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/apidocs/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 matplotlib.dates as mdates
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	0.06451300
LTCBTC	0.00310800
BNBBTC	0.00788500
NEOBTC	0.00076600
QTUMETH	0.00345400
...	...
SHIBAUD	0.00003560
RAREBTC	0.00003495
RAREBNB	0.00443600
RAREBUSD	2.08000000
RAREUSDT	2.08000000

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']))
```

```
59462.93
```

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	59463.01000000	0.02000000
1	59464.00000000	0.04000000
2	59464.01000000	0.02000000
3	59464.20000000	0.00066000
4	59465.00000000	0.50000000

In [13]:
```
"""
Pulling historical daily data
"""
```

Out[13]: '\nPulling historical daily data\n'

In [14]:
```
btc_daily_data = client.get_historical_klines('BTCUSDT', Client.KLINE_INTERVAL_1DAY, '1 Jan 2020')
```

In [15]:
```
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 [16]:
```
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 [17]:
```
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 [18]:
```
btc_ohlcv_daily = btc_daily_df.iloc[:,0:6]
btc_ohlcv_daily = btc_ohlcv_daily.set_index('Open Time')
btc_ohlcv_daily
```

Out[18]:

Open Time	Open	High	Low	Close	Volume
2020-01-01	7195.24	7255.00	7175.15	7200.85	16792.388165
2020-01-02	7200.77	7212.50	6924.74	6965.71	31951.483932
2020-01-03	6965.49	7405.00	6871.04	7344.96	68428.500451
2020-01-04	7345.00	7404.00	7272.21	7354.11	29987.974977
2020-01-05	7354.19	7495.00	7318.00	7358.75	38331.085604
...	...	...	...	...	...
2021-10-11	54659.01	57839.04	54415.06	57471.35	52933.165751
2021-10-12	57471.35	57680.00	53879.00	55996.93	53471.285500
2021-10-13	55996.91	57777.00	54167.19	57367.00	55808.444920
2021-10-14	57370.83	58532.54	56818.05	57347.94	43053.336781
2021-10-15	57347.94	59998.00	56850.00	59465.00	22326.228281

654 rows × 5 columns

In [19]:
```
btc_ohlcv_daily.to_csv("Resources/daily_btc_ohclv_2021.csv")
```

In [20]:
```
"""
Pulling historical minute data
"""
```

Out[20]: '\nPulling historical minute data \n'

In [21]:
```
historical_minute = client.get_historical_klines('BTCUSDC', Client.KLINE_INTERVAL_1MINUTE, '5 day ago UTC')
```

In [22]:
```
hist_min = pd.DataFrame(historical_minute)
```

In [23]:
```
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 [24]:
```
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 [25]:
```
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 [26]:

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

Out[26]:

	Open	High	Low	Close	Volume
Open Time					
2021-10-10 05:53:00	55732.23	55772.11	55732.23	55772.11	0.17013
2021-10-10 05:54:00	55767.97	55782.86	55723.42	55723.42	0.41344
2021-10-10 05:55:00	55734.54	55740.82	55698.48	55698.48	0.16777
2021-10-10 05:56:00	55703.21	55703.21	55669.43	55672.82	0.18831
2021-10-10 05:57:00	55705.32	55705.32	55656.61	55659.76	0.09099
...	...	...	...	...	...
2021-10-15 05:48:00	59627.99	59633.80	59600.00	59616.07	0.43744
2021-10-15 05:49:00	59633.46	59644.87	59591.56	59591.56	0.53721
2021-10-15 05:50:00	59588.07	59588.59	59500.00	59526.73	3.75097
2021-10-15 05:51:00	59530.88	59563.64	59488.92	59488.92	0.77968
2021-10-15 05:52:00	59500.05	59500.05	59458.22	59465.68	0.42926

7200 rows × 5 columns

In [27]:

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

In [28]:

```
"""
Next, we will be using the daily data for our Recurrent Neural Network. We are using Recurrent Neural Network for its use of time series and sequential data
RNN specializes in using information from prior inputs and uses it to influence current inputs and outputs, and the cycle repeats.
"""
```

Out[28]:

```
'\nNext, we will be using the daily data for our Recurrent Neural Network. We are using Recurrent Neural Network for its use of time series and sequential d
ata. \nRNN specializes in using information from prior inputs and uses it to influence current inputs and outputs, and the cycle repeats. \n'
```

In [29]:

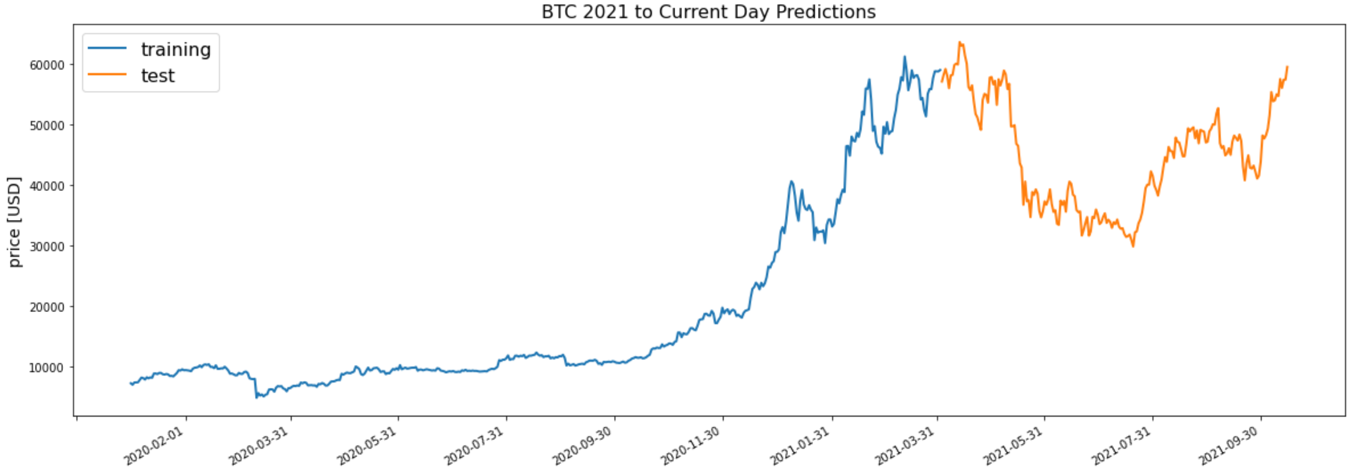
```
btc_df = pd.read_csv(Path("Resources/daily_btc_ohclv_2021.csv"),
                     index_col= "Open Time")
target_col = 'Close'
```

In [30]:

```
# 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 [31]:

```
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)
    fmt_bimonthly = mdates.MonthLocator(interval=2)
    ax.xaxis.set_major_locator(fmt_bimonthly)
    ax.set_title(title, fontsize=16)
    fig.autofmt_xdate()
    ax.legend(loc='best', fontsize=16)

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



In [32]:
```
"""
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[32]:
```
'\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 [33]:
```
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 [34]:
```
def extract_window_data(btc_df, window_len=10, 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 [35]:
```
# J.Guanzon Comment: We want to use the data from Jan-Jun 2021 and use the rest of the data to train and predict the rest of the data.
X_train= btc_df[:"2021-06-01"]
X_test = btc_df["2021-06-01":]
y_train = btc_df.loc[:"2021-06-01",target_col]
y_test = btc_df.loc["2021-06-01":,target_col]
```

In [36]:
```
# 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 [37]:
```
def build_lstm_model(input_data, output_size, neurons=100, activ_func='relu', dropout=0.2, loss='mse', optimizer='adam'):
    model = Sequential()
    stm= LSTM(neurons, input_shape=(input_data.shape[1], input_data.shape[2]))
    model.add(stm)
    model.add(Dropout(dropout))
```

```
    model.add(Dense(units=output_size))
    model.add(Activation(activ_func))
    model.compile(loss=loss, optimizer=optimizer)
    return model
```

In [38]:

```
np.random.seed(46)
window_len = 10
test_size = 0.3
zero_base = True
lstm_neurons = 100
epochs = 50
batch_size = 32
loss = 'mse'
dropout = 0.2
optimizer = 'adam'
```

In [39]:

```
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 1/50
14/14 [==============================] - 3s 13ms/step - loss: 0.0143
Epoch 2/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0088
Epoch 3/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0083
Epoch 4/50
14/14 [==============================] - 0s 11ms/step - loss: 0.0078
Epoch 5/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0075
Epoch 6/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0073
Epoch 7/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0071
Epoch 8/50
14/14 [==============================] - 0s 11ms/step - loss: 0.0069
Epoch 9/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0069
Epoch 10/50
14/14 [==============================] - 0s 11ms/step - loss: 0.0068
Epoch 11/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0067
Epoch 12/50
14/14 [==============================] - 0s 11ms/step - loss: 0.0064
Epoch 13/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0065
Epoch 14/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0066
Epoch 15/50
14/14 [==============================] - 0s 13ms/step - loss: 0.0064
Epoch 16/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0062
Epoch 17/50
14/14 [==============================] - 0s 11ms/step - loss: 0.0062
Epoch 18/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0064
Epoch 19/50
14/14 [==============================] - 0s 11ms/step - loss: 0.0064
Epoch 20/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0063
Epoch 21/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0063
Epoch 22/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0063
Epoch 23/50
14/14 [==============================] - 0s 11ms/step - loss: 0.0063
Epoch 24/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0064
Epoch 25/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0063
Epoch 26/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0062
Epoch 27/50
14/14 [==============================] - 0s 13ms/step - loss: 0.0062
Epoch 28/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0061
Epoch 29/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0062
Epoch 30/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0061
Epoch 31/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0060
Epoch 32/50
14/14 [==============================] - 0s 11ms/step - loss: 0.0062
Epoch 33/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0062
Epoch 34/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0061
Epoch 35/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0060
Epoch 36/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0059
Epoch 37/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0060
Epoch 38/50
```

```
14/14 [==============================] - 0s 12ms/step - loss: 0.0060
Epoch 39/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0059
Epoch 40/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0061
Epoch 41/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0059
Epoch 42/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0059
Epoch 43/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0060
Epoch 44/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0059
Epoch 45/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0059
Epoch 46/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0059
Epoch 47/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0059
Epoch 48/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0060
Epoch 49/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0058
Epoch 50/50
14/14 [==============================] - 0s 12ms/step - loss: 0.0059
```

In [40]:

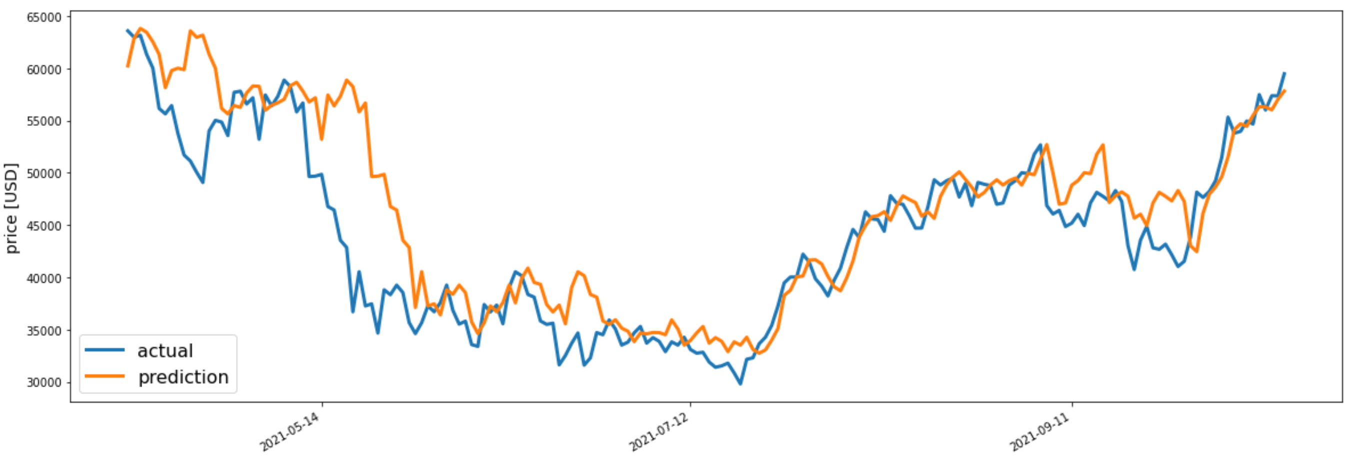
```
targets = test[target_col][window_len:]
preds = model.predict(X_test).squeeze()
mean_absolute_error(preds, y_test)
```

Out[40]:

```
0.06590164230202528
```

In [41]:

```
# 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)
```



In [ ]: