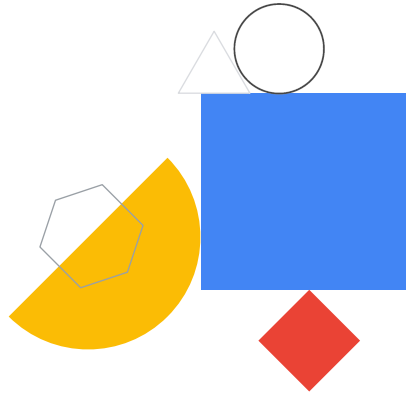



# Custom Model Building with SQL in BigQuery ML



BigQuery Machine Learning allows you to build machine learning models using SQL syntax. This is a really powerful feature since anyone who knows only SQL can build ML models. BigQuery ML is a step further in democratizing Machine Learning and AI.



# Module agenda

01 BigQuery ML for Quick Model Building

---

02 Supported Models

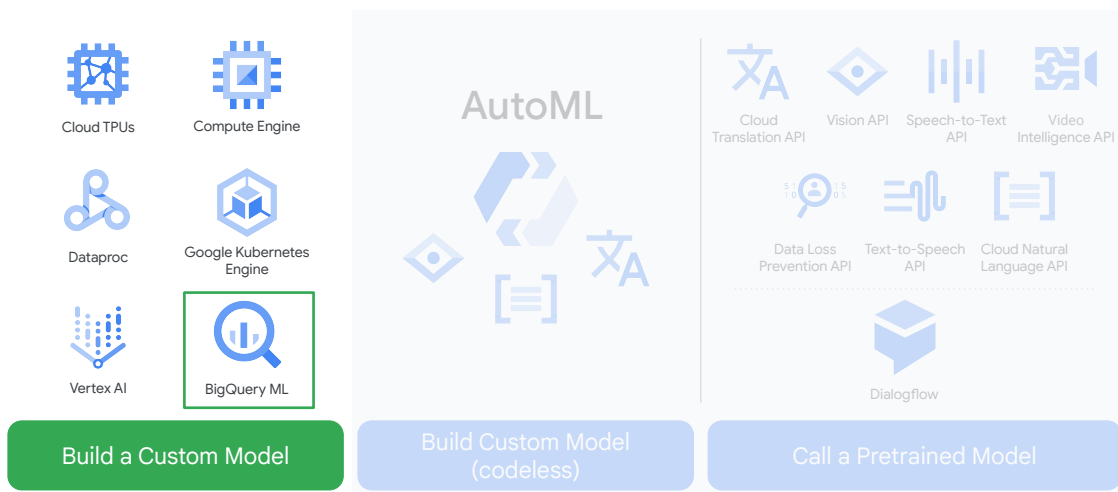
First, we'll introduce BigQuery ML and walk through the steps for an end-to-end use case. We'll then discuss the supported models. At the end, you'll work through a couple of labs, so you can get your hands on BigQuery ML.



## BigQuery ML for Quick Model Building

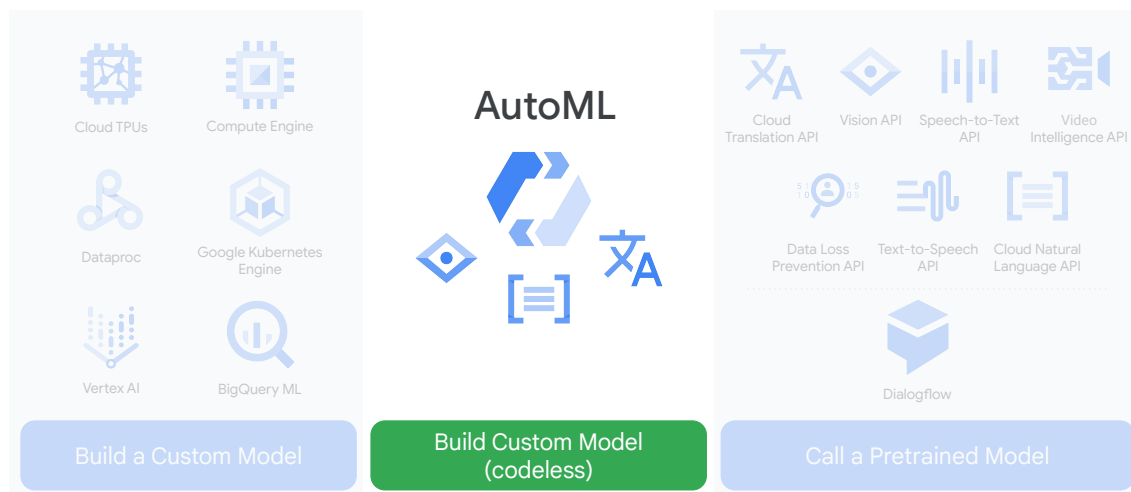
Let's start by identifying where BigQuery ML fits into the greater picture of Google Cloud's AI and machine learning options.

# BigQuery ML is a way to build custom models



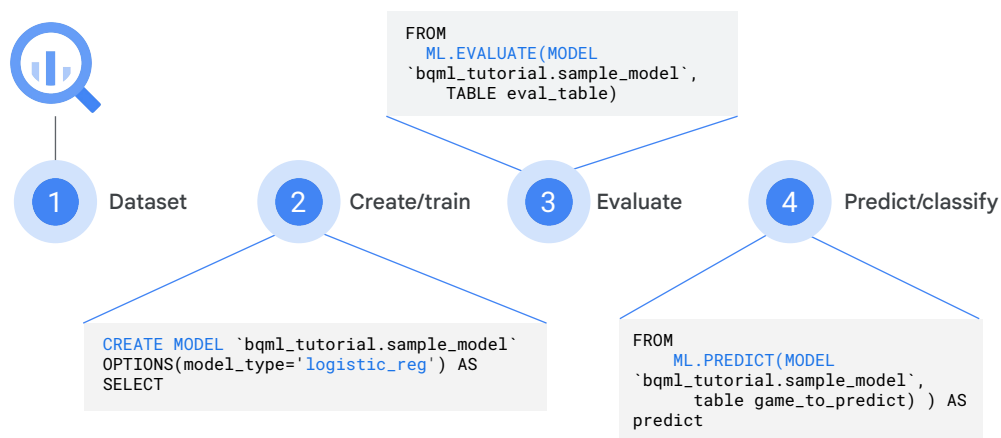
Well, unlike the ML APIs, you are able to create your own custom models with BigQuery ML.

# BigQuery ML is a way to build custom models



We'll get to AutoML later, but in short, AutoML allows us to leverage Google's ML models in certain cases to build your own models from scratch using transfer learning and a form of Neural Architecture Search.

# Working with BigQuery ML



Google Cloud

To work with BigQuery ML, there are only a few steps from beginning to inference.

- First, you must **write a query** on data stored in BigQuery to extract your training data.
- Then, you can **create a model**, where you specify a model type and other hyperparameters.
- After the model is trained, you can **evaluate the model** and verify that it meets your requirements.
- Finally, you can **make predictions** using your model on data extracted from BigQuery.

## Where was this article published?

- 1 Techcrunch
- 2 GitHub
- 3 NY Times

### Unlikely Partnership in House Gives Lawmakers Hope for Border Deal

Representatives Nita M. Lowey and Kay Granger are the first women to lead the House Appropriations Committee. Their bond gives lawmakers optimism for the work to come.

By EMILY COCHRANE



### Fitbit's newest fitness tracker is just for employees and health insurance members

Fitbit has a new fitness tracker, but it's one that you can't buy in stores. The company quietly uncorked the Inspire on Friday, releasing its first product that is available only to co...



1 hour ago Jon Russell

### Downloading the Android Studio Project Folder

FTC Engineering edited this page on Sep 19, 2017 · 1 revision

Downloading the Android Studio Project Folder

Are you familiar with Hacker News? Hacker News is a social news aggregator website focused on computer science and entrepreneurship. I won't get into all of the details, but let's ask a question. Given these three articles on the right, where were they actually published? Techcrunch, GitHub or The New York Times? Of course, you may be able to tell based on the style of the articles, but what if you wanted to build an ML model to do this?

## SQL query to extract data

```
SELECT
  url, title
FROM
  `bigquery-public-data.hacker_news.stories`
WHERE
  LENGTH(title) > 10
  AND LENGTH(url) > 0
LIMIT 10
```

\*no clusters, no indexes,  
ad hoc query!

uri	title
http://www.bbc.co.uk/news/business-27732743	Vodafone reveals direct government wiretaps
https://www.kickstarter.com/projects/appdocu/a...	Doc - App: The Human Story
http://www.starwebworld.com/android-jelly-bean...	Android Jelly Bean: Streaming Audio Through th...
http://www.myplanetdigital.com/digital_strateg...	Why Canadian Tech Entrepreneurs Need to Man/Wo...
http://startupislandconference.com/index.html	StartupConference June 13. - 16. 2013, HVAR Cr...
http://kopimism.org/	Kopimism Hactivism Meetup Tomorrow (Sunday) in...
http://uneearthedgadget.com/xbox-live-gold-2/14...	Xbox Live Gold Membership Is It Really Worth ~...
https://evertale.com	Evertale changes the way people remember
http://www.racketboy.com/retro/commodore-amiga...	Commodore Amiga: A Beginner's Guide
http://www.extremetech.com/extreme/156393-cold...	Cold fusion reactor 'independently verified'

First, let's write an ad hoc query to explore your data. Here's a quick example where you're simply looking at the title of the article and the url. You can extract the publisher information from the URL using regular expression functions.



# Use regex to get **source** + train on **words** of title

txtclass\_words LINK SHARING

```

1 WITH extracted AS (
2   SELECT source, REGEXP_REPLACE(LOWER(REGEXP_REPLACE(title, '[a-zA-Z0-9 $-]', ' ')), '
3   FROM
4   (SELECT
5     ARRAY_REVERSE(SPLIT(REGEXP_EXTRACT(url, '.*?/(.*)?$', '')[OFFSET(1)] AS source
6     title
7   FROM
8     bigquery-public-data.hacker_news.stories
9   WHERE
10    REGEXP_CONTAINS(REGEXP_EXTRACT(url, '.*?/(.*)?$', '')[OFFSET(1)], '.com$')
11    AND LENGTH(title) > 10
12  )
13 )
14 , ds AS (
15   SELECT ARRAY_CONCAT(SPLIT(title, ' '), ['NULL', 'NULL', 'NULL', 'NULL', 'NULL']) AS words
16   extracted
17   WHERE (source = 'github' OR source = 'nytimes' OR source = 'techcrunch')
18 )
19 SELECT
20   source,
21   words[OFFSET(0)] AS word1,
22   words[OFFSET(1)] AS word2,
23   words[OFFSET(2)] AS word3,
24   words[OFFSET(3)] AS word4,
25   words[OFFSET(4)] AS word5
26 FROM ds

```

Run Save query Save view More This query will process 204

Query results SAVE RESULTS EXPLORE IN DATA STUDIO

37293	nytimes	the	socratic	shrink	NULL	NULL
37294	nytimes	still	stuck	in	a	climate
37295	nytimes	as	unlimited	data	plans	are
37296	nytimes	disney	s	neuroscience	advertising	lab
37297	nytimes	hold	that	thought	the	google

That's exactly what you'll do using the query on this slide. The plan will be the following:

Extract the publication source using a REGEXP\_EXTRACT function and then separate out the first 5 words of each title, or replace them with NULL values in the case the title is too short. The goal will then be to decide if the article title is from Github, The New York Times or TechCrunch using BigQuery ML.

# Create model

Query to extract training data

```
CREATE OR REPLACE MODEL advdata.txtclass
OPTIONS(model_type='logistic_reg', input_label_cols=['source'])
AS
```

```
WITH extracted AS (
...
), ds AS (
SELECT ARRAY_CONCAT(SPLIT(title, " "), ['NULL', 'NULL', 'NULL',
'NULL', 'NULL']) AS words, source FROM extracted
WHERE (source = 'github' OR source = 'nytimes' OR source =
'techcrunch')
)

SELECT
source,
words[OFFSET(0)] AS word1,
words[OFFSET(1)] AS word2,
words[OFFSET(2)] AS word3,
words[OFFSET(3)] AS word4,
words[OFFSET(4)] AS word5
FROM ds
```

Let's look at the syntax needed to create a model. You have the CREATE OR REPLACE MODEL statement at the top where you give the name of the model, the options of the model and other hyperparameters. After the AS clause you have your query which defines the training set from the previous slide. That's it! In this case you're specifying that you want to build a logistic regression model. This is a type of classification model you can use to classify your input as being either a GitHub, New York Times, or TechCrunch article.

[\[https://towardsdatascience.com/choosing-between-tensorflow-keras-bigquery-ml-and-automl-natural-language-for-text-classification-6b1c9fc21013\]](https://towardsdatascience.com/choosing-between-tensorflow-keras-bigquery-ml-and-automl-natural-language-for-text-classification-6b1c9fc21013)

## Evaluate model

```
SELECT * FROM ML.EVALUATE(MODEL advdata.txtclass)
```

precision	recall	accuracy	f1_score	log_loss	roc_auc
0.783	0.783	0.79	0.783	0.858	0.918

Actual labels	Predicted labels			
	github	nytimes	techcrunch	% samples
github	88.8%	5.29%	5.9%	37.83%
nytimes	6.34%	70.92%	22.74%	31.26%
techcrunch	5.54%	19.35%	75.11%	30.9%

(BigQuery ML splits the training data and reports evaluation statistics on the held-out set)

To get the metrics, you simply call ML.EVALUATE on your trained model or click on the model in the BigQuery UI and click the Evaluation tab.

For the metrics, they're on a score from 0 to 1. Generally speaking, the closer to 1 the better but it really depends on your metric.

Precision means for the articles you made a guess on, how accurate were those guesses. High precision means a low false positive rate meaning you really punish a model's precision if it makes a ton of bad guesses.

Recall on the other hand, is the ratio of correctly predicted positive observations to the all observations in actual class. How many did you get right out of both True Positives and False Negatives.

Accuracy is simply true positives plus true negatives over the entire set of observations.

There are other metrics like f1\_score, log\_loss, and roc\_curves that you can use as well.

You can also see the confusion matrix, where you can see where the model made incorrect predictions. As you can see here, the model had some confusion between Techcrunch and New York Times articles.

## Predict using trained model

```
SELECT * FROM ML.PREDICT(MODEL advdata.txtclass,(
  SELECT 'government' AS word1, 'shutdown' AS word2, 'leaves' AS word3,
  'workers' AS word4, 'reeling' AS word5
  UNION ALL SELECT 'unlikely', 'partnership', 'in', 'house', 'gives'
  UNION ALL SELECT 'fitbit', 's', 'fitness', 'tracker', 'is'
  UNION ALL SELECT 'downloading', 'the', 'android', 'studio', 'project'
))
```

“Batch  
prediction”

Row	Predicted_source	word1	word2	word3	word4	word5
1	nytimes	government	shutdown	leaves	workers	reeling
2	nytimes	unlikely	partnership	in	house	gives
3	techcrunch	fitbit	s	fitness	tracker	is
4	techcrunch	downloading	the	android	studio	project

If the model meets your requirements, great! You're ready to predict with your model. You don't need to worry about deploying your model in a separate process. It's automatically available to serve predictions in BigQuery. Here's an example of performing batch prediction using BigQuery ML. The first example here has “government, shutdown, leaves, workers, and reeling” as the first five words of the title. The model's prediction in this case is “The New York Times”.



## Supported Models

Before you start your first lab using BigQuery ML, I'll give a brief overview of the various model types that are currently supported. For the article prediction task in the previous lesson, recall that you utilized a logistic regression model. There are other models that could have been used. Let's look at the model options available in BigQuery ML to classify things.

# Linear Classifier (Logistic regression)

```
#standardsql
CREATE OR REPLACE MODEL flights.ontime
OPTIONS
  (model_type='logistic_reg', input_label_cols=['on_time']) AS
SELECT
  IF(arr_delay < 15, 1, 0) AS on_time,
  carrier,
  origin,
  dest,
  dep_delay,
  taxi_out,
  distance
FROM
  `cloud-training-demos.flights.tzcorr`
WHERE
  arr_delay IS NOT NULL
```

First, you have another example of linear classification, or what is also called logistic regression.

In this example, you're using a flights arrivals dataset to predict whether a flight will be on time or not - a binary outcome. For logistic regression, you simply need to specify the type of model and the label you're trying to predict. For more advanced users there are other options, such as if you want your model to use regularization.

# DNN Classifier

```
#standardsql
CREATE OR REPLACE MODEL flights.ontime
OPTIONS
  (model_type='dnn_classifier', hidden_units = [47,29,18],
   input_label_cols=['on_time']) AS
SELECT
  IF(arr_delay < 15, 1, 0) AS on_time,
  carrier,
  origin,
  dest,
  dep_delay,
  taxi_out,
  distance
FROM
  `cloud-training-demos.flights.tzcorr`
WHERE
  arr_delay IS NOT NULL
```

While logistic regression models are the Swiss Army knife of machine learning, deep neural networks, or DNNs, allow you to better model non-linear relationships in your data.

An example of a non-linear relationship would be in car depreciation. A car loses a vast majority of its value within the first few years, after which the value more or less stabilizes. We won't go into the details of DNNs in this course, but know that you can use them in BigQuery ML. All you have to do is specify DNN as the model type in the CREATE OR REPLACE MODEL header.

This slide shows how to do that for the same flight arrival time example used in the previous slide.

## xgboost Classifier

```
#standardsql
CREATE OR REPLACE MODEL flights.ontime
OPTIONS
  (model_type='boosted_tree_classifier', input_label_cols=['on_time']) AS
SELECT
  IF(arr_delay < 15, 1, 0) AS on_time,
  carrier,
  origin,
  dest,
  dep_delay,
  taxi_out,
  distance
FROM
  `cloud-training-demos.flights.tzcorr`
WHERE
  arr_delay IS NOT NULL
```

If you're familiar with gradient boosted trees and XGBoost, you can use this algorithm to train models in BigQuery ML.

In this example, you specify the model type as “boosted\_tree\_classifier” to use gradient boosted trees as your classifier. There are a number of hyperparameters you can select for this model, including the maximal tree depth.

Again, I won't go into fine detail here, but the idea behind gradient boosted trees is to iteratively fit a decision tree to the residuals of preceding decision trees.



# Linear Regression

```
CREATE OR REPLACE MODEL
  taxi.taxifare_dnn OPTIONS (model_type='linear_reg', labels=['fare_amount']) AS
SELECT
  fare_amount,
  hourofday, dayofweek,
  pickuplon, pickuplat, dropofflon, dropofflat,
  passenger_count
FROM
  `taxi.taxi3m`
```

Now we'll look at the model options available in BigQuery ML to forecast things like numeric values.

So far I've only talked about classification. You can also use BigQuery ML to do regression. In this example you're fitting a linear regression model to predict taxi fare based on features such as the hour of day, pickup and drop-off location, and the day of the week.

# DNN Regression

```
CREATE OR REPLACE MODEL
  taxi.taxifare_dnn OPTIONS (model_type='dnn_regressor',
    hidden_units=[144,89,55], labels=['fare_amount']) AS
SELECT
  fare_amount,
  hourofday, dayofweek,
  pickuplon, pickuplat, dropofflon, dropofflat,
  passenger_count
FROM
  `taxi.taxi3m`
```

If you need a model more complex than linear regression you can use a DNN regressor in BigQuery ML. All you have to do is specify the model type as “dnn\_regressor” and define the hidden units, just like in the classification version of DNNs.

## xgboost Regression

```
CREATE OR REPLACE MODEL
  taxi.taxifare_xgboost
  OPTIONS (model_type='boosted_tree_regressor', labels=['fare_amount']) AS
SELECT
  fare_amount,
  hourofday, dayofweek,
  pickuplon, pickuplat, dropofflon, dropofflat,
  passenger_count
FROM
  `taxi.taxi3m`
```

Just like for classification, you can also use XGBoost for regression! Just specify you want to use a “boosted\_tree\_regressor”.

# Train on TF, predict with BigQuery

```
CREATE OR REPLACE MODEL advdata.txtclass_tf2
OPTIONS (model_type='tensorflow',

model_path='gs://cloud-training-demos-ml/txtcls/trained_finetune_native/export/exporter/1
549825580/*')
```

```
SELECT
  input,
  (SELECT AS STRUCT(p, ['github', 'nytimes', 'techcrunch'][ORDINAL(s)]) prediction FROM
    (SELECT p, ROW_NUMBER() OVER() AS s FROM
      (SELECT * FROM UNNEST(dense_1) AS p))
    ORDER BY p DESC LIMIT 1).*

FROM ML.PREDICT(MODEL advdata.txtclass_tf2,
(
  SELECT 'Unlikely Partnership in House Gives Lawmakers Hope for Border Deal' AS input
  UNION ALL SELECT "Fitbit\'s newest fitness tracker is just for employees and health
insurance members"
  UNION ALL SELECT "Show HN: Hello, a CLI tool for managing social media"
))
```

What if you have already trained a model in TensorFlow? Well, you can import your saved model into BigQuery to serve batch predictions.

In this example, the first statement is where you load your model into BigQuery. You specify the model name, the model path in Cloud Storage, and you tell BigQuery that the model type is “tensorflow”.

When you want to serve predictions, you can do it as shown on the bottom half of the query in this example. Notice the ML.PREDICT statement being used to generate predictions, and then that actual query is being used to print the predictions in a user-friendly format.

## Recommendation engine (matrix factorization)

```
create or replace model models.suggested_products_1or2_example
options(model_type='matrix_factorization',
        user_col='user_id', item_col='product_id', rating_col='rating',
        l2_reg=10)
AS

with purchases AS (
  select product_id, user_id from
  operations.orders_with_lines, unnest(order_lines)
),

total_purchases as (
  select product_id, user_id, count(*) as numtimes
  from purchases
  group by product_id, user_id
)

select
product_id, user_id,
IF(numtimes < 2, 1, 2) AS rating
FROM total_purchases
```

Google Cloud

Let's now look at the model options available in BigQuery ML to recommend things based on ratings.

You can also serve recommendations to users in BigQuery ML! The type of model being used for this is called a matrix factorization model. In short, matrix factorization models are a form of collaborative filtering algorithms used in recommender systems.

Matrix factorization algorithms work by decomposing the user-item interaction matrix. That is, the matrix consisting of users and recommendations gets broken apart into the product of two lower dimensionality rectangular matrices. The full details of this can be found in many different online sources for recommendation systems.

This family of methods became widely known during the Netflix Prize Challenge due to its effectiveness in making recommendations.

To build a recommendation engine in BigQuery ML you must set the model type to "matrix factorization" and provide a table of user-item interactions.

## So what do we recommend for a given set of users?

```
with users AS (  
  SELECT  
    user_id, count(*) as num_orders  
  from operations.orders_with_lines  
  group by user_id  
  order by num_orders desc  
  limit 10  
,  
  
  products as (  
    select product_id, count(*) as num_orders  
    from operations.orders_with_lines, unnest(order_lines)  
    group by product_id  
    order by num_orders desc  
    limit 10  
  )  
  
SELECT * FROM ML.PREDICT(MODEL models.suggested_products_1or2,  
(SELECT user_id, product_id  
FROM users, products)  
)
```

Once you have a trained recommender model, you can use it to serve recommendations! Here's an example query where you're getting your users and products with a query, and getting the suggested products using the ML.PREDICT statement.

## So what do we recommend for a given set of users?

Row	predicted_rating	user_id	product_id
1	1.5746015507788755	101797	26209
2	1.8070705987455633	101797	13176
3	1.7171094544245578	101797	27845
4	1.9763373899260837	101797	47209
5	1.8659380090171271	101797	21137
6	1.721610848530093	101797	47766
7	1.9516130703939483	101797	21903

Here's what the output will look like for a single user. You could sort by the predicted rating for each user\_id and give the top 5 or so recommendations using a simple SQL query.

# Clustering

```
CREATE OR REPLACE MODEL
demos_eu.london_station_clusters
OPTIONS(model_type='kmeans', num_clusters=4,
standardize_features = true) AS
```

```
WITH hs AS ...,
stationstats AS ...
```

```
SELECT * except(station_name, isweekday)
from stationstats
```

- 1 4 clusters (hardcoded)
- 2 Standardize features since different dynamic ranges
- 3 Remove the cluster "id" fields (keep just the attributes)

Finally, I'll discuss an unsupervised learning algorithm using K-Means Clustering that's available in BigQuery ML. I won't go into the details of the algorithm, but in short, in this case you don't have a label that you are trying to predict. Rather, you're trying to explore patterns in your data.

In this example, you'll cluster the data into four clusters, standardize the features to account for different feature value ranges, which is really important for clustering, and remove any id fields so that you're not grouping based on uniquely identifying information.



## Which cluster?

```
WITH hs AS ...,
stationstats AS ...,

SELECT * except(nearest_centroids_distance)
FROM ML.PREDICT(MODEL demos_eu.london_station_clusters,
(SELECT * FROM stationstats WHERE
REGEXP_CONTAINS(station_name, 'Kennington'))))
```

Row	CENTROID_ID	station_name	isweekday	duration	num_trips	bikes_count	distance_from_city_center
1	3	Kennington Lane Tesco, Vauxhall	weekday	911.5810637908974	5471	9	1.8345619962343163
2	3	Kennington Lane Rail Bridge, Vauxhall	weekday	979.3919952622995	20263	19	2.175032834765301
3	4	Doddington Grove, Kennington	weekday	1397.7189755200225	7067	28	1.468140527379382
4	4	Kennington Cross, Kennington	weekday	911.5238777770538	15349	35	1.4625875338501981

After you've trained a clustering model you can easily see which cluster your data belongs to with a quick SQL query. Here's an example of the query to do so. The CENTROID\_ID column refers to the cluster assigned to each row of data. The dataset used here is for a bike rental company that has stations spread out across London.

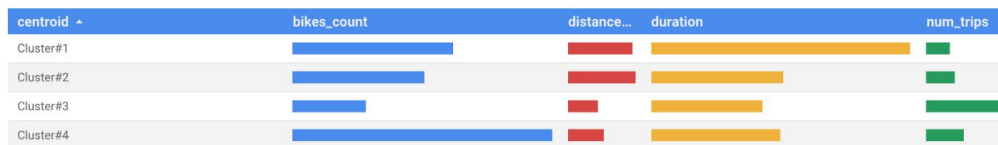
## Find cluster attributes

```
WITH T AS (  
  SELECT  
    centroid_id,  
    ARRAY_AGG(STRUCT(numerical_feature AS name, ROUND(feature_value,1) AS value) ORDER BY  
      centroid_id) AS cluster  
  FROM ML.CENTROIDS(MODEL demos_eu.london_station_clusters)  
  GROUP BY centroid_id  
)  
SELECT  
  CONCAT('Cluster#', CAST(centroid_id AS STRING)) AS centroid,  
  (SELECT value from unnest(cluster) WHERE name = 'duration') AS duration,  
  (SELECT value from unnest(cluster) WHERE name = 'num_trips') AS num_trips,  
  (SELECT value from unnest(cluster) WHERE name = 'bikes_count') AS bikes_count,  
  (SELECT value from unnest(cluster) WHERE name = 'distance_from_city_center') AS  
  distance_from_city_center  
FROM T  
ORDER BY centroid_id ASC
```

Once you have clusters, how do you actually get utility out of them? The key is to do some analysis on the data in each cluster. You can explore feature ranges and aggregate statistics to try to understand what is “special” about the individual clusters.

## Visualize attributes in Google Data Studio

Row	centroid	duration	num_trips	bikes_count	distance_from_city_center
1	Cluster#1	3079.5	3026.1	14.0	6.2
2	Cluster#2	1564.0	3635.1	11.5	6.5
3	Cluster#3	1319.6	9654.8	6.4	2.9
4	Cluster#4	1527.7	4846.8	22.6	3.5



For example, you could use Google Data Studio to visualize some of the attributes of the data in the clusters. For example, here you can see that Cluster 1 has the longest duration rides.

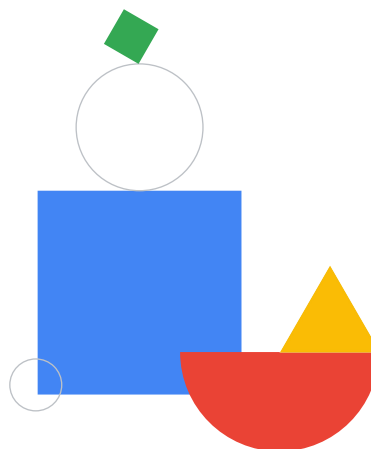
## Reminder: BigQuery ML Cheatsheet

- **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>))`

We've covered a lot of syntax in BigQuery ML. Here's a quick cheat sheet with syntax for training a model, evaluating it, and making predictions, among other things. Use this cheat sheet to help you when you first start building models in BigQuery ML.

## Lab Intro

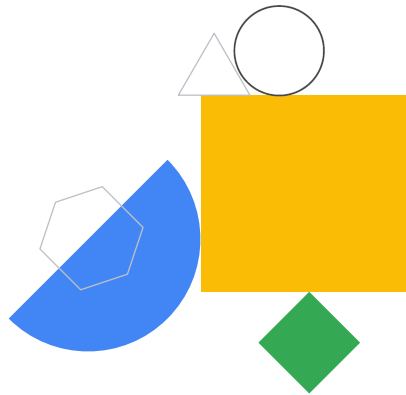
Predict Bike Trip Duration with a  
Regression Model in BigQuery ML



Let's get your hands on BigQuery ML. In this first of two labs, you'll predict bike trip duration using BigQuery ML by building a linear regression model. After training your model, you'll evaluate your model performance and extract the model weights to understand feature importance.

## Lab Intro

Movie Recommendations in  
BigQuery ML



In this second BigQuery ML lab, you'll generate movie recommendations. You'll train a recommendation model using a matrix factorization approach. Then, you'll see how to make product predictions for both single users and batch users.

## Summary

You can train and evaluate machine learning models directly in BigQuery.

The key takeaway of this course is that you can build powerful machine learning models in BigQuery. If your data lives in BigQuery, this is a very useful functionality since you don't have to worry about moving your data elsewhere to train your model.