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/ap
idocs/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	0.06129200
LTCBTC	0.00306300
BNBBTC	0.00715400
NEOBTC	0.00076700
QTUMETH	0.00359900
...	...
SHIBAUD	0.00004191
RAREBTC	0.00005021
RAREBNB	0.00701300
RAREBUSD	2.85600000
RAREUSDT	2.84800000

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']))
```

```
56657.98
```

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	56657.56000000	3.00038000
1	56659.02000000	0.38560000
2	56659.33000000	0.47385000
3	56659.34000000	0.66400000
4	56661.06000000	0.32000000

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	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	56653.02	5915.337850

285 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	High	Low	Close	Volume
Open Time					
2021-10-07 03:05:00	55177.45	55237.19	55177.45	55216.81	2.69380
2021-10-07 03:06:00	55216.62	55218.37	55169.75	55184.25	2.28414
2021-10-07 03:07:00	55184.25	55195.87	55097.70	55134.25	3.72313
2021-10-07 03:08:00	55133.76	55133.76	55082.45	55115.33	2.04664

Open Time	Open	High	Low	Close	Volume
2021-10-07 03:09:00	55106.50	55122.00	55075.79	55121.23	4.11692
...	...	...	...	...	...
2021-10-12 03:00:00	56777.99	56777.99	56743.28	56756.52	0.40967
2021-10-12 03:01:00	56756.37	56786.12	56756.37	56786.12	0.40826
2021-10-12 03:02:00	56784.89	56791.74	56758.97	56758.97	0.45269
2021-10-12 03:03:00	56734.27	56740.36	56660.90	56666.79	0.58205
2021-10-12 03:04:00	56681.77	56681.77	56671.95	56671.95	0.13018

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]:

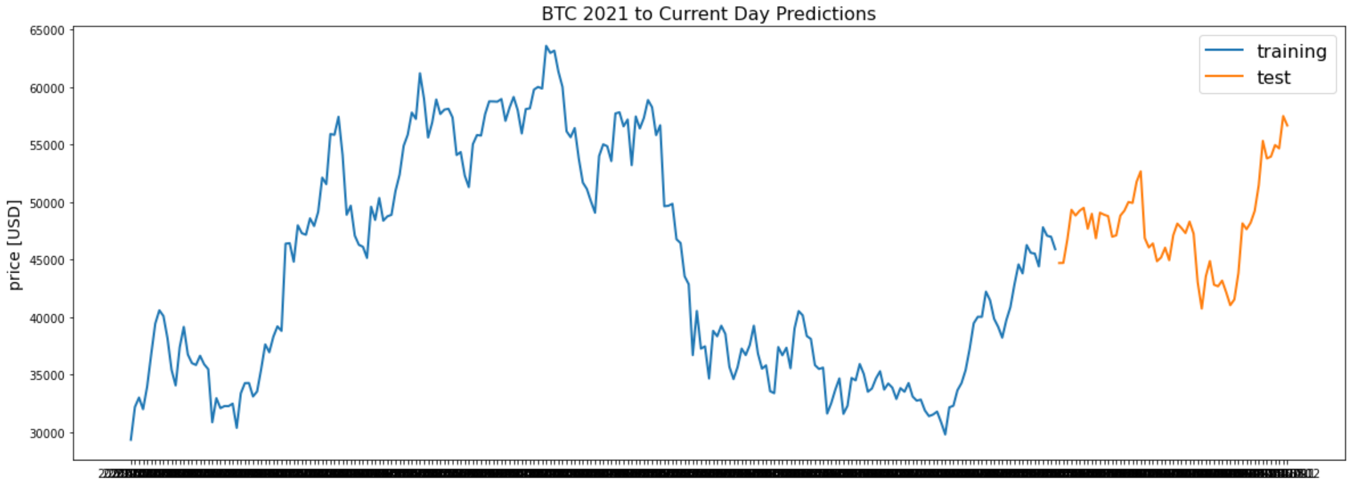
Open Time	Open	High	Low	Close	Volume
2021-01-01	28923.63	29600.00	28624.57	29331.69	54182.925011
2021-01-02	29331.70	33300.00	28946.53	32178.33	129993.873362
2021-01-03	32176.45	34778.11	31962.99	33000.05	120957.566750
2021-01-04	33000.05	33600.00	28130.00	31988.71	140899.885690
2021-01-05	31989.75	34360.00	29900.00	33949.53	116049.997038

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

# 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=150, 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(42)
```

```
window_len = 5
test_size = 0.2
zero_base = True
lstm_neurons = 150
epochs = 20
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 1/20
7/7 [==============================] - 1s 4ms/step - loss: 0.0108
Epoch 2/20
7/7 [==============================] - 0s 4ms/step - loss: 0.0076
Epoch 3/20
7/7 [==============================] - 0s 4ms/step - loss: 0.0054
Epoch 4/20
7/7 [==============================] - 0s 4ms/step - loss: 0.0049
Epoch 5/20
7/7 [==============================] - 0s 4ms/step - loss: 0.0045
Epoch 6/20
7/7 [==============================] - 0s 5ms/step - loss: 0.0041
Epoch 7/20
7/7 [==============================] - 0s 4ms/step - loss: 0.0039
Epoch 8/20
7/7 [==============================] - 0s 5ms/step - loss: 0.0037
Epoch 9/20
7/7 [==============================] - 0s 5ms/step - loss: 0.0035
Epoch 10/20
7/7 [==============================] - 0s 5ms/step - loss: 0.0035
Epoch 11/20
7/7 [==============================] - 0s 5ms/step - loss: 0.0034
Epoch 12/20
7/7 [==============================] - 0s 5ms/step - loss: 0.0034
Epoch 13/20
7/7 [==============================] - 0s 5ms/step - loss: 0.0031
Epoch 14/20
7/7 [==============================] - 0s 5ms/step - loss: 0.0031
Epoch 15/20
7/7 [==============================] - 0s 5ms/step - loss: 0.0030
Epoch 16/20
7/7 [==============================] - 0s 5ms/step - loss: 0.0030
Epoch 17/20
7/7 [==============================] - 0s 5ms/step - loss: 0.0029
Epoch 18/20
7/7 [==============================] - 0s 5ms/step - loss: 0.0029
Epoch 19/20
7/7 [==============================] - 0s 5ms/step - loss: 0.0028
Epoch 20/20
7/7 [==============================] - 0s 5ms/step - loss: 0.0028
```

In [38]:

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

Out[38]:

```
0.03260490619507155
```

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)
```