

# Assignment 7: optional catchup assignment 2 - VERTEX AI - for midterm and quiz - this will catch up midterm.

Time Series Forecasting with Vertex AI and BigQuery ML

Reference: <https://codelabs.developers.google.com/codelabs/time-series-forecasting-with-cloud-ai-platform#0>

Objectives:

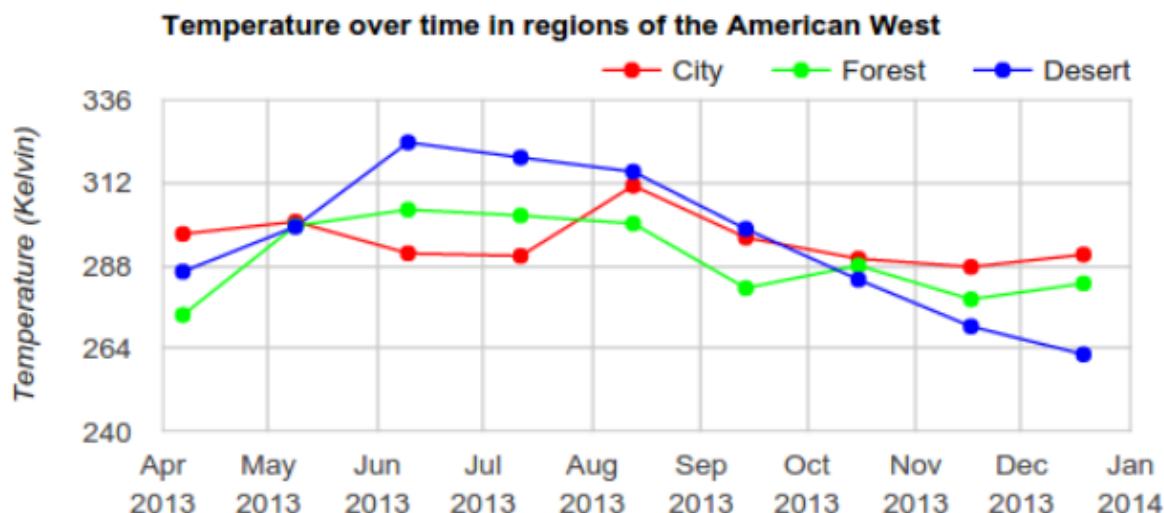
- Transform data so that it can be used in an ML model
- Visualize and explore data
- Use BigQuery ML to create a time-series forecasting model
- Build a time-series forecasting model with TensorFlow using LSTM and CNN architectures

## Introduction to Time-Series Forecasting

The focus of this codelab is on how to *apply* time-series forecasting techniques using the Google Cloud Platform

Time Series Data

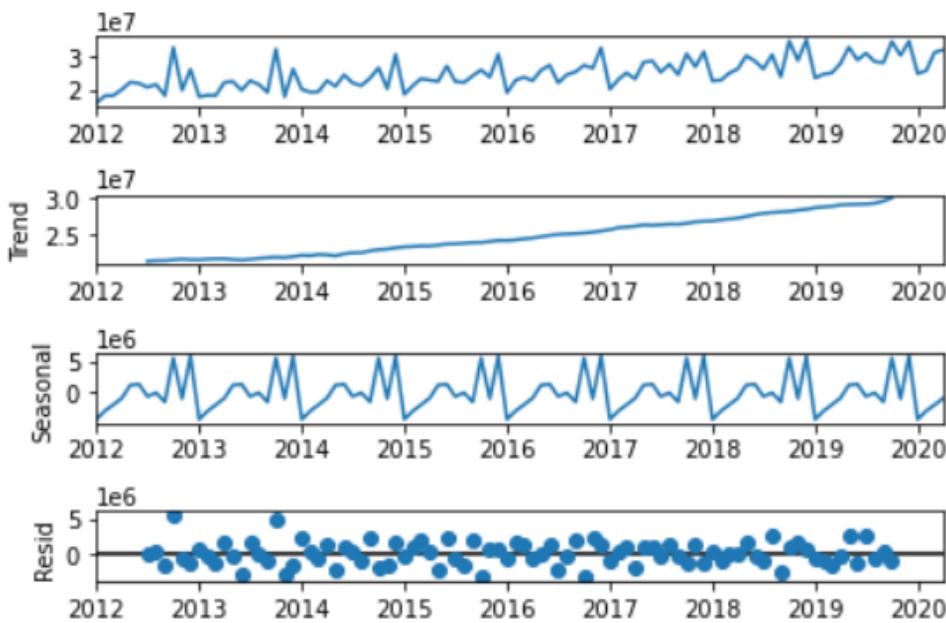
It's a dataset with data recorded at regular time intervals. A time-series dataset contains both time and at least one variable that is dependent on time.



## Components

A time-series can be decomposed into components:

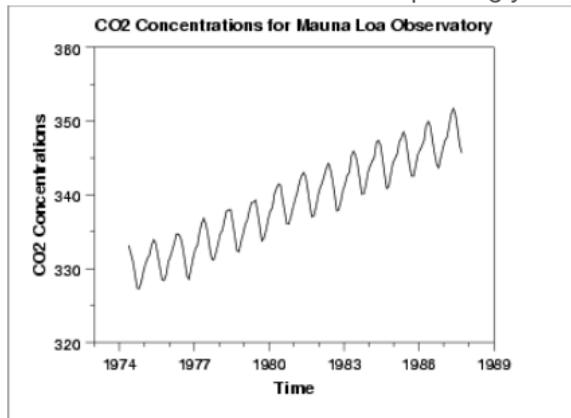
- **Trend:** moves up or down in a reasonably predictable pattern
- **Seasonal:** repeats over a specific period such as a day, week, month, season, etc.
- **Random:** residual fluctuations



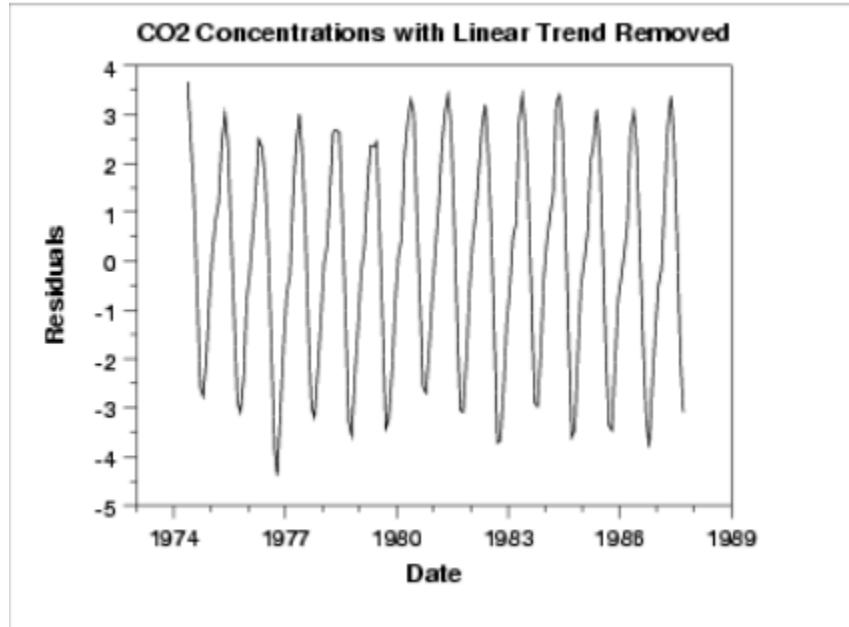
## Stationarity

For best results in forecasting, time-series data should be made [stationary](#), where statistical properties such as mean and variance are constant over time. Techniques such as differencing and detrending can be applied to raw data to make it more stationary.

The plot below of CO<sub>2</sub> concentration shows a repeating yearly pattern with an upward trend



After removing the linear trend, the data is more suitable for forecasting, as it now has a constant mean.



## Using Time Series Data for Machine Learning

To use time-series data in a machine learning problem, it needs to be transformed so that previous values can be used to predict future values. This table shows an example of how lagged variables are created to help predict the target.

Original Data		LAGGED VARIABLES (FEATURES)			TARGET
Date	Value	Date	t-2	t-1	Actual
Jan/2000	11	Jan/2000			11
Feb/2000	12	Feb/2000			12
Mar/2000	13	Mar/2000	11	12	13
Apr/2000	14	Apr/2000	12	13	14
May/2000	15	May/2000	13	14	15
Jun/2000	16	Jun/2000	14	15	16
Jul/2000	16	Jul/2000	15	16	16
Aug/2000	15	Aug/2000	16	16	15
Sep/2000	14	Sep/2000	16	15	14
Oct/2000	13	Oct/2000	15	14	13
Nov/2000	12	Nov/2000	14	13	12
Dec/2000	11	Dec/2000	13	12	11

# Setup your Notebook environment

## Step 1: Enable APIs

The BigQuery connector uses the BigQuery Storage API

The screenshot shows the Google Cloud Platform interface. On the left, there's a sidebar with various icons and a search bar at the top. The main content area has a title "The BigQuery connector uses the BigQuery Storage API". Below this, there are two tabs: "DOCUMENTATION & TUTORIALS" and "MARKETPLACE". The "DOCUMENTATION & TUTORIALS" tab is active, displaying a list of links such as "Import data from Cloud Storage to BigQuery", "Overview of BigQuery storage | BigQuery", "SKU Groups - BigQuery Storage | Documentation", etc. The "MARKETPLACE" tab shows a list of products, with "BigQuery Storage API" highlighted. On the right, there's a sidebar titled "Endpoints" with a "CREATE ENDPOINT" button. It lists three endpoints: "default\_pred\_v1", "beans-model-pipeline\_endpoint", and "beans-model-pipeline\_endpoint". The "default\_pred\_v1" endpoint is selected. At the bottom, there's a URL bar with the address <https://console.cloud.google.com/marketplace/product/google/bigquerystorage.googleapis.com?q=search&referrer=search&project=cmpe260-334300>.

## Step 2: Create a Vertex AI Workbench notebook

Navigate to Vertex AI → Workbench

The screenshot shows the Google Cloud Platform interface. The left sidebar has sections for "Financial Services", "Healthcare", "Life Sciences", "Dataprep", and "ARTIFICIAL INTELLIGENCE". Under "ARTIFICIAL INTELLIGENCE", "Vertex AI" is expanded, showing sub-options like "Dashboard", "Datasets", "Features", "Labeling tasks", "Workbench", "Pipelines", "Training", "Experiments", "Models", "Endpoints", "Batch predictions", and "Metadata". The "Workbench" option is currently selected. The main content area shows a "Compute Engine" dashboard with a graph of CPU utilization over time. To the right, there are sections for "Google Cloud Platform status", "Billing", and "Monitoring". The URL in the address bar is <https://console.cloud.google.com/vertex-ai/workbench?project=cmpe260-334300>.

Select **TensorFlow Enterprise 2.3 (with LTS)** instance type **without GPUs**:

The screenshot shows the Google Cloud Platform Vertex AI Workbench interface. On the left, there's a sidebar with various options like Vertex AI, Dashboard, Datasets, Features, Labeling tasks, Workbench (which is selected), Pipelines, Training, Experiments, Models, Endpoints, Batch predictions, and Metadata. Below that is a Marketplace section. The main area is titled "Notebooks" and has a "NEW NOTEBOOK" button. It lists several managed notebook environments:

- Python 3**: As of the MI environment. Includes scikit-learn, pandas and more.
- TensorFlow Enterprise**: Includes Keras, scikit-learn, pandas, NLTK and more.
- PyTorch 1.9**: Includes scikit-learn, pandas, NLTK and more.
- R 4.1**: Includes basic R packages, scikit-learn, pandas, NLTK and more.
- RAPIDS 0.18 [EXPERIMENTAL]**: Optimized for NVIDIA GPUs.
- Kaggle Python [BETA]**: Python image for Kaggle Notebooks, supporting hundreds of machine learning libraries popular on Kaggle.
- Theia IDE [EXPERIMENTAL]**: IDE with notebook support including scikit-learn, pandas, and more.
- Smart Analytics Frameworks**: BigQuery, Apache Beam, Apache Spark, Apache Hive, and more.

A modal window titled "TensorFlow Enterprise 2.3 (with LTS)" is open, showing details about the environment, including "Without GPUs".

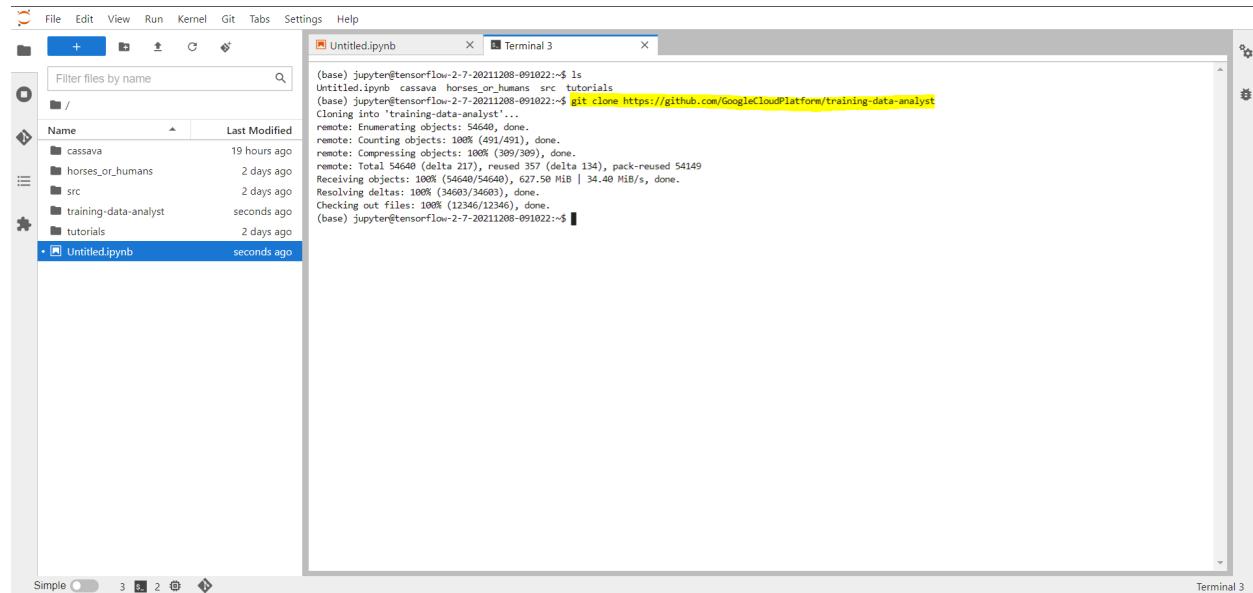
Then, create a **Python 3** notebook from JupyterLab:

The screenshot shows the JupyterLab interface. On the left is a file browser with a sidebar showing a directory structure with files like cassava, horses\_or\_humans, src, and tutorials. A file named "Untitled.ipynb" is selected. The main area is titled "Untitled.ipynb" and shows three sections: Notebook, Console, and Other. Under Notebook, there are two icons: "Python 3" and "Python [conda envroot] \*". Under Console, there are also two icons: "Python 3" and "Python [conda envroot] \*". Under Other, there are icons for Terminal, Text File, Markdown File, Python File, and Show Contextual Help.

## Step 3: Download lab materials

Run the following command in the Terminal:

```
git clone https://github.com/GoogleCloudPlatform/training-data-analyst
```



The screenshot shows a Jupyter Notebook interface. On the left is a file browser with a list of files and folders under the root directory. In the center is a terminal window displaying the output of a git clone command. The command cloned the 'training-data-analyst' repository from GitHub. The terminal output shows the progress of cloning, including the enumeration of objects, compression, and receiving objects.

```
(base) jupyter@tensorflow-2-7-20211208-091022:~$ ls
Untitled.ipynb cassava horses_or_humans src tutorials
(base) jupyter@tensorflow-2-7-20211208-091022:~$ git clone https://github.com/GoogleCloudPlatform/training-data-analyst
Cloning into 'training-data-analyst'...
remote: Enumerating objects: 54640, done.
remote: Counting objects: 1080 (491/491), done.
remote: Compressing objects: 100% (309/309), done.
remote: Total 54640 (delta 217), reused 357 (delta 134), pack-reused 54149
Receiving objects: 100% (54640/54640), 627.50 MB | 34.40 MB/s, done.
Resolving deltas: 100% (34683/34683), done.
Checking out files: 100% (12541/12546), done.
(base) jupyter@tensorflow-2-7-20211208-091022:~$
```

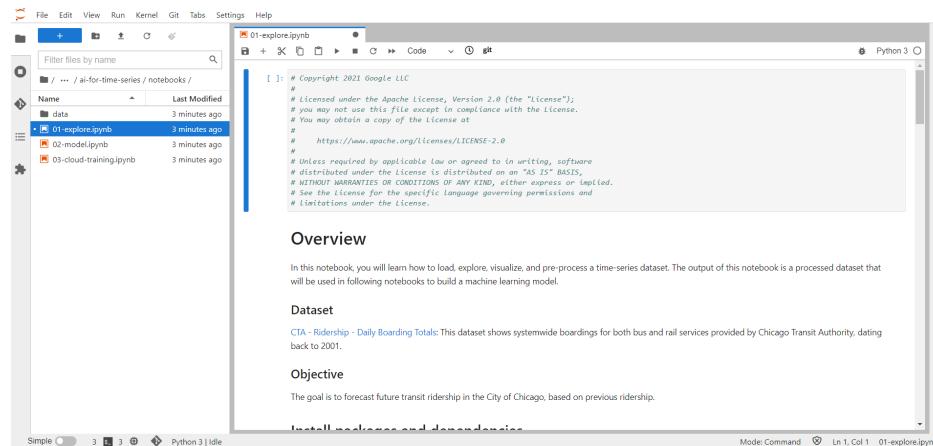
## Explore and Visualize Data

The following are the tasks in this section:

- Create a query that groups data into a time-series
- Fill missing values
- Visualize data
- Decompose time-series into trend and seasonal components

### Step 1

In Vertex AI Workbench, navigate to `training-data-analyst/courses/ai-for-time-series/notebooks` and open `01-explore.ipynb`.



The screenshot shows a Jupyter Notebook interface with the file `01-explore.ipynb` selected in the sidebar. The main area displays the code and comments for this notebook. The code includes a copyright notice and a detailed dataset description. Below the code, there are sections titled 'Overview' and 'Dataset'. The 'Overview' section provides a brief description of the dataset, stating it shows systemwide boardings for both bus and rail services provided by Chicago Transit Authority, dating back to 2001. The 'Dataset' section describes the 'CITA - Ridership - Daily Boarding Totals' dataset.

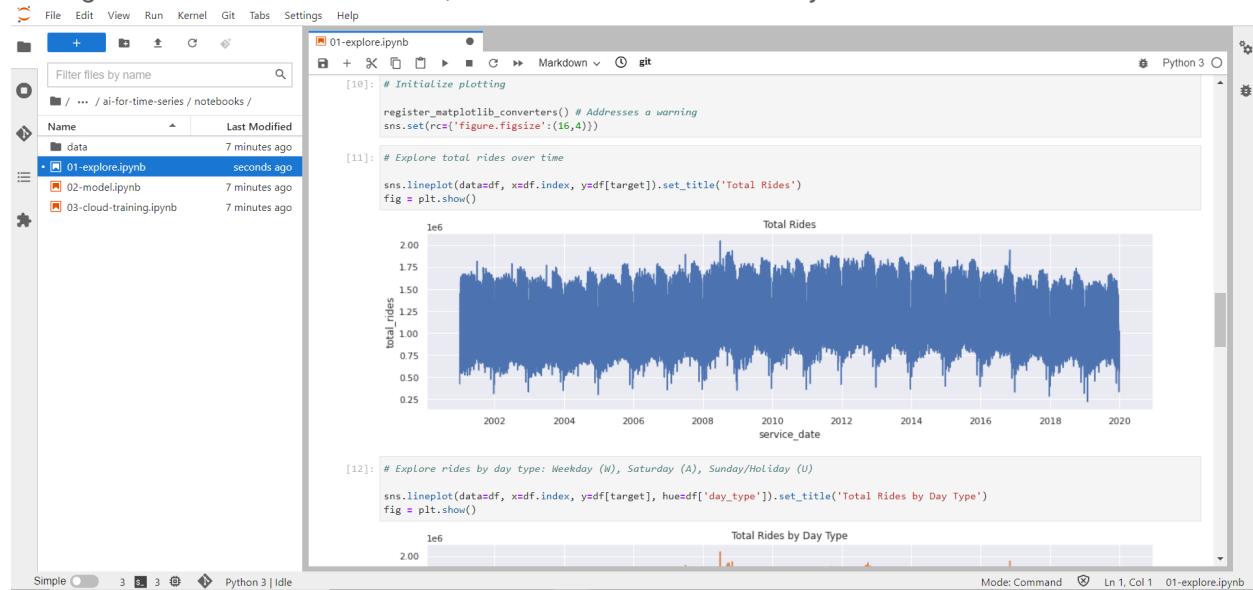
```
1: # Copyright 2021 Google LLC
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#     https://www.apache.org/licenses/LICENSE-2.0
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
```

**Overview**  
In this notebook, you will learn how to load, explore, visualize, and pre-process a time-series dataset. The output of this notebook is a processed dataset that will be used in following notebooks to build a machine learning model.

**Dataset**  
[CITA - Ridership - Daily Boarding Totals](#): This dataset shows systemwide boardings for both bus and rail services provided by Chicago Transit Authority, dating back to 2001.

## Step 2

Clear all the cells in the notebook (Edit > Clear All Outputs), change the region, project and bucket settings in one of the first few cells, and then Run the cells one by one.

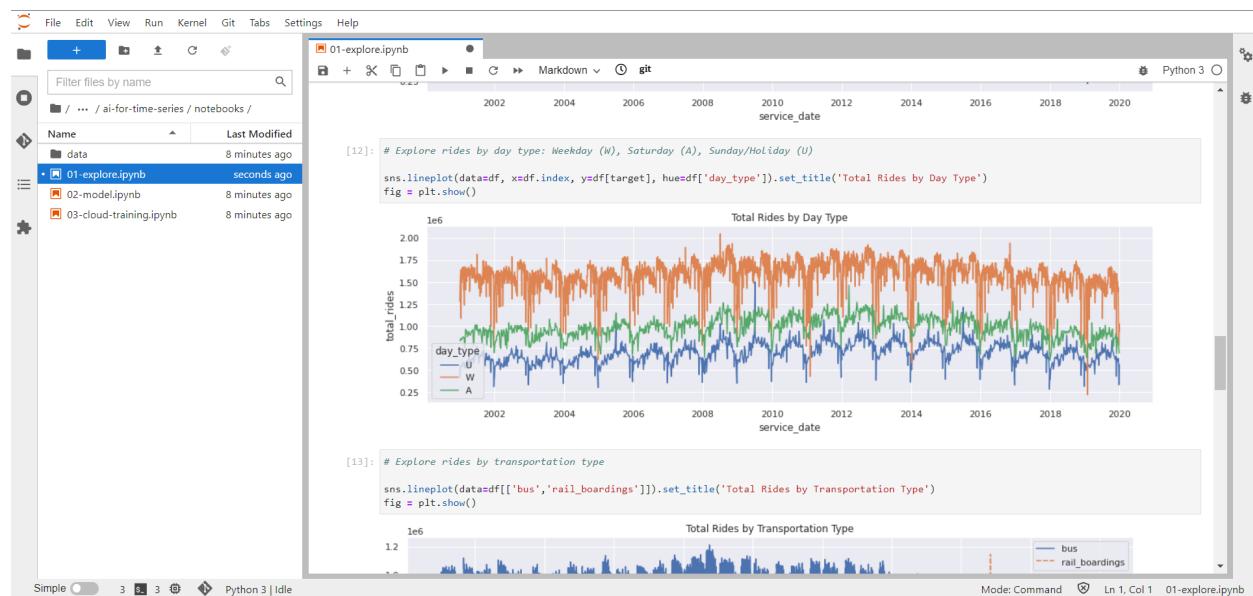


The screenshot shows the Jupyter Notebook interface with the file tree on the left and the code editor on the right. The current file is '01-explore.ipynb'. The code in cell [10] initializes plotting and sets the figure size. Cell [11] explores total rides over time using a line plot. The resulting plot, titled 'Total Rides', shows a highly volatile blue line representing daily ride counts from approximately 2002 to 2020, with values ranging from 0.25 to 2.00 million. The x-axis is labeled 'service\_date' and ranges from 2002 to 2020. The y-axis is labeled 'total\_rides' on a logarithmic scale from 0.25 to 2.00.

```
[10]: # Initialize plotting
register_matplotlib_converters() # Addresses a warning
sns.set(rc={'figure.figsize':(16,4)})

[11]: # Explore total rides over time
sns.lineplot(data=df, x=df.index, y=df[target]).set_title('Total Rides')
fig = plt.show()
```

Mode: Command | Ln 1, Col 1 | 01-explore.ipynb



The screenshot shows the Jupyter Notebook interface with the file tree on the left and the code editor on the right. The current file is '01-explore.ipynb'. The code in cell [12] explores rides by day type using a line plot. The resulting plot, titled 'Total Rides by Day Type', shows four colored lines representing different day types: Weekday (blue), Saturday (orange), Sunday/Holiday (green), and Unknown (red). The x-axis is labeled 'service\_date' and ranges from 2002 to 2020. The y-axis is labeled 'total\_rides' on a logarithmic scale from 0.25 to 2.00. A legend on the left identifies the colors: blue for W, orange for U, green for A, and red for S. Cell [13] explores rides by transportation type using a line plot. The resulting plot, titled 'Total Rides by Transportation Type', shows two lines: 'bus' (solid blue) and 'rail\_boardings' (dashed orange). The x-axis is labeled 'service\_date' and ranges from 2002 to 2020. The y-axis is labeled 'total\_rides' on a logarithmic scale from 0.25 to 1.20.

```
[12]: # Explore rides by day type: Weekday (W), Saturday (A), Sunday/Holiday (U)
sns.lineplot(data=df, x=df.index, y=df[target], hue=df['day_type']).set_title('Total Rides by Day Type')
fig = plt.show()

[13]: # Explore rides by transportation type
sns.lineplot(data=df[['bus','rail_boardings']].set_title('Total Rides by Transportation Type')
fig = plt.show()
```

Mode: Command | Ln 1, Col 1 | 01-explore.ipynb

The screenshot shows a Jupyter Notebook interface with several tabs at the top: File, Edit, View, Run, Kernel, Git, Tabs, Settings, Help. The left sidebar shows a file tree with notebooks like '01-explore.ipynb' (selected), '02-model.ipynb', and '03-cloud-training.ipynb'. The main area displays a line plot titled 'Total Rides by Transportation Type' with two series: 'bus' (blue line) and 'rail\_boardings' (orange dashed line). The x-axis represents 'service\_date' from 2002 to 2020, and the y-axis represents ride counts from 0.2 to 1.2e6. Below the plot, a section titled 'TODO 2: Review summary statistics' contains a bullet list and a code snippet.

```
[13]: # Explore rides by transportation type
sns.lineplot(data=df[['bus','rail_boardings']].set_title('Total Rides by Transportation Type')
fig = plt.show()
```

**TODO 2: Review summary statistics**

- How many records are in the dataset?
- What is the average # of riders per day?

```
[ ]: df[target].describe().apply(lambda x: round(x))
```

Mode: Command | Ln 1, Col 1 | 01-explore.ipynb

## Create a Model with BigQuery Time Series Forecasting

Following are the tasks in this section:

- Import your time series input data into a BigQuery table
- Create a time series model using BQML syntax
- Learn how to evaluate your model parameters and accuracy
- Forecast using your model

### Step 1

We are going to create a BigQuery table with the raw data from the CSV we just explored

From the `training-data-analyst/courses/ai-for-time-series/notebooks/data` directory, right-click on `cta_ridership.csv` and **Download** it to your local environment.

The screenshot shows a Jupyter Notebook interface with several tabs at the top: File, Edit, View, Run, Kernel, Git, Tabs, Settings, Help. The left sidebar shows a file tree with notebooks like '01-explore.ipynb' (selected), '02-model.ipynb', and '03-cloud-training.ipynb', and a CSV file 'cta\_ridership.csv'. The main area displays a scatter plot of data points. Below the plot, sections for 'Export data' and 'Conclusion' are shown.

**Export data**

This will generate a CSV file, which you will use in the next labs of this quest. Inspect the CSV file to see what the data looks like.

```
[19]: df[[target]].to_csv(processed_file, index=True, index_label=ts_col)
```

**Conclusion**

You've successfully completed the exploration and visualization lab. You've learned how to:

- Create a query that groups data into a time series
- Visualize data
- Decompose time series into trend and seasonal components

Mode: Command | Ln 1, Col 1 | 01-explore.ipynb

## Step 2

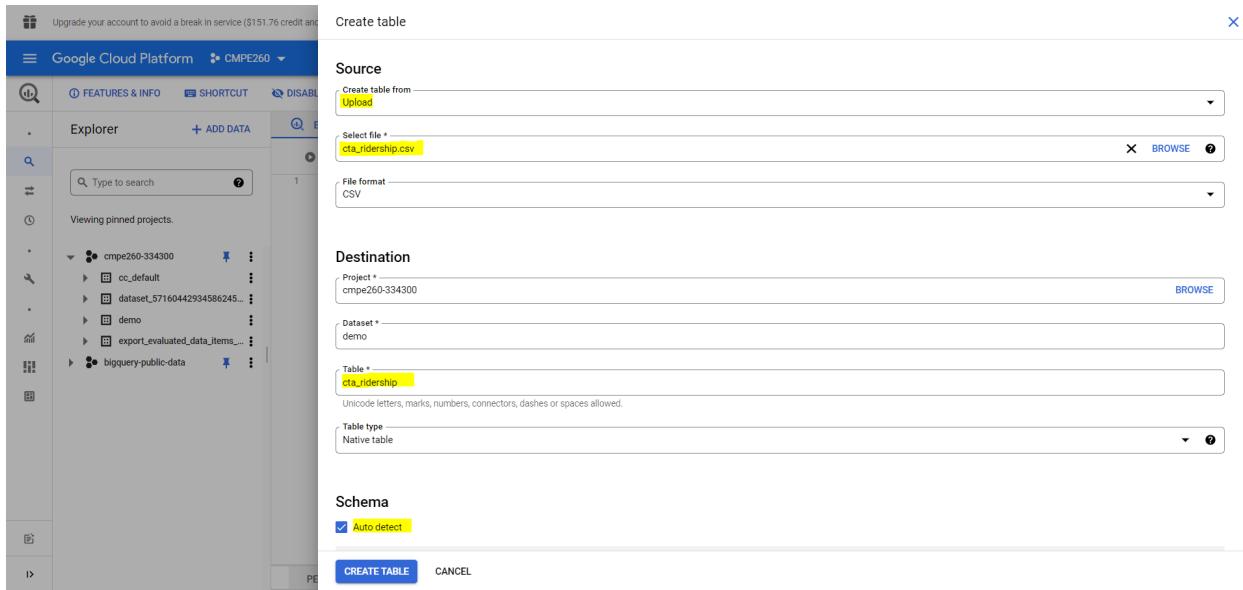
We will upload this data into a BigQuery table.

**Navigate to 'BigQuery'**

The screenshot shows the Google Cloud Platform dashboard with the navigation menu open. The 'BigQuery' option is highlighted in yellow. The menu tree under 'BigQuery' includes 'ANALYSIS', 'MIGRATION', and 'ADMINISTRATION'. The URL in the address bar is <https://console.cloud.google.com/bigquery?project=cmpe260-334300>.

**Create a Dataset 'demo':**

The screenshot shows the 'Create dataset' dialog box. The 'Dataset ID' field contains 'demo'. The 'Data location' dropdown is set to 'us-central1 (Iowa)'. Under 'Default table expiration', there is a checkbox for 'Enable table expiration' which is unchecked. The 'Encryption' section shows 'Google-managed encryption key' selected. At the bottom are 'CREATE DATASET' and 'CANCEL' buttons.



## Step 3

Create our model

Run the below query in the query editor

```

1 CREATE OR REPLACE MODEL
2   'demo.cta_ridership_model' OPTIONS(MODEL_TYPE='ARIMA',
3   TIME_SERIES_TIMESTAMP_COL='service_date',
4   TIME_SERIES_DATA_COL='total_rides',
5   HOLIDAY_REGION='us') AS
6   SELECT
7     service_date, total_rides
8   FROM
9   | 'demo.cta_ridership'

```

## Step 4

### Querying the model

The screenshot shows the Google Cloud Platform BigQuery interface. The left sidebar displays a project structure with a 'Models' section containing a single entry: 'ctaridership\_model'. The main area shows a query editor with the following SQL code:

```

1 SELECT
2   *
3   FROM
4   [ML.EVALUATE(MODEL `demo.cta_ridership_model`)]

```

The results pane shows the output of the query:

Row	non_seasonal_p	non_seasonal_d	non_seasonal_q	has_drift	log_likelihood	AIC	variance	seasonal_periods
1	0	1	5	true	-84291.96365702226	168597.92731404453	2.0899385335999274E9	WEEKLY
2	0	1	5	false	-84293.2544933968	168598.5089867936	2.090729530268954E9	WEEKLY

Below the table, there are buttons for 'PERSONAL HISTORY', 'PROJECT HISTORY', and 'SAVED QUERIES'.

## Step 5

### Ready to forecast with the [ML.FORECAST](#) function

The screenshot shows the Google Cloud Platform BigQuery interface. The left sidebar displays a project structure with a 'Models' section containing a single entry: 'ctaridership\_model'. The main area shows a query editor with the following SQL code:

```

1 SELECT
2   *
3   FROM
4   [ML.FORECAST(MODEL `demo.cta_ridership_model` ,
5   STRUCT(7 AS horizon))]

```

The results pane shows the output of the query:

Row	forecast_timestamp	forecast_value	standard_error	confidence_level	prediction_interval_lower_bound	prediction_interval_upper_bound	confidence_interval_lower_bound	confidence_interval_upper_bound
1	2020-01-01 00:00:00 UTC	672673.4067125214	45715.84554178048	0.95	583232.2707817592	762114.5426432837	583232.2707817592	762114.5426432837
2	2020-01-02 00:00:00 UTC	1028503.0865139209	45806.99994822557	0.95	938883.6108070718	1118122.56222077	938883.6108070718	1118122.56222077
3	2020-01-03 00:00:00 UTC	1177797.9717879058	46287.610736162766	0.95	1087238.2012160213	1268357.7423597903	1087238.2012160213	1268357.7423597903
4	2020-01-04 00:00:00 UTC	638818.6543572493	47234.50401336978	0.95	546406.3267711629	731230.9819433356	546406.3267711629	731230.9819433356
5	2020-01-05 00:00:00 UTC	473730.228701511	48149.235480549505	0.95	379528.2673343356	567932.190679666	379528.2673343356	567932.190679666
6	2020-01-06 00:00:00 UTC	1163952.3480790951	48179.67357435147	0.95	1069690.8358573639	1258213.8603008264	1069690.8358573639	1258213.8603008264
7	2020-01-07 00:00:00 UTC	1131827.4618996186	48210.09245065515	0.95	1037506.4364212014	1226148.487378036	1037506.4364212014	1226148.487378036

Below the table, there are buttons for 'PERSONAL HISTORY', 'PROJECT HISTORY', and 'SAVED QUERIES'.

# Build a Custom Forecasting Model

- 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

## Step 1

In Vertex AI Workbench, navigate to [training-data-analyst/courses/ai-for-time-series/notebooks](#) and open `02-model.ipynb`.

The screenshot shows the Vertex AI Workbench interface. On the left, there is a file browser window titled 'File browser' showing files in the directory `/.../ai-for-time-series/notebooks/`. The files listed are `01-explore.ipynb`, `02-model.ipynb` (which is selected), `03-cloud-training.ipynb`, and `cta_ridership.csv`. The main workspace shows two tabs: `01-explore.ipynb` and `02-model.ipynb`. The `02-model.ipynb` tab is active and displays the following content:

```
The goal is to forecast future transit ridership in the City of Chicago, based on previous ridership.

Install packages and dependencies
Restarting the kernel may be required to use new packages.

[ ]: %pip install -U statsmodels scikit-learn --user
Note: To restart the Kernel, navigate to Kernel > Restart Kernel... on the Jupyter menu.

Import libraries and define constants
[ ]: import os
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import warnings

from google.cloud import storage
from pandas.plotting import register_matplotlib_converters
from sklearn.preprocessing import StandardScaler
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from statsmodels.tools.sm_exceptions import ConvergenceWarning
from tensorflow.keras import Sequential
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.layers import Conv1D, Dense, Dropout, Flatten, LSTM, MaxPooling1D
warnings.filterwarnings('ignore') # Address warning
```

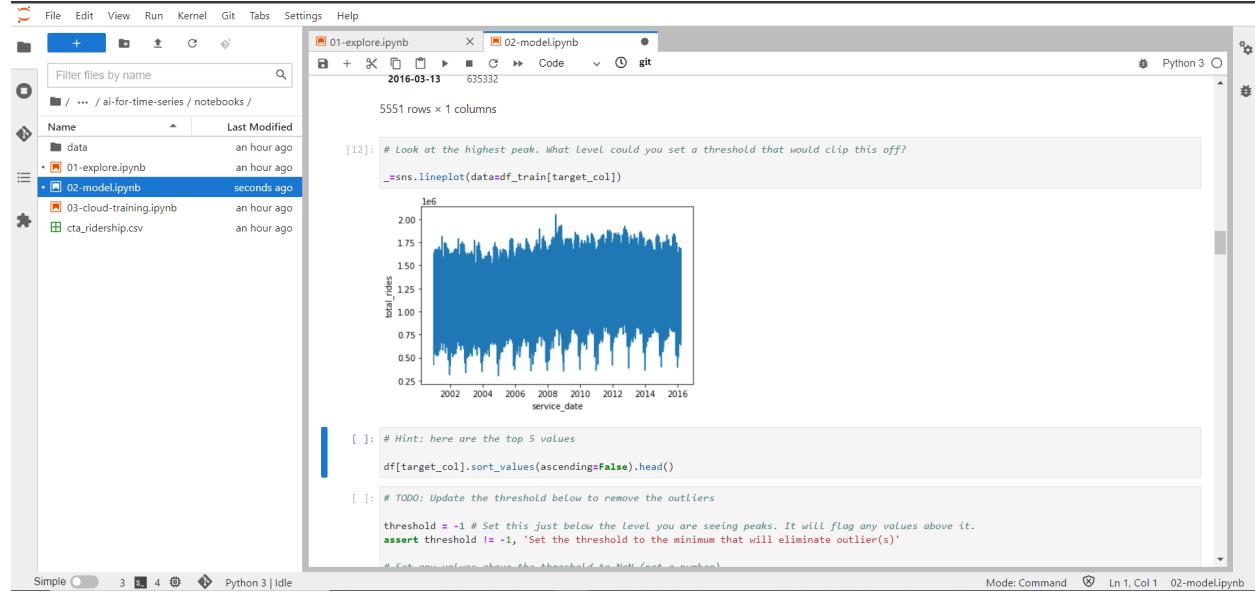
## Step 2

Clear all the cells in the notebook (Edit > Clear All Outputs)

## Step 3

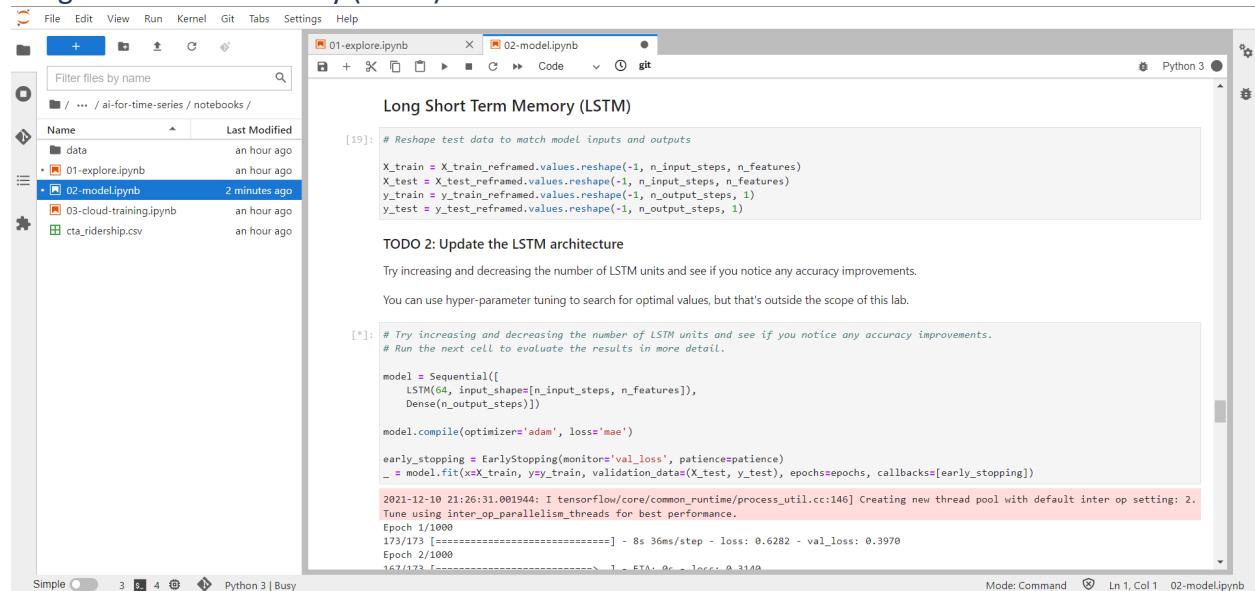
Then Run the cells one by one

## Results:



The screenshot shows a Jupyter Notebook interface with two tabs open: '01-explore.ipynb' and '02-model.ipynb'. The '02-model.ipynb' tab is active, displaying a line plot titled 'total\_rides' versus 'service\_date' from 2002 to 2016. The plot shows a highly volatile time series with values ranging from approximately 0.50 to 2.00. Below the plot, a code cell contains the command `sns.lineplot(data=df_train[target_col])`. Another code cell below it contains the command `df[target_col].sort_values(ascending=False).head()`. A third code cell contains the comment `# TODO: Update the threshold below to remove the outliers` followed by `threshold = -1 # Set this just below the level you are seeing peaks. It will flag any values above it.` and `assert threshold != -1, 'Set the threshold to the minimum that will eliminate outlier(s)'`. At the bottom of the notebook, there is a note: `# Cut down outliers above the threshold (A few outliers are removed)`.

## Long Short Term Memory (LSTM)



The screenshot shows a Jupyter Notebook interface with two tabs open: '01-explore.ipynb' and '02-model.ipynb'. The '02-model.ipynb' tab is active, displaying a section titled 'Long Short Term Memory (LSTM)'. It contains code for reshaping test data: `X_train = X_train_reframed.values.reshape(-1, n_input_steps, n_features)`, `X_test = X_test_reframed.values.reshape(-1, n_input_steps, n_features)`, `y_train = y_train_reframed.values.reshape(-1, n_output_steps, 1)`, and `y_test = y_test_reframed.values.reshape(-1, n_output_steps, 1)`. Below this, a section titled 'TODO 2: Update the LSTM architecture' is present with the instruction 'Try increasing and decreasing the number of LSTM units and see if you notice any accuracy improvements.' and the note 'You can use hyper-parameter tuning to search for optimal values, but that's outside the scope of this lab.' A code cell at the bottom contains the command `# Try increasing and decreasing the number of LSTM units and see if you notice any accuracy improvements.` and `# Run the next cell to evaluate the results in more detail.`. This cell also includes the definition of the 'model' Sequential model with LSTM and Dense layers, compilation with 'adam' optimizer and 'mae' loss, and fitting of the model with early stopping. The output of the cell shows TensorFlow logs and training progress: Epoch 1/1000, 173/173 [=====] - 8s 36ms/step - loss: 0.6282 - val\_loss: 0.3970, Epoch 2/1000, 167/173 [=====] - ETA: 0s - loss: 0.3140.

File Edit View Run Kernel Git Tabs Settings Help

Filter files by name

Name Last Modified

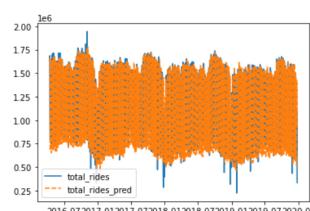
- data an hour ago
- Lstm\_export seconds ago
- 01-explore.ipynb an hour ago
- 02-model.ipynb** 2 minutes ago
- 03-cloud-training.ipynb an hour ago
- cta\_ridership.csv an hour ago

01-explore.ipynb x 02-model.ipynb

```
# The plot will show the R^2 value (0 lowest -> 1 highest) and the MAE (mean absolute error) for the entire prediction window.
# It will also show individual plots for 1 day out, 2 days out, etc. comparing the actual vs the predicted value.

== t+1 ==
R^2: 0.81
MAPE: 0.092
MAE: 80675.552

== t+1 ==
R^2: 0.854
MAPE: 0.08
MAE: 68836.469


service_date
```

```
== t+2 ==
R^2: 0.816
MAPE: 0.091
MAE: 78368.578
```

Simple 3 4 Python 3 | Idle Mode: Command Ln 1, Col 1 02-model.ipynb

## Convolutional Neural Network (CNN)

File Edit View Run Kernel Git Tabs Settings Help

Filter files by name

Name Last Modified

- data an hour ago
- Lstm\_export a minute ago
- 01-explore.ipynb an hour ago
- 02-model.ipynb** seconds ago
- 03-cloud-training.ipynb an hour ago
- cta\_ridership.csv an hour ago

01-explore.ipynb x 02-model.ipynb

### Convolutional Neural Network (CNN)

#### TODO 3: Update the CNN architecture

Try adjusting the # of filters (pattern types) and kernel size (size of the sliding window)

```
[ ]: 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_train, y_train, validation_data=(X_test, y_test), epochs=epochs, callbacks=[early_stopping])

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

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

evaluate(y_pred_cnn)
```

#### Naïve Models

So-called "naive models" can be surprisingly hard to beat. These can serve as a useful benchmark for your model's performance.

#### Random Walk

Mode: Command Ln 1, Col 1 02-model.ipynb

The screenshot shows a Jupyter Notebook interface with two open files: '01-explore.ipynb' and '02-model.ipynb'. The '02-model.ipynb' file is active, displaying code for a CNN model. The code includes predictions on test data, evaluation metrics (R2: 0.767, MAPE: 0.103), and a plot comparing total rides (blue line) and predicted total rides (orange line) from July 2016 to January 2020. The plot shows a highly volatile time series with significant seasonal patterns.

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

evaluate(y_pred_cnn)

==== t+1(1-7) ====
R2: 0.767
MAPE: 0.103
MAE: 96414.134

==== t+1+ ====
R2: 0.81
MAPE: 0.099
MAE: 86185.094
```

le6  
2.00  
1.75  
1.50  
1.25  
1.00  
0.75  
0.50  
0.25  
0.00  
total\_rides  
total\_rides\_pred  
2016-07-2017-07-2018-01-2018-07-2019-01-2019-07-2020-01  
service\_date

Naïve Models

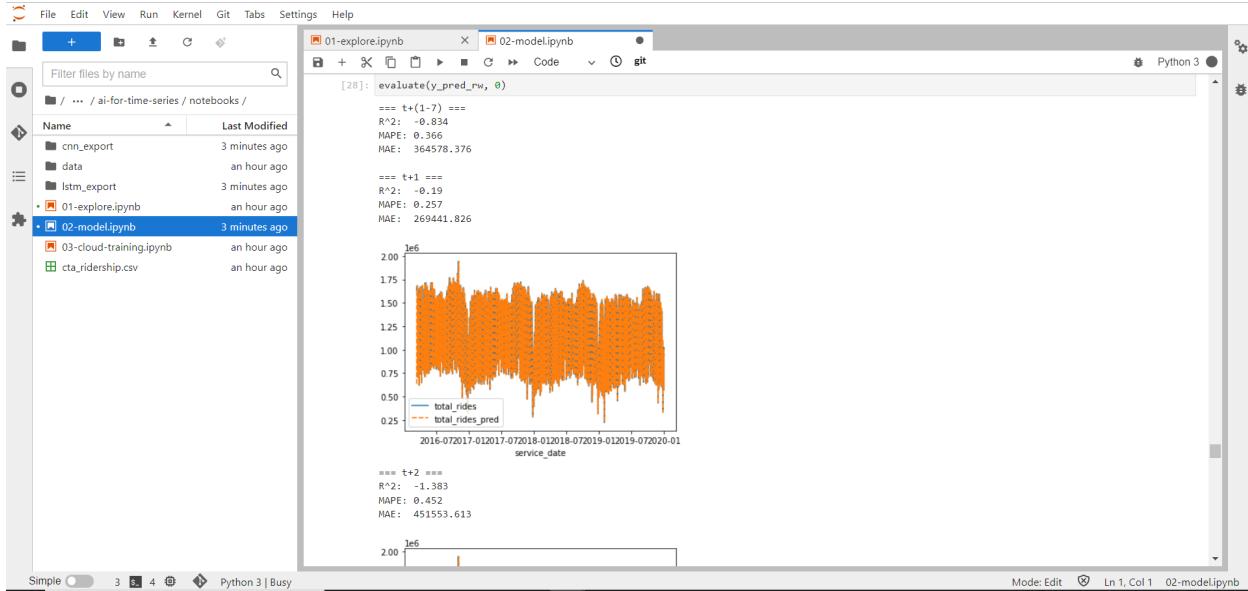
The screenshot shows the Jupyter Notebook interface with two open notebooks: '01-explore.ipynb' and '02-model.ipynb'. The left sidebar displays a file tree under the path '/.../ai-for-time-series/notebooks/'. The notebook '02-model.ipynb' is currently active, showing code for a Random Walk model. The code imports ARIMA from statsmodels.tsa.arima.model and uses it to predict future values based on historical data. It iterates through the range of predictions, fitting the model to the updated history at each step and making forecasts for the next step. The final cell evaluates the predictions.

```
[ ]: from statsmodels.tsa.arima.model import ARIMA

hist = df_train[target_col].copy() # Predict based on historical data. Start with the training data
hist.index.freq = pd.infer_freq(hist.index) # To avoid warnings, explicitly specify the dataframe frequency
n_pred = len(df_test) + 1 # Number of predictions: 1 on the training set; and then 1 for each additional
y_pred_rw = np.empty([n_pred,n_output_steps]) # Create an array to hold predictions, with a number of predictions equal to the test set size,
# and n_output_steps columns

for t in range(n_pred):
    mod = ARIMA(hist, order=(0, 1, 0))
    res = mod.fit()
    pred = res.forecast(n_output_steps)
    y_pred_rw[t] = pred.values
    if t < n_pred - 1:
        hist.loc[df_test.iloc[t].name] = df_test[target_col][t] # Append the latest test data row to the history, for fitting the next model
        hist.index.freq = pd.infer_freq(hist.index)

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



## Seasonal Naïve

The screenshot shows a Jupyter Notebook interface with two tabs open: '01-explore.ipynb' and '02-model.ipynb'. The '02-model.ipynb' tab is active, displaying code. A section titled 'Seasonal Naïve' is highlighted in blue. The code implements a walk-forward approach using SARIMAX to predict future values based on historical data, taking into account seasonal patterns.

```

Seasonal Naïve

Similar to random walk, but instead of using the previous value, you'll use the value from the previous seasonal period. For example, if you're predicting July's forecast, you'll use last July's value, rather than June's value.

[*]: # You will use a walk-forward approach, in which a model is fit on all historical data available.
[*]: # As you progress through the test set to evaluate the model, you will be creating new models for each row in the test set.
[*]: # Each new model will be fit on not only the training data, but on prior test data.

from statsmodels.tsa.statespace.sarimax import SARIMAX

hist = df_train[target_col].copy() # Predict based on historical data. Start with the training data
hist.index.freq = pd.infer_freq(hist.index) # To avoid warnings, explicitly specify the dataframe frequency
n_pred = len(df_test) + 1 # Number of predictions: 1 on the training set; and then 1 for each additional
y_pred_sn = np.empty([n_pred,n_output_steps]) # Create an array to hold predictions, with a number of predictions equal to the test set size,
y_pred_sn[0] = hist[-1].values[0]

for t in range(n_pred):
    mod = SARIMAX(hist, order=(0, 0, 0), seasonal_order=(0, 1, 0, n_seasons))
    res = mod.fit(disp=False)
    pred = res.forecast(n_output_steps)
    y_pred_sn[t] = pred.values
    if t < n_pred - 1:
        hist.loc[df_test.iloc[t].name] = df_test[target_col][t] # Append the latest test data row to the history, for fitting the next model
        hist.index.freq = pd.infer_freq(hist.index)

[*]: evaluate(y_pred_sn, 0)

```

Mode: Command Ln 1, Col 1 02-model.ipynb

The screenshot shows a Jupyter Notebook interface with two tabs open: '01-explore.ipynb' and '02-model.ipynb'. The '02-model.ipynb' tab is active, displaying code in cell [30] that evaluates predictions for time steps t+1 and t+2. The output shows R-squared values of 0.675 and 0.676, MAPE values of 0.11, and MAE values of 108722.34 and 108556.529 respectively. Below the code is a line plot titled 'total\_rides' and 'total\_rides\_pred' from July 2017 to January 2020. The plot shows a highly volatile time series with a blue line for actual data and an orange dashed line for predicted data.

```
[30]: evaluate(y_pred_sn, 0)
      === t+1 ===
      R^2: 0.675
      MAPE: 0.11
      MAE: 108722.34

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

## Statistical Models

The screenshot shows a Jupyter Notebook interface with two tabs open: '01-explore.ipynb' and '02-model.ipynb'. The '02-model.ipynb' tab is active, displaying a section titled 'Statistical Models' with a brief description of exponential smoothing. Below this is a section titled 'Exponential Smoothing' containing code for training a model on historical data and making predictions for the test set. The code uses the `ExponentialSmoothing` function from the statsmodels library, specifying parameters like seasonal periods, trends, and damped trends.

### Statistical Models

You will next implement a popular statistical method for time-series analysis, *exponential smoothing*. Exponential smoothing estimates future data by weighting recent observations more heavily. The [Holt-Winters exponential smoothing](#) method used here uses a "triple" exponential smoothing approach that also considers trend and seasonality.

You can also ensemble classical and machine learning methods for a potentially even more accurate result.

### Exponential Smoothing

```
[31]: with warnings.catch_warnings():
    warnings.simplefilter("ignore", category=ConvergenceWarning)

[*]: use a walk-forward approach, in which a model is fit on all historical data available.
progress through the test set to evaluate the model, you will be creating new models for each row in the test set.
model will be fit on not only the training data, but on prior test data.

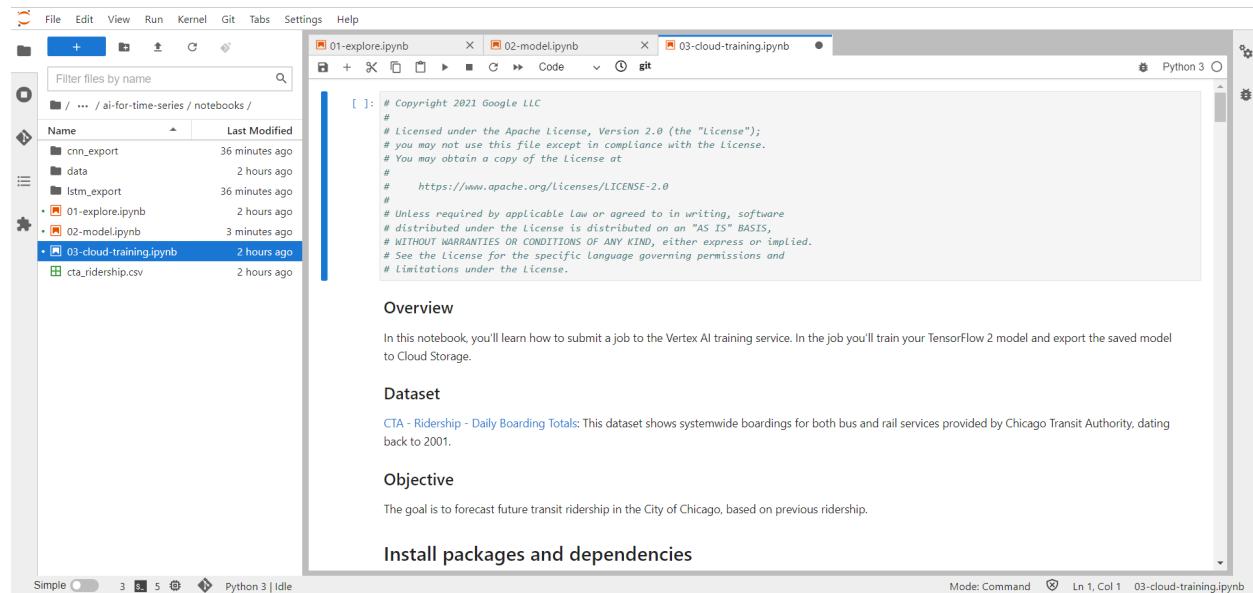
train[target_col].copy() # Predict based on historical data. Start with the training data
freq = pd.infer_freq(hist.index) # To avoid warnings, explicitly specify the dataframe frequency
en(df_test) + 1 # Number of predictions: 1 on the training set; and then 1 for each additional
= np.empty([n_pred,n_output_steps]) # Create an array to hold predictions, with a number of predictions equal to the test set size, each cont
range(n_pred):
ExponentialSmoothing(hist, seasonal_periods=seasons, trends='add', seasonal='add', damped_trend=True, use_boxcox=False, initialization_metho
mod.fit(method='L-BFGS-B') # Use a different minimizer to avoid convergence warnings
res.forecast(n_output_steps)
les[t] = pred.values
n_pred = 1
st.loc[df_test.iloc[t].name] = df_test[target_col][t] # Append the latest test data row to the history, for fitting the next model
st.index.freq = pd.infer_freq(hist.index)
```

## Train and Predict in the Cloud

- Prepare data and models for training in the cloud
- Train your model and monitor the progress of the job with AI Platform Training
- Predict using the model with AI Platform Prediction

### Step 1

In Vertex AI Workbench, navigate to `training-data-analyst/courses/ai-for-time-series/notebooks` and open `03-cloud-training.ipynb`.



The screenshot shows the Vertex AI Workbench interface. On the left, there is a file browser window titled "Filter files by name" showing a directory structure under "... / ai-for-time-series / notebooks /". The files listed are: cnn\_export, data, lstm\_export, 01-explore.ipynb, 02-model.ipynb, 03-cloud-training.ipynb (which is selected), and cta\_ridership.csv. The main workspace shows the content of the 03-cloud-training.ipynb notebook. The code cell contains the following Apache License 2.0 header:

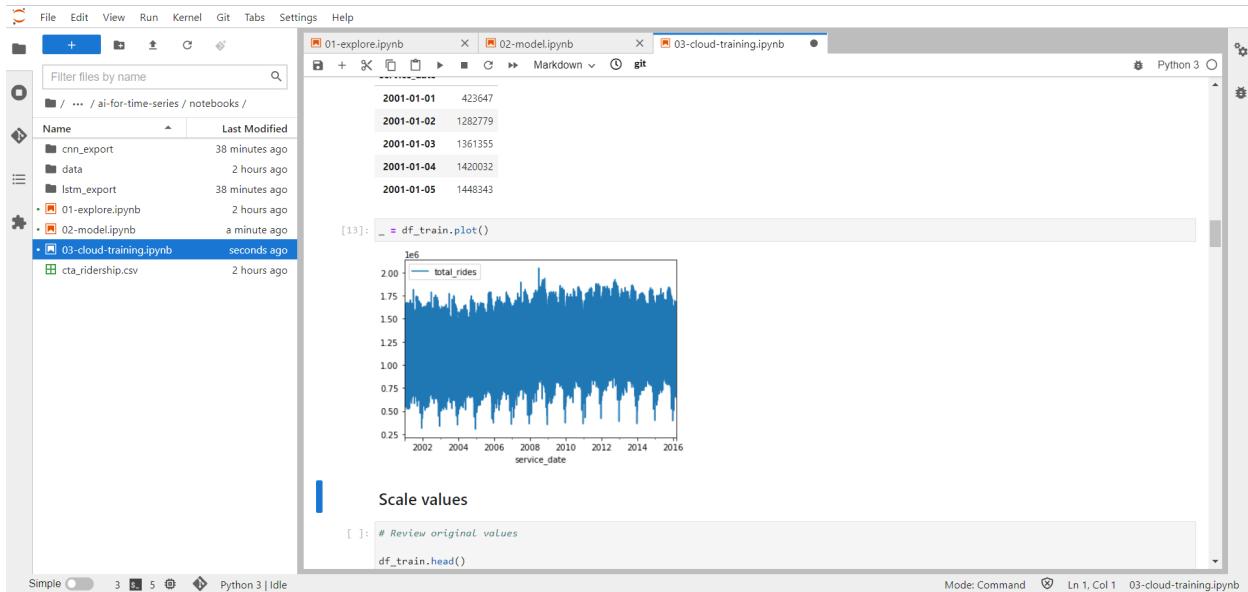
```
[ ]: # Copyright 2021 Google LLC
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
```

The notebook content includes sections like "Overview", "Dataset", "Objective", and "Install packages and dependencies". The status bar at the bottom right indicates "Mode: Command" and "Ln 1, Col 1 03-cloud-training.ipynb".

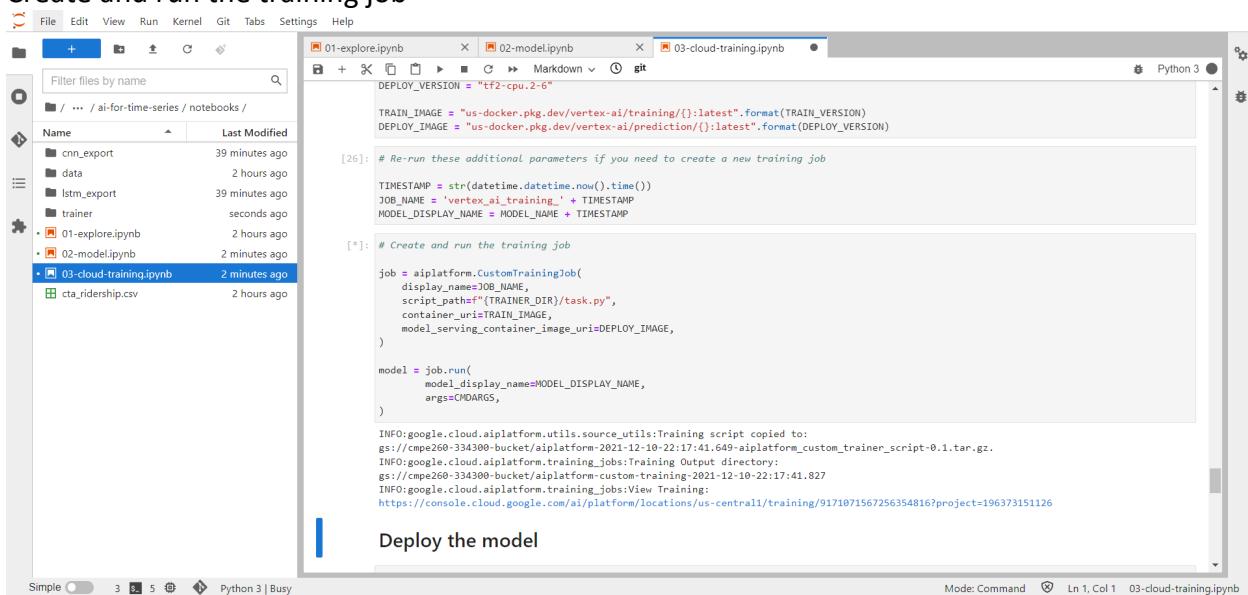
### Step 2

Clear all the cells in the notebook (Edit > Clear All Outputs), change the region, project and bucket settings in one of the first few cells, and then Run the cells one by one.

### Step 3



## Create and run the training job



## Deploy the Model

The screenshot shows a Jupyter Notebook interface with three tabs: 01-explore.ipynb, 02-model.ipynb, and 03-cloud-training.ipynb. The 03-cloud-training.ipynb tab is active. On the left, a file browser lists files including 'cnn\_export', 'data', 'Istm\_export', 'trainer', '01-explore.ipynb', '02-model.ipynb', '03-cloud-training.ipynb' (selected), and 'cta\_ridership.csv'. The main area contains code for deploying a model:

```
[*]: DEPLOYED_NAME = f'{MODEL_NAME}_deployed-{TIMESTAMP}'  
endpoint = model.deploy(  
    deployed_model_display_name=DEPLOYED_NAME,  
    machine_type='n1-standard-4',  
    min_replica_count=1,  
    max_replica_count=1,  
    traffic_split={"0": 100},  
)
```

Below this, a section titled 'Get predictions on deployed model' contains:

```
[ ]: # Get predictions for the first test instance  
raw_predictions = endpoint.predict(instances=x_test.tolist()).predictions[0]  
predicted_values = inverse_scale(np.array([raw_predictions]).round())  
actual_values = inverse_scale(np.array([y_test[0]]))  
  
[ ]: # Print prediction and compare to actual value  
print('Predicted riders:', predicted_values)  
print('Actual riders: ', actual_values)
```

At the bottom, a 'Cleanup' section includes the command:

```
[ ]: delete_training_job = True
```

The screenshot shows a Jupyter Notebook interface with three tabs: 01-explore.ipynb, 02-model.ipynb, and 03-cloud-training.ipynb. The 03-cloud-training.ipynb tab is active. On the left, a file browser lists files including 'task.py', 'x.test.npy', 'x.train.npy', 'y.test.npy', and 'y.train.npy'. The main area contains code for deploying a model, with log output from Google Cloud AI Platform:

```
[28]: DEPLOYED_NAME = f'{MODEL_NAME}_deployed-{TIMESTAMP}'  
endpoint = model.deploy(  
    deployed_model_display_name=DEPLOYED_NAME,  
    machine_type='n1-standard-4',  
    min_replica_count=1,  
    max_replica_count=1,  
    traffic_split={"0": 100},  
)  
  
INFO:google.cloud.aiplatform.training_jobs:Model available at projects/196373151126/locations/us-central1/models/6811346990723497984  
  
Deploy the model
```

The log continues with deployment details:

```
INFO:google.cloud.aiplatform.models:Creating Endpoint  
INFO:google.cloud.aiplatform.models>Create Endpoint backing LRO: projects/196373151126/locations/us-central1/endpoints/6100204860110536704/o  
perations/6542053572056449024  
INFO:google.cloud.aiplatform.models:Endpoint created. Resource name: projects/196373151126/locations/us-central1/endpoints/6100204860110536704  
04  
INFO:google.cloud.aiplatform.models>To use this Endpoint in another session:  
INFO:google.cloud.aiplatform.models:endpoint = aiplatform.Endpoint('projects/196373151126/locations/us-central1/endpoints/6100204860110536704  
4')  
INFO:google.cloud.aiplatform.models:Deploying model to Endpoint : projects/196373151126/locations/us-central1/endpoints/6100204860110536704  
INFO:google.cloud.aiplatform.models:Deploy Endpoint model backing LRO: projects/196373151126/locations/us-central1/endpoints/610020486011053  
6704/operations/5546758054407569408  
INFO:google.cloud.aiplatform.models:Endpoint model deployed. Resource name: projects/196373151126/locations/us-central1/endpoints/6100204860  
110536704
```

Below this, a section titled 'Get predictions on deployed model' contains:

```
[ ]: # Get predictions for the first test instance  
raw_predictions = endpoint.predict(instances=x_test.tolist()).predictions[0]
```

## Prediction Results

The screenshot shows a Jupyter Notebook interface with three tabs: '01-explore.ipynb', '02-model.ipynb', and '03-cloud-training.ipynb'. The '03-cloud-training.ipynb' tab is active, displaying Python code and its output. The code uses a deployed endpoint to predict rider counts from test data. The output shows predicted values and actual values.

```
[29]: # Get predictions for the first test instance
raw_predictions = endpoint.predict(instances=x_test.tolist())
predicted_values = inverse_scale(np.array([raw_predictions])).round()

actual_values = inverse_scale(np.array([y_test[0]]))

[30]: # Print prediction and compare to actual value
print('Predicted riders:', predicted_values)
print('Actual riders:', actual_values)

Predicted riders: [1653388. 1622200. 1638429. 980612. 634831. 1584257. 1636946.]
Actual riders: [[1647321. 1668584. 1687618. 1060043. 786217. 1517370. 1506995.]]

Cleanup
```

```
[ ]: delete_training_job = True
delete_model = True
delete_endpoint = True

# Warning: Setting this to true will delete everything in your bucket
delete_bucket = False

# Delete the training job
job.delete()

[31]: delete_training_job = True
delete_model = True
delete_endpoint = True

# Warning: Setting this to true will delete everything in your bucket
delete_bucket = False

# Delete the training job
job.delete()

# Delete the endpoint
endpoint.delete(forces=True)

# Delete the model
model.delete()

# Warning: uncomment this section only if you want to delete the entire bucket
# if delete_bucket and "BUCKET" in globals():
#     ! gsutil -m rm -r $BUCKET
```

## Cleanup

The screenshot shows a Jupyter Notebook interface with three tabs: '01-explore.ipynb', '02-model.ipynb', and '03-cloud-training.ipynb'. The '03-cloud-training.ipynb' tab is active, displaying Python code and its output. The code performs cleanup operations, including deleting the training job, endpoint, and model, and then deleting the entire bucket if specified.

```
print('Actual riders:', actual_values)

Predicted riders: [1653388. 1622200. 1638429. 980612. 634831. 1584257. 1636946.]
Actual riders: [[1647321. 1668584. 1687618. 1060043. 786217. 1517370. 1506995.]]

Cleanup
```

```
[31]: delete_training_job = True
delete_model = True
delete_endpoint = True

# Warning: Setting this to true will delete everything in your bucket
delete_bucket = False

# Delete the training job
job.delete()

# Delete the endpoint
endpoint.delete(forces=True)

# Delete the model
model.delete()

# Warning: uncomment this section only if you want to delete the entire bucket
# if delete_bucket and "BUCKET" in globals():
#     ! gsutil -m rm -r $BUCKET
```

## Cleanup

From the Workbench UI in your Cloud Console, select the notebook and then select **Stop**:

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](#).

Notebook name	Zone	Auto-upgrade	Environment	Machine type	GPUs	Permission	Last modified
tensorflow-2-3-20211209-130458	OPEN JUPYTERLAB	us-central1-a	TensorFlow:2.3	4 vCPUs, 15 GB RAM	None	Service account	Dec 9, 2021, 3:35:42 PM
<b>tensorflow-2-7-20211208-091022</b>	OPEN JUPYTERLAB	us-central1-a	TensorFlow:2.7	4 vCPUs, 15 GB RAM	None	Service account	Dec 9, 2021, 3:39:09 PM