google Sheet can be used to write a query without being ingested by BigQuery first. One thing to note here: inconsistency might result from saving and processing data separately. To avoid that risk, consider using Dataflow to build a streaming data pipeline into BigQuery.
- multi-cloud data, which is data stored in multiple cloud services, such as AWS or Azure.
- and public datasets. If you don't have data of your own, you can analyze any of the datasets available in the public dataset marketplace.

After the data is stored in BigQuery, it's fully managed and is automatically replicated, backed up, and set up to autoscale.

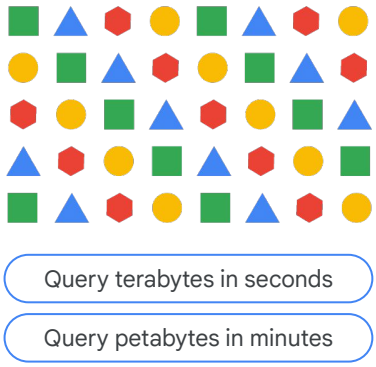
Patterns to load data into BigQuery



There are three basic patterns to load data into BigQuery.

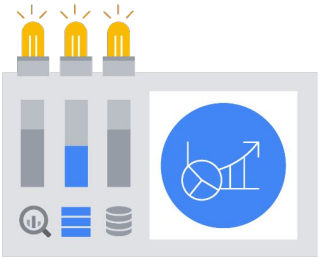
- The first is a **batch load**, where source data is loaded into a BigQuery table in a single batch operation. This can be a one-time operation or it can be automated to occur on a schedule. A batch load operation can create a new table or append data into an existing table.
- The second is **streaming,** where smaller batches of data are streamed continuously so that the data is available for querying in near-real time.
- And the third is **generated data**, where SQL statements are used to insert rows into an existing table or to write the results of a query to a table.

BigQuery can run analytic queries over large datasets



Query terabytes in seconds

Query petabytes in minutes

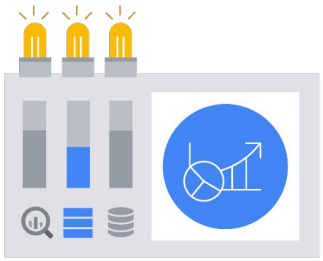


Analyze data

Of course, the purpose of BigQuery is not to just save data; it's for analyzing data and helping to make business decisions.

BigQuery is optimized for running analytic queries over large datasets. It can perform queries on terabytes of data in seconds and petabytes in minutes. This performance lets you analyze large datasets efficiently and get insights in near real time.

BigQuery analytics features



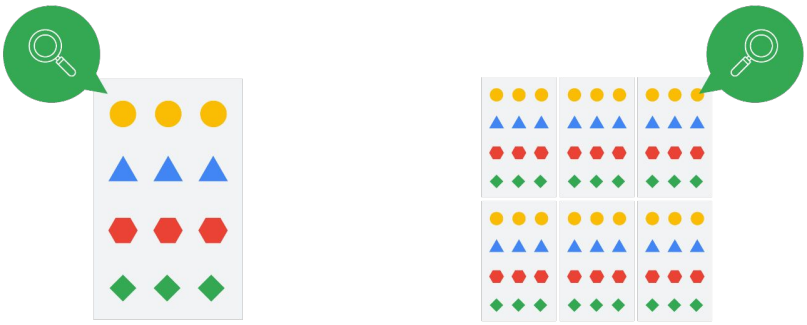
- Ad hoc analysis
- Geospatial analytics
- Building machine learning models
- Building BI dashboards

Let's look at the analytics features that are available in BigQuery.

BigQuery supports:

- **Ad hoc analysis** using Standard SQL, the BigQuery SQL dialect.
- **Geospatial analytics** using geography data types and Standard SQL geography functions.
- **Building machine learning models** using BigQuery ML.
- Building rich, interactive **business intelligence dashboards** using BigQuery BI Engine.

BigQuery runs interactive and batch queries



Interactive queries

Executed as needed

Batch queries

Queries starts when resources are available

By default, BigQuery runs interactive queries, which means that the queries are executed as needed.

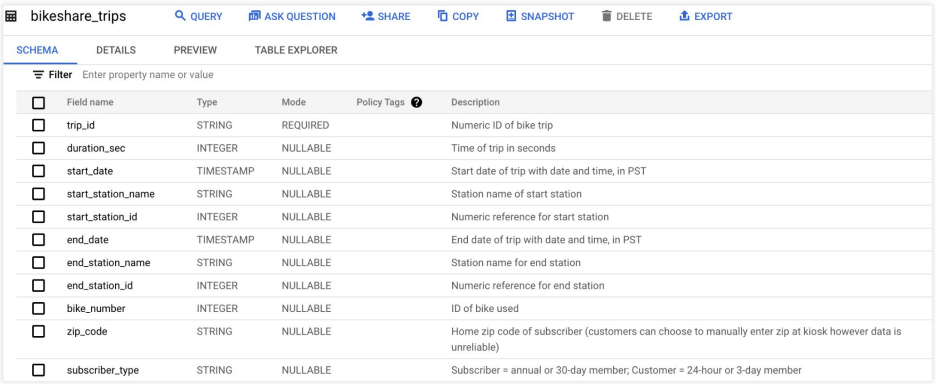
BigQuery also offers batch queries, where each query is queued on your behalf and the query starts when idle resources are available, usually within a few minutes.

BigQuery demo

Exploring bike share data with BigQuery

Discover the insights of a public dataset using BigQuery.

- **Goal**: Identify the most popular stations in San Francisco to pick up a shared bike
- **Dataset**: San Francisco bike share trips
- **Schema** (partial):



Field name	Type	Mode	Policy Tags	Description
trip_id	STRING	REQUIRED		Numeric ID of bike trip
duration_sec	INTEGER	NULLABLE		Time of trip in seconds
start_date	TIMESTAMP	NULLABLE		Start date of trip with date and time, in PST
start_station_name	STRING	NULLABLE		Station name of start station
start_station_id	INTEGER	NULLABLE		Numeric reference for start station
end_date	TIMESTAMP	NULLABLE		End date of trip with date and time, in PST
end_station_name	STRING	NULLABLE		Station name for end station
end_station_id	INTEGER	NULLABLE		Numeric reference for end station
bike_number	INTEGER	NULLABLE		ID of bike used
zip_code	STRING	NULLABLE		Home zip code of subscriber (customers can choose to manually enter zip at kiosk however data is unreliable)
subscriber_type	STRING	NULLABLE		Subscriber = annual or 30-day member; Customer = 24-hour or 3-day member

Google Cloud

As any data analyst will tell you, exploring a dataset with SQL is often one of the first steps to uncover hidden insights.

In this section:

- We'll show you how to use BigQuery to uncover insights from a public dataset.
- The goal is to find the most popular stations across San Francisco to pick up bikes.
- You can find this public dataset in Bigquery by following these steps:
  - Navigate to the Google Cloud console > BigQuery > Add data > Public dataset > Search San Francisco bike share > Under **san_francisco_bikeshare**, choose **bikeshare_trips**
  - Click the three dots besides the dataset to start the query, for more details please refer to the video.
- From the schema, you can find the dataset include information about the:
  - TripID
  - Trip duration
  - Start station, date
  - End station, date
  - Bike number
  - Subscriber information, etc.
- Take a moment to consider this question: How can you find the most popular station using SQL)?

BigQuery demo

SQL code in BigQuery

```sql
SELECT
    start_station_name,
    COUNT(trip_id) AS num_trips
FROM
    `bigquery-public-data.san_francisco_bikeshare.bikeshare_trips`
WHERE start_date > '2017-12-31 00:00:00 UTC'
GROUP BY
    start_station_name
ORDER BY
    num_trips DESC
LIMIT
    10
```

Query results (partial)

Query complete (1.2 sec elapsed, 81.4 MB processed)

Row	start_station_name	num_trips
1	San Francisco Ferry Building	9926
2	San Francisco Caltrain	9740
3	Berry St at 4th St	9041
4	Market St at 10th St	8934

Please find the full demo video here.

The goal of this code is to find the ten most popular stations, based on the number of trips that start from that station.

- SELECT: The fields to display in the query results
- FROM: Specify the dataset name
- WHERE: The query condition
- GROUP BY: Count the number of trips by start station
- ORDER BY: Order the result
- LIMIT: Specify the number of the query results (top 10 stations in this case)

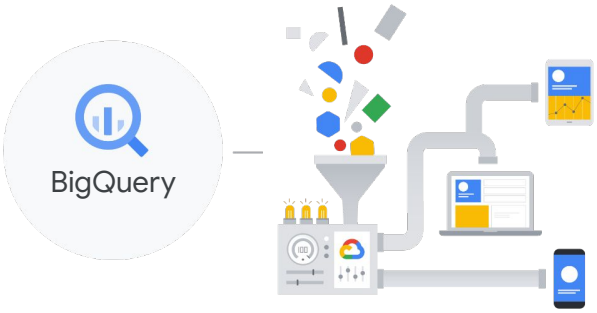
Partial query results are displayed on the right.

- What insights do you get from these?
- How might you use these insights to improve the business?
- Can you think of any other analysis based on this dataset?

Introduction to BigQuery ML

Although BigQuery started out solely as a data warehouse, over time it has evolved to provide features that support the data-to-AI lifecycle.

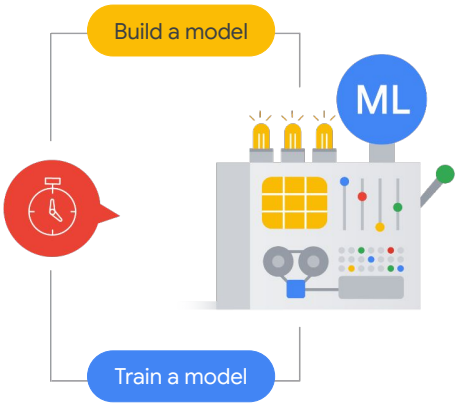
BigQuery ML



- Capabilities to build ML models
- ML project phases
- Key ML commands in SQL

In this section of the course, we'll explore BigQuery's capabilities for building machine learning models and the ML project phases, and walk you through the key ML commands in SQL.

Building ML models can be time-intensive



01 Export data from your datastore into an integrated development environment (IDE)

02 Transform the data and perform feature engineering

03 Build the model in TensorFlow and train it locally or on a virtual machine

\+ To improve model performance, you need to get more data and create new features

If you've worked with ML models before, you know that building and training them can be very time-intensive.

- You must first export data from your datastore into an IDE (integrated development environment) such as Jupyter Notebook or Google Colab and then transform the data and perform your feature engineering steps before you can feed it into a training model.
- Then finally, you need to build the model in TensorFlow, or a similar library, and train it locally on a computer or on a virtual machine.

To improve the model performance, you also need to go back and forth to get more data and create new features. This process will need to be repeated, but it's so time-intensive that you'll probably stop after a few iterations. Also, I just mentioned TensorFlow and feature engineering; in the past, if you weren't familiar with these technologies, ML was left to the data scientists on your team and was not available to you.

Step 1: Create a model with a SQL statement

```
1 CREATE MODEL numbikes.model
2 OPTIONS
3   (model_type='linear_reg', labels=['num_trips']) AS
4 WITH bike_data AS
5 (
6 SELECT
7   COUNT(*) as num_trips,
```


There are two steps needed to start:

Step 1: **Create** a model with a SQL statement. Here we can use the bikeshare dataset as an example.

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

```
1 SELECT
2  predicted_num_trips,num_trips,trip_data
3 FROM
4  ml.PREDICT(MODEL `numbikes.model`, (WITH bike_data AS
5 (
6 SELECT
7   COUNT(*) as num_trips,
```

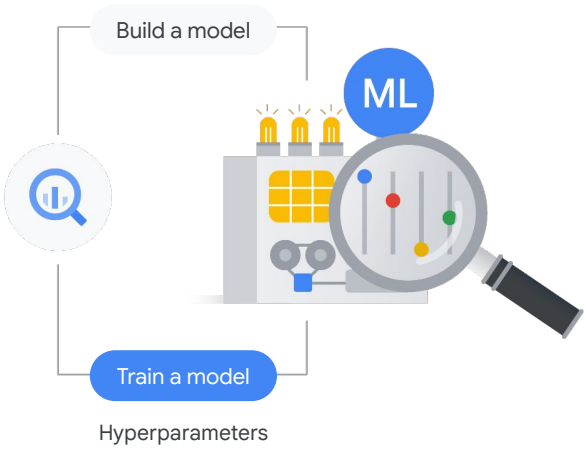

Step 2: Write a SQL **prediction** query and invoke ml.Predict.

And that's it! You now have a model and can view the results.

Additional steps might include activities like evaluating the model, but if you know basic SQL, you can now implement ML; pretty cool!

Hyperparameters

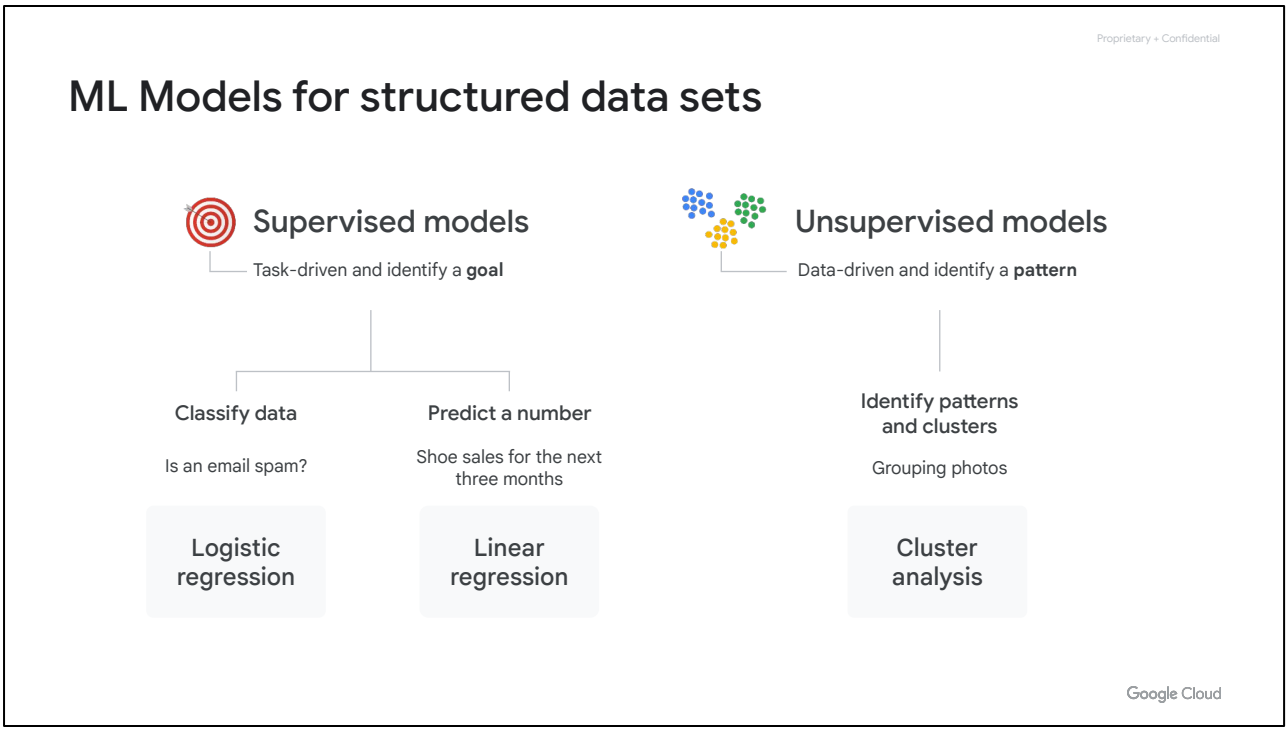
- Manually control hyperparameters
- Automatically tune hyperparameters



BigQuery ML was designed to be simple, like building a model in two steps. That simplicity extends to defining the machine learning hyperparameters, which let you tune the model to achieve the best training result.

Hyperparameters are the settings applied to a model before the training starts, like the learning rate.

With BigQuery ML, you can either manually control the hyperparameters or hand it to BigQuery starting with a default hyperparameter setting and then automatic tuning.



When using a structured dataset in BigQuery ML, you need to choose the appropriate model type. Choosing which type of ML model depends on your business goal and the datasets.
BigQuery supports **supervised models** and **unsupervised models**.

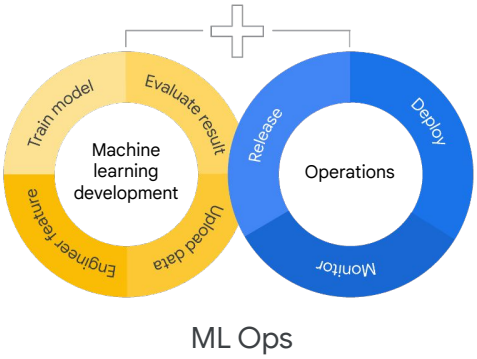
- **Supervised models** are task-driven and identify a goal.
  - Within a supervised model, if your goal is to classify data, like whether an email is spam, use **logistic regression**.
  - If your goal is to predict a number, like shoe sales for the next three months, use **linear regression**.
- Alternatively, **unsupervised models** are data-driven and identify a pattern.
  - Within an unsupervised model, if your goal is to identify patterns or clusters and then determine the best way to group them, like grouping random photos of flowers into categories, you should use **cluster analysis**.

Once you have your problem outlined, it's time to decide on the best model. Categories include **classification** and **regression models.** There are also **other model** options to choose from, along with **ML ops**.

Logistic regression is an example of a classification model, and linear regression is an example of a regression model. **We recommend that you start with these options**, and use the results to benchmark to compare against more complex models such as DNN (deep neural networks), which may take more time and computing resources to train and deploy.

Machine learning operations

- Importing TensorFlow models
- Exporting models from BigQuery ML
- Hyperparameter tuning



ML Ops

In addition to providing different types of machine learning models, BigQuery ML supports features to deploy, monitor, and manage the ML production, called ML Ops, which is short for machine learning operations.

Options include:

- Importing TensorFlow models for batch prediction
- Exporting models from BigQuery ML for online prediction
- And hyperparameter tuning using Vertex AI Vizier

We'll explore ML Ops in more detail later in this course.

Using BigQuery ML to predict customer lifetime value

Predict customer lifetime value with a model

Lifetime value (LTV)

Estimate how much revenue or profit you can expect from a customer given their history and customers with similar patterns

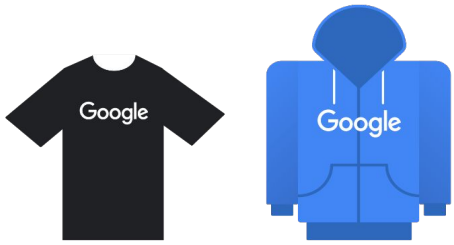
Now that you're familiar with the types of ML models available to choose from, high-quality data must be used to teach the models what they need to learn. The best way to learn the key concepts of machine learning on structured datasets is through an example.

In this scenario, we'll predict customer lifetime value with a model.

Lifetime value, or LTV, is a common metric in marketing used to estimate how much revenue or profit you can expect from a customer given their history and customers with similar patterns.

Google Analytics ecommerce data set

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 Women's Quilted Insulated Vest Black
Google Women's Lightweight Microfleece Jacket
Women's Weatherblock Shell Jacket Black
Men's Weatherblock Shell Jacket Black



Identify high-value customers

We'll use a Google Analytics ecommerce dataset from Google's own merchandise store that sells branded items like t-shirts and jackets.

The goal is to identify high-value customers and bring them to our store with special promotions and incentives.

Fields

Customer lifetime pageviews | Total visits | Average time spent on the site | Total revenue | Ecommerce transactions

Row	fullVisitorID	distinct_days_visited	ltv_pageviews	ltv_visits	ltv_avg_time_on_site_s	ltv_revenue	ltv_transactions	label
1	7813149964104484438	79	7395	138	479.63	624572	67	
2	7713012430069756739	2	514	6	1954.33	18194	35	High value customer
3	6760732402251466726	30	868	41	723.55	481282	34	
4	552667592603848032	1	466	1	7013.0	8796	25	High value customer
5	1957458976293878100	148	4303	284	796.46	7711343	22	
6	4983264713004875783	2	366	4	3807.5	7485	21	High value customer
7	2402527199731150932	28	559	31	906.61	327010	19	

Having explored the available fields, you may find some useful in determining whether a customer is high value based on their behavior on our website.

These fields include:

- customer lifetime pageviews,
- total visits,
- average time spent on the site,
- total revenue brought in,
- and ecommerce transactions on the site.

Remember that in machine learning, you 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 weren't useful at all to the model in predicting the outcome -- you'll see later how to determine this.

Labels

Row	fullVisitorID	distinct_days_visited	ltv_pageviews	ltv_visits	ltv_avg_time_on_site_s	ltv_revenue	ltv_transactions	label
1	7813149964104484438	79	7395	138	479.63	624572	67	
2	7713012430069756739	2	514	6	1954.33	18194	35	High value customer
3	6760732402251466726	30	868	41	723.55	481282	34	
4	552667592603848032	1	466	1	7013.0	8796	25	High value customer
5	1957458976293878100	148	4303	284	796.46	7711343	22	
6	4983264713004875783	2	366	4	3807.5	7485	21	High value customer
7	2402527199731150932	28	559	31	906.61	327010	19	

Example / Observation / Instance

Label

Label

Linear regression

Logistic regression

Google Cloud

Now that we have some data, we can prepare to feed it into the model. Incidentally, to keep this example simple, we're only using seven records, but we'd need tens of thousands of records to train a model effectively.

Before we feed the data into the model, we first need to define our data and columns in the language that data scientists and other ML professionals use.

Using the Google Merchandise Store example, a record or row in the dataset is called an **example, an observation, or an instance**.

A **label** is a correct answer, and you know it's correct because it comes from historical data. This is what you need to train the model on in order to predict *future* data. Depending on what you want to predict, a label can be either a numeric variable, which requires a linear regression model, or a categorical variable, which requires a logistic regression model.

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 lifetime revenue, we could use revenue as the label and predict the same for newer customers with that same spending trajectory.

This means forecasting a number, so we can use a **linear regression** as a starting point to model.

Labels could also be categorical variables like binary values, such as *High Value Customer or not*. To predict a categorical variable, if you recall from the previous section, you need to use a **logistic regression** model.

Knowing what you're trying to predict, such as a class or a number, will greatly influence the type of model you'll use.

Features

Row	fullVisitorID	distinct_days_visited	ltv_pageviews	ltv_visits	ltv_avg_time_on_site_s	ltv_revenue	ltv_transactions	label
1	7813149964104484438	79	7395	138	479.63	624572	67	
2	7713012430069756739	2	514	6	1954.33	18194	35	High value customer
3	6760732402251466726	30	868	41	723.55	481282	34	
4	552667592603848032	1	466	1	7013.0	8796	25	High value customer
5	1957458976293878100	148	4303	284	796.46	7711343	22	
6	4983264713004875783	2	366	4	3807.5	7485	21	High value customer
7	2402527199731150932	28	559	31	906.61	327010	19	

Features

Sifting through data can be time consuming

But what do we call all the other data columns in the data table?

Those columns are called features, or potential features. Each column of data is like a cooking ingredient you can use from the kitchen pantry. Too many ingredients, however, can ruin a dish!

The process of sifting through data can be time consuming. 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.

Feature engineering

Row	fullVisitorID	distinct_days_visited	ltv_pageviews	ltv_visits	ltv_avg_time_on_site_s	ltv_revenue	ltv_transactions	label
1	7813149964104484438	79	7395	138	479.63	624572	67	
2	7713012430069756739	2	514	6	1954.33	18194	35	High value customer
3	6760732402251466726	30	868	41	723.55	481282	34	
4	552667592603848032	1	466	1	7013.0	8796	25	High value customer
5	1957458976293878100	148	4303	284	796.46	7711343	22	
6	4983264713004875783	2	366	4	3807.5	7485	21	High value customer
7	2402527199731150932	28	559	31	906.61	327010	19	

Feature engineering

Calculate fields

You can even combine or transform feature columns in a process called feature engineering.

If you've ever created calculated fields in SQL, you've already executed the basics of feature engineering.

BigQuery ML does much of the hard work

BigQuery ML

01 Automatically one-hot encoding categorical values

02 Automatically splits the dataset into training data and evaluation data

BigQuery ML does much of the hard work for you, like automatically one-hot encoding categorical values. One-hot encoding is a method of converting categorical data to numeric data to prepare it for model training. From there, BigQuery ML automatically splits the dataset into training data and evaluation data.

Predicting on future data

Historical Training Data (Known LTV)

Row	fullVisitorID	distinct_days_visited	ltv_pageviews	ltv_visits	ltv_avg_time_on_site_s	ltv_revenue	ltv_transactions	label
1	7813149964104484438	79	7395	138	479.63	624572	67	
2	7713012430069756739	2	514	6	1954.33	18194	35	High value customer
3	6760732402251466726	30	868	41	723.55	481282	34	
4	552667592603848032	1	466	1	7013.0	8796	25	High value customer
5	1957458976293878100	148	4303	284	796.46	7711343	22	
6	4983264713004875783	2	366	4	3807.5	7485	21	High value customer
7	2402527199731150932	28	559	31	906.61	327010	19	

Future Data (Unknown LTV)

8	7904807859681747547	3	42	3	1162.0	null	null	????????????????????
9	4405445121320750966	51	358	62	517.36	null	null	????????????????????
10	1419607020881916790	5	22	5	711.0	null	null	????????????????????

And finally, there is predicting on future data.

Let's say new data comes in that you don't have a label for, so you don't know whether it is for a high-value customer. You do, however, have a rich history of labeled examples for you to train a model on.

Predicting results!

Historical Training Data (Known LTV)

Row	fullVisitorID	distinct_days_visited	ltv_pageviews	ltv_visits	ltv_avg_time_on_site_s	ltv_revenue	ltv_transactions	label
1	7813149964104484438	79	7395	138	479.63	624572	67	
2	7713012430069756739	2	514	6	1954.33	18194	35	High value customer
3	6760732402251466726	30	868	41	723.55	481282	34	
4	552667592603848032	1	466	1	7013.0	8796	25	High value customer
5	1957458976293878100	148	4303	284	796.46	7711343	22	
6	4983264713004875783	2	366	4	3807.5	7485	21	High value customer
7	2402527199731150932	28	559	31	906.61	327010	19	

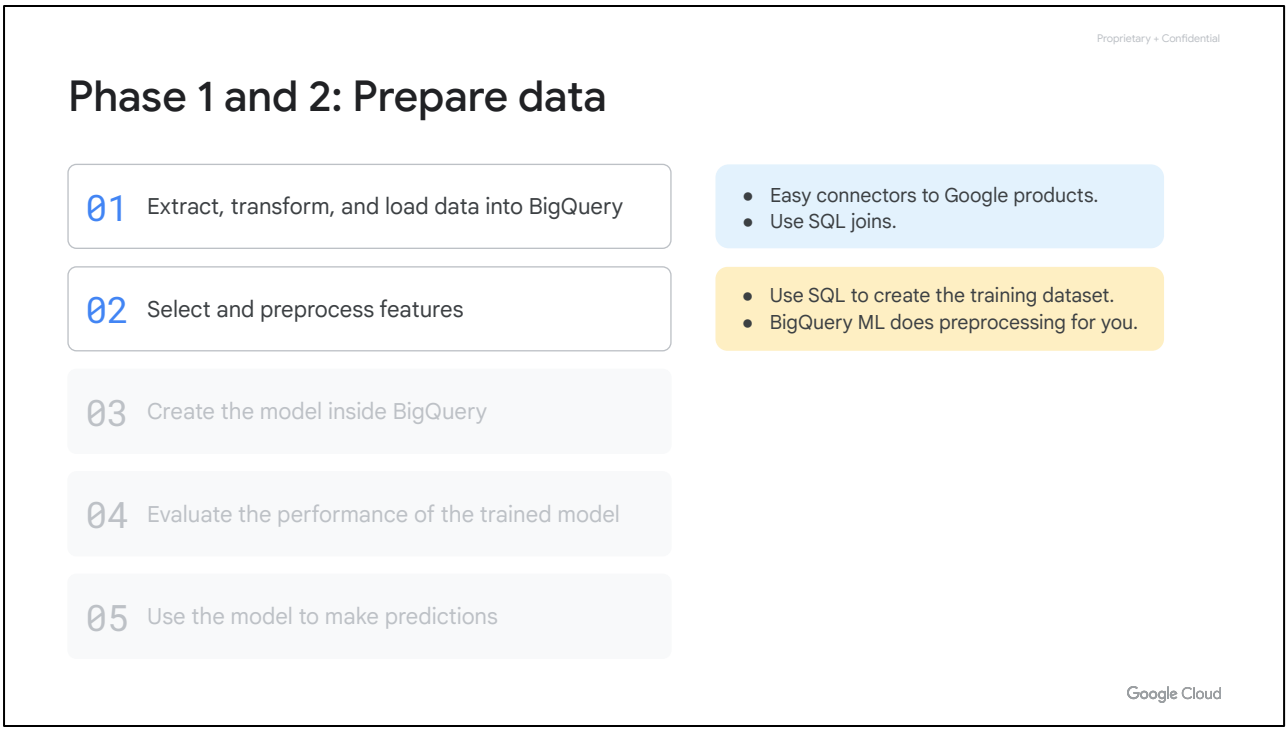
Future Data (Unknown LTV)

8	7904807859681747547	3	42	3	1162.0	null	null	High value customer
9	4405445121320750966	51	358	62	517.36	null	null	
10	1419607020881916790	5	22	5	711.0	null	null	High value customer

Google Cloud

So if we train a model on the known historical data, and are happy with the performance, then we can use it to predict on future datasets!

BigQuery ML project phases



Let's explore the key phases of a machine learning project.

In **phase 1,** you **extract, transform, and load data into BigQuery**, if it isn't there already. If you're already using other Google products, like YouTube for example, look out for easy connectors to get that data into BigQuery before you build your own pipeline. You can enrich your existing data warehouse with other data sources by using SQL joins.

In **phase 2**, you **select and preprocess features**. You can use SQL to create the training dataset for the model to learn from. You'll recall that BigQuery ML does some of the preprocessing for you, like one-hot encoding of your categorical variables. One-hot encoding converts your categorical data into numeric data that is required by a training model.

Phase 3: Create ML model

01 Extract, transform, and load data into BigQuery

02 Select and preprocess features

03 Create the model inside BigQuery

04 Evaluate the performance of the trained model

05 Use the model to make predictions

Use the “CREATE MODEL” command.

```
#standardSQL
CREATE MODEL
ecommerce.classification

OPTIONS
  (
   model_type='logistic_reg',
   input_label_cols =
   'will_buy_later'
   ) AS

# SQL query with training data
```


In **phase 3**, you **create the model** inside BigQuery. This is done by using the “CREATE MODEL” command. Give it a name, specify the model type, and pass in a SQL query with your training dataset. From there, you can run the query.

Phase 4: Evaluate ML model

01 Extract, transform, and load data into BigQuery

02 Select and preprocess features

03 Create the model inside BigQuery

04 Evaluate the performance of the trained model

05 Use the model to make predictions

Execute an ML.EVALUATE query.

```
#standardSQL
SELECT
   roc_auc,
   accuracy,
   precision,
   recall
FROM
  ML.EVALUATE(MODEL
`ecommerce.classification`)

# SQL query with evaluation data
```


In **phase 4**, after your model is trained, you can execute an ML.EVALUATE query to **evaluate the performance** of the trained model on your evaluation dataset. It's here that you can analyze loss metrics like a Root Mean Squared Error for forecasting models and area-under-the-curve, accuracy, precision, and recall, for classification models. We'll explore these metrics later in the course.

Phase 5: Make prediction

01 Extract, transform, and load data into BigQuery

02 Select and preprocess features

03 Create the model inside BigQuery

04 Evaluate the performance of the trained model

05 Use the model to make predictions

Invoke the ml.PREDICT command.

```
#standardSQL
SELECT * FROM
   ML.PREDICT
(MODEL
`ecommerce.classification`)

# SQL query with test data
```


In **phase 5**, the final phase, when you're happy with your model performance, you can then use it to **make predictions**. To do so, invoke the ml.PREDICT command on your newly trained model to return with predictions and the model's confidence in those predictions. With the results, your label field will have "predicted" added to the field name. This is your model's prediction for that label.

BigQuery ML key commands

Now that you're familiar with the key phases of an ML project, let's explore some of the key commands of BigQuery ML.

Create a model with CREATE MODEL

```
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
```


You'll remember from an earlier that you can create a model with just the CREATE MODEL command.

If you want to overwrite an existing model, use the CREATE OR REPLACE MODEL command.

Models have OPTIONS, which you can specify. The most important, and the only one required, is the model type.

Inspect what a 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
```

Output

Each feature has a weight from -1 to 1.

Closer to 0 = the **less important** the feature is to the prediction.

Closer to -1 or 1 = the **more important** the feature is to the prediction.

You can inspect what a model learned with the ML.WEIGHTS command and filtering on an input column.

The output of ML.WEIGHTS is a numerical value, and each feature has a weight from -1 to 1. That value indicates how important the feature is for predicting the result, or label. If the number is closer to 0, the feature isn't important for the prediction. However, if the number is closer to -1 or 1, then the feature is more important for predicting the result.

Evaluate the model's performance with ML.EVALUATE

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


To evaluate the model's performance, you can run an ML.EVALUATE command against a trained model. You get different performance metrics depending on the model type you chose.

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`)
)
)
```


And if you want to make batch predictions, you can use the ML.PREDICT command on a trained model, and pass through the dataset you want to make the prediction on.

BigQuery ML commands for supervised models

Labels	Identify column as 'label' or specify column in OPTIONS using `input_label_cols`.
Features	Data columns that are part of your SELECT statement, after your CREATE MODEL statement. `SELECT * FROM ML.FEATURE_INFO(MODEL `mydataset.mymodel`)`
Model object	An object created in BigQuery that resides in your BigQuery dataset.

Model types	Linear Regression, logistic Regression, etc.
	`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>))`

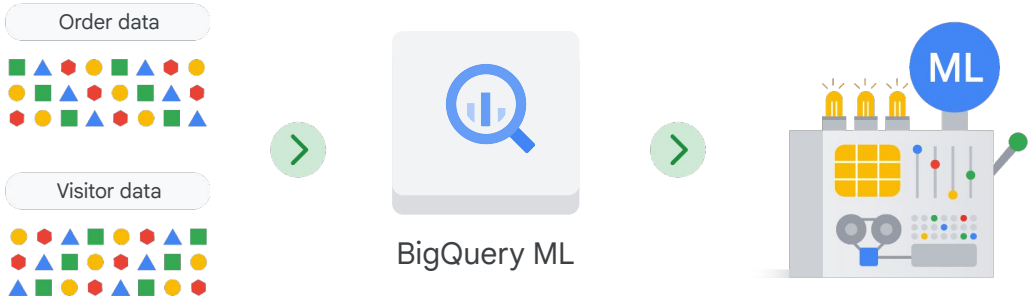
Now let's explore a consolidated list of BigQuery ML commands for supervised models.

- First in BigQuery ML, you need a field in your training dataset titled **LABEL**, or you need to specify which field, or fields, your labels are using as the input label columns in your model OPTIONS.
- Second, your model **features** are the data columns that are part of your SELECT statement after your CREATE MODEL statement. After a model is trained, you can use the ML.FEATURE_INFO command to get statistics and metrics about that column for additional analysis.
- Next is the **model object** itself. This is an object created in BigQuery that resides in your BigQuery dataset. You train many different models, which will all be objects stored under your BigQuery dataset, much like your tables and views. Model objects can display information for when it was last updated or how many training runs it completed.
- 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, such as next year's sales, consider linear regression for forecasting. If it's a discrete class like high, medium, low, or spam/not-spam, consider using logistic regression for classification.
- While the model is running, and even after it's complete, you can view **training progress** with ML.TRAINING_INFO.
- As mentioned earlier, you can **inspect weights** to see what the model learned about the importance of each feature as it relates to the label you're predicting. The importance is indicated by the weight of each feature.

- You can see how well the model performed against its **evaluation** dataset by using ML.EVALUATE.
- And lastly, getting **predictions** is as simple as writing **ML.PREDICT** and referencing your model name and prediction dataset.

Lab:
Predicting purchases with BigQuery ML

Hands-on lab



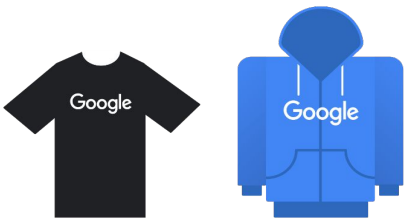
shop.googlemerchandisestore.com

Now it's time to get some hands-on practice building a machine learning model in BigQuery.

In the next lab, you'll use ecommerce data from the Google Merchandise Store website: https://shop.googlemerchandisestore.com/

The site's visitor and order data have been loaded into BigQuery, and you'll build a machine learning model to predict whether a visitor will return for more purchases later.

You'll practice



1. Loading data using BigQuery
2. Querying and exploring the dataset
3. Creating a training and evaluation dataset
4. Creating a classification model
5. Evaluating model performance
6. Predicting and ranking purchase probability

You'll get practice:

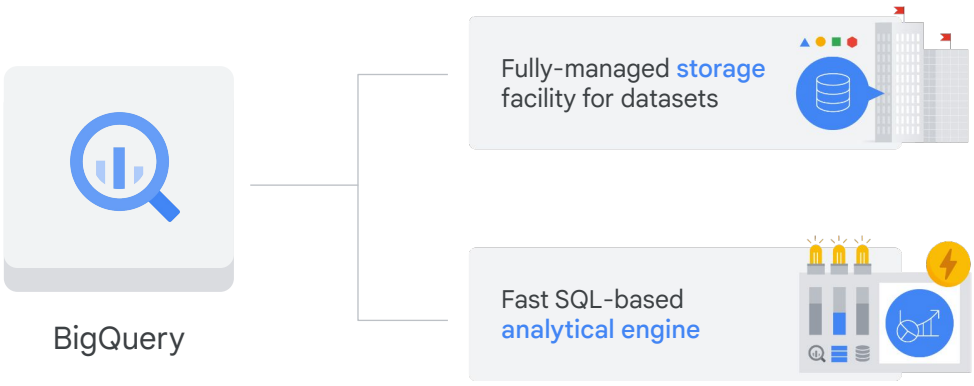
- Loading data into BigQuery from a public dataset.
- Querying and exploring the ecommerce dataset.
- Creating a training and evaluation dataset to be used for batch prediction.
- Creating a classification (logistic regression) model in BigQueryML.
- Evaluating the performance of your machine learning model.
- And predicting and ranking the probability that a visitor will make a purchase

Summary

Well done on completing another lab! Hopefully you now feel more comfortable building custom machine learning models with BigQuery ML!

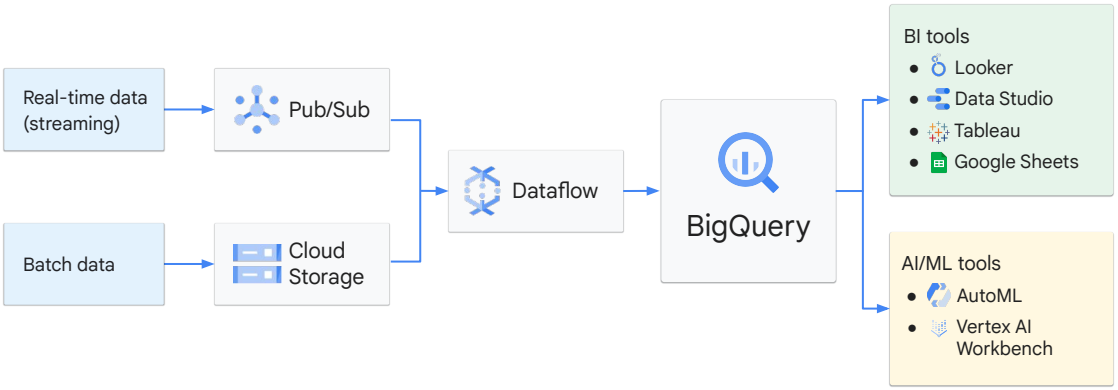
Let's review what we explored in this module of the course.

Storage and analytics



Our focus was on BigQuery, the data warehouse that provides two services in one. It's a fully-managed storage facility for datasets, and a fast SQL-based analytical engine.

BigQuery sits between data processes and data uses



BigQuery sits between data processes and data uses, like a common staging area. It gets data from ingestion and processing and outputs data to BI tools such as Looker and Data Studio and ML tools such as Vertex AI.

After the data is in BigQuery, business analysts, BI developers, data scientists, and machine learning engineers can be granted access to the data for their own insights.

Key phases of a BigQuery ML project

01 Extract, transform, and load data into BigQuery

02 Select and preprocess features

03 Create the model inside BigQuery

04 Evaluate the performance of the trained model

05 Use the model to make predictions

BigQuery ML

In addition to traditional data warehouses, BigQuery offers machine learning features. This means you can use BigQuery to directly build ML models in five key phases.

In **phase 1,** you **extract, transform, and load data into BigQuery**, if it isn't there already.

In **phase 2**, you **select and preprocess features**. You can use SQL to create the training dataset for the model to learn from.

In **phase 3**, you **create the ML model** inside BigQuery.

In **phase 4**, after your model is trained, you can execute an ML.EVALUATE query to **evaluate the performance of the trained model** on your evaluation dataset.

And in **phase 5**, the final phase, when you're happy with your model performance, you can then use it to **make predictions**.