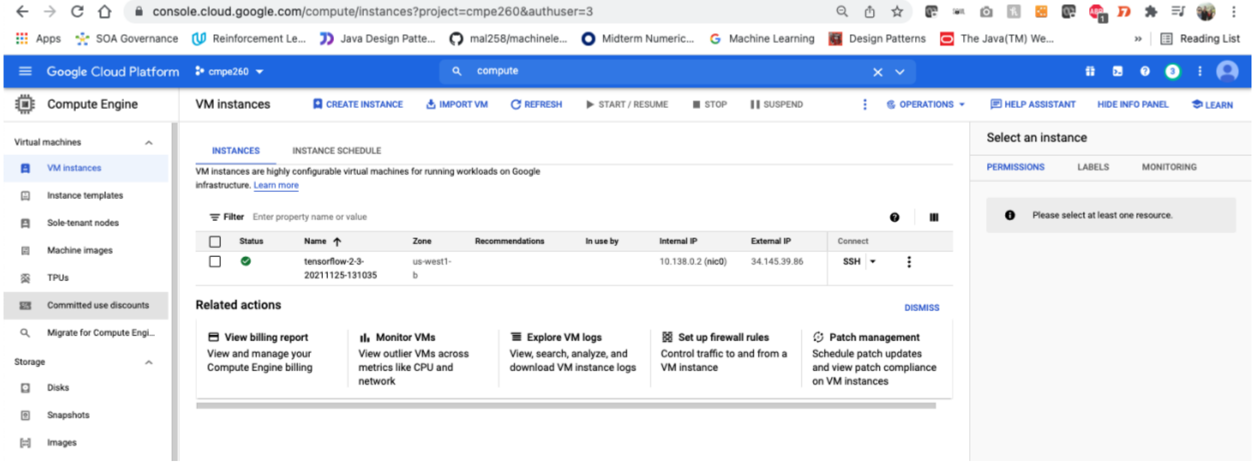
Vertex AI Workbench: Train a TensorFlow model with data from BigQuery

1) Enable the Compute Engine API



console.cloud.google.com/compute/instances?project=cmpe260&authuser=3

VM instances are highly configurable virtual machines for running workloads on Google infrastructure. Learn more

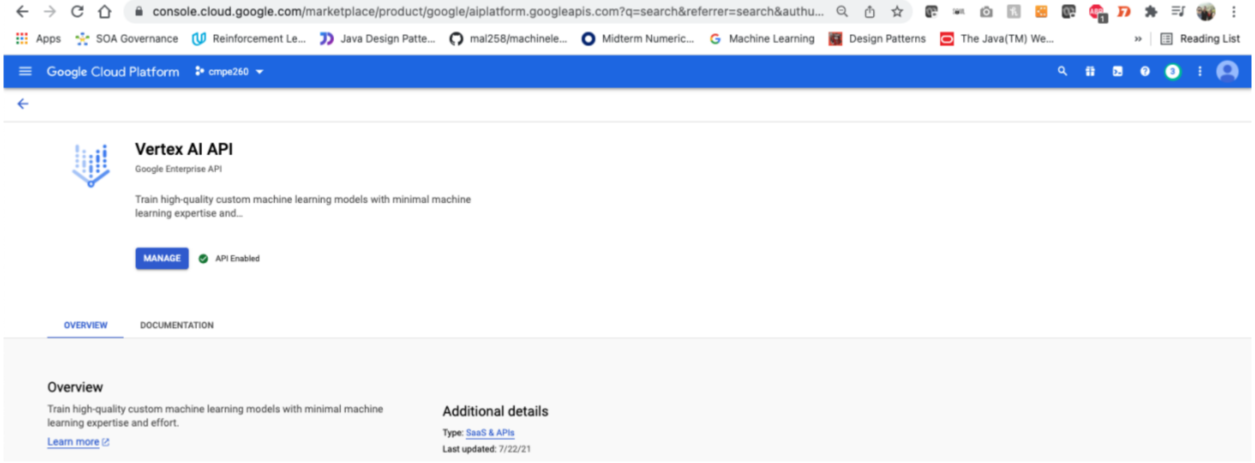
□	Status	Name	Zone	Recommendations	In use by	Internal IP	External IP	Connect
□	✓	tensorflow-2-3-20211125-131035	us-west1-b			10.138.0.2 (nic0)	34.145.39.86	SSH

Related actions — DISMISS

- View billing report: View and manage your Compute Engine billing
- Monitor VMs: View outlier VMs across metrics like CPU and network
- Explore VM logs: View, search, analyze, and download VM instance logs
- Set up firewall rules: Control traffic to and from a VM instance
- Patch management: Schedule patch updates and view patch compliance on VM instances

Please select at least one resource.

2) Enable Vertex AI API



console.cloud.google.com/marketplace/product/google/aiplatform.googleapis.com?q=search&referrer=search&authu...

Vertex AI API

Google Enterprise API

Train high-quality custom machine learning models with minimal machine learning expertise and...

✓ API Enabled

Overview

Train high-quality custom machine learning models with minimal machine learning expertise and effort.

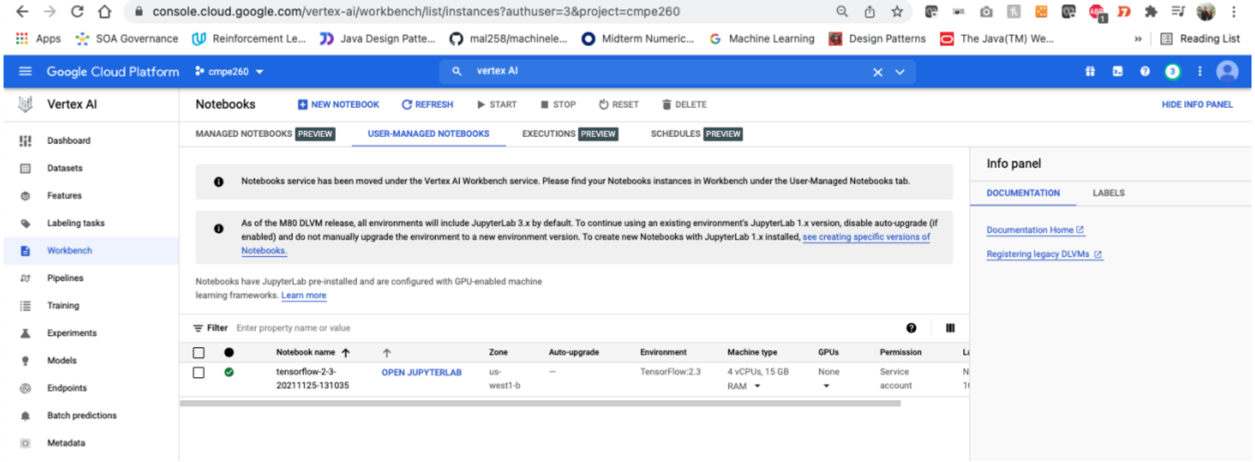
Learn more

Additional details

Type: SaaS & APIs

Last updated: 7/22/21

3) Create Vertex AI Workbench instance



console.cloud.google.com/vertex-ai/workbench/list/instances?authuser=3&project=cmpe260

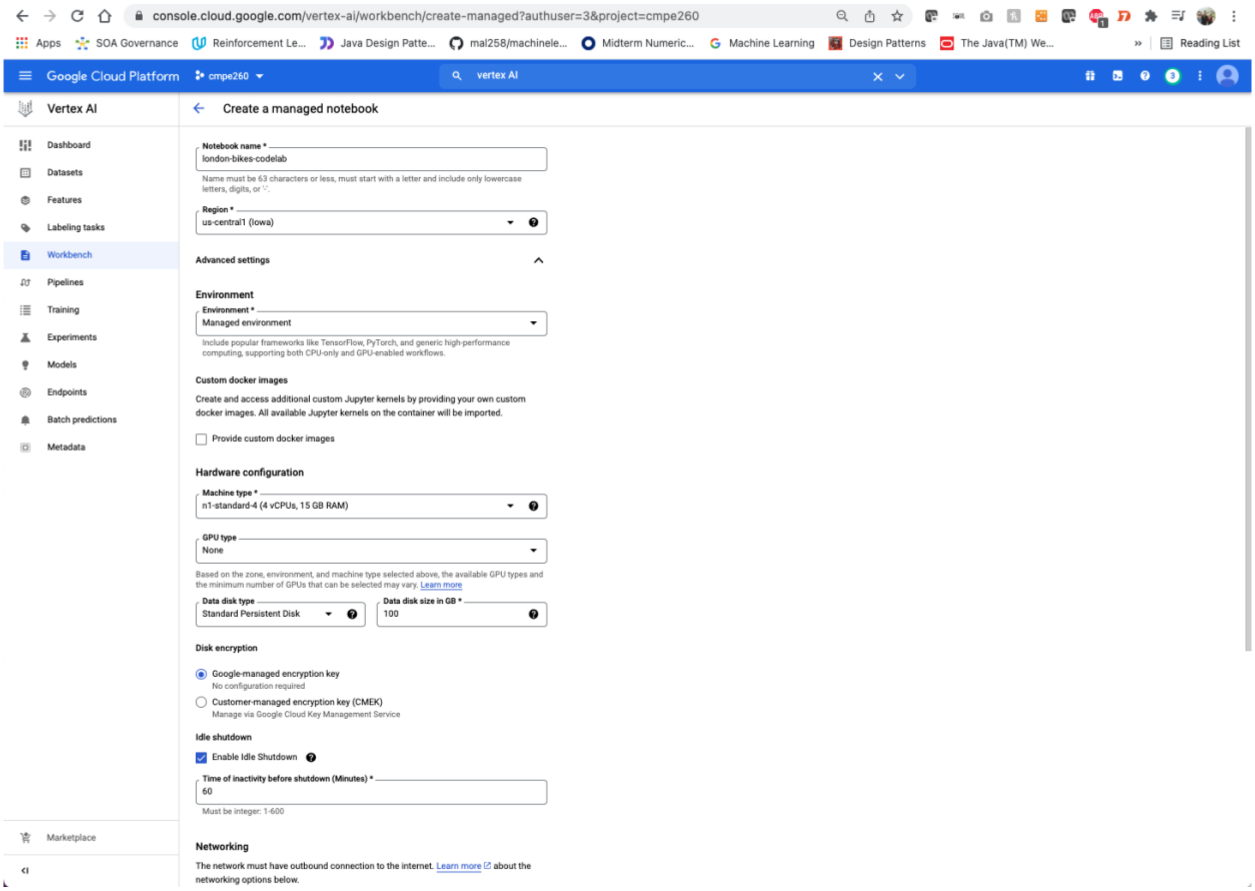
Notebooks service has been moved under the Vertex AI Workbench service. Please find your Notebooks instances in Workbench under the User-Managed Notebooks tab.

As of the M80 DLVM release, all environments will include JupyterLab 3.x by default. To continue using an existing environment's JupyterLab 1.x version, disable auto-upgrade (if enabled) and do not manually upgrade the environment to a new environment version. To create new Notebooks with JupyterLab 1.x installed, see creating specific versions of Notebooks.

Notebooks have JupyterLab pre-installed and are configured with GPU-enabled machine learning frameworks. Learn more

□	●	Notebook name		Zone	Auto-upgrade	Environment	Machine type	GPUs	Permission	La
□	✓	tensorflow-2-3-20211125-131035	OPEN JUPYTERLAB	us-west1-b	–	TensorFlow:2.3	4 vCPUs, 15 GB RAM	None	Service account	N 1

4) create a new notebook



Create a managed notebook

Notebook name *: london-bikes-codelab

Name must be 63 characters or less, must start with a letter and include only lowercase letters, digits, or '-'.

Region *: us-central1 (Iowa)

Advanced settings

Environment

Environment *: Managed environment

Include popular frameworks like TensorFlow, PyTorch, and generic high-performance computing, supporting both CPU-only and GPU-enabled workflows.

Custom docker images

Create and access additional custom Jupyter kernels by providing your own custom docker images. All available Jupyter kernels on the container will be imported.

☐ Provide custom docker images

Hardware configuration

Machine type *: n1-standard-4 (4 vCPUs, 15 GB RAM)

GPU type: None

Based on the zone, environment, and machine type selected above, the available GPU types and the minimum number of GPUs that can be selected may vary. Learn more

Data disk type: Standard Persistent Disk

Data disk size in GB *: 100

Disk encryption

● Google-managed encryption key
No configuration required

○ Customer-managed encryption key (CMEK)
Manage via Google Cloud Key Management Service

Idle shutdown

☑ Enable Idle Shutdown

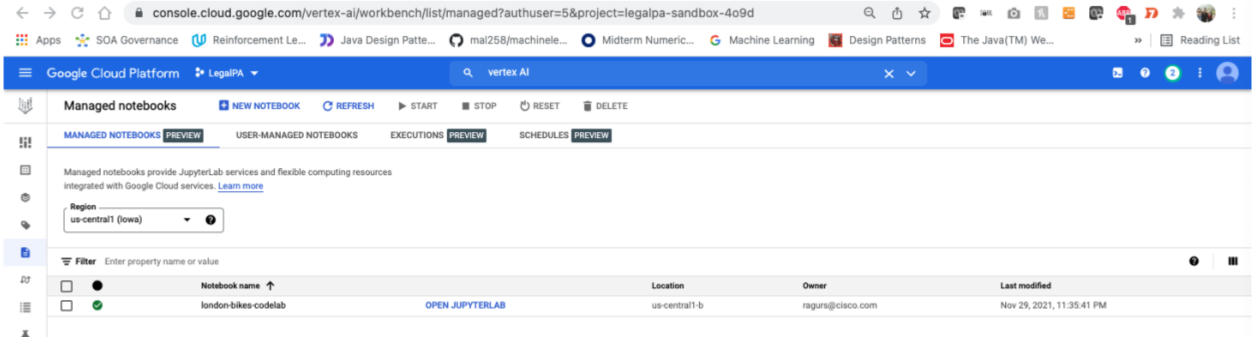
Time of inactivity before shutdown (Minutes) *: 60

Must be integer: 1-600

Networking

The network must have outbound connection to the internet. Learn more about the networking options below.

5)

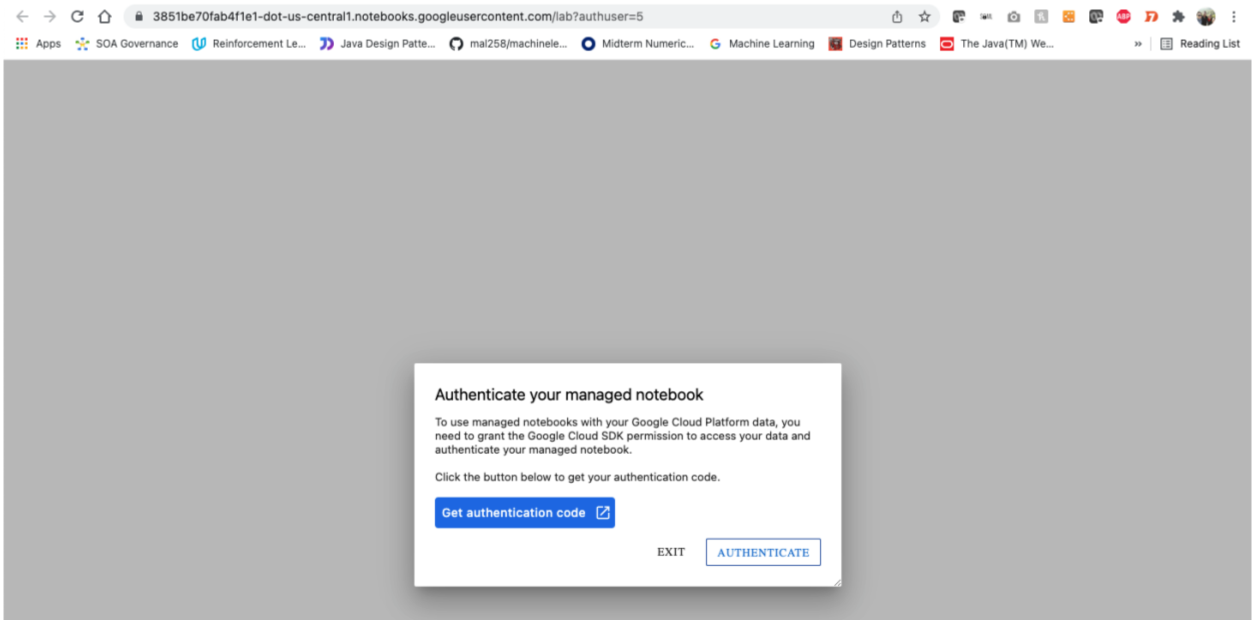


Managed notebooks

Managed notebooks provide JupyterLab services and flexible computing resources integrated with Google Cloud services. Learn more

Region: us-central1 (Iowa)

☐	●	Notebook name ↑		Location	Owner	Last modified
☐	✓	london-bikes-codelab	OPEN JUPYTERLAB	us-central1-b	ragurs@cisco.com	Nov 29, 2021, 11:35:41 PM



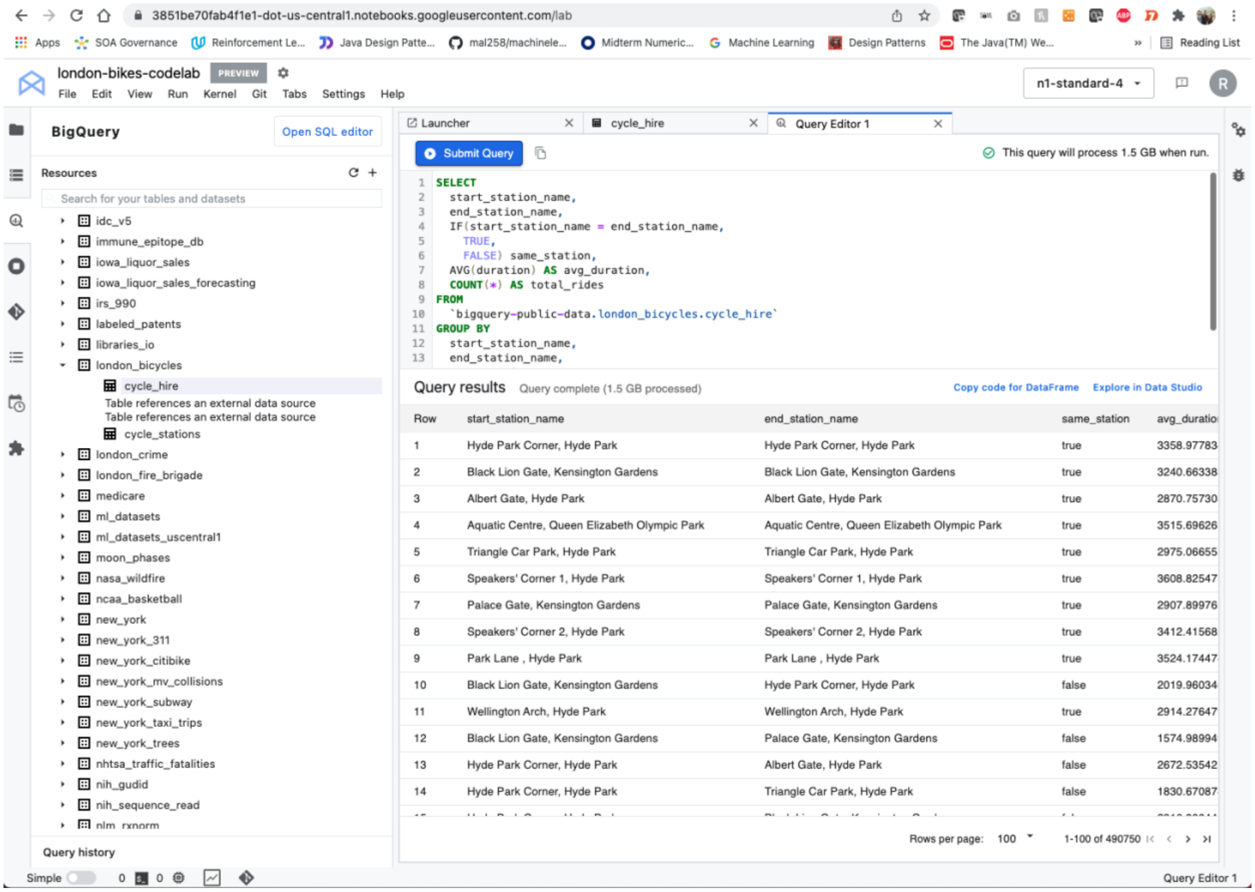
Authenticate your managed notebook

To use managed notebooks with your Google Cloud Platform data, you need to grant the Google Cloud SDK permission to access your data and authenticate your managed notebook.

Click the button below to get your authentication code.

6)

Dataset exploration in BigQuery



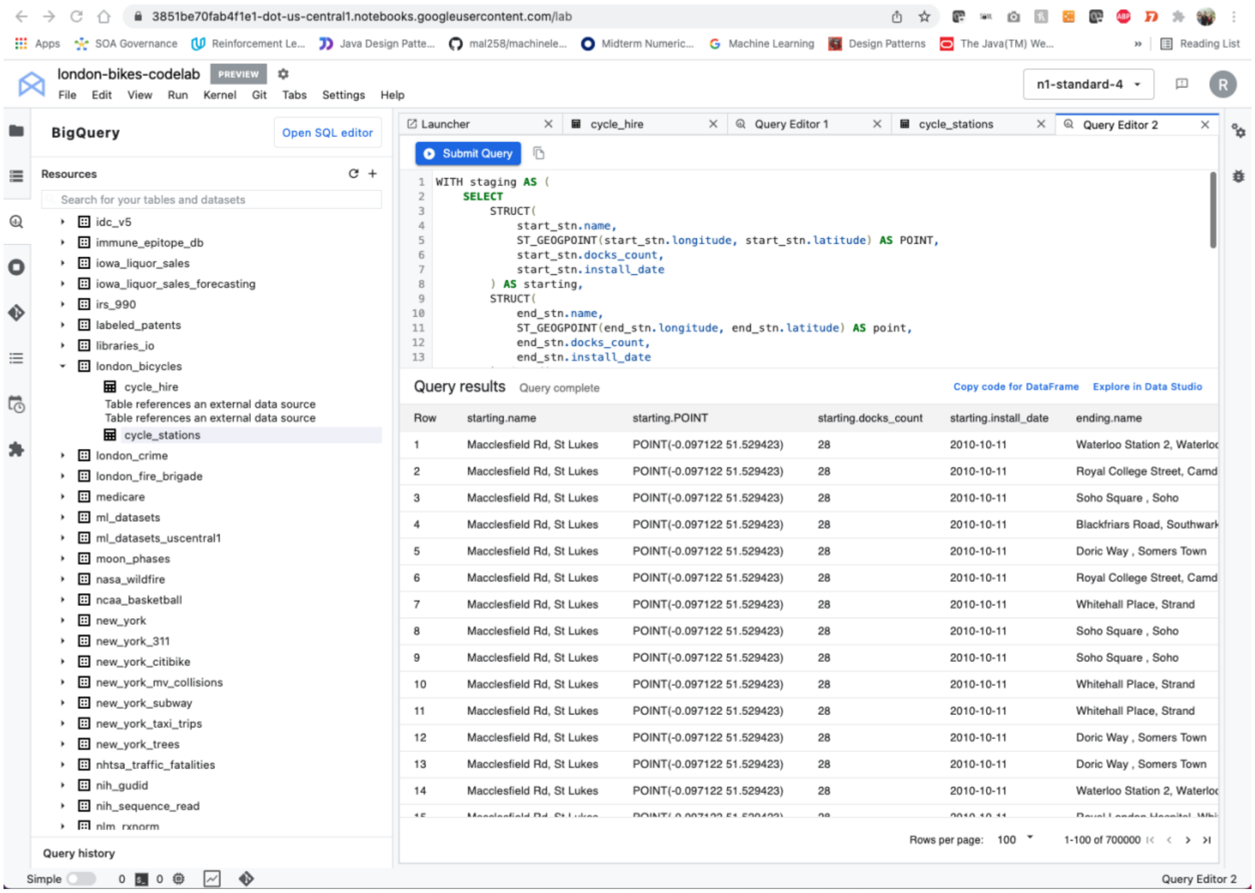
This query will process 1.5 GB when run.

```sql
SELECT
  start_station_name,
  end_station_name,
  IF(start_station_name = end_station_name,
    TRUE,
    FALSE) same_station,
  AVG(duration) AS avg_duration,
  COUNT(*) AS total_rides
FROM
  `bigquery-public-data.london_bicycles.cycle_hire`
GROUP BY
  start_station_name,
  end_station_name,
```

Query results Query complete (1.5 GB processed)

Row	start_station_name	end_station_name	same_station	avg_duratio
1	Hyde Park Corner, Hyde Park	Hyde Park Corner, Hyde Park	true	3358.97783
2	Black Lion Gate, Kensington Gardens	Black Lion Gate, Kensington Gardens	true	3240.66338
3	Albert Gate, Hyde Park	Albert Gate, Hyde Park	true	2870.75730
4	Aquatic Centre, Queen Elizabeth Olympic Park	Aquatic Centre, Queen Elizabeth Olympic Park	true	3515.69626
5	Triangle Car Park, Hyde Park	Triangle Car Park, Hyde Park	true	2975.06655
6	Speakers' Corner 1, Hyde Park	Speakers' Corner 1, Hyde Park	true	3608.82547
7	Palace Gate, Kensington Gardens	Palace Gate, Kensington Gardens	true	2907.89976
8	Speakers' Corner 2, Hyde Park	Speakers' Corner 2, Hyde Park	true	3412.41568
9	Park Lane , Hyde Park	Park Lane , Hyde Park	true	3524.17447
10	Black Lion Gate, Kensington Gardens	Hyde Park Corner, Hyde Park	false	2019.96034
11	Wellington Arch, Hyde Park	Wellington Arch, Hyde Park	true	2914.27647
12	Black Lion Gate, Kensington Gardens	Palace Gate, Kensington Gardens	false	1574.98994
13	Hyde Park Corner, Hyde Park	Albert Gate, Hyde Park	false	2672.53542
14	Hyde Park Corner, Hyde Park	Triangle Car Park, Hyde Park	false	1830.67087

Rows per page: 100 1-100 of 490750

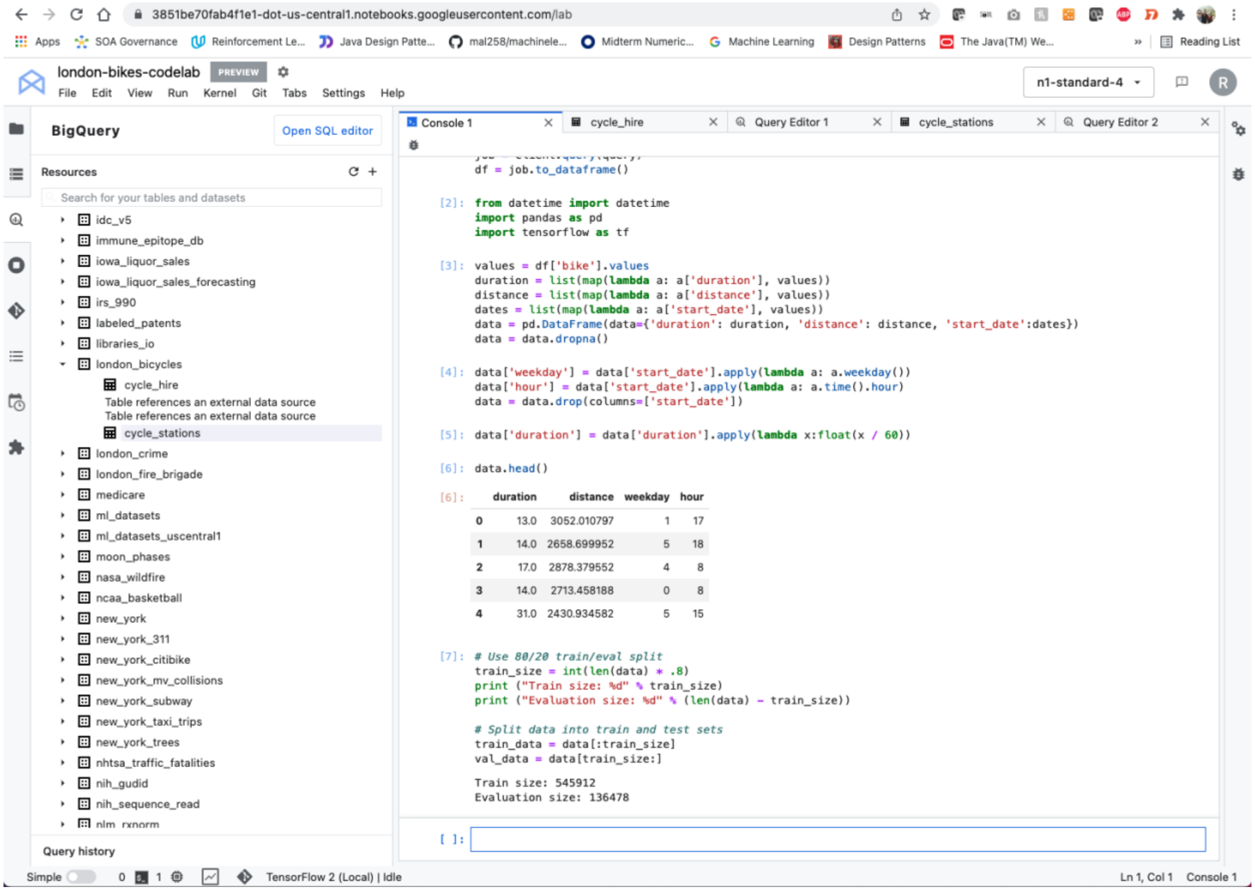


```
WITH staging AS (
    SELECT
        STRUCT(
            start_stn.name,
            ST_GEOGPOINT(start_stn.longitude, start_stn.latitude) AS POINT,
            start_stn.docks_count,
            start_stn.install_date
        ) AS starting,
        STRUCT(
            end_stn.name,
            ST_GEOGPOINT(end_stn.longitude, end_stn.latitude) AS point,
            end_stn.docks_count,
            end_stn.install_date
```

Query results Query complete

Row	starting.name	starting.POINT	starting.docks_count	starting.install_date	ending.name
1	Macclesfield Rd, St Lukes	POINT(-0.097122 51.529423)	28	2010-10-11	Waterloo Station 2, Waterloo
2	Macclesfield Rd, St Lukes	POINT(-0.097122 51.529423)	28	2010-10-11	Royal College Street, Camd
3	Macclesfield Rd, St Lukes	POINT(-0.097122 51.529423)	28	2010-10-11	Soho Square , Soho
4	Macclesfield Rd, St Lukes	POINT(-0.097122 51.529423)	28	2010-10-11	Blackfriars Road, Southwark
5	Macclesfield Rd, St Lukes	POINT(-0.097122 51.529423)	28	2010-10-11	Doric Way , Somers Town
6	Macclesfield Rd, St Lukes	POINT(-0.097122 51.529423)	28	2010-10-11	Royal College Street, Camd
7	Macclesfield Rd, St Lukes	POINT(-0.097122 51.529423)	28	2010-10-11	Whitehall Place, Strand
8	Macclesfield Rd, St Lukes	POINT(-0.097122 51.529423)	28	2010-10-11	Soho Square , Soho
9	Macclesfield Rd, St Lukes	POINT(-0.097122 51.529423)	28	2010-10-11	Soho Square , Soho
10	Macclesfield Rd, St Lukes	POINT(-0.097122 51.529423)	28	2010-10-11	Whitehall Place, Strand
11	Macclesfield Rd, St Lukes	POINT(-0.097122 51.529423)	28	2010-10-11	Whitehall Place, Strand
12	Macclesfield Rd, St Lukes	POINT(-0.097122 51.529423)	28	2010-10-11	Doric Way , Somers Town
13	Macclesfield Rd, St Lukes	POINT(-0.097122 51.529423)	28	2010-10-11	Doric Way , Somers Town
14	Macclesfield Rd, St Lukes	POINT(-0.097122 51.529423)	28	2010-10-11	Waterloo Station 2, Waterloo
15	Macclesfield Rd, St Lukes	POINT(-0.097122 51.529423)	28	2010-10-11	Royal London Hospital, Whi

7)Creating the dataframe



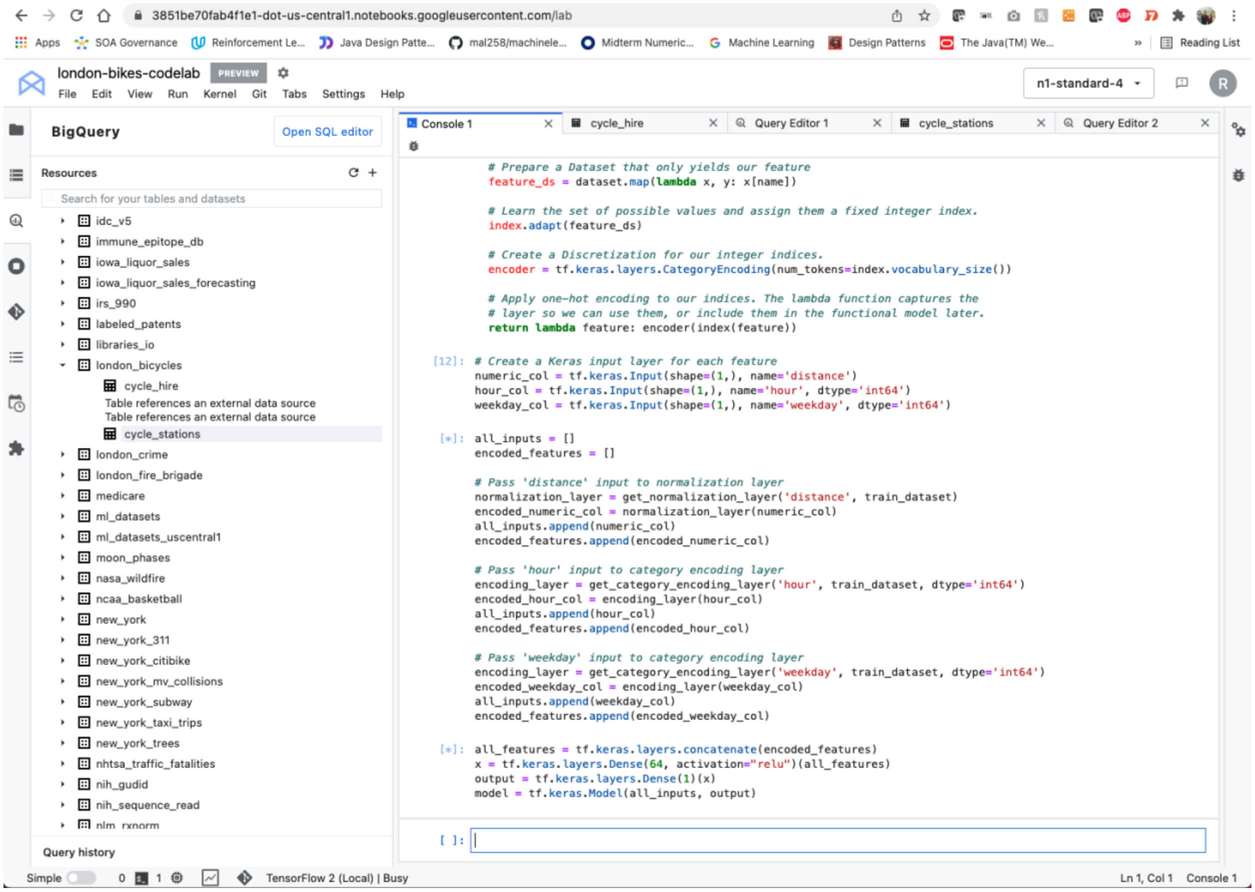
```
       df = job.to_dataframe()

[2]:   from datetime import datetime
       import pandas as pd
       import tensorflow as tf

[3]:   values = df['bike'].values
       duration = list(map(lambda a: a['duration'], values))
       distance = list(map(lambda a: a['distance'], values))
       dates = list(map(lambda a: a['start_date'], values))
       data = pd.DataFrame(data={'duration': duration, 'distance': distance, 'start_date':dates})
       data = data.dropna()

[4]:   data['weekday'] = data['start_date'].apply(lambda a: a.weekday())
       data['hour'] = data['start_date'].apply(lambda a: a.time().hour)
       data = data.drop(columns=['start_date'])

[5]:   data['duration'] = data['duration'].apply(lambda x:float(x / 60))

[6]:   data.head()
```

[6]:

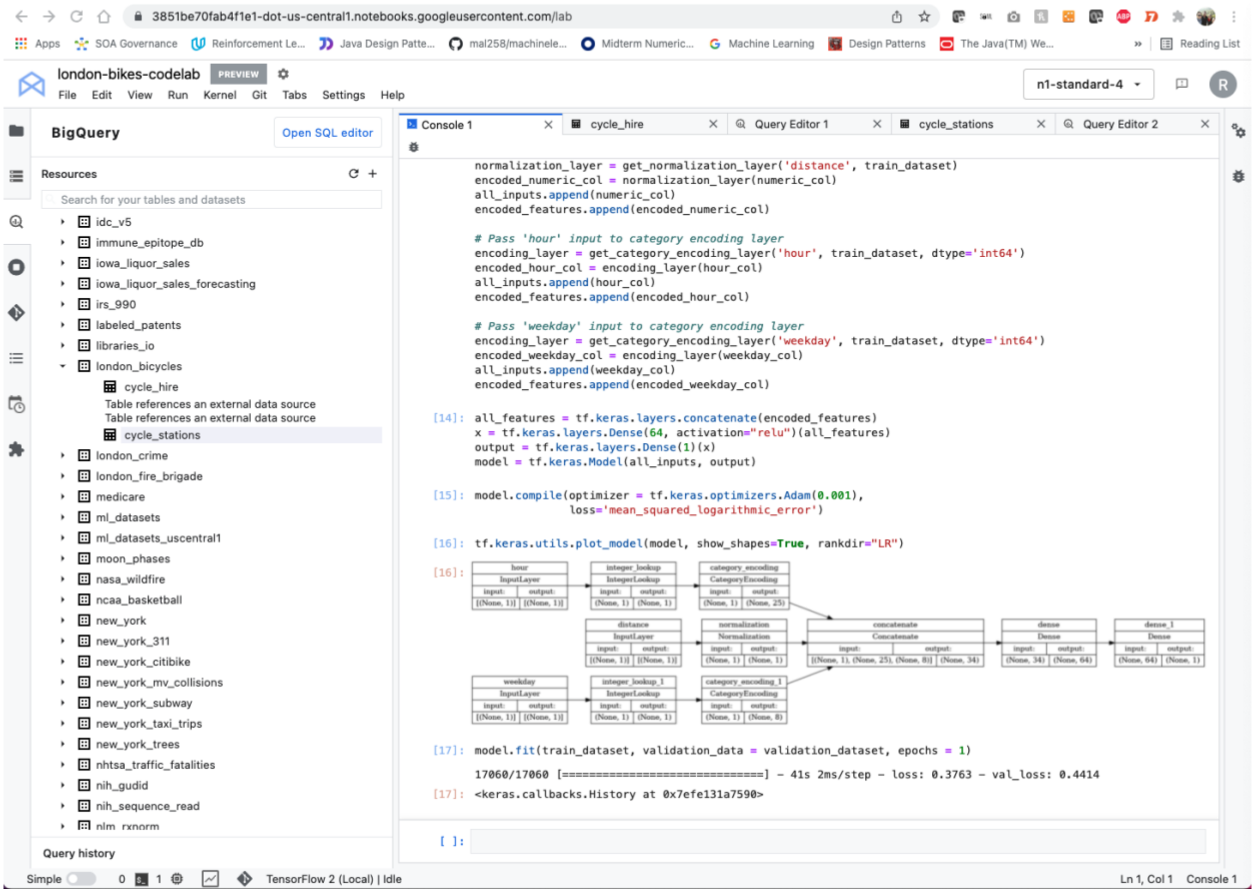
	duration	distance	weekday	hour
0	13.0	3052.010797	1	17
1	14.0	2658.699952	5	18
2	17.0	2878.379552	4	8
3	14.0	2713.458188	0	8
4	31.0	2430.934582	5	15

```
[7]:   # Use 80/20 train/eval split
       train_size = int(len(data) * .8)
       print ("Train size: %d" % train_size)
       print ("Evaluation size: %d" % (len(data) - train_size))

       # Split data into train and test sets
       train_data = data[:train_size]
       val_data = data[train_size:]

       Train size: 545912
       Evaluation size: 136478

[ ]:
```

Training model on tensorflow kernel



```
        # Prepare a Dataset that only yields our feature
        feature_ds = dataset.map(lambda x, y: x[name])

        # Learn the set of possible values and assign them a fixed integer index.
        index.adapt(feature_ds)

        # Create a Discretization for our integer indices.
        encoder = tf.keras.layers.CategoryEncoding(num_tokens=index.vocabulary_size())

        # Apply one-hot encoding to our indices. The lambda function captures the
        # layer so we can use them, or include them in the functional model later.
        return lambda feature: encoder(index(feature))

[12]: # Create a Keras input layer for each feature
      numeric_col = tf.keras.Input(shape=(1,), name='distance')
      hour_col = tf.keras.Input(shape=(1,), name='hour', dtype='int64')
      weekday_col = tf.keras.Input(shape=(1,), name='weekday', dtype='int64')

 [*]: all_inputs = []
      encoded_features = []

      # Pass 'distance' input to normalization layer
      normalization_layer = get_normalization_layer('distance', train_dataset)
      encoded_numeric_col = normalization_layer(numeric_col)
      all_inputs.append(numeric_col)
      encoded_features.append(encoded_numeric_col)

      # Pass 'hour' input to category encoding layer
      encoding_layer = get_category_encoding_layer('hour', train_dataset, dtype='int64')
      encoded_hour_col = encoding_layer(hour_col)
      all_inputs.append(hour_col)
      encoded_features.append(encoded_hour_col)

      # Pass 'weekday' input to category encoding layer
      encoding_layer = get_category_encoding_layer('weekday', train_dataset, dtype='int64')
      encoded_weekday_col = encoding_layer(weekday_col)
      all_inputs.append(weekday_col)
      encoded_features.append(encoded_weekday_col)

 [*]: all_features = tf.keras.layers.concatenate(encoded_features)
      x = tf.keras.layers.Dense(64, activation="relu")(all_features)
      output = tf.keras.layers.Dense(1)(x)
      model = tf.keras.Model(all_inputs, output)

 [ ]:
```



```
      normalization_layer = get_normalization_layer('distance', train_dataset)
      encoded_numeric_col = normalization_layer(numeric_col)
      all_inputs.append(numeric_col)
      encoded_features.append(encoded_numeric_col)

      # Pass 'hour' input to category encoding layer
      encoding_layer = get_category_encoding_layer('hour', train_dataset, dtype='int64')
      encoded_hour_col = encoding_layer(hour_col)
      all_inputs.append(hour_col)
      encoded_features.append(encoded_hour_col)

      # Pass 'weekday' input to category encoding layer
      encoding_layer = get_category_encoding_layer('weekday', train_dataset, dtype='int64')
      encoded_weekday_col = encoding_layer(weekday_col)
      all_inputs.append(weekday_col)
      encoded_features.append(encoded_weekday_col)

[14]: all_features = tf.keras.layers.concatenate(encoded_features)
      x = tf.keras.layers.Dense(64, activation="relu")(all_features)
      output = tf.keras.layers.Dense(1)(x)
      model = tf.keras.Model(all_inputs, output)

[15]: model.compile(optimizer = tf.keras.optimizers.Adam(0.001),
                    loss='mean_squared_logarithmic_error')

[16]: tf.keras.utils.plot_model(model, show_shapes=True, rankdir="LR")

[16]:

[17]: model.fit(train_dataset, validation_data = validation_dataset, epochs = 1)

      17060/17060 [==============================] - 41s 2ms/step - loss: 0.3763 - val_loss: 0.4414
[17]: <keras.callbacks.History at 0x7efe131a7590>

[ ]:
```