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 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	0.06127900
LTCBTC	0.00307400
BNBBTC	0.00716100
NEOBTC	0.00077100
QTUMETH	0.00362700
...	...
SHIBAUD	0.00004091
RAREBTC	0.00005154
RAREBNB	0.00717300
RAREBUSD	2.93100000
RAREUSDT	2.92800000

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("final_code/Resources/binance_tickers.csv")
```

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

56935.85

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	56935.86000000	3.35308000
1	56936.62000000	0.08781000
2	56939.09000000	0.01756000
3	56939.10000000	0.20584000
4	56939.11000000	0.50004000

In [13]:
```
# J.Guanzon Comment: Pulling historical daily data
btc_daily_data = client.get_historical_klines('BTCUSDT', Client.KLINE_INTERVAL_1DAY, '1 Jan 2020')
```

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 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
...	...	...	...	...	...
2021-10-08	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	56935.85	4312.257750

651 rows × 5 columns

In [18]:
```
btc_ohlcv_daily.to_csv("final_code/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 Time	Open	High	Low	Close	Volume
2021-10-07 02:30:00	55105.75	55105.75	55047.60	55089.16	0.38310
2021-10-07 02:31:00	55096.54	55096.54	55047.50	55071.27	0.47088
2021-10-07 02:32:00	55049.20	55073.46	55047.27	55073.46	0.07063
2021-10-07 02:33:00	55064.59	55074.26	55038.05	55055.32	0.08762

	Open	High	Low	Close	Volume
Open Time					
2021-10-07 02:34:00	55055.46	55060.96	55031.61	55037.94	0.22975
...	...	...	...	...	...
2021-10-12 02:25:00	56998.20	56998.20	56945.77	56945.77	1.02229
2021-10-12 02:26:00	56938.67	56971.85	56936.14	56955.34	0.51384
2021-10-12 02:27:00	56958.81	57016.71	56958.81	57007.88	0.50804
2021-10-12 02:28:00	57009.53	57018.21	56981.95	56981.95	0.22369
2021-10-12 02:29:00	56956.22	56957.08	56947.25	56955.22	0.14894

7200 rows × 5 columns

In [25]:

```
btc_ohlcv_minute.to_csv("final_code/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("final_code/Resources/daily_btc_ohclv_2021.csv"),
                     index_col= "Open Time")
target_col = 'Close'
```

In [28]:

```
btc_df.head()
```

Out[28]:

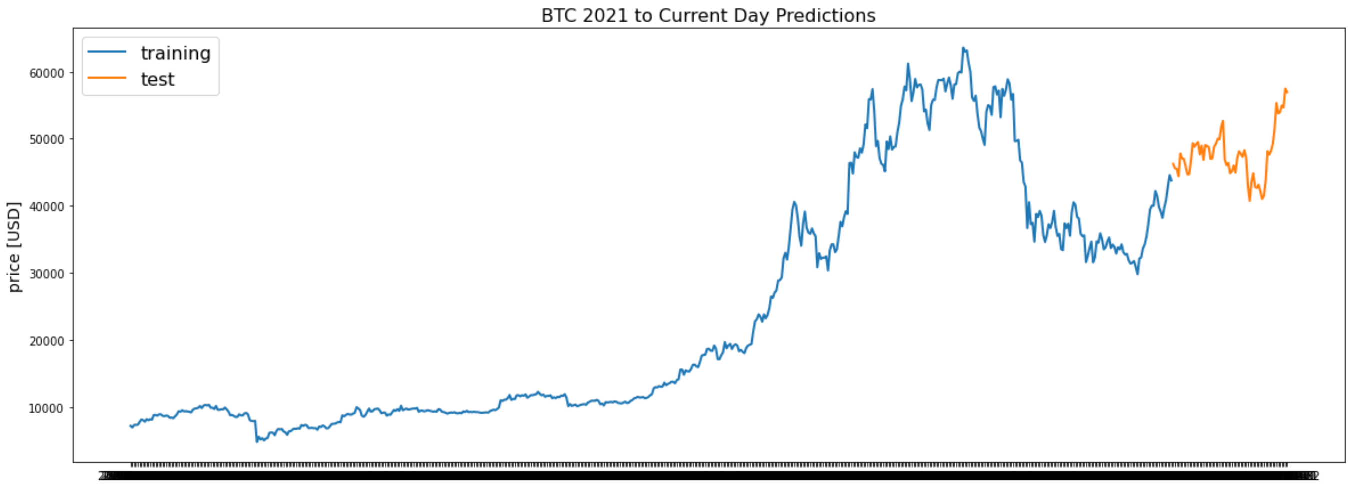
	Open	High	Low	Close	Volume
Open Time					
2020-01-01	7195.24	7255.0	7175.15	7200.85	16792.388165
2020-01-02	7200.77	7212.5	6924.74	6965.71	31951.483932
2020-01-03	6965.49	7405.0	6871.04	7344.96	68428.500451
2020-01-04	7345.00	7404.0	7272.21	7354.11	29987.974977
2020-01-05	7354.19	7495.0	7318.00	7358.75	38331.085604

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]:
```
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]:
```
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]:
```
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
```
In [35]:
```
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]:
```
np.random.seed(30)
window_len = 5
test_size = 0.2
zero_base = True
lstm_neurons = 100
epochs = 50
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 1/50
17/17 [==============================] - 1s 3ms/step - loss: 0.0067
Epoch 2/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0042
Epoch 3/50
```

```
17/17 [==============================] - 0s 3ms/step - loss: 0.0035
Epoch 4/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0032
Epoch 5/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0031
Epoch 6/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0029
Epoch 7/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0026
Epoch 8/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0025
Epoch 9/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0026
Epoch 10/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0024
Epoch 11/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0023
Epoch 12/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0023
Epoch 13/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0024
Epoch 14/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0022
Epoch 15/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0022
Epoch 16/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0023
Epoch 17/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0023
Epoch 18/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0022
Epoch 19/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0021
Epoch 20/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0021
Epoch 21/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0021
Epoch 22/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0020
Epoch 23/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0022
Epoch 24/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0021
Epoch 25/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0020
Epoch 26/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0021
Epoch 27/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0020
Epoch 28/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0021
Epoch 29/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0020
Epoch 30/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0020
Epoch 31/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0019
Epoch 32/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0019
Epoch 33/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0021
Epoch 34/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0019
Epoch 35/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0019
Epoch 36/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0020
Epoch 37/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0020
Epoch 38/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0020
Epoch 39/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0019
Epoch 40/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0020
Epoch 41/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0019
Epoch 42/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0019
Epoch 43/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0019
Epoch 44/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0019
Epoch 45/50
17/17 [==============================] - 0s 4ms/step - loss: 0.0019
Epoch 46/50
17/17 [==============================] - 0s 4ms/step - loss: 0.0020
Epoch 47/50
17/17 [==============================] - 0s 5ms/step - loss: 0.0019
Epoch 48/50
17/17 [==============================] - 0s 4ms/step - loss: 0.0019
Epoch 49/50
17/17 [==============================] - 0s 4ms/step - loss: 0.0019
Epoch 50/50
17/17 [==============================] - 0s 4ms/step - loss: 0.0018
```

In [38]:

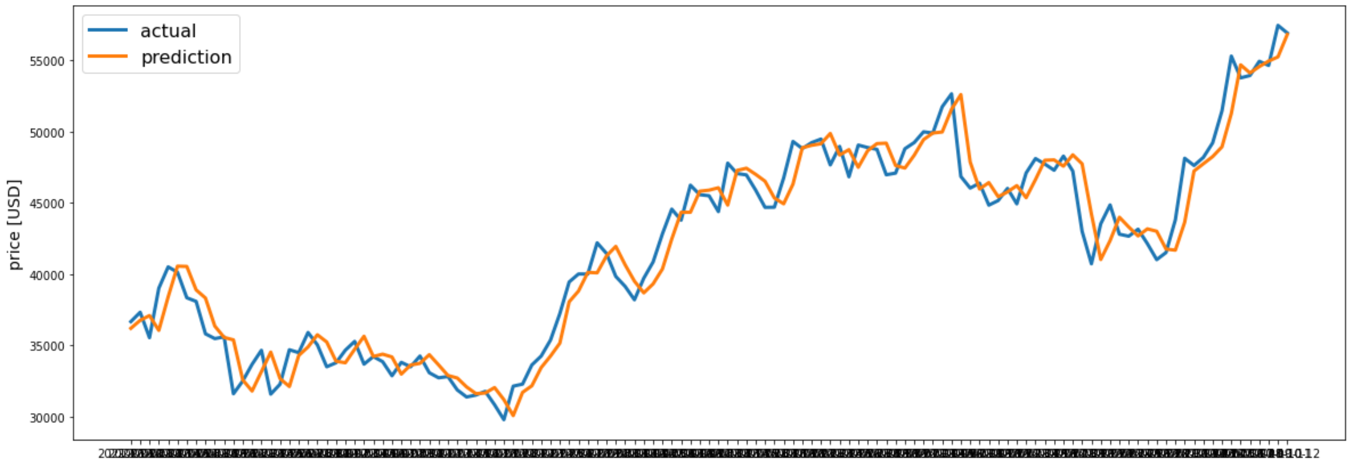
```
targets = test[target_col][window_len:]
preds = model.predict(X_test).squeeze()
mean_absolute_error(preds, y_test)
```

```
0.030991054497023857
```

Out[38]:

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)
```



In [ ]:

In [ ]:

In [ ]: