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	0.06238700
LTCBTC	0.00309400
BNBBTC	0.00821700
NEOBTC	0.00079400
QTUMETH	0.00380100
...	...
SHIBAUD	0.00004021
RAREBTC	0.00004372
RAREBNB	0.00533400
RAREBUSD	2.47000000
RAREUSDT	2.46900000

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']))
```

56382.57

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	56382.58000000	1.40438000
1	56382.62000000	0.00100000
2	56385.58000000	0.33629000
3	56385.59000000	0.17737000
4	56385.87000000	0.62216000

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 Time	Open	High	Low	Close	Volume
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
...	...	...	...	...	...
2021-10-09	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	57680.00	53879.00	55996.93	53471.285500
2021-10-13	55996.91	56599.99	55825.90	56380.17	4790.914300

286 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 Time	Open	High	Low	Close	Volume
2021-10-08 02:57:00	53826.15	53842.86	53813.87	53842.86	1.09835
2021-10-08 02:58:00	53885.64	53905.89	53842.65	53842.65	0.95274
2021-10-08 02:59:00	53850.75	53877.20	53846.88	53863.59	0.16733

	Open	High	Low	Close	Volume
Open Time					
2021-10-08 03:00:00	53851.93	53882.65	53851.93	53882.39	0.10060
2021-10-08 03:01:00	53927.69	53927.69	53892.56	53894.27	0.11298
...	...	...	...	...	...
2021-10-13 02:52:00	56347.61	56347.61	56288.71	56320.78	2.62600
2021-10-13 02:53:00	56323.28	56335.42	56323.27	56328.07	0.26958
2021-10-13 02:54:00	56330.36	56351.55	56324.53	56335.49	0.42222
2021-10-13 02:55:00	56340.34	56370.46	56335.43	56365.10	0.31977
2021-10-13 02:56:00	56367.08	56389.71	56364.71	56381.68	1.14081

7200 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]:

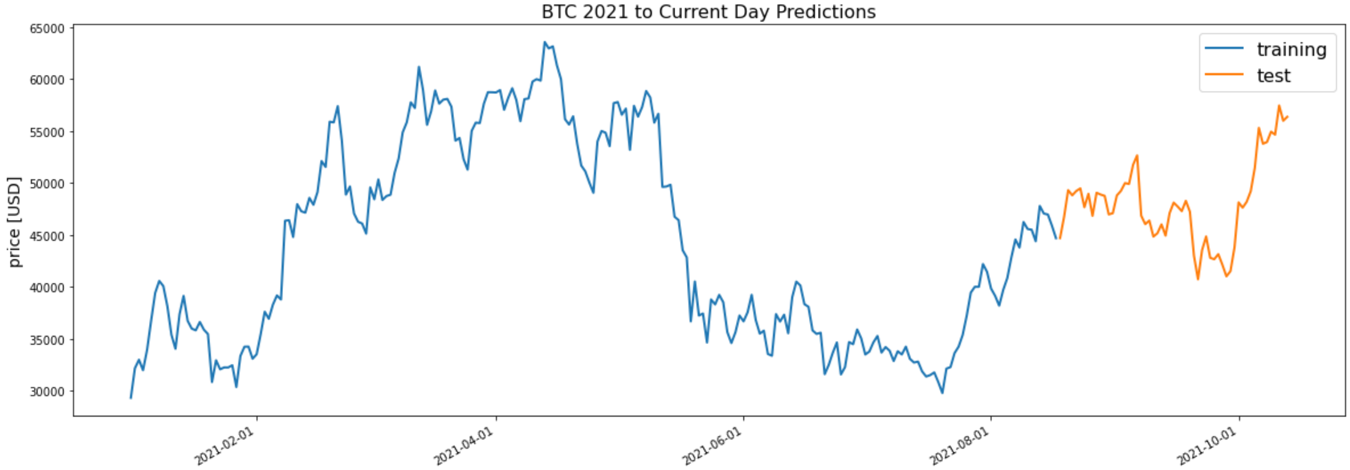
	Open	High	Low	Close	Volume
Open Time					
2021-01-01	28923.63	29600.00	28624.57	29331.69	54182.925011
2021-01-02	29331.70	33300.00	28946.53	32178.33	129993.873362
2021-01-03	32176.45	34778.11	31962.99	33000.05	120957.566750
2021-01-04	33000.05	33600.00	28130.00	31988.71	140899.885690
2021-01-05	31989.75	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)
    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 [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=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 [34]:

```
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 [35]:

```
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 [36]:

```
def build_lstm_model(input_data, output_size, neurons=100, activ_func='linear', 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 [37]:

```
np.random.seed(42)
window_len = 10
test_size = 0.2
zero_base = True
lstm_neurons = 100
epochs = 50
batch_size = 32
loss = 'mse'
dropout = 0.2
optimizer = 'adam'
```

In [38]:

```
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 4ms/step - loss: 0.0117
Epoch 2/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0082
Epoch 3/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0065
Epoch 4/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0059
Epoch 5/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0057
Epoch 6/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0055
Epoch 7/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0047
Epoch 8/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0045
Epoch 9/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0042
Epoch 10/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0038
Epoch 11/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0036
Epoch 12/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0041
Epoch 13/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0037
Epoch 14/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0034
Epoch 15/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0034
Epoch 16/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0036
Epoch 17/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0035
Epoch 18/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0034
Epoch 19/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0032
Epoch 20/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0029
Epoch 21/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0032
Epoch 22/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0030
Epoch 23/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0031
Epoch 24/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0029
Epoch 25/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0032
Epoch 26/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0030
Epoch 27/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0030
Epoch 28/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0030
Epoch 29/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0028
Epoch 30/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0028
Epoch 31/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0031
Epoch 32/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0026
Epoch 33/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0027
Epoch 34/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0030
Epoch 35/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0029
Epoch 36/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0028
Epoch 37/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0026
Epoch 38/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0027
```

```
Epoch 39/50
7/7 [==============================] - 0s 5ms/step - loss: 0.0028
Epoch 40/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0027
Epoch 41/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0026
Epoch 42/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0027
Epoch 43/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0028
Epoch 44/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0028
Epoch 45/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0027
Epoch 46/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0025
Epoch 47/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0028
Epoch 48/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0025
Epoch 49/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0026
Epoch 50/50
7/7 [==============================] - 0s 4ms/step - loss: 0.0027
```

In [39]:

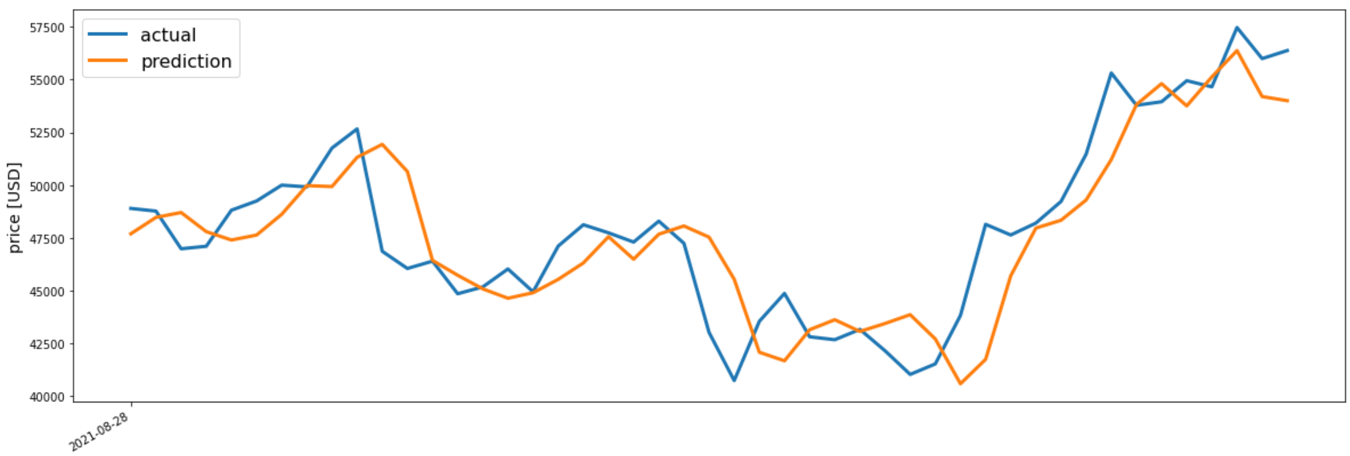
```
targets = test[target_col][window_len:]
preds = model.predict(X_test).squeeze()
mean_absolute_error(preds, y_test)
```

Out[39]:

```
0.03568001325820969
```

In [40]:

```
# 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 [ ]:

In [ ]: