

```
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  0.06134600
LTCBTC  0.00307200
BNBBTC  0.00716300
NEOBTC  0.00077000
QTUMETH 0.00360000
...      ...
SHIBAUD 0.00004160
RAREBTC 0.00005077
RAREBNB 0.00712200
RAREBUSD 2.89100000
RAREUSDT 2.89200000

1695 rows x 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\n'

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

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

56772.04

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	56772.05000000	0.47582000
1	56779.00000000	0.09807000
2	56779.44000000	0.27301000
3	56779.45000000	0.66555000
4	56779.50000000	2.80732000

In [13]:

```
# J.Guanzon Comment: Pulling historical daily data
btc_daily_data = client.get_historical_klines('BTCUSD', 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_ohlc_daily = btc_daily_df.iloc[:,0:6]
btc_ohlc_daily = btc_ohlc_daily.set_index('Open Time')
btc_ohlc_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	56772.05	5145.655120

285 rows × 5 columns

In [18]:

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

Out[24]:

	Open	High	Low	Close	Volume
Open Time					
2021-10-07 02:48:00	55114.95	55114.95	55055.39	55072.11	1.22394
2021-10-07 02:49:00	55069.12	55072.24	54990.87	55004.59	2.76886
2021-10-07 02:50:00	55004.59	55007.38	54947.04	55007.38	2.46780
2021-10-07 02:51:00	55015.89	55028.06	54950.43	54990.18	3.26457

	Open	High	Low	Close	Volume
Open Time					
2021-10-07 02:52:00	54996.66	55065.94	54976.60	55064.27	2.51096
...
2021-10-12 02:44:00	57007.41	57007.41	56924.30	56929.00	0.96076
2021-10-12 02:45:00	56924.94	56936.35	56865.66	56873.73	0.83581
2021-10-12 02:46:00	56863.43	56863.43	56817.23	56818.27	0.72059
2021-10-12 02:47:00	56818.03	56858.48	56789.25	56789.25	1.00023
2021-10-12 02:48:00	56786.83	56802.86	56786.83	56789.95	0.22433

7201 rows x 5 columns

```
In [25]: btc_ohlcv_minute.to_csv("Resources/minute_btc_ohlcv_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_ohlcv_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 accuracy

# 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

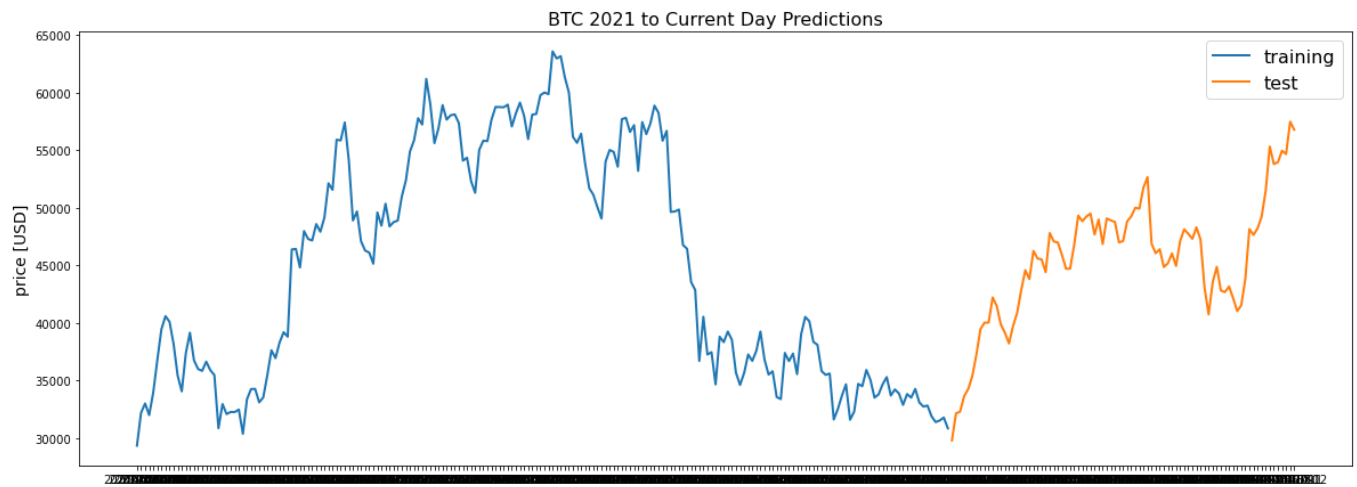
train, test = train_test_split(btc_df, test_size=0.3)

# 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 [ ]:
```

```
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

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]: 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(45)
window_len = 5
test_size = 0.3
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
7/7 [=====] - 1s 3ms/step - loss: 0.0102
Epoch 2/50
7/7 [=====] - 0s 3ms/step - loss: 0.0075
Epoch 3/50
7/7 [=====] - 0s 3ms/step - loss: 0.0056
Epoch 4/50
7/7 [=====] - 0s 4ms/step - loss: 0.0050
Epoch 5/50
7/7 [=====] - 0s 4ms/step - loss: 0.0048
Epoch 6/50
7/7 [=====] - 0s 3ms/step - loss: 0.0047
Epoch 7/50
7/7 [=====] - 0s 3ms/step - loss: 0.0047
Epoch 8/50
7/7 [=====] - 0s 3ms/step - loss: 0.0046
Epoch 9/50
7/7 [=====] - 0s 3ms/step - loss: 0.0040
Epoch 10/50
7/7 [=====] - 0s 3ms/step - loss: 0.0041
Epoch 11/50
7/7 [=====] - 0s 2ms/step - loss: 0.0038
Epoch 12/50
7/7 [=====] - 0s 3ms/step - loss: 0.0039
Epoch 13/50
7/7 [=====] - 0s 3ms/step - loss: 0.0036
Epoch 14/50
7/7 [=====] - 0s 3ms/step - loss: 0.0036
Epoch 15/50
7/7 [=====] - 0s 3ms/step - loss: 0.0036
Epoch 16/50
7/7 [=====] - 0s 3ms/step - loss: 0.0038
Epoch 17/50
7/7 [=====] - 0s 3ms/step - loss: 0.0034
Epoch 18/50
7/7 [=====] - 0s 3ms/step - loss: 0.0035
Epoch 19/50
7/7 [=====] - 0s 3ms/step - loss: 0.0033
Epoch 20/50
7/7 [=====] - 0s 3ms/step - loss: 0.0032
Epoch 21/50
7/7 [=====] - 0s 3ms/step - loss: 0.0031
Epoch 22/50
7/7 [=====] - 0s 4ms/step - loss: 0.0036
Epoch 23/50
7/7 [=====] - 0s 5ms/step - loss: 0.0032
Epoch 24/50
7/7 [=====] - 0s 4ms/step - loss: 0.0030
Epoch 25/50
7/7 [=====] - 0s 3ms/step - loss: 0.0032
Epoch 26/50
7/7 [=====] - 0s 3ms/step - loss: 0.0032
Epoch 27/50
7/7 [=====] - 0s 3ms/step - loss: 0.0031
Epoch 28/50
7/7 [=====] - 0s 3ms/step - loss: 0.0032
Epoch 29/50
7/7 [=====] - 0s 3ms/step - loss: 0.0030
Epoch 30/50
7/7 [=====] - 0s 3ms/step - loss: 0.0028
Epoch 31/50
7/7 [=====] - 0s 3ms/step - loss: 0.0029
Epoch 32/50
7/7 [=====] - 0s 3ms/step - loss: 0.0029
Epoch 33/50
7/7 [=====] - 0s 3ms/step - loss: 0.0027
Epoch 34/50
7/7 [=====] - 0s 3ms/step - loss: 0.0030
Epoch 35/50
7/7 [=====] - 0s 3ms/step - loss: 0.0027
Epoch 36/50
7/7 [=====] - 0s 3ms/step - loss: 0.0026
Epoch 37/50
7/7 [=====] - 0s 3ms/step - loss: 0.0028
Epoch 38/50
7/7 [=====] - 0s 3ms/step - loss: 0.0030
Epoch 39/50
7/7 [=====] - 0s 3ms/step - loss: 0.0028
Epoch 40/50

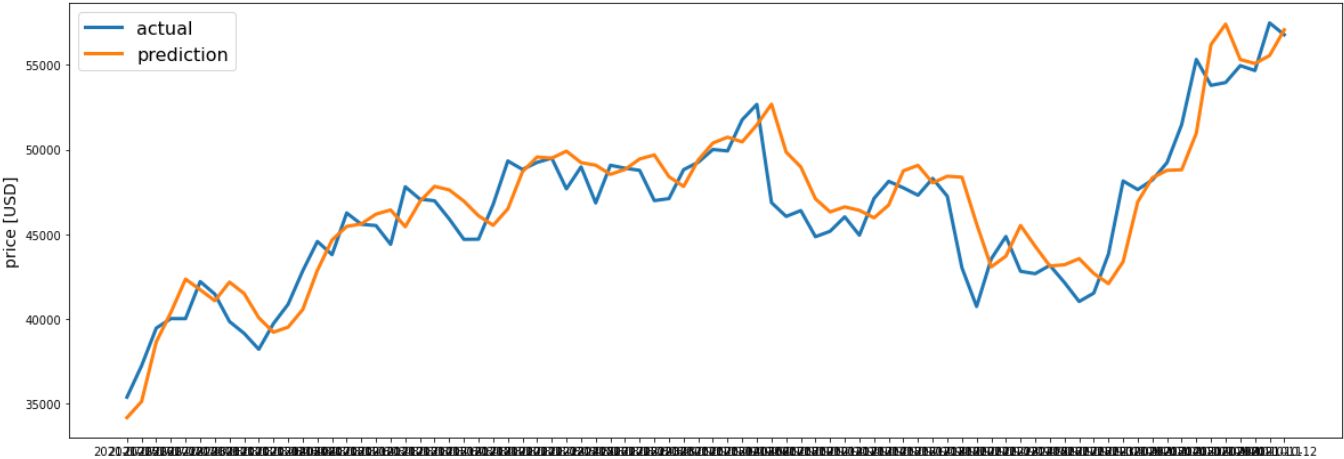
```

```
7/7 [=====] - 0s 3ms/step - loss: 0.0027
Epoch 41/50
7/7 [=====] - 0s 3ms/step - loss: 0.0030
Epoch 42/50
7/7 [=====] - 0s 3ms/step - loss: 0.0028
Epoch 43/50
7/7 [=====] - 0s 3ms/step - loss: 0.0029
Epoch 44/50
7/7 [=====] - 0s 3ms/step - loss: 0.0028
Epoch 45/50
7/7 [=====] - 0s 3ms/step - loss: 0.0028
Epoch 46/50
7/7 [=====] - 0s 3ms/step - loss: 0.0028
Epoch 47/50
7/7 [=====] - 0s 3ms/step - loss: 0.0027
Epoch 48/50
7/7 [=====] - 0s 3ms/step - loss: 0.0028
Epoch 49/50
7/7 [=====] - 0s 3ms/step - loss: 0.0026
Epoch 50/50
7/7 [=====] - 0s 3ms/step - loss: 0.0027
```

```
In [38]: targets = test[target_col][window_len:]
preds = model.predict(X_test).squeeze()
mean_absolute_error(preds, y_test)
```

Out[38]: 0.033879393134016646

```
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 [ ]:
```