

▼ CS145: Project 3 | ML Warmup (10 points)

Notes (read carefully!):

- Be sure you read the instructions on each cell and understand what it is doing before running it.
- Don't forget that if you can always re-download the starter notebook from the course website if you need to.
- You may create new cells to use for testing, debugging, exploring, etc., and this is in fact encouraged! Just make sure that the final answer for each question is **in its own cell** and **clearly indicated**.
- Colab will not warn you about how many bytes your SQL query will consume. **Be sure to check on the BigQuery UI first before running queries here!**
- This project may be done alone or in pairs.
- See the assignment handout for submission instructions.

▼ Setting Up BigQuery and Dependencies

Run the cells below (shift + enter) to authenticate your project.

Note that you need to fill in the `project_id` variable with the Google Cloud project id you are using for this course. You can see your project ID by going to <https://console.cloud.google.com/cloud-resource-manager>

```
# Run this cell to authenticate yourself to BigQuery
from google.colab import auth
auth.authenticate_user()
project_id = "project3-db-346816"
```

```
# Initialize BigQuery client
from google.cloud import bigquery
client = bigquery.Client(project=project_id)
```

▼ Overview

This part of Project 3 is meant serve as a brief tutorial for Machine Learning with BigQuery, since you will be using BigQuery Prediction in the final portion of your explorations.

Don't worry if you've never studied Machine Learning before. This notebook will guide you through everything you need to know to be successful in the open-ended part of Project 3.

In the next two sections, we'll give you a bird's eye intro to machine learning and a primer on how BigQuery makes machine learning easy. In the third and last section, you'll walk through an example of how to train and use a machine learning model in BigQuery.

▼ Section 1: Machine Learning in a Nutshell

Basic Machine Learning tasks can be framed in terms of **inputs** X , **target values** Y (sometimes called labels), **training data** (pairs of observed data points (x_i, y_i)), and a function $h : X \rightarrow Y$ historically called the **hypothesis function** that maps inputs to target values.

Given these primitives, we can think of the canonical Machine Learning task as follows:

Given that I've seen a ton of training data $(x_1, y_1), \dots, (x_m, y_m)$, how can I come up with a good function h so that on an *unseen* input value x_{m+1} , the value of $h(x_{m+1})$ is a good "prediction" y_{m+1} ?

Example 1: Three Point Shots

Say elements of X are the number of three point shots scored by a team in a basketball game, and elements of Y are 0 or 1 indicating whether that team lost or won the game. Our training data could look like this:

$$T = \{(2, 0), (3, 0), (6, 0), (5, 0), (10, 0), (11, 0), (5, 1), (15, 1), (18, 0), (17, 1), (16, 1), (16, 1)\}$$

In this case, we'd *train* a machine learning model on T to effectively generate an h that would give us reasonable values of y for unseen values of x , i.e., predict whether a game was won or not based on how many three pointers were scored on that game. For example, we might expect that $h(1) \approx 0$, and $h(20) \approx 1$. Note that in this case, we'd like h to output not only a 0 or 1 but a *probability* for how likely a game is to be won, hence the approximate equalities.

▼ Example 2: GitHub Revisited

For a richer example, let's say we are trying to predict how many *forks* (i.e., copies of the repo by GitHub users besides the original owner) a Github repo will have at some point in time -- here Y will represent the number of forks of a repo. As you might have seen in Project 2, such questions are usually not easily answerable with only one or two statistics of a Github repo. We usually want to think about several *features* together. What are the watch and star count of the repo? How many contributors does the repo have? How many files does the repo have? What is the age of the repo in years?

The values in X *need not be single real numbers*, they can be lists of real numbers as well, and we can use feature engineering to come up with these feature lists.

Feature engineering is the most informal process by which we use domain-knowledge to extract numerical features from some entity (e.g., a GitHub repo in this example), to provide them as training data to a machine learning model.

Here is how a simple feature engineering process for predicting the fork count of a GitHub repo may pan out:

Simple feature engineering process

1. Using domain knowledge, you hypothesize that watch count, star count, number of commits, and age of the repo will probably be good indicators of its fork count. You also toss in the average commit length of the repo because you know it has some non-trivial relationship with watch count based on anecdotal evidence from a certain project in your friend's database course.
2. You write some code in your favorite programming language (or SQL if using BigQuery!) to extract these five features from your set of 1,000 GitHub repos, creating 1,000 tuples that look like this:

$((45, 100, 200, 2.4, 127.65), 30), ((65, 302, 100, 1.2, 164.1), 132) \dots$

3. You train your model on this data and evaluate it on another 100 repos **which your model has not yet seen**. If the quality of your results (the accuracy of your predictions) is not so good, you may attempt to improve the quality of your features. If performance is good, you are done and have a decent model!
4. If you think the current features you thought about are not good enough, you can go back to 1 and brainstorm more.

Once you have honed in on a good set features which you have trained with and evaluated, you can now predict using your model. This should be done on an additional test dataset that **your model has also not yet seen**.

Evaluating your Models

In Example 2, we said that the feature engineering process involves a key step in which you *evaluate* how good your model (aka hypothesis function h) is doing. Usually this consists of:

1. Running your hypothesis function h on a set of inputs X which you have not already seen to get outputs $h(x_{m+1}), \dots, h(x_{m+k})$
2. Comparing how close your predicted values are to the ground-truth labels y_{m+1}, \dots, y_{m+k} using a reasonable statistical metric(s).

The "reasonable statistical metric" varies depending on the nature of your labels.

Here's a very brief overview of when you might want to use different metrics:

- **Accuracy** - when your classes are balanced (roughly same number of examples expected for each category)
- **F1-score** - when your classes are very unbalanced (some categories are expected to have way more examples than other categories)
- **Recall** - when you are more willing to have false positives than false negatives (e.g., predicting rare cancer - you'd rather have a false alarm than miss an actual case)
- **Precision** - when you are more willing to have false negatives than false positives (e.g., predicting spam emails - you'd rather have some spam in your inbox than have important emails go to spam)
- **RMS error** - when trying to predict a real value

There are many more ways to evaluate models, but deeper discussion is beyond the scope of this assignment and course.

Data Splits

Overall, when doing machine learning, you should have three datasets:

- Training dataset - the bulk of your data, which you use to let your model learn a hypothesis function h
- Validation dataset - a small portion of your data, used to evaluate your trained model as you try out different features and other possible hyperparameters.
- Test dataset - a small portion of your data, used to evaluate your **final model** when you are done picking features and tuning your model. You should not be running any models on this data to help choose relevant features.

Common splits for machine learning datasets are to take 80% of data for training, 10% for validation, and 10% for testing, but many other splits are acceptable depending on how much data is available.

▼ Types of Models

BigQuery supports three types of models: *linear regression*, *binary logistic regression*, and *multiclass logistic regression*.

- A *linear regression model* predicts a number, i.e., $Y = \mathbb{R}$.
- A *binary logistic regression model* makes a binary prediction by giving the confidence of an event, e.g., is an email spam or not?
- A *multiclass logistic regression model* is a generalization of the binary logistic regression model. E.g., what is the sentiment bucket of a sentence, from 1 (negative) to 5 (positive).

Example 1 above is a binary logistic regression model, and Example 2 is a linear regression model. You can use any of these three models in your project.

If you have not already studied machine learning and are interested in digging into more details, reading section 1 of the CS229 notes [here](#) will cover the basic topics discussed here and expand on them. However, the information in this notebook will be sufficient to complete Project 3.

▼ Section 2: BigQuery and ML

In the previous section, we did not cover how the hypothesis function h is actually generated from training data. Luckily for us, BigQuery abstracts the details of this process away from us and instead exposes a nice SQL interface for ML which we already know how to work with!

Machine Learning in BigQuery consists of three steps: creating a model, evaluating the model, and using the model to make predictions.

▼ Creating a Model

This step consists of telling BigQuery that you want to create a model. You tell BigQuery what type of model you want to create, and you write SQL to gather the features and ground-truth values for the model.

The create model statement could look like this:

```
# Don't run me! My tables don't exist. I'm just here as an example.
%%bigquery --project $project_id

CREATE MODEL `my_awesome_model`
OPTIONS(model_type='logistic_reg') AS
SELECT
  IF(my_awesome_database.ground_truth IS NULL, 0, 1) AS label,
  IFNULL(my_awesome_database.feature1, "") AS feature1,
  my_awesome_database.feature2 AS feature2,
  my_awesome_database.feature3 AS feature3,
  my_awesome_database.feature2 * my_awesome_database.feature3 AS feature4
FROM
  `my_awesome_database`
WHERE
  my_awesome_database.date BETWEEN 2010 AND 2015
```

One thing to note: `CREATE MODEL` will fail if the model with that name has already been created. If you're retraining your model, for example, you'll want to use `CREATE OR REPLACE MODEL` as the first line instead.

See this page for documentation: <https://cloud.google.com/bigquery/docs/reference/standard-sql/bigqueryml-syntax-create>

▼ Evaluating the Model

Once you've created your model, BigQuery has already trained it for you – you already have a h at your disposal ready to evaluate! We evaluate h by asking BigQuery to predict the Y values of **new data unseen by the model** and compare them to ground-truth values.

To evaluate a model you'd do something like this:

```
# Don't run me! My tables don't exist. I'm just here as an example.
%%bigquery --project $project_id

SELECT
  *
FROM
  ML.EVALUATE(MODEL `my_awesome_model`, (
SELECT
  IF(my_awesome_database.ground_truth IS NULL, 0, 1) AS label,
  IFNULL(my_awesome_database.feature1, "") AS feature1,
  my_awesome_database.feature2 AS feature2,
  my_awesome_database.feature3 AS feature3,
  my_awesome_database.feature2 * my_awesome_database.feature3 AS feature4
FROM
  `my_awesome_database`
WHERE
  my_awesome_database.date BETWEEN 2016 AND 2017))
```

Note that we are evaluating on data between 2016 and 2017, even though we trained on data between 2010 and 2015. **This is important!!** If we did not do this, we would be "cheating" since the model has already seen a training value corresponding to the one you are trying to evaluate. If the model was a simple lookup table, it would get 100% accuracy on everything it's already seen trivially. Also, we'll generally use a much larger amount of data for training than for evaluating or testing.

See this page for documentation: <https://cloud.google.com/bigquery/docs/reference/standard-sql/bigqueryml-syntax-evaluate>. Note the `ML.EVALUATE` function is one of three functions you can use to evaluate your model, depending on your task.

▼ Exercising the Model

If your model achieves good evaluation metrics (see [this](#) section of the BigQuery ML tutorial for data analysts for context on what 'good' evaluation metrics are), you can now utilize your model to predict values. Again, this should be done on data that the model has not seen.

Assuming you have a trained model, you can predict values like this:

```
# Don't run me! My tables don't exist. I'm just here as an example.
%%bigquery --project $project_id

SELECT
  my_awesome_database.key,
  predicted_label
FROM
  ML.PREDICT(MODEL `my_awesome_model`, (
SELECT
  IFNULL(my_awesome_database.feature1, "") AS feature1,
  my_awesome_database.feature2 AS feature2,
  my_awesome_database.feature3 AS feature3,
  my_awesome_database.feature2 * my_awesome_database.feature3 AS feature4
FROM
  `my_awesome_database`
WHERE
  my_awesome_database.date BETWEEN 2018.01 AND 2018.02))
```

See this page for documentation: <https://cloud.google.com/bigquery/docs/reference/standard-sql/bigqueryml-syntax-predict>.

For more details and an end-to-end example in BigQuery, read the following article:

<https://cloud.google.com/bigquery/docs/bigqueryml-analyst-start>.

▼ Section 3: Now it's Your Turn!

Let's now dive into an exercise using BigQuery and ML! This is a fairly simple warm-up problem to help you gain hands-on experience working with BQML. You'll get to dive into much more depth with your open-ended project! You'll be going through the three steps described in the previous section on your own.

For this problem, we're going to be working with the Austin bikeshare dataset available in BigQuery public dataset. Take a moment to familiarize yourself with the data we have at hand.

Notice we have various pieces of information about each trip - for example, the stations where the biker started and ended, with the corresponding latitude/longitude, the date of the ride, the subscriber type, and the duration of the trip in minutes.

Our goal in this exercise will be the following:

Given attributes about a ride, can we predict whether a bike ride will be a "quick" ride? Let's define a "quick" ride as a ride that takes less than 15 minutes.

Note this is a *binary logistic regression task*, or classification task, where, given attributes about a ride, we predict one of two labels: 1 = quick (< 15 minutes); 0 = not quick (>= 15 minutes).

Once we've trained our model, we can then use it to help predict on unlabeled data. In particular, we can use it to help fill in missing data - some bike rides have a different start/end station, but have a duration of 0 minutes (likely missing data).

Let's dive in!

▼ Step 1: Look at the data (1 point)

In any ML task, it's important to first explore the data. Investigating correlations between attributes as you did in Project 2 can help you determine which attributes may be useful as training features, and will be important for your final project. And, looking into the distribution of labels, you want the prediction to give you a better understanding of the distribution of your data (such as whether your dataset is *balanced*). For this exercise, we'll dig into the latter.

a) What percentage of rides are "quick"? Recall that we have: quick ride: < 15 minutes; not quick: >= 15 minutes. Filter out rides with a duration of 0 minutes.

```
%%bigquery --project $project_id

# YOUR QUERY HERE
SELECT COUNTIF(duration_minutes<15)*100/(SELECT COUNT(*) FROM `cs145-fa21-326819.project3.austin_bikeshare_trips`)
AS Percentage_of_Quick_Rides ,(SELECT COUNT(trip_id) FROM `cs145-fa21-326819.project3.austin_bikeshare_trips`
WHERE duration_minutes=0) AS No_of_Zero_Duration_Rides
FROM `cs145-fa21-326819.project3.austin_bikeshare_trips`;
```

	Percentage_of_Quick_Rides	No_of_Zero_Duration_Rides
0	49.487625	11011

b) What percentage of rides have a different start/end station, but have a value of 0 for their duration? How many rides is this? Write a query that returns the count in one column and the percentage in another. The denominator for the percentage should be all rides, regardless of duration.

Hint: [COUNTIF](#) may be helpful.

```
%%bigquery --project $project_id

# YOUR QUERY HERE
SELECT COUNTIF(start_station_name<>end_station_name AND duration_minutes=0) AS Number_of_Rides,
COUNTIF(start_station_id<>end_station_id AND duration_minutes=0)*100/
(SELECT COUNT(*) FROM `cs145-fa21-326819.project3.austin_bikeshare_trips`) AS Percentage
FROM `cs145-fa21-326819.project3.austin_bikeshare_trips`;
```

	Number_of_Rides	Percentage
0	2201	0.321149

▼ Step 2: Create a dataset to store the model

When you create and train a model, BigQuery will store the model in a dataset. Before training, you'll first need to create a new empty dataset. Note that you only need to do this step once. If you later update your model, it can replace the existing one.

You can also do this step in the UI (see 'create your dataset': <https://cloud.google.com/bigquery/docs/bigqueryml-analyst-start>).

Let's call our dataset `bqml_bikeshare`. After either running the cell below, or creating the dataset with the BigQuery UI, you should see the dataset name appear in the left column of the UI.

```
# Run this cell to create a dataset to store your model, or create in the UI

model_dataset_name = 'bqml_bikeshare'

dataset = bigquery.Dataset(client.dataset(model_dataset_name))
dataset.location = 'US'
client.create_dataset(dataset)
```

▼ Step 3: Extract training data from BigQuery (2 points)

Write a SQL query that extracts training data from the dataset. These are features that you want to feed into your model. For this part, you do not need to do feature engineering - you can simply pull raw features from the tables that you think may be helpful.

Your query should return a column called `label` with the target label value (our "Y" value), and additional columns for some features you want to use (our "X" values). Note:

- recall: `label` value is 1 for quick rides (< 15 minutes), and 0 otherwise (>= 15 minutes)
- `duration_minutes` cannot be a training feature - we're trying to predict (a boolean version of) this
- filter out any rides with a duration of 0 minutes

Display the first 10 rows of the table returned by your query.

```
%%bigquery --project $project_id
SELECT duration_minutes,
       IF(duration_minutes < 15,1,0) AS label,
       start_station_name AS feature1,
```

```
end_station_name AS feature2
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`
WHERE duration_minutes > 0 LIMIT 10
```

	duration_minutes	label	feature1	feature2
0	39	0	Zilker Park West	Nueces @ 3rd
1	31	0	Toomey Rd @ South Lamar	Toomey Rd @ South Lamar
2	31	0	Toomey Rd @ South Lamar	Toomey Rd @ South Lamar
3	30	0	Toomey Rd @ South Lamar	Toomey Rd @ South Lamar
4	19	0	State Capitol @ 14th & Colorado	State Capitol @ 14th & Colorado
5	17	0	State Capitol @ 14th & Colorado	State Capitol @ 14th & Colorado
6	6	1	Waller & 6th St.	Waller & 6th St.
7	12	1	Waller & 6th St.	Waller & 6th St.
8	47	0	Pease Park	Pease Park
9	1	1	5th & San Marcos	5th & San Marcos

▼ Step 4: Train a simple model (1 point)

First, an important note: it's important to have separate datasets to train, evaluate, and finally test your model. We'll want 3 different subsets of data:

1. **Training set:** used to train a model.

- we'll train on rides before 2017 (start_time < '2017-01-01'), with duration time > 0

2. **Evaluation set:** used to evaluate model after training. This should not be data used during training. It can be used multiple times to evaluate and compare the performance of different models.

- we'll evaluate on the next 5 months (start_time between '2017-01-01' and '2017-06-01'), with duration time > 0

3. **Test set:** *should only be used once at the end of your entire training process* to say how your model does on real data. This should not be the same as either training or eval data. Using the test set to tune your model is bad, since it means you are starting to overfit your model (i.e., making your model artificially good on a certain dataset at the possible expense of it doing poorly on new data) to that test set as well.

- we'll test on the 5 months after that (start_time between '2017-06-01' and '2017-11-01'), with duration time > 0

Note that for all these datasets, we'll filter out rides with duration time = 0. For the purposes of this problem, we'll consider this to be incomplete data.

Now, let's go ahead and train a simple model. **Create a model, using the query you wrote above to tell the model what features and ground-truth labels to use.** Remember that we're training only on rides before 2017 (start_time < '2017-01-01'), and with a duration time > 0.

Note: it may take a few minutes to run the query. Also, you may get the error `Table has no schema: call 'client.get_table()'`. This is because notebook cells try to print out the table returned from a SQL query, but the query to create/train a model doesn't return any table at all, so the notebook complains. The model is still trained successfully though. You may ignore this, and can click the (X) in the top left of the output to clear the error message.

```
%%bigquery --project $project_id

# YOUR QUERY HERE

CREATE OR REPLACE MODEL
  `bqml_bikeshare.bikeshare_model` OPTIONS(model_type='logistic_reg') AS
SELECT
  duration_minutes,
  IF
    (duration_minutes < 15,1,0) AS label,
  start_station_name AS feature1,
  end_station_name AS feature2
FROM
```

```
`bigquery-public-data.austin_bikeshare.bikeshare_trips`  
WHERE  
  duration_minutes > 0  
  AND start_time < '2017-01-01'
```

You can get training statistics on your model by running the following cell:

```
%%bigquery --project $project_id  
  
# Run cell to view training stats  
  
SELECT  
  *  
FROM  
  ML.TRAINING_INFO(MODEL `bqml_bikeshare.bikeshare_model`)
```

	training_run	iteration	loss	eval_loss	learning_rate	duration_ms
0	0	12	0.462577	0.471745	3.2	8721
1	0	11	0.466526	0.475104	12.8	8979
2	0	10	0.462452	0.462697	6.4	8772

▼ Step 5: Evaluate (1 point)

Evaluate your model on unseen evaluation data.

Recall for our evaluation set, we're using the 5 months following what we trained on (use: start_time between '2017-01-01' and '2017-06-01'), with duration time > 0.

```

7          0          5  0.551502  0.561152          6.4          8585

%%bigquery --project $project_id

# YOUR QUERY HERE
SELECT
  *
FROM
  ML.EVALUATE(MODEL `bqml_bikeshare.bikeshare_model`, (
SELECT
  duration_minutes,
IF
  (duration_minutes < 15,1,0) AS label,
  start_station_name AS feature1,
  end_station_name AS feature2
FROM
  `bigquery-public-data.austin_bikeshare.bikeshare_trips`
WHERE
  duration_minutes > 0
  AND start_time between '2017-01-01' and '2017-06-01'
))

```

	precision	recall	accuracy	f1_score	log_loss	roc_auc
0	0.800023	0.846969	0.831873	0.822827	0.44346	0.908212

▼ Step 6: Improving our model (3 points)

In general, we can't just throw raw data into the model and expect it to work: in practice, you'll iterate on improving your features and re-training/re-evaluating your model. Let's try the following: add engineered features -> re-train model -> re-evaluate model.

a) Let's add an engineered feature! You suspect that there is a relationship between the distance between the start and end stations and whether it will be a "quick" ride. Let's add the distance between the start station and the end station as a feature. (1 point)

Extend your query from step 3 to also have a feature for the euclidean distance between the start and end station.

You may find the following useful:

- [Example](#) from Dr. Lakshmanan's invited talk
- `ST_GeogPoint(longitude, latitude)` - creates geography point from longitude, latitude values
- `ST_DISTANCE(start_pt, end_pt)` - computes distance between 2 geographic points (more [here](#))

You are welcome, but not required, to experiment with other engineered features as well.

Display the first 10 rows of the table returned by your query.

```
%%bigquery --project $project_id

# YOUR QUERY HERE
# YOUR QUERY HERE

# YOUR QUERY HERE


WITH r1 AS (
SELECT
  trip_id,
  duration_minutes,
  start_station_id,
  footprint_length,
  footprint_width
from
  `bigquery-public-data.austin_bikeshare.bikeshare_trips` ,
```



```
`bigquery-public-data.austin_bikeshare.bikeshare_stations`
WHERE
  start_station_id=station_id
Order by footprint_length DESC, footprint_width DESC
LIMIT 10
),
r2 AS (
  SELECT
    trip_id,
    duration_minutes,
    end_station_name,
    footprint_length,
    footprint_width
  from
    `bigquery-public-data.austin_bikeshare.bikeshare_trips` ,
    `bigquery-public-data.austin_bikeshare.bikeshare_stations`
  WHERE
    end_station_name=name
Order by footprint_length DESC, footprint_width DESC
LIMIT 100

)
SELECT IF (t1.duration_minutes < 15,1,0) AS label,

  ST_DISTANCE(ST_GeogPoint(r1.footprint_width,r1.footprint_length),
    ST_GeogPoint(r2.footprint_length,r2.footprint_width)) dist,
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips` t1 ,r1,r2
where r1.trip_id = r2.trip_id
ORDER BY
  dist DESC
LIMIT
  100
```

	label	dist	
0	0	7.114306e+06	
1	0	7.114306e+06	
2	0	7.114306e+06	
3	0	7.114306e+06	
4	1	7.114306e+06	
...	
95	0	7.114306e+06	
96	1	7.114306e+06	
97	1	7.114306e+06	
98	1	7.114306e+06	

b) Let's train our model again (using the same training set as before) with the added features. You can replace the existing one, or create a new one with a different name. (1 point)

Note: it may take a few minutes to run the query. Also, you may again get the error `Table has no schema: call 'client.get_table()'`. The model is still trained successfully though. You may ignore this, and can click the (X) in the top left of the output to clear the error message.

Let's get our training stats again:

```
%%bigquery --project $project_id

# YOUR QUERY HERE
CREATE OR REPLACE MODEL `bqml_bikeshare.bikeshare_model_v2` -- we'll call our model 'bikeshare_model_v2'
-- TODO: complete query
OPTIONS(model_type='logistic_reg') AS
SELECT
```

```
label AS label,  
dist AS feature1  
FROM  
`cs145-255023.bqml_bikeshare.bikeshare_dataset2`  
WHERE  
start_time < '2017-01-01' AND duration_minutes>0 AND dist>0
```

You'll should hopefully find that the loss is a bit lower (better) than before, on both on the training data and on BigQuery's automatic evaluation set (it withholds some data you passed in as training data for reporting eval stats).

```
%%bigquery --project $project_id  
  
# Run cell to view training stats  
  
SELECT  
*  
FROM  
ML.TRAINING_INFO(MODEL `bqml_bikeshare.bikeshare_model_v2`)
```

c) Now let's evaluate our re-trained model on our evaluation set. You can use a similar evaluation query from step 5, but with your updated features (note: you may need to change the model name in the query if your new model has a different name). (1 point)

```
%%bigquery --project $project_id  
  
# YOUR QUERY HERE  
SELECT  
*  
FROM  
ML.EVALUATE(MODEL `bqml_bikeshare.bikeshare_model_v2`, (  
SELECT  
label AS label,  
dist AS feature1  
FROM  
`cs145-255023.bqml_bikeshare.bikeshare_dataset2`
```

```
WHERE  
start_time BETWEEN '2017-01-01' AND '2017-06-01' AND duration_minutes>0 AND dist>0))
```

▼ Step 7: Evaluate final model on test set (1 point)

Once you're done training your model (in practice, you'll likely iterate on updating your model, retraining on the training set, and re-evaluating on the evaluation set several times), you'll evaluate your final model on a test set. The test set consists of examples that have not been used at all before, neither during training nor during evaluation.

Again, the test set should **only be used once at the end of your entire training process**, to see how your model does on real data. You should only run the cell below once you are finished modifying your features.

Recall that for our test set, we're using the 5 months after our evaluation set (rides with start_time between '2017-06-01' and '2017-11-01', with duration time > 0).

Evaluate your model once on this test set. The query is almost identical to the previous one, except now you use the test set.

```
%%bigquery --project $project_id  
  
# YOUR QUERY HERE  
SELECT  
*  
FROM  
  ML.EVALUATE(MODEL `bqml_bikeshare.bikeshare_model_v2`, (  
SELECT  
  label AS label,  
  dist AS feature1  
FROM  
  `cs145-255023.bqml_bikeshare.bikeshare_dataset2`  
WHERE  
  start_time BETWEEN '2017-06-01' AND '2017-11-01' AND duration_minutes>0 AND dist>0))
```

▼ Step 8: Use the trained model to predict (1 point)

Once you've trained your model, you can use it to make predictions! Let's try to use it to fill in some of the missing data.

Now, let's go ahead and predict on rides that had a duration time of 0 minutes, but had different start/end stations. Does our model think these were quick rides?

Notice that these samples were never used during training/evaluation/testing, since we filtered out rides with a duration of 0.

Display the features used for prediction and the predicted label for 10 examples. The predicted label will be called `predicted_label1`.

```
%%bigquery --project $project_id

# YOUR QUERY HERE
SELECT
*
FROM
  ML.PREDICT(MODEL `bqml_bikeshare.bikeshare_model_v2`, (
SELECT
  dist AS feature1
FROM
  `cs145-255023.bqml_bikeshare.bikeshare_dataset2`
WHERE
  duration_minutes=0 AND dist>0.0))
```

✓ 1s completed at 9:42 PM

