In [1]:
```
"""
Data (Daily & Minute): Binance API-Will need Binance API keys to be able to pull the data.
Binance API Documentation: https://binance-docs.github.io/apidocs/spot/en/#introduction

"""
```

Out[1]:
```
'\nData (Daily & Minute): Binance API-Will need Binance API keys to be able to pull the data. \nBinance API Documentation: https://binance-docs.github.io/ap
idocs/spot/en/#introduction\n\n'
```

In [2]:
```
# J.Guanzon Comment-Imports needed to run this file
from binance import Client, ThreadedWebsocketManager, ThreadedDepthCacheManager
import pandas as pd
import mplfinance as mpl
import mplfinance as mpf
import os
import json
import requests
from keras.models import Sequential
from keras.layers import Activation, Dense, Dropout, LSTM
import matplotlib.pyplot as plt
import numpy as np
from pathlib import Path
import seaborn as sns
from sklearn.metrics import mean_absolute_error
%matplotlib inline
```

In [3]:
```
# Pull API keys from .env file
api_key = os.environ.get("api_key")
api_secret = os.environ.get("api_secret")
```

In [4]:
```
client = Client(api_key, api_secret)
```

In [5]:
```
# J.Guanzon Comment: Gather tickers for all
tickers = client.get_all_tickers()
```

In [6]:
```
ticker_df = pd.DataFrame(tickers)
```

In [7]:
```
ticker_df.set_index('symbol', inplace=True)
ticker_df
```

Out[7]:

	price
symbol	
ETHBTC	0.06133600
LTCBTC	0.00306500
BNBBTC	0.00716100
NEOBTC	0.00076900
QTUMETH	0.00360000
...	...
SHIBAUD	0.00004143
RAREBTC	0.00004935
RAREBNB	0.00688700
RAREBUSD	2.79900000
RAREUSDT	2.79200000

1695 rows × 1 columns

In [8]:
```
"""
Ability to save csv file of all tickers.
Allows the user to see what types of cryptocurrencies are out there.
For now, we will only focus on Bitcoin
"""
```

Out[8]:
```
' \nAbility to save csv file of all tickers.\nAllows the user to see what types of cryptocurrencies are out there.\nFor now, we will only focus on Bitcoin
\n'
```

In [9]:
```
ticker_df.to_csv("Resources/binance_tickers.csv")
```

In [10]:
```
display(float(ticker_df.loc['BTCUSDT']['price']))
```

```
56739.99
```

In [11]:
```
depth = client.get_order_book(symbol='BTCUSDT')
```

In [12]:
```
depth_df = pd.DataFrame(depth['asks'])
depth_df.columns = ['Price', 'Volume']
depth_df.head()
```

Out[12]:

	Price	Volume
0	56740.00000000	8.57157000
1	56742.11000000	0.11966000
2	56745.36000000	0.02820000
3	56747.05000000	0.05281000
4	56747.37000000	0.20750000

In [13]:

```
# J.Guanzon Comment: Pulling historical daily data
btc_daily_data = client.get_historical_klines('BTCUSDT', Client.KLINE_INTERVAL_1DAY, '1 Jan 2021')
```

In [14]:

```
btc_daily_df = pd.DataFrame(btc_daily_data)
btc_daily_df.columns = ['Open Time', 'Open', 'High', 'Low', 'Close', 'Volume', 'Close Time', 'Quote Asset Volume',
                    'Number of Trades', 'TB Base Volume', 'TB Quote Volume', 'Ignore']
```

In [15]:

```
btc_daily_df['Open Time'] = pd.to_datetime(btc_daily_df['Open Time']/1000, unit='s')
btc_daily_df['Close Time'] = pd.to_datetime(btc_daily_df['Close Time']/1000, unit='s')
```

In [16]:

```
numeric_columns = ['Open', 'High', 'Low', 'Close', 'Volume', 'Quote Asset Volume', 'TB Base Volume', 'TB Quote Volume']
btc_daily_df[numeric_columns] = btc_daily_df[numeric_columns].apply(pd.to_numeric, axis=1)
```

In [17]:

```
btc_ohlcv_daily = btc_daily_df.iloc[:,0:6]
btc_ohlcv_daily = btc_ohlcv_daily.set_index('Open Time')
btc_ohlcv_daily
```

Out[17]:

	Open	High	Low	Close	Volume
Open Time					
2021-01-01	28923.63	29600.00	28624.57	29331.69	54182.925011
2021-01-02	29331.70	33300.00	28946.53	32178.33	129993.873362
2021-01-03	32176.45	34778.11	31962.99	33000.05	120957.566750
2021-01-04	33000.05	33600.00	28130.00	31988.71	140899.885690
2021-01-05	31989.75	34360.00	29900.00	33949.53	116049.997038
...	...	...	...	...	...
2021-10-08	53785.22	56100.00	53617.61	53951.43	46160.257850
2021-10-09	53955.67	55489.00	53661.67	54949.72	55177.080130
2021-10-10	54949.72	56561.31	54080.00	54659.00	89237.836128
2021-10-11	54659.01	57839.04	54415.06	57471.35	52933.165751
2021-10-12	57471.35	57471.35	56588.00	56740.00	5679.002750

285 rows × 5 columns

In [18]:

```
btc_ohlcv_daily.to_csv("Resources/daily_btc_ohclv_2021.csv")
```

In [19]:

```
# J.Guanzon Comment: Pulling historical minute data
historical_minute = client.get_historical_klines('BTCUSDC', Client.KLINE_INTERVAL_1MINUTE, '5 day ago UTC')
```

In [20]:

```
hist_min = pd.DataFrame(historical_minute)
```

In [21]:

```
hist_min.columns = ['Open Time', 'Open', 'High', 'Low', 'Close', 'Volume', 'Close Time', 'Quote Asset Volume',
                    'Number of Trades', 'TB Base Volume', 'TB Quote Volume', 'Ignore']
```

In [22]:

```
hist_min['Open Time'] = pd.to_datetime(hist_min['Open Time']/1000, unit='s')
hist_min['Close Time'] = pd.to_datetime(hist_min['Close Time']/1000, unit='s')
```

In [23]:

```
numeric_columns = ['Open', 'High', 'Low', 'Close', 'Volume', 'Quote Asset Volume', 'TB Base Volume', 'TB Quote Volume']
hist_min[numeric_columns] = hist_min[numeric_columns].apply(pd.to_numeric, axis=1)
```

In [24]:

```
btc_ohlcv_minute = hist_min.iloc[:,0:6]
btc_ohlcv_minute = btc_ohlcv_minute.set_index('Open Time')
btc_ohlcv_minute
```

Out[24]:

	Open	High	Low	Close	Volume
Open Time					
2021-10-07 02:57:00	55117.50	55118.17	55067.64	55067.64	1.78542
2021-10-07 02:58:00	55082.99	55113.22	55070.03	55070.50	1.98333
2021-10-07 02:59:00	55099.85	55126.99	55059.01	55113.87	2.68683
2021-10-07 03:00:00	55113.87	55185.82	55102.77	55138.53	5.07494

	Open	High	Low	Close	Volume
Open Time					
2021-10-07 03:01:00	55135.03	55187.98	55128.39	55167.09	9.04292
...	...	...	...	...	...
2021-10-12 02:53:00	56822.42	56822.42	56701.97	56743.31	1.74763
2021-10-12 02:54:00	56751.64	56751.64	56714.40	56724.88	0.69013
2021-10-12 02:55:00	56723.72	56723.72	56688.88	56688.88	0.61865
2021-10-12 02:56:00	56714.84	56753.33	56714.84	56753.33	0.37080
2021-10-12 02:57:00	56767.04	56767.04	56757.33	56757.33	0.03692

7201 rows × 5 columns

In [25]:

```
btc_ohlcv_minute.to_csv("Resources/minute_btc_ohclv_2021.csv")
```

In [26]:

```
"""
Next, we will be using the daily data for our Recurrent Neural Network. We are using Recurrent Neural Network.

"""
```

Out[26]: '\nNext, we will be using the daily data for our Recurrent Neural Network. We are using Recurrent Neural Network.\n\n'

In [27]:

```
btc_df = pd.read_csv(Path("Resources/daily_btc_ohclv_2021.csv"),
                     index_col= "Open Time")
target_col = 'Close'
```

In [28]:

```
btc_df.head()
```

Out[28]:

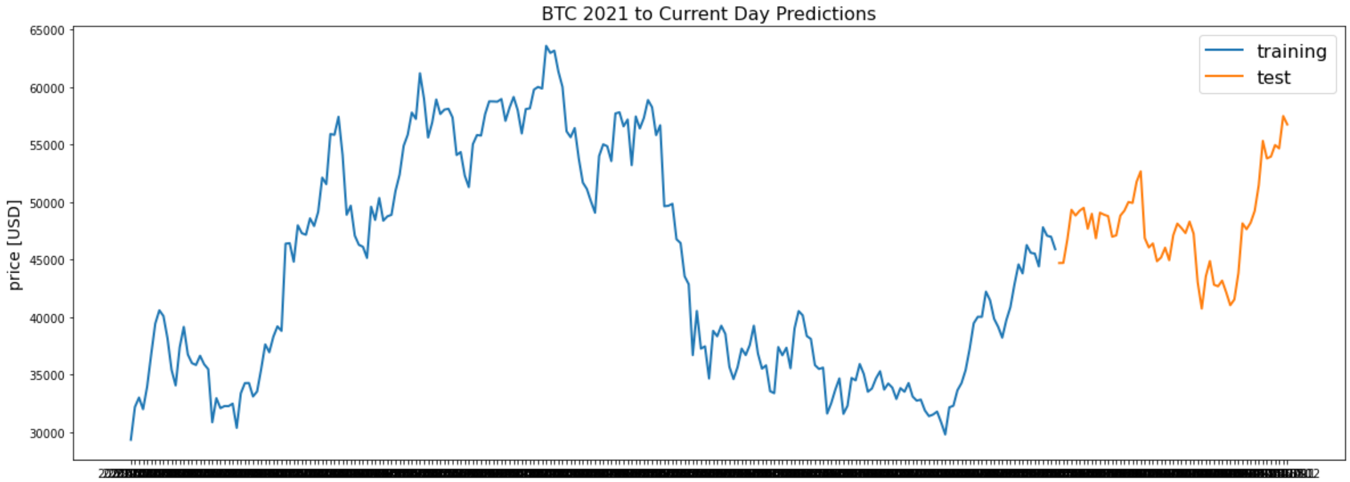
	Open	High	Low	Close	Volume
Open Time					
2021-01-01	28923.63	29600.00	28624.57	29331.69	54182.925011
2021-01-02	29331.70	33300.00	28946.53	32178.33	129993.873362
2021-01-03	32176.45	34778.11	31962.99	33000.05	120957.566750
2021-01-04	33000.05	33600.00	28130.00	31988.71	140899.885690
2021-01-05	31989.75	34360.00	29900.00	33949.53	116049.997038

In [29]:

```
# J.Guanzon Comment: Using an 80/20 split for our training data and testing data. Testing 2 other testing sizes to see if there are any differnces in accura

def train_test_split(btc_df, test_size=0.2):
  split_row = len(btc_df) - int(test_size * len(btc_df))
  train_data = btc_df.iloc[:split_row]
  test_data = btc_df.iloc[split_row:]
  return train_data, test_data

train, test = train_test_split(btc_df, test_size=0.2)

# def train_test_split(btc_df, test_size=0.3):
#   split_row = len(btc_df) - int(test_size * len(btc_df))
#   train_data = btc_df.iloc[:split_row]
#   test_data = btc_df.iloc[split_row:]
#   return train_data, test_data

# train, test = train_test_split(btc_df, test_size=0.3)

# def train_test_split(btc_df, test_size=0.1):
#   split_row = len(btc_df) - int(test_size * len(btc_df))
#   train_data = btc_df.iloc[:split_row]
#   test_data = btc_df.iloc[split_row:]
#   return train_data, test_data

# train, test = train_test_split(btc_df, test_size=0.1)
```

In [30]:

```
def line_plot(line1, line2, label1=None, label2=None, title='', lw=2):
    fig, ax = plt.subplots(1, figsize=(20, 7))
    ax.plot(line1, label=label1, linewidth=lw)
    ax.plot(line2, label=label2, linewidth=lw)
    ax.set_ylabel('price [USD]', fontsize=14)
    ax.set_title(title, fontsize=16)
    ax.legend(loc='best', fontsize=16)

line_plot(train[target_col], test[target_col], 'training', 'test', title='BTC 2021 to Current Day Predictions')
```



In [31]:
```
"""
Next, we have to prep the data for RNN by normalizing the numeric columns in the dataset to a common scale, without distorting differences in the range of v
"""
```

Out[31]:
```
'\nNext, we have to prep the data for RNN by normalizing the numeric columns in the dataset to a common scale, without distorting differences in the range o
f values.\n'
```

In [32]:
```python
def normalise_zero_base(df):
    return df / df.iloc[0] - 1

def normalise_min_max(df):
    return (df - df.min()) / (data.max() - df.min())
```

In [33]:
```python
def extract_window_data(btc_df, window_len=5, zero_base=True):
    window_data = []
    for idx in range(len(btc_df) - window_len):
        tmp = btc_df[idx: (idx + window_len)].copy()
        if zero_base:
            tmp = normalise_zero_base(tmp)
        window_data.append(tmp.values)
    return np.array(window_data)
```

In [34]:
```python
def prepare_data(btc_df, target_col, window_len=10, zero_base=True, test_size=0.2):
    train_data, test_data = train_test_split(btc_df, test_size=test_size)
    X_train = extract_window_data(train_data, window_len, zero_base)
    X_test = extract_window_data(test_data, window_len, zero_base)
    y_train = train_data[target_col][window_len:].values
    y_test = test_data[target_col][window_len:].values
    if zero_base:
        y_train = y_train / train_data[target_col][:-window_len].values - 1
        y_test = y_test / test_data[target_col][:-window_len].values - 1

    return train_data, test_data, X_train, X_test, y_train, y_test

# def prepare_data(btc_df, target_col, window_len=10, zero_base=True, test_size=0.3):
#     train_data, test_data = train_test_split(btc_df, test_size=test_size)
#     X_train = extract_window_data(train_data, window_len, zero_base)
#     X_test = extract_window_data(test_data, window_len, zero_base)
#     y_train = train_data[target_col][window_len:].values
#     y_test = test_data[target_col][window_len:].values
#     if zero_base:
#         y_train = y_train / train_data[target_col][:-window_len].values - 1
#         y_test = y_test / test_data[target_col][:-window_len].values - 1

#     return train_data, test_data, X_train, X_test, y_train, y_test

# def prepare_data(btc_df, target_col, window_len=10, zero_base=True, test_size=0.1):
#     train_data, test_data = train_test_split(btc_df, test_size=test_size)
#     X_train = extract_window_data(train_data, window_len, zero_base)
#     X_test = extract_window_data(test_data, window_len, zero_base)
#     y_train = train_data[target_col][window_len:].values
#     y_test = test_data[target_col][window_len:].values
#     if zero_base:
#         y_train = y_train / train_data[target_col][:-window_len].values - 1
#         y_test = y_test / test_data[target_col][:-window_len].values - 1

#     return train_data, test_data, X_train, X_test, y_train, y_test
```

In [35]:
```python
def build_lstm_model(input_data, output_size, neurons=100, activ_func='linear', dropout=0.2, loss='mse', optimizer='adam'):
    model = Sequential()
    model.add(LSTM(neurons, input_shape=(input_data.shape[1], input_data.shape[2])))
    model.add(Dropout(dropout))
    model.add(Dense(units=output_size))
    model.add(Activation(activ_func))
    model.compile(loss=loss, optimizer=optimizer)
    return model
```

In [36]:
```python
np.random.seed(42)
```

```
window_len = 5
test_size = 0.2
zero_base = True
lstm_neurons = 100
epochs = 100
batch_size = 32
loss = 'mse'
dropout = 0.2
optimizer = 'adam'
```

In [37]:

```
train, test, X_train, X_test, y_train, y_test = prepare_data(
    btc_df, target_col, window_len=window_len, zero_base=zero_base, test_size=test_size)
model = build_lstm_model(
    X_train, output_size=1, neurons=lstm_neurons, dropout=dropout, loss=loss,
    optimizer=optimizer)
history = model.fit(
    X_train, y_train, epochs=epochs, batch_size=batch_size, verbose=1, shuffle=True)
```

```
Epoch 1/100
7/7 [==============================] - 1s 4ms/step - loss: 0.0089
Epoch 2/100
7/7 [==============================] - 0s 5ms/step - loss: 0.0060
Epoch 3/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0054
Epoch 4/100
7/7 [==============================] - 0s 5ms/step - loss: 0.0048
Epoch 5/100
7/7 [==============================] - 0s 5ms/step - loss: 0.0044
Epoch 6/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0044
Epoch 7/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0043
Epoch 8/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0038
Epoch 9/100
7/7 [==============================] - 0s 3ms/step - loss: 0.0039
Epoch 10/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0036
Epoch 11/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0036
Epoch 12/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0032
Epoch 13/100
7/7 [==============================] - 0s 3ms/step - loss: 0.0033
Epoch 14/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0031
Epoch 15/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0032
Epoch 16/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0029
Epoch 17/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0030
Epoch 18/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0030
Epoch 19/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0030
Epoch 20/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0031
Epoch 21/100
7/7 [==============================] - 0s 3ms/step - loss: 0.0029
Epoch 22/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0028
Epoch 23/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0031
Epoch 24/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0027
Epoch 25/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0028
Epoch 26/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0027
Epoch 27/100
7/7 [==============================] - 0s 3ms/step - loss: 0.0026
Epoch 28/100
7/7 [==============================] - 0s 3ms/step - loss: 0.0028
Epoch 29/100
7/7 [==============================] - 0s 3ms/step - loss: 0.0028
Epoch 30/100
7/7 [==============================] - 0s 3ms/step - loss: 0.0026
Epoch 31/100
7/7 [==============================] - 0s 3ms/step - loss: 0.0026
Epoch 32/100
7/7 [==============================] - 0s 3ms/step - loss: 0.0026
Epoch 33/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0024
Epoch 34/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0027
Epoch 35/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0027
Epoch 36/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0025
Epoch 37/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0025
Epoch 38/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0027
Epoch 39/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0026
Epoch 40/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0025
Epoch 41/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0026
```

```
Epoch 42/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0025
Epoch 43/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0026
Epoch 44/100
7/7 [==============================] - 0s 3ms/step - loss: 0.0023
Epoch 45/100
7/7 [==============================] - 0s 3ms/step - loss: 0.0026
Epoch 46/100
7/7 [==============================] - 0s 3ms/step - loss: 0.0026
Epoch 47/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0025
Epoch 48/100
7/7 [==============================] - 0s 3ms/step - loss: 0.0025
Epoch 49/100
7/7 [==============================] - 0s 3ms/step - loss: 0.0024
Epoch 50/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0024
Epoch 51/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0024
Epoch 52/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0025
Epoch 53/100
7/7 [==============================] - 0s 3ms/step - loss: 0.0023
Epoch 54/100
7/7 [==============================] - 0s 3ms/step - loss: 0.0024
Epoch 55/100
7/7 [==============================] - 0s 3ms/step - loss: 0.0024
Epoch 56/100
7/7 [==============================] - 0s 3ms/step - loss: 0.0024
Epoch 57/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0023
Epoch 58/100
7/7 [==============================] - 0s 3ms/step - loss: 0.0025
Epoch 59/100
7/7 [==============================] - 0s 3ms/step - loss: 0.0022
Epoch 60/100
7/7 [==============================] - 0s 5ms/step - loss: 0.0023
Epoch 61/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0024
Epoch 62/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0023
Epoch 63/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0024
Epoch 64/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0024
Epoch 65/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0022
Epoch 66/100
7/7 [==============================] - 0s 3ms/step - loss: 0.0023
Epoch 67/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0023
Epoch 68/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0024
Epoch 69/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0022
Epoch 70/100
7/7 [==============================] - 0s 5ms/step - loss: 0.0024
Epoch 71/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0024
Epoch 72/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0022
Epoch 73/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0022
Epoch 74/100
7/7 [==============================] - 0s 5ms/step - loss: 0.0022
Epoch 75/100
7/7 [==============================] - 0s 5ms/step - loss: 0.0022
Epoch 76/100
7/7 [==============================] - 0s 5ms/step - loss: 0.0022
Epoch 77/100
7/7 [==============================] - 0s 5ms/step - loss: 0.0023
Epoch 78/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0022
Epoch 79/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0023
Epoch 80/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0022
Epoch 81/100
7/7 [==============================] - 0s 5ms/step - loss: 0.0023
Epoch 82/100
7/7 [==============================] - 0s 5ms/step - loss: 0.0022
Epoch 83/100
7/7 [==============================] - 0s 5ms/step - loss: 0.0022
Epoch 84/100
7/7 [==============================] - 0s 5ms/step - loss: 0.0023
Epoch 85/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0022
Epoch 86/100
7/7 [==============================] - 0s 5ms/step - loss: 0.0022
Epoch 87/100
7/7 [==============================] - 0s 5ms/step - loss: 0.0023
Epoch 88/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0022
Epoch 89/100
7/7 [==============================] - 0s 5ms/step - loss: 0.0024
Epoch 90/100
7/7 [==============================] - 0s 5ms/step - loss: 0.0021
Epoch 91/100
7/7 [==============================] - 0s 3ms/step - loss: 0.0021
Epoch 92/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0021
```

```
Epoch 93/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0022
Epoch 94/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0022
Epoch 95/100
7/7 [==============================] - 0s 3ms/step - loss: 0.0022
Epoch 96/100
7/7 [==============================] - 0s 3ms/step - loss: 0.0021
Epoch 97/100
7/7 [==============================] - 0s 5ms/step - loss: 0.0021
Epoch 98/100
7/7 [==============================] - 0s 4ms/step - loss: 0.0022
Epoch 99/100
7/7 [==============================] - 0s 5ms/step - loss: 0.0021
Epoch 100/100
7/7 [==============================] - 0s 5ms/step - loss: 0.0021
```

In [38]:

```
targets = test[target_col][window_len:]
preds = model.predict(X_test).squeeze()
mean_absolute_error(preds, y_test)
```

Out[38]:

```
0.029340654388983985
```

In [39]:

```
# Plotting predictions against the actual.
preds = test[target_col].values[:-window_len] * (preds + 1)
preds = pd.Series(index=targets.index, data=preds)
line_plot(targets, preds, 'actual', 'prediction', lw=3)
```