In [ ]:

```
"""
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

"""
```

In [203…

```
# 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 [204…

```
# Pull API keys from .env file
api_key = os.environ.get("api_key")
api_secret = os.environ.get("api_secret")
```

In [205…

```
client = Client(api_key, api_secret)
```

In [206…

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

In [207…

```
ticker_df = pd.DataFrame(tickers)
```

In [208…

```
ticker_df.set_index('symbol', inplace=True)
ticker_df
```

Out[208…

	price
symbol	
ETHBTC	0.06134300
LTCBTC	0.00307800
BNBBTC	0.00717100
NEOBTC	0.00077300
QTUMETH	0.00362700
...	...
SHIBAUD	0.00004132
RAREBTC	0.00005108
RAREBNB	0.00713600
RAREBUSD	2.90700000
RAREUSDT	2.90800000

1695 rows × 1 columns

In [209…

```
"""
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[209…

```
' \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 [210…

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

In [211…

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

```
57081.1
```

In [212…

```
depth = client.get_order_book(symbol='BTCUSDT')
```

In [213…

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

Out[213…

	Price	Volume

	Price	Volume
0	57078.01000000	1.26994000
1	57078.73000000	0.08759000
2	57082.62000000	0.05307000
3	57082.63000000	1.05132000
4	57082.80000000	0.02358000

In [214…

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

In [215…

```
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 [216…

```
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 [217…

```
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 [218…

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

Out[218…

	Open	High	Low	Close	Volume
Open Time					
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	57075.83	4077.492980

651 rows × 5 columns

In [219…

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

In [220…

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

In [221…

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

In [222…

```
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 [223…

```
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 [224…

```
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 [225…

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

Out[225…

	Open	High	Low	Close	Volume
Open Time					
2021-10-07 02:18:00	54925.49	54950.15	54907.49	54917.73	0.08934
2021-10-07 02:19:00	54932.82	54943.77	54885.20	54893.85	2.19582
2021-10-07 02:20:00	54930.46	54930.46	54879.32	54919.38	0.18727
2021-10-07 02:21:00	54908.70	54963.23	54908.70	54926.43	0.54984

	Open	High	Low	Close	Volume
Open Time					
2021-10-07 02:22:00	54919.22	55044.62	54919.22	55033.85	0.25225
...	...	...	...	...	...
2021-10-12 02:13:00	56994.64	57061.11	56994.64	57061.11	0.51440
2021-10-12 02:14:00	57045.11	57052.36	57033.19	57052.36	0.37159
2021-10-12 02:15:00	57032.24	57032.24	56978.27	56978.27	0.06760
2021-10-12 02:16:00	56984.24	57096.20	56984.24	57084.11	1.17687
2021-10-12 02:17:00	57096.58	57126.91	57086.22	57086.33	1.27463

7200 rows × 5 columns

In [226…

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

In [227…

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

"""
```

Out[227…

```
'\nNext, we will be using the daily data for our Recurrent Neural Network. We are using Recurrent Neural Network.\n\n'
```

In [228…

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

In [229…

```
btc_df.head()
```

Out[229…

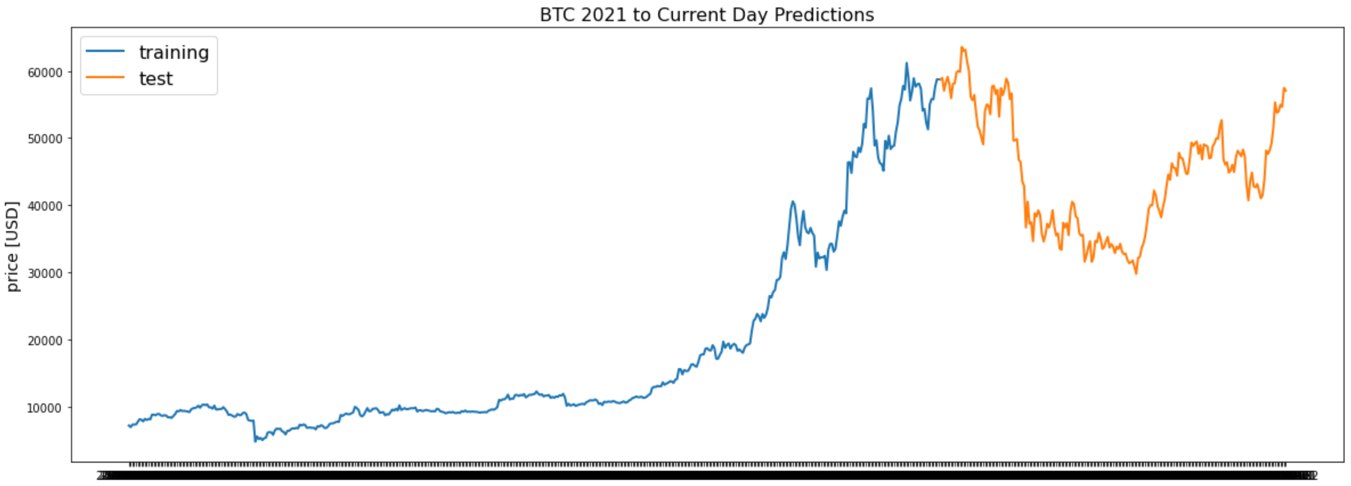
	Open	High	Low	Close	Volume
Open Time					
2020-01-01	7195.24	7255.0	7175.15	7200.85	16792.388165
2020-01-02	7200.77	7212.5	6924.74	6965.71	31951.483932
2020-01-03	6965.49	7405.0	6871.04	7344.96	68428.500451
2020-01-04	7345.00	7404.0	7272.21	7354.11	29987.974977
2020-01-05	7354.19	7495.0	7318.00	7358.75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

In [239…

```
# 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 [240…

```
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 [241…

```
"""
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[241…

```
'\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 [242…

```
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 [243…

```
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 [244…

```
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 [245…

```
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 [246…

```
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 [247…

```
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 4ms/step - loss: 0.0074
Epoch 2/50
17/17 [==============================] - 0s 4ms/step - loss: 0.0040
Epoch 3/50
```

```
17/17 [==============================] - 0s 4ms/step - loss: 0.0034
Epoch 4/50
17/17 [==============================] - 0s 4ms/step - loss: 0.0032
Epoch 5/50
17/17 [==============================] - 0s 4ms/step - loss: 0.0030
Epoch 6/50
17/17 [==============================] - 0s 5ms/step - loss: 0.0029
Epoch 7/50
17/17 [==============================] - 0s 5ms/step - loss: 0.0028
Epoch 8/50
17/17 [==============================] - 0s 5ms/step - loss: 0.0028
Epoch 9/50
17/17 [==============================] - 0s 5ms/step - loss: 0.0027
Epoch 10/50
17/17 [==============================] - 0s 6ms/step - loss: 0.0024
Epoch 11/50
17/17 [==============================] - 0s 5ms/step - loss: 0.0024
Epoch 12/50
17/17 [==============================] - 0s 5ms/step - loss: 0.0022
Epoch 13/50
17/17 [==============================] - 0s 5ms/step - loss: 0.0024
Epoch 14/50
17/17 [==============================] - 0s 4ms/step - loss: 0.0024
Epoch 15/50
17/17 [==============================] - 0s 5ms/step - loss: 0.0023
Epoch 16/50
17/17 [==============================] - 0s 5ms/step - loss: 0.0022
Epoch 17/50
17/17 [==============================] - 0s 4ms/step - loss: 0.0022
Epoch 18/50
17/17 [==============================] - 0s 4ms/step - loss: 0.0026
Epoch 19/50
17/17 [==============================] - 0s 4ms/step - loss: 0.0021
Epoch 20/50
17/17 [==============================] - 0s 4ms/step - loss: 0.0020
Epoch 21/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0022
Epoch 22/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0021
Epoch 23/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0021
Epoch 24/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0024
Epoch 25/50
17/17 [==============================] - 0s 4ms/step - loss: 0.0021
Epoch 26/50
17/17 [==============================] - 0s 5ms/step - loss: 0.0022
Epoch 27/50
17/17 [==============================] - 0s 4ms/step - loss: 0.0021
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: 0.0021
Epoch 32/50
17/17 [==============================] - 0s 4ms/step - loss: 0.0021
Epoch 33/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0020
Epoch 34/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0020
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 4ms/step - loss: 0.0020
Epoch 38/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0020
Epoch 39/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0022
Epoch 40/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0019
Epoch 41/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0020
Epoch 42/50
17/17 [==============================] - 0s 3ms/step - loss: 0.0019
Epoch 43/50
17/17 [==============================] - 0s 4ms/step - loss: 0.0019
Epoch 44/50
17/17 [==============================] - 0s 4ms/step - loss: 0.0019
Epoch 45/50
17/17 [==============================] - 0s 4ms/step - loss: 0.0019
Epoch 46/50
17/17 [==============================] - 0s 4ms/step - loss: 0.0019
Epoch 47/50
17/17 [==============================] - 0s 4ms/step - loss: 0.0019
Epoch 48/50
17/17 [==============================] - 0s 5ms/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 [248…

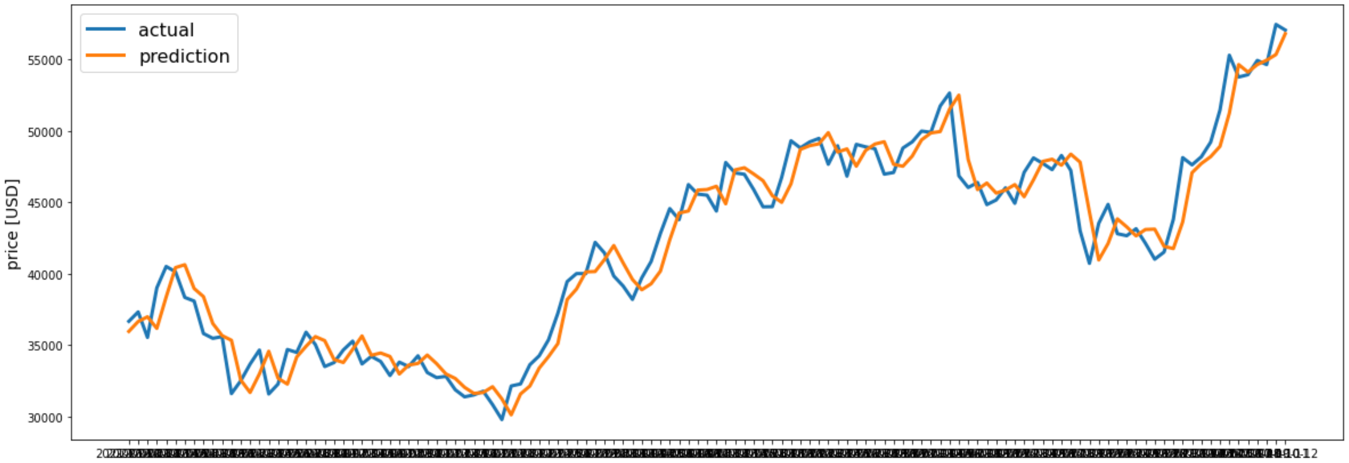
```python
targets = test[target_col][window_len:]
preds = model.predict(X_test).squeeze()
mean_absolute_error(preds, y_test)
```

```
0.03148357759556417
```

Out[248…

In [249…

```
# 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 [ ]:

```
```