```
#@title 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.
```

Licensed under the Apache License, Version 2.0 (the "License");

```
!pip install tf-nightly-2.0-preview
```

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
print(tf.__version__)
```

```
2.3.0
```

```
#The final week we'll be using synthetic data
def plot_series(time, series, format="-", start=0, end=None):
    plt.plot(time[start:end], series[start:end], format)
    plt.xlabel("Time")
```

Saved successfully!

```
def trend(time, slope=0):
    return slope * time

def seasonal_pattern(season_time):
    """Just an arbitrary pattern, you can change it if you wish"""
    return np.where(season_time < 0.1,
                    np.cos(season_time * 6 * np.pi),
                    2 / np.exp(9 * season_time))

def seasonality(time, period, amplitude=1, phase=0):
    """Repeats the same pattern at each period"""
    season_time = ((time + phase) % period) / period
    return amplitude * seasonal_pattern(season_time)

def noise(time, noise_level=1, seed=None):
    rnd = np.random.RandomState(seed)
    return rnd.randn(len(time)) * noise_level

time = np.arange(10 * 365 + 1, dtype="float32")
```

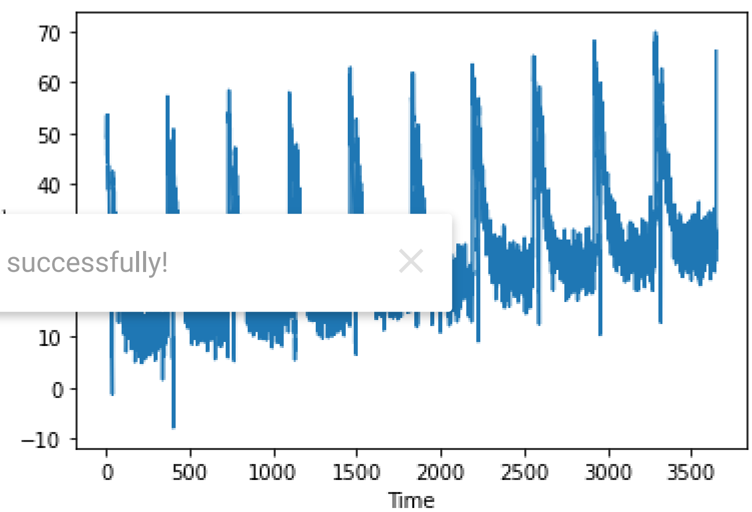
```
baseline = 10
series = trend(time, 0.1)
baseline = 10
amplitude = 40
slope = 0.005
noise_level = 3

# Create the series
series = baseline + trend(time, slope) + seasonality(time, period=365, amplitude=amplitude)
# Update with noise
series += noise(time, noise_level, seed=51)

split_time = 3000
time_train = time[:split_time]
x_train = series[:split_time]
time_valid = time[split_time:]
x_valid = series[split_time:]

window_size = 20
batch_size = 32
shuffle_buffer_size = 1000

plot_series(time, series)
```



Saved successfully!

```
#structure and format our data, shuffle it
#keep only rows that're of length window_size
#break it up into different batches of batch_size
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
  dataset = tf.data.Dataset.from_tensor_slices(series)
  dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)
  dataset = dataset.flat_map(lambda window: window.batch(window_size + 1))
  dataset = dataset.shuffle(shuffle_buffer).map(lambda window: (window[:-1], window[-1]))
  dataset = dataset.batch(batch_size).prefetch(1)
  return dataset


tf.keras.backend.clear_session()
```

```
tf.random.set_seed(51)
np.random.seed(51)

#clear any past variables and avoid conficts with other models
tf.keras.backend.clear_session()
dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
  #lambda function that prepares our windowed data for RNN - LSTM's
  #RNN's require input of 3D, but our windowed data is 2D, this expands the
  #data adding another dimension making it 3D and making it capable of being
  #inputted into a LSTM
  tf.keras.layers.Lambda(lambda x: tf.expand_dims(x, axis=-1), input_shape=[None]),
  #everything except the last RNN layer requires the return_sequences=True
  #cuz that's what passes the data from each memory cell onto the next RNN
  tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32, return_sequences=True)),
  tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
  #windowed data is in scale of 10 multiples, the default activation of Lambda
  #is tanh, (-1<x<1), so in order to bring them to appropriate scaling we multiply
  #by 100
  tf.keras.layers.Lambda(lambda x: x*100.0)
])

#learning rate callback that changes upon each epoch to help us identify the best
#learning rate
lr_schedule = tf.keras.callbacks.LearningRateScheduler(
    lambda epoch: 1e-8 * 10**(epoch / 20))
optimizer = tf.keras.optimizers.SGD(lr=1e-8, momentum=0.9)
#huber loss - a loss function that's less sensitive to outliers
                                    isy, this is an amazing fit for time series
                                    uber(),
              optimizer=optimizer,
              metrics=["mae"])
history = model.fit(dataset, epochs=100, callbacks=[lr_schedule])
```

Saved successfully!

```
94/94 [==============================] - 2s 21ms/step - loss: 6.3137 - mae: 6.7889
Epoch 62/100
94/94 [==============================] - 2s 21ms/step - loss: 6.1412 - mae: 6.6159
Epoch 63/100
94/94 [==============================] - 2s 21ms/step - loss: 5.9703 - mae: 6.4448
Epoch 64/100
94/94 [==============================] - 2s 21ms/step - loss: 5.8274 - mae: 6.3015
Epoch 65/100
94/94 [==============================] - 2s 21ms/step - loss: 5.6743 - mae: 6.1475
Epoch 66/100
94/94 [==============================] - 2s 21ms/step - loss: 5.6147 - mae: 6.0880
Epoch 67/100
94/94 [==============================] - 2s 21ms/step - loss: 5.4587 - mae: 5.9313
Epoch 68/100
94/94 [==============================] - 2s 22ms/step - loss: 5.3111 - mae: 5.7825
Epoch 69/100
94/94 [==============================] - 2s 21ms/step - loss: 5.2701 - mae: 5.7416
Epoch 70/100
94/94 [==============================] - 2s 21ms/step - loss: 5.1421 - mae: 5.6131
Epoch 71/100
94/94 [==============================] - 2s 21ms/step - loss: 5.0837 - mae: 5.5542
Epoch 72/100
94/94 [==============================] - 2s 21ms/step - loss: 5.0200 - mae: 5.4903
Epoch 73/100
94/94 [==============================] - 2s 21ms/step - loss: 5.0438 - mae: 5.5151
Epoch 74/100
94/94 [==============================] - 2s 21ms/step - loss: 4.9762 - mae: 5.4475
Epoch 75/100
94/94 [==============================] - 2s 21ms/step - loss: 4.8398 - mae: 5.3102
Epoch 76/100
94/94 [==============================] - 2s 21ms/step - loss: 4.8549 - mae: 5.3254
Epoch 77/100
94/94 [==============================] - 2s 21ms/step - loss: 4.6858 - mae: 5.1543
```

Saved successfully!

```
=======] - 2s 21ms/step - loss: 4.6779 - mae: 5.1472
94/94 [==============================] - 2s 21ms/step - loss: 4.5024 - mae: 4.9707
Epoch 80/100
94/94 [==============================] - 2s 21ms/step - loss: 4.5554 - mae: 5.0252
Epoch 81/100
94/94 [==============================] - 2s 21ms/step - loss: 4.5170 - mae: 4.9868
Epoch 82/100
94/94 [==============================] - 2s 21ms/step - loss: 4.6028 - mae: 5.0729
Epoch 83/100
94/94 [==============================] - 2s 21ms/step - loss: 4.3870 - mae: 4.8545
Epoch 84/100
94/94 [==============================] - 2s 21ms/step - loss: 4.2792 - mae: 4.7461
Epoch 85/100
94/94 [==============================] - 2s 22ms/step - loss: 4.1954 - mae: 4.6620
Epoch 86/100
94/94 [==============================] - 2s 21ms/step - loss: 4.2248 - mae: 4.6916
Epoch 87/100
94/94 [==============================] - 2s 21ms/step - loss: 4.2740 - mae: 4.7425
Epoch 88/100
94/94 [==============================] - 2s 22ms/step - loss: 4.1223 - mae: 4.5885
Epoch 89/100
94/94 [==============================] - 2s 21ms/step - loss: 4.1143 - mae: 4.5820
Epoch 90/100
```

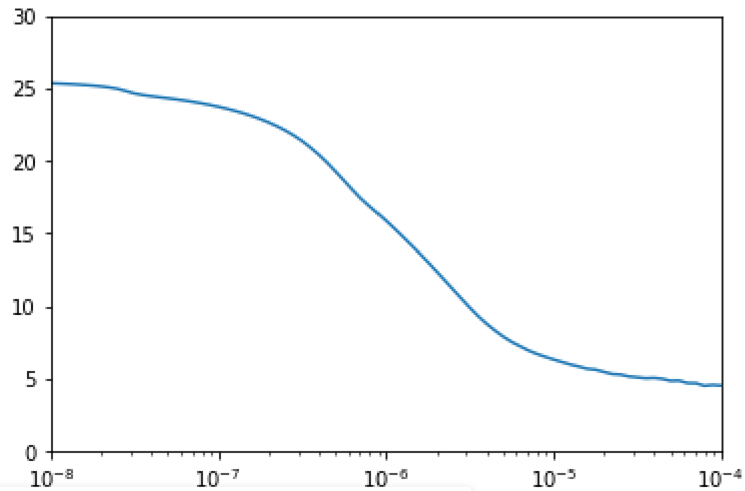
```
94/94 [==============================] - 2s 21ms/step - loss: 4.0710 - mae: 4.5389
Epoch 91/100
94/94 [==============================] - 2s 21ms/step - loss: 4.0176 - mae: 4.4851
Epoch 92/100
94/94 [==============================] - 2s 21ms/step - loss: 4.0602 - mae: 4.5276
Epoch 93/100
94/94 [==============================] - 2s 21ms/step - loss: 4.1645 - mae: 4.6346
Epoch 94/100
94/94 [==============================] - 2s 21ms/step - loss: 3.8520 - mae: 4.3189
Epoch 95/100
94/94 [==============================] - 2s 21ms/step - loss: 3.8656 - mae: 4.3339
Epoch 96/100
94/94 [==============================] - 2s 22ms/step - loss: 3.8771 - mae: 4.3445
Epoch 97/100
94/94 [==============================] - 2s 21ms/step - loss: 3.9997 - mae: 4.4689
Epoch 98/100
94/94 [==============================] - 2s 21ms/step - loss: 3.9991 - mae: 4.4694
Epoch 99/100
94/94 [==============================] - 2s 21ms/step - loss: 3.8375 - mae: 4.3064
Epoch 100/100
94/94 [==============================] - 2s 21ms/step - loss: 3.9458 - mae: 4.4154
```

Saved successfully!

```
#get the best learning rate from this graph
plt.semilogx(history.history["lr"], history.history["loss"])
plt.axis([1e-8, 1e-4, 0, 30])

# FROM THIS PICK A LEARNING RATE
```

```
(1e-08, 0.0001, 0.0, 30.0)
```



Saved successfully!

```
tf.keras.backend.clear_session()
tf.random.set_seed(51)
np.random.seed(51)

tf.keras.backend.clear_session()
dataset = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)

model = tf.keras.models.Sequential([
  tf.keras.layers.Lambda(lambda x: tf.expand_dims(x, axis=-1), input_shape=[None]),
  tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32, return_sequences=True)),
  tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
  tf.keras.layers.Lambda(lambda x: x*100.0)
])

model.compile(loss="mse", optimizer=tf.keras.optimizers.SGD(lr=5e-5, momentum=0.9),metrics=["
history = model.fit(dataset,epochs=500,verbose=1)

# FIND A MODEL AND A LR THAT TRAINS TO AN MAE < 3
```

```
Epoch 472/500
94/94 [==============================] - 2s 21ms/step - loss: 19.4532 - mae: 2.9152
Epoch 473/500
94/94 [==============================] - 2s 21ms/step - loss: 20.4924 - mae: 3.0378
Epoch 474/500
94/94 [==============================] - 2s 21ms/step - loss: 20.2130 - mae: 3.0051
Epoch 475/500
94/94 [==============================] - 2s 21ms/step - loss: 19.5352 - mae: 2.9343
Epoch 476/500
94/94 [==============================] - 2s 21ms/step - loss: 19.5098 - mae: 2.9374
Epoch 477/500
94/94 [==============================] - 2s 21ms/step - loss: 20.2979 - mae: 3.0056
Epoch 478/500
94/94 [==============================] - 2s 21ms/step - loss: 19.5791 - mae: 2.9389
Epoch 479/500
94/94 [==============================] - 2s 21ms/step - loss: 19.6814 - mae: 2.9517
Epoch 480/500
94/94 [==============================] - 2s 21ms/step - loss: 20.1362 - mae: 3.0059
Epoch 481/500
94/94 [==============================] - 2s 21ms/step - loss: 19.5522 - mae: 2.9394
Epoch 482/500
94/94 [==============================] - 2s 21ms/step - loss: 19.7473 - mae: 2.9658
Epoch 483/500
94/94 [==============================] - 2s 21ms/step - loss: 19.6441 - mae: 2.9433
Epoch 484/500
94/94 [==============================] - 2s 21ms/step - loss: 20.3404 - mae: 3.0270
Epoch 485/500
94/94 [==============================] - 2s 21ms/step - loss: 19.4280 - mae: 2.9295
Epoch 486/500
94/94 [==============================] - 2s 21ms/step - loss: 20.2188 - mae: 3.0018
Epoch 487/500
94/94 [==============================] - 2s 21ms/step - loss: 19.6343 - mae: 2.9508
Epoch 488/500
======] - 2s 21ms/step - loss: 19.5411 - mae: 2.9416
=============================] - 2s 21ms/step - loss: 19.7661 - mae: 2.9623
Epoch 490/500
94/94 [==============================] - 2s 21ms/step - loss: 19.7665 - mae: 2.9602
Epoch 491/500
94/94 [==============================] - 2s 21ms/step - loss: 19.8102 - mae: 2.9600
Epoch 492/500
94/94 [==============================] - 2s 21ms/step - loss: 20.1886 - mae: 3.0009
Epoch 493/500
94/94 [==============================] - 2s 21ms/step - loss: 19.5779 - mae: 2.9419
Epoch 494/500
94/94 [==============================] - 2s 21ms/step - loss: 20.0223 - mae: 2.9869
Epoch 495/500
94/94 [==============================] - 2s 21ms/step - loss: 19.7759 - mae: 2.9499
Epoch 496/500
94/94 [==============================] - 2s 21ms/step - loss: 19.5615 - mae: 2.9412
Epoch 497/500
94/94 [==============================] - 2s 21ms/step - loss: 20.0849 - mae: 2.9841
Epoch 498/500
94/94 [==============================] - 2s 21ms/step - loss: 20.0793 - mae: 2.9973
Epoch 499/500
94/94 [==============================] - 2s 21ms/step - loss: 20.2574 - mae: 3.0116
Epoch 500/500
```

Saved successfully!

```
94/94 [==============================] - 2s 21ms/step - loss: 19.5709 - mae: 2.9369
```

Saved successfully!