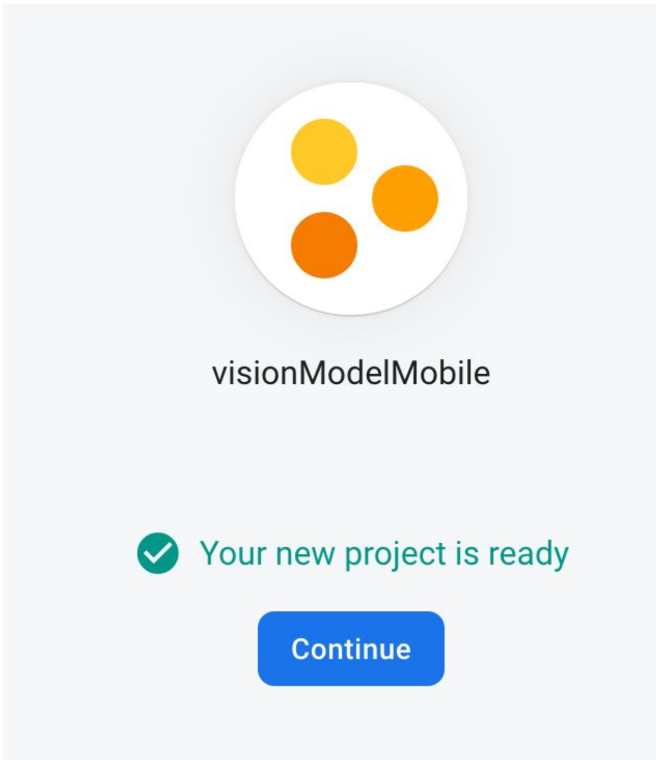
Content

AutoML Vision in ML Kit

Setup iOS project on ML Kit



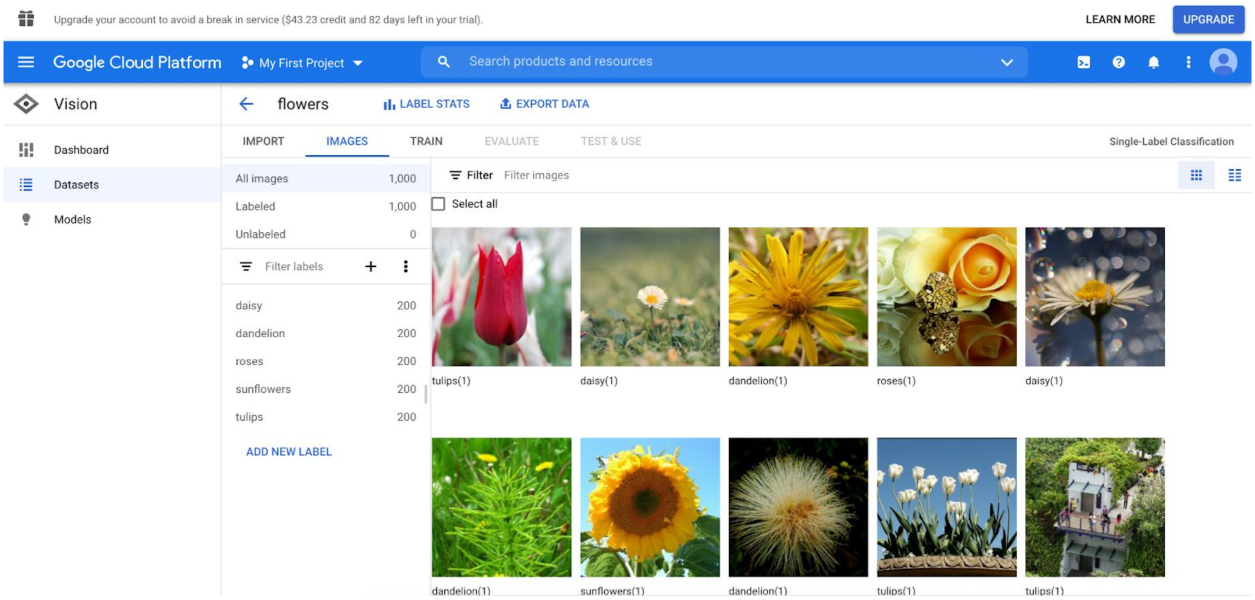
Download a zip archive that contains the project source code.

Install cocoapods on mac:

sudo xcrun gem install cocoapods

Create dataset

Create a dataset on google cloud platform and import flower_photos.zip to it.



You have enough images to start training

Unlabeled images aren't used. Your dataset will be automatically split into Train, Validation, and Test sets .

Ideally, each label should have at least **10 images**. Fewer images often result in inaccurate precision and recall. You must also have at least **8, 1, 1 images** each assigned to your Train, Validation and Test sets.

Labels	Images	Train	Validation	Test
daisy	200	160	20	20
dandelion	200	160	20	20
roses	200	160	20	20
sunflowers	200	160	20	20
tulips	200	160	20	20

START TRAINING

Create and train model

1 Define your model

Model name *

flowers_20210307

Cloud hosted

Host your model on Google Cloud for online predictions

Edge

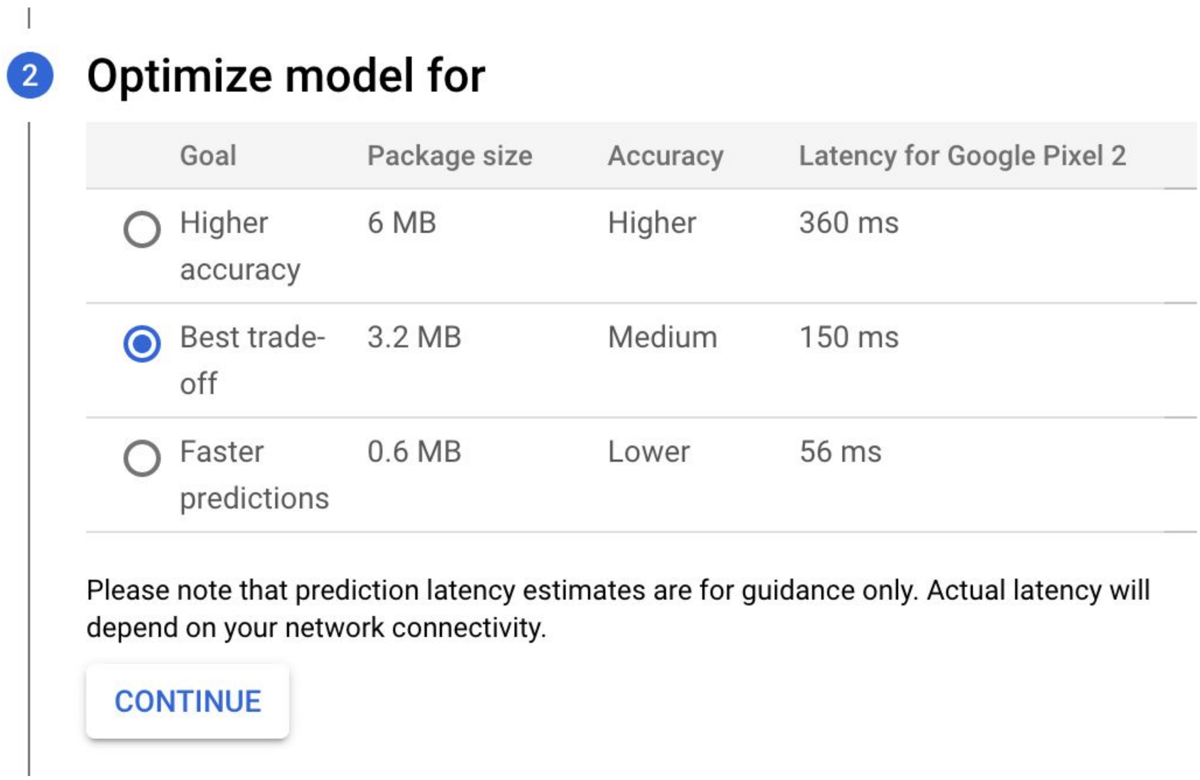
Download your model for offline/mobile use

CONTINUE

2 Optimize model for

3 Set a node hour budget

START TRAINING CANCEL



2 Optimize model for

	Goal	Package size	Accuracy	Latency for Google Pixel 2
○	Higher accuracy	6 MB	Higher	360 ms
◉	Best trade-off	3.2 MB	Medium	150 ms
○	Faster predictions	0.6 MB	Lower	56 ms

Please note that prediction latency estimates are for guidance only. Actual latency will depend on your network connectivity.

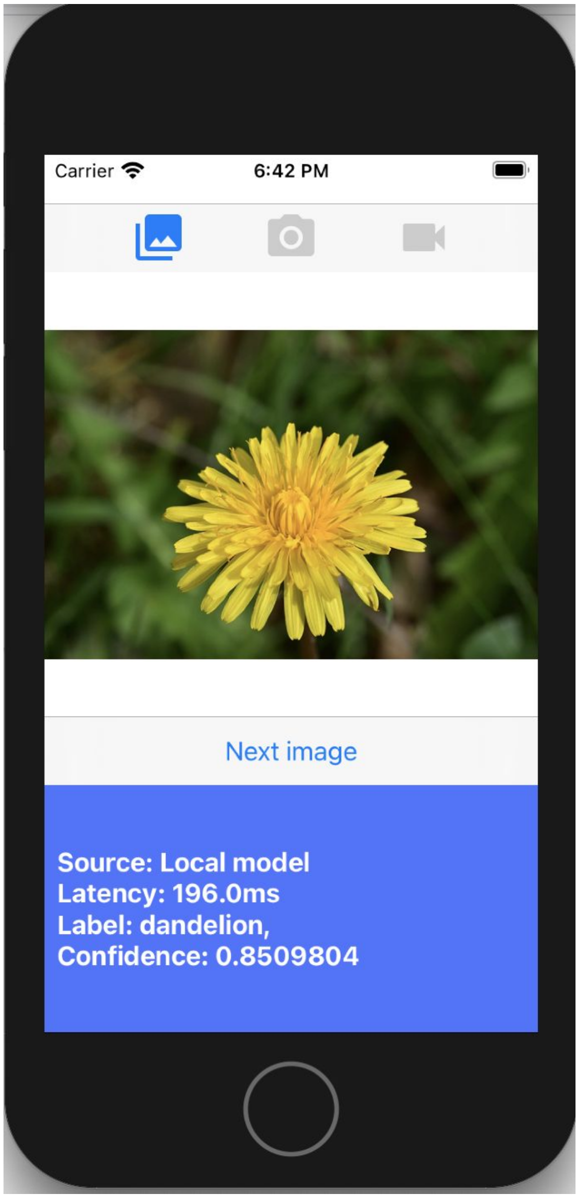
CONTINUE

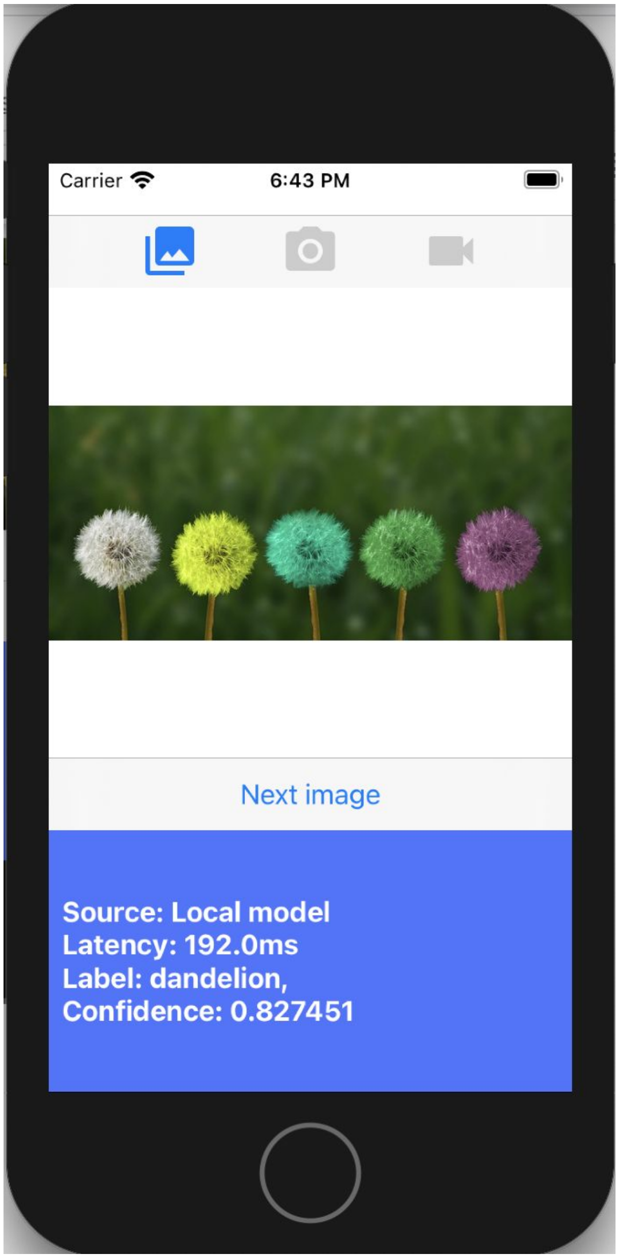
Download image classification model

Download the trained model file ImageClassifier.swift to local project folder

Test the model with iOS app

1. Open Terminal and go to **ios/mlkit-automl/** folder
2. Run pod install to download dependencies via Cocoapods
3. Run ‘open MLVisionExample.xcworkspace/’ to open the project workspace in Xcode.


Carrier
6:42 PM
Next image
Source: Local model
Latency: 196.0ms
Label: dandelion,
Confidence: 0.8509804

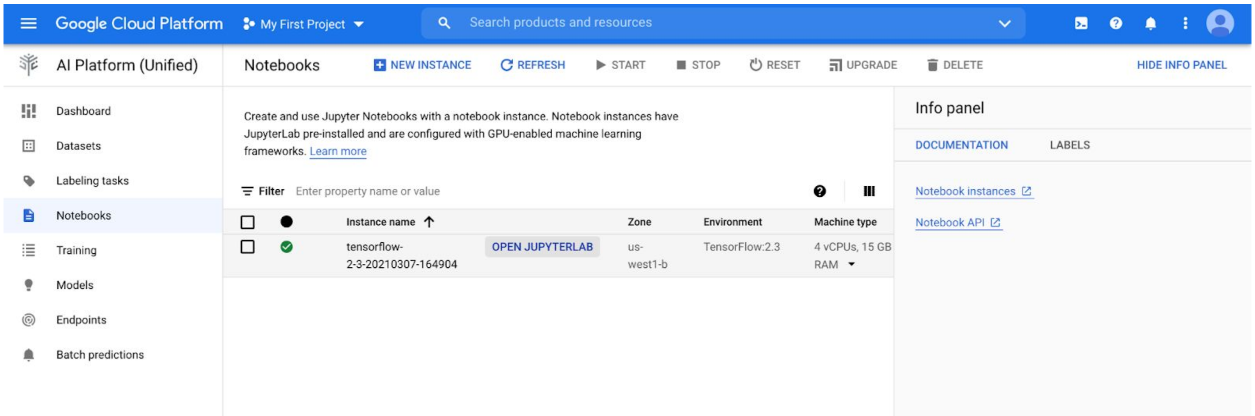


Conclusion

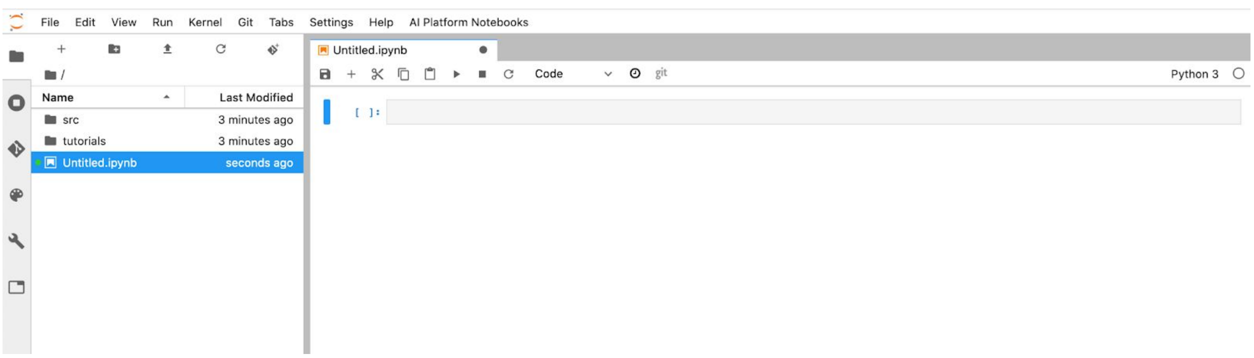
We have gone through an end-to-end training of an image classification model with training data using AutoML, and then use the model in a mobile app using ML Kit.

Series Forecasting Auto ML

Create a new instance, select the latest TensorFlow Enterprise 2.x instance type without GPUs:



create a **Python 3** notebook from JupyterLab:

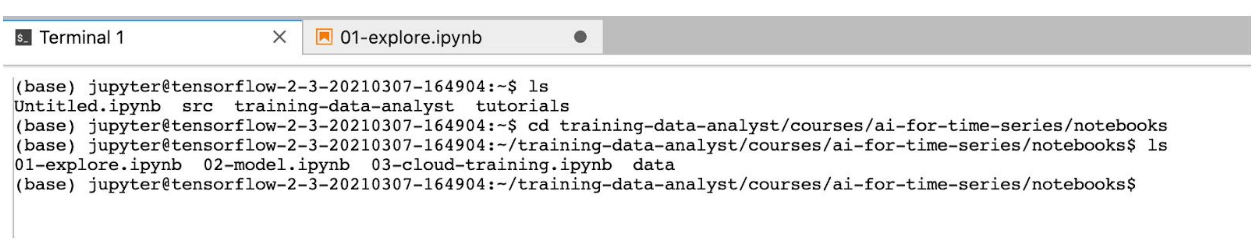


Notebook workspace setup

Create a new Terminal window from the JupyterLab interface: File -> New -> Terminal.

From there, clone the source material with this command:

Create a new terminal, File -> New -> Terminal
And clone the code from github:
git clone https://github.com/GoogleCloudPlatform/training-data-analyst



Explore and visualize data

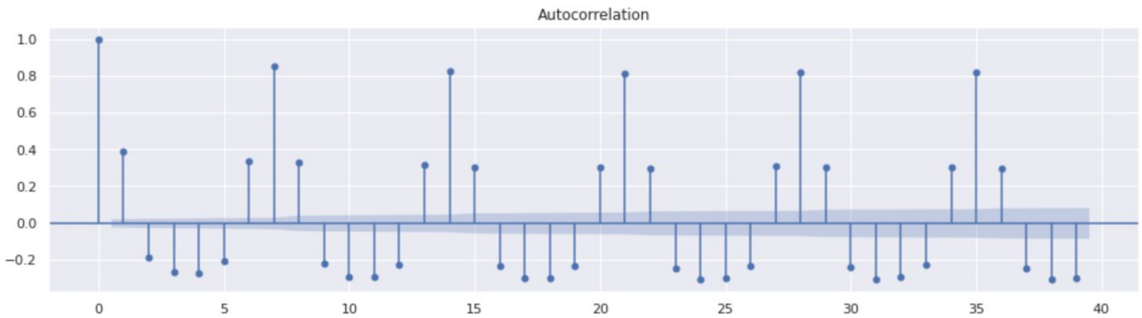
Running results of 01-explore.ipynb:

Auto-correlation

Next, we will create an auto-correlation plot, to show how correlated a time-series is with itself. Each point on the x-axis indicates the correlation at a given lag. The shaded area indicates the confidence interval.

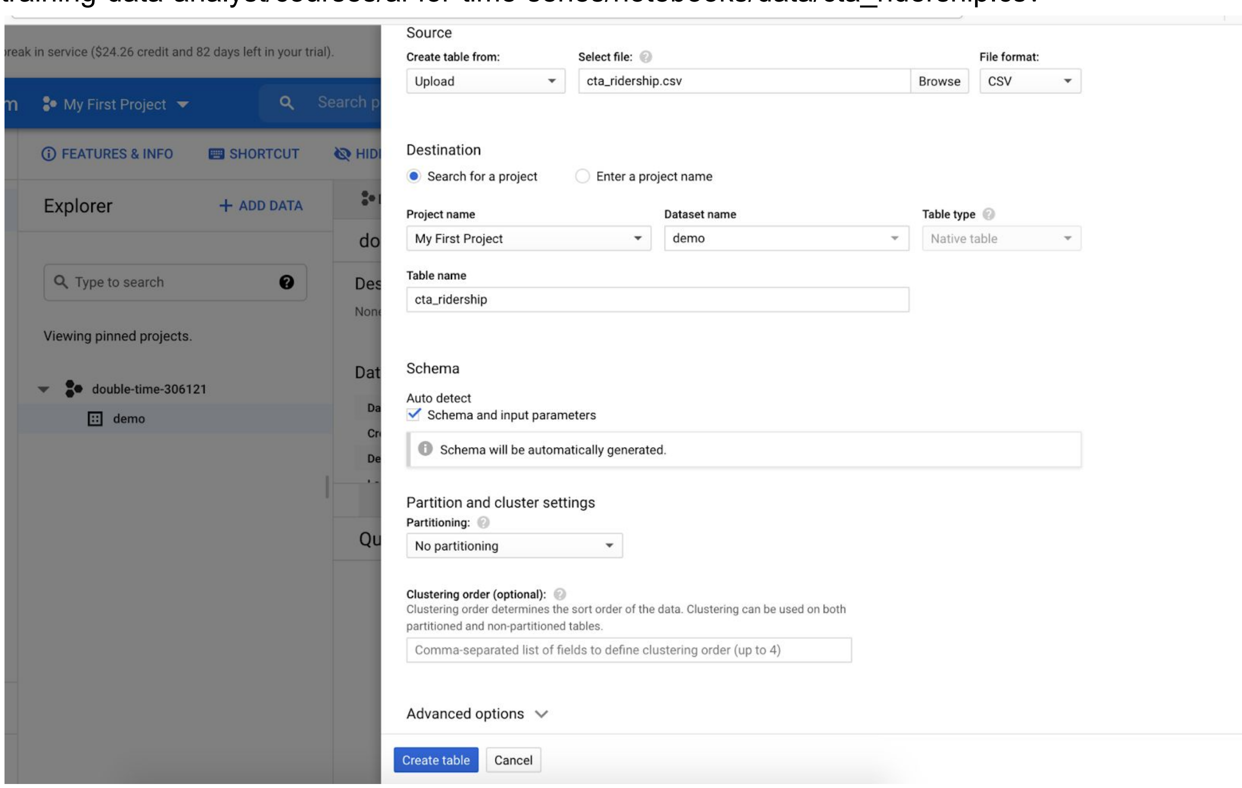
Note that the correlation gradually decreases over time, but reflects weekly seasonality (e.g. `t-7` and `t-14` stand out).

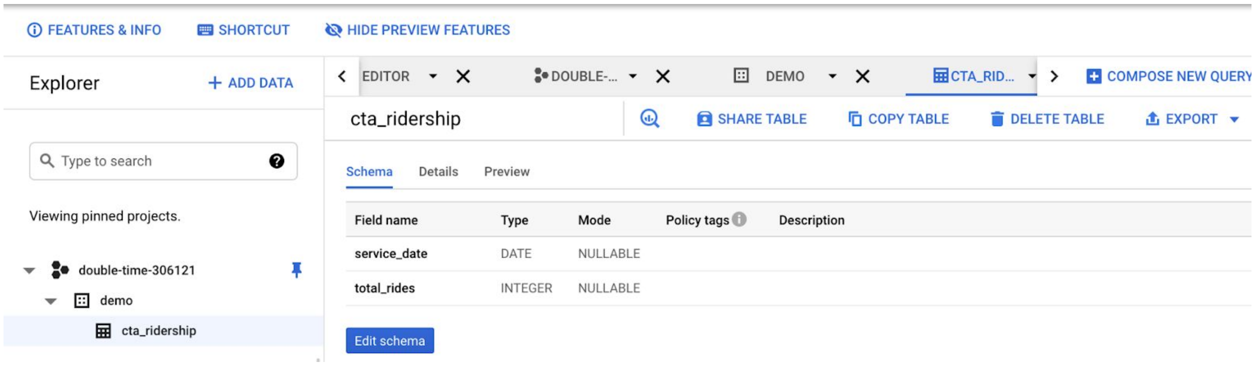
22]:
```
plot_acf(df[target])

fig = plt.show()
```



BigQuery model for Time Series Forecasting

Create table with: https://github.com/GoogleCloudPlatform/training-data-analyst training-data-analyst/courses/ai-for-time-series/notebooks/data/cta_ridership.csv

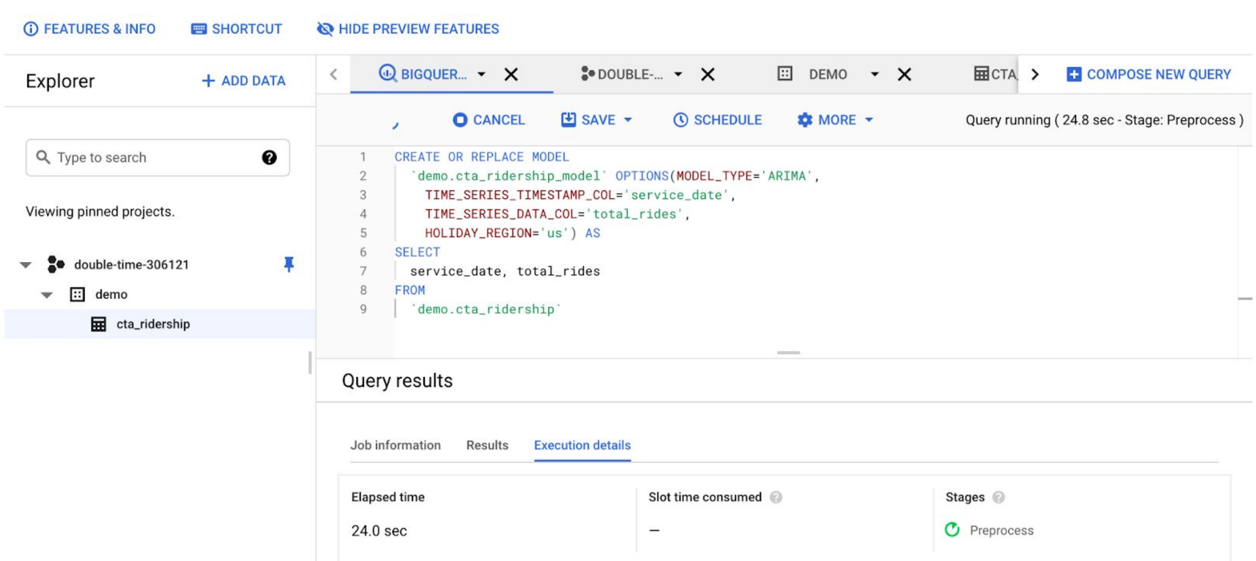


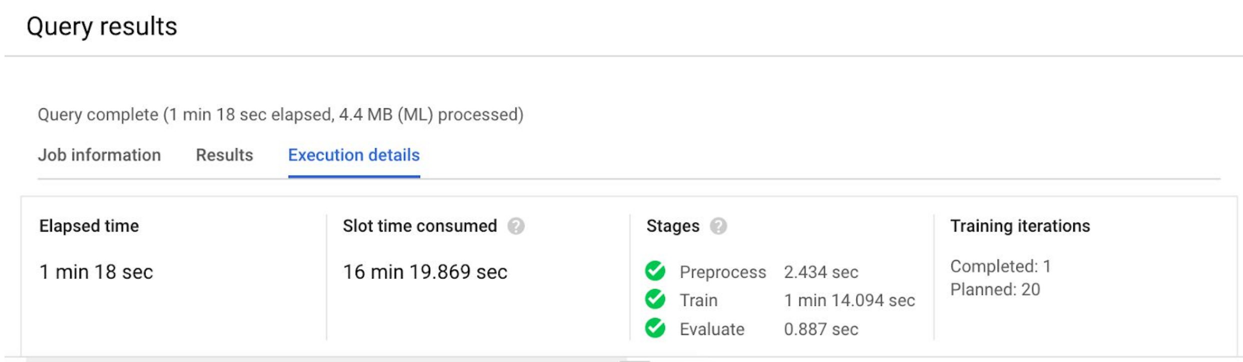


[BigQuery ML](#) provides a straightforward syntax similar to SQL that enables you to create a wide variety of model types.

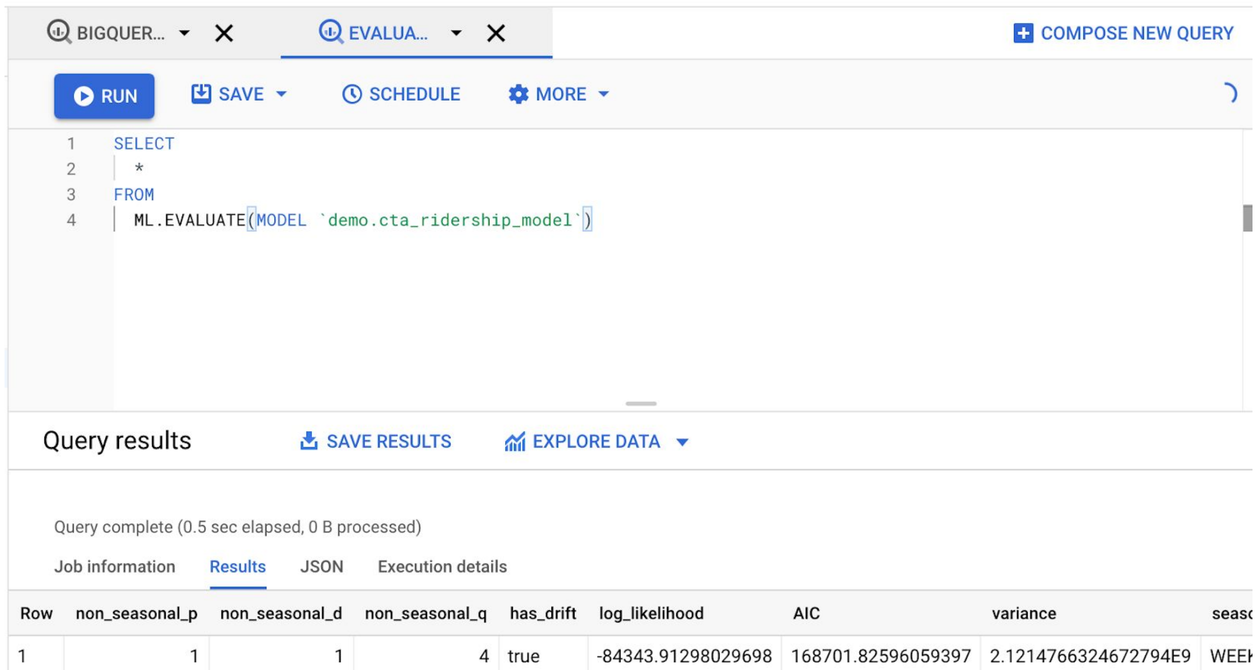
Put below in query editor:

```
CREATE OR REPLACE MODEL
 `demo.cta_ridership_model` OPTIONS(MODEL_TYPE='ARIMA',
   TIME_SERIES_TIMESTAMP_COL='service_date',
   TIME_SERIES_DATA_COL='total_rides',
   HOLIDAY_REGION='us') AS
SELECT
 service_date, total_rides
FROM
 `demo.cta_ridership`
```

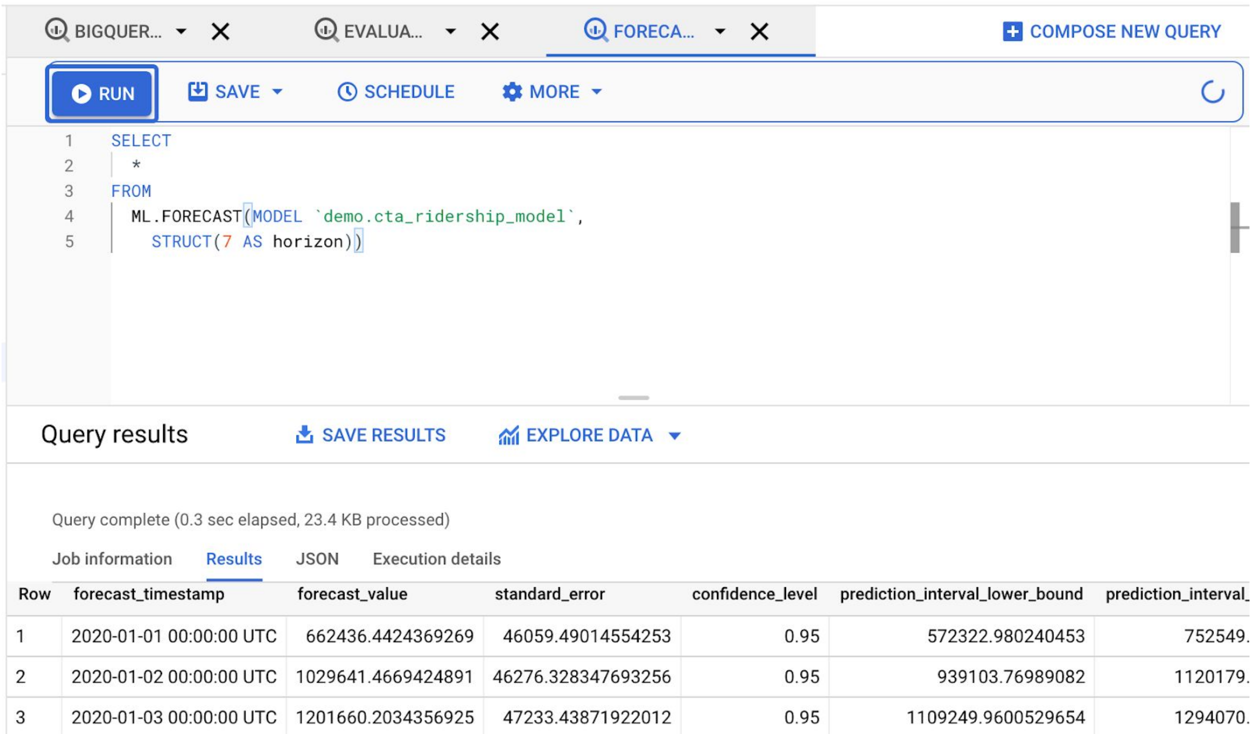




When it is finished, create a query to evaluate the model:



Create a query for forecast



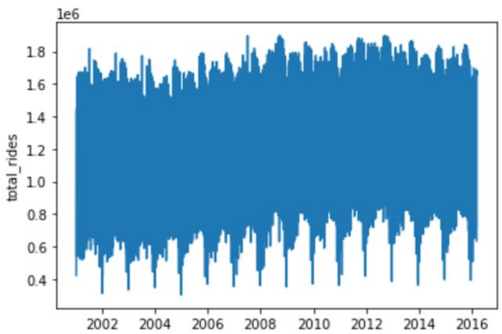
we've created a time series model with just a few BQML queries.

Custom Forecasting Model

Remove outliers

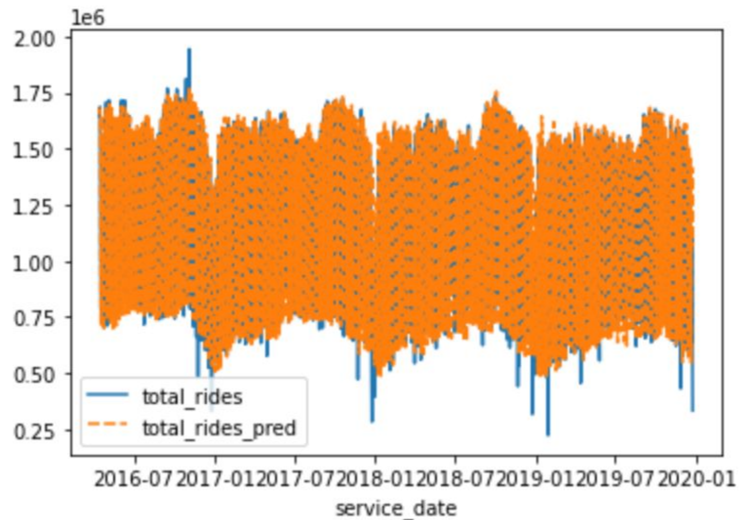
[52]:
```python
# TODO: Update the threshold below to remove the outliers
mydf = df_train
threshold = 1900000# Set this just below the level you are seeing peaks. It will flag any values above it.
assert threshold != -1, 'Set the threshold to the minimum that will eliminate outlier(s)'

# Set any values above the threshold to NaN (not a number)
df_train.loc[df_train[target_col] > threshold, target_col] = np.nan

# Interpolate the missing values (e.g. [3, NaN, 5] becomes [3, 4, 5])
df_train = df_train.interpolate()
```

[53]:
```python
# Review the updated chart to see if outliers still exist
# NOTE: If you set the threshold too low, rerun starting from the

_=sns.lineplot(data=df_train[target_col])
```



Long Short Term Memory (LSTM)¶

```
=== t+(1-7) ===
R^2:  0.803
MAPE: 0.093
MAE:  81658.308

=== t+1 ===
R^2:  0.851
MAPE: 0.079
MAE:  69050.793
```




```
=== t+2 ===
R^2:  0.815
MAPE: 0.09
MAE:  78365.961
```

Convolutional Neural Network (CNN)

```
=== t+1 ===
R^2:  0.806
MAPE: 0.092
MAE:  88857.139
```


```
=== t+2 ===
R^2:  0.771
MAPE: 0.103
MAE:  95308.853
```

Naive Model

```
[65]: evaluate(y_pred_rw, 0)
```

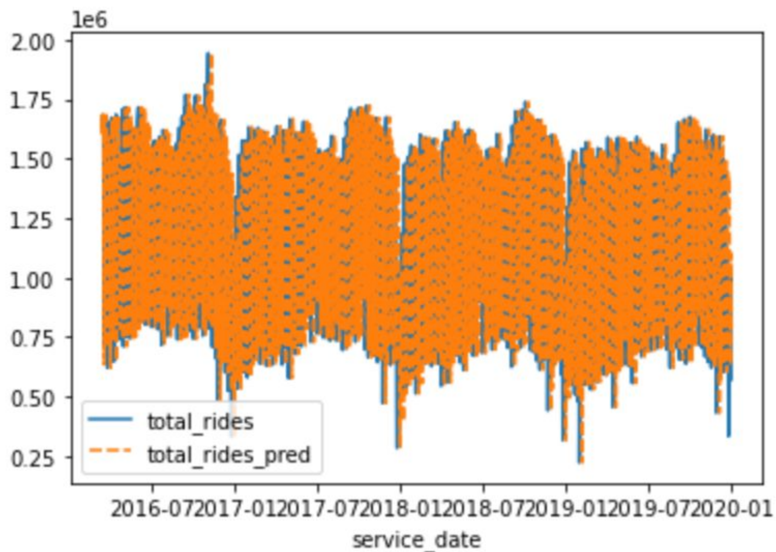
```
=== t+(1-7) ===
R^2:  -0.834
MAPE: 0.366
MAE:  364578.376

=== t+1 ===
R^2:  -0.19
MAPE: 0.257
MAE:  269441.826
```


```
=== t+2 ===
R^2:  -1.383
```

Seasonal Naive

[67]:
```
evaluate(y_pred_sn, 0)
```

```
=== t+(1-7) ===
R^2:  0.675
MAPE: 0.11
MAE:  108722.34

=== t+1 ===
R^2:  0.676
MAPE: 0.11
MAE:  108556.529
```




```
=== t+2 ===
R^2:  0.675
MAPE: 0.11
MAE:  108611.732
```

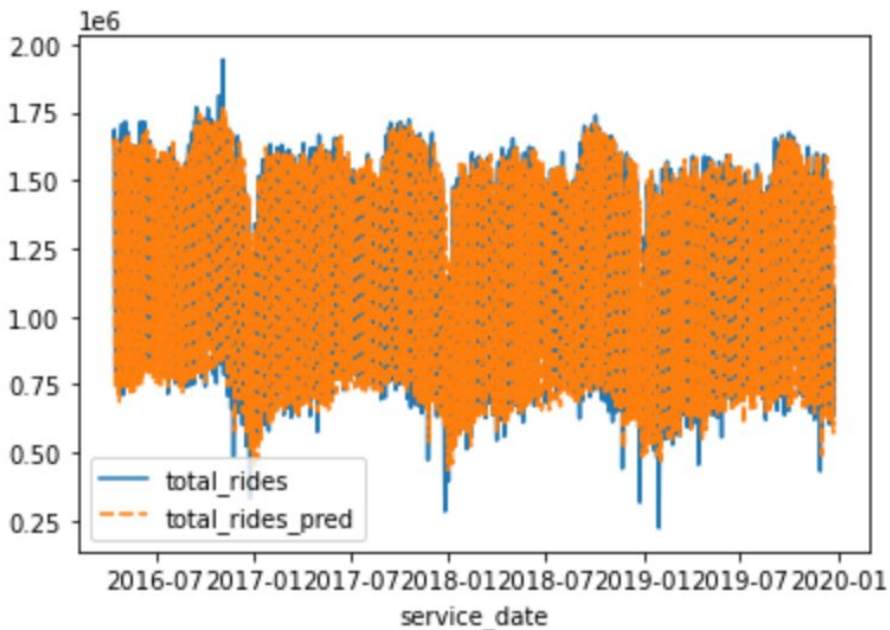
Exponential Smoothing

```
=== t+1 ===
R^2:  0.834
MAPE: 0.095
MAE:  86742.015
```


```
=== t+2 ===
R^2:  0.799
MAPE: 0.106
MAE:  96757.964
```

Ensemble ML and Statistical Models

```
=== t+(1-7) ===
R^2:  0.815
MAPE: 0.09
MAE:  80983.073

=== t+1 ===
R^2:  0.857
MAPE: 0.077
MAE:  68731.174
```



```
from tensorflow.keras.layers import AveragePooling1D

# TODO: Try adjusting the # of filters (pattern types) and kernel size (size of the sliding window)
model = Sequential([
    Conv1D(filters=32, kernel_size=3, input_shape=[n_input_steps, n_features]),
    Flatten(),
    Dense(n_output_steps)])

model.compile(optimizer='adam', loss='mae')

early_stopping = EarlyStopping(monitor='val_loss', patience=5)
_ = model.fit(x=X_train, y=y_train, validation_data=(X_test, y_test), epochs=epochs, callbacks=[early_stopping])
```

```
Epoch 1/1000
173/173 [==============================] - 1s 5ms/step - loss: 0.2960 - val_loss: 0.2493
Epoch 2/1000
173/173 [==============================] - 1s 5ms/step - loss: 0.2512 - val_loss: 0.2498
Epoch 3/1000
173/173 [==============================] - 1s 7ms/step - loss: 0.2496 - val_loss: 0.2524
Epoch 4/1000
173/173 [==============================] - 1s 6ms/step - loss: 0.2492 - val_loss: 0.2481
Epoch 5/1000
173/173 [==============================] - 1s 5ms/step - loss: 0.2485 - val_loss: 0.2469
Epoch 6/1000
173/173 [==============================] - 1s 6ms/step - loss: 0.2475 - val_loss: 0.2470
Epoch 7/1000
173/173 [==============================] - 1s 5ms/step - loss: 0.2477 - val_loss: 0.2459
```

Predict

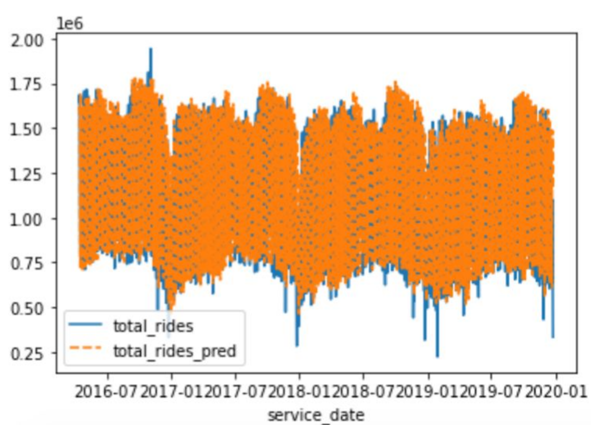
train_split =

```
[62]: model.save('./cnn_export/')
```

```
INFO:tensorflow:Assets written to: ./cnn_export/assets
```

```
[63]: preds = model.predict(X_test)
y_pred_cnn = inverse_scale(preds)

evaluate(y_pred_cnn)
```

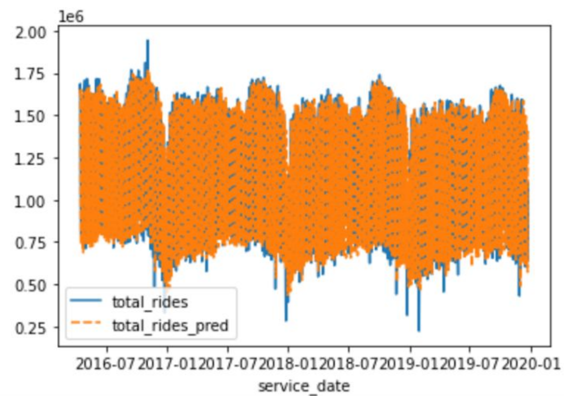
```
=== t+(1-7) ===
R^2:  0.768
MAPE: 0.104
MAE:  96710.292

=== t+1 ===
R^2:  0.806
MAPE: 0.092
MAE:  88857.139
```



Naïve Models

[74]:
```
models = [y_pred_lstm, y_pred_cnn, y_pred_es_trunc]
weights = [2, 1, 1]

y_pred_ensemble = np.average( np.array(models), axis=0, weights=weights)

evaluate(y_pred_ensemble, 0, y_true_trunc)
```

```
=== t+(1-7) ===
R^2:  0.815
MAPE: 0.09
MAE:  80983.073

=== t+1 ===
R^2:  0.857
MAPE: 0.077
MAE:  68731.174
```



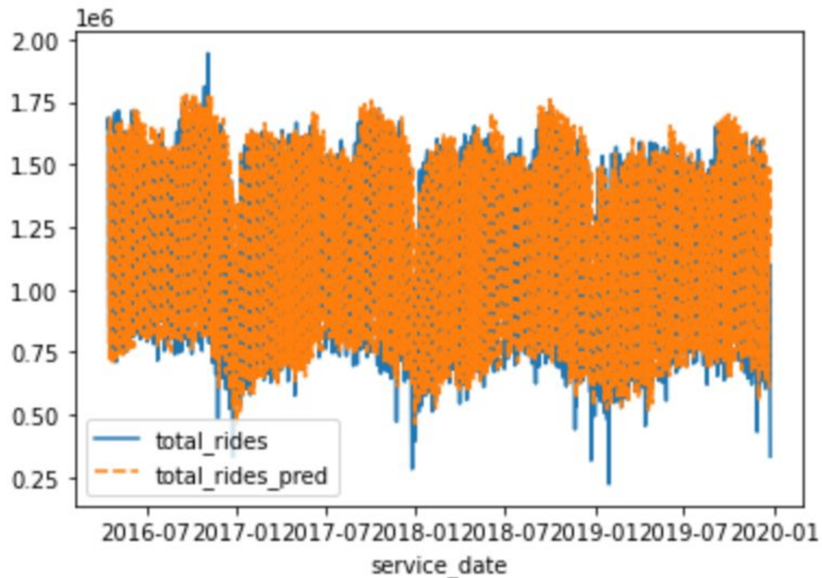
Train and Predict in the Cloud

[25]:
```
# List the contents of the bucket to ensure they were copied properly

!gsutil ls $BUCKET_URI/$TRAINER_DIR
```

```
gs://cloud-ai-platform-1221d4f9-ffb8-49f9-b973-02477d2a76b5/trainer/__init__.py
gs://cloud-ai-platform-1221d4f9-ffb8-49f9-b973-02477d2a76b5/trainer/model.py
gs://cloud-ai-platform-1221d4f9-ffb8-49f9-b973-02477d2a76b5/trainer/x_test.npy
gs://cloud-ai-platform-1221d4f9-ffb8-49f9-b973-02477d2a76b5/trainer/x_train.npy
gs://cloud-ai-platform-1221d4f9-ffb8-49f9-b973-02477d2a76b5/trainer/y_test.npy
gs://cloud-ai-platform-1221d4f9-ffb8-49f9-b973-02477d2a76b5/trainer/y_train.npy
```

```
y_pred_cnn = inverse_scale(preds)

evaluate(y_pred_cnn)
```

```
=== t+(1-7) ===
R^2:  0.768
MAPE: 0.104
MAE:  96710.292

=== t+1 ===
R^2:  0.806
MAPE: 0.092
MAE:  88857.139
```




```
=== t+2 ===
R^2:  0.771
MAPE: 0.103
MAE:  95308.853
```

Conclusion

We have learned

- Transform data so that it can be used in an ML model
- Visualize and explore data
- Remove outliers from the data
- Perform multi-step forecasting
- Include additional features in a time-series model
- Learn about neural network architectures for time-series forecasting: LSTM and CNN

- Learn about statistical models, including Holt-Winters Exponential Smoothing
- Ensemble models