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]:

symbol	price
ETHBTC	0.06235600
LTCBTC	0.00309100
BNBBTC	0.00825400
NEOBTC	0.00079300
QTUMETH	0.00377300
...	...
SHIBAUD	0.00004099
RAREBTC	0.00004259
RAREBNB	0.00515200
RAREBUSD	2.39000000
RAREUSDT	2.38900000

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']))
```

```
56306.03
```

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	56306.03000000	0.07008000
1	56306.07000000	0.00031000
2	56310.20000000	0.14635000
3	56310.21000000	0.03553000
4	56311.22000000	0.04000000

In [13]:
```
"""
Pulling historical daily data
"""
```

Out[13]:
```
'\nPulling historical daily data\n'
```

In [14]:
```
btc_daily_data = client.get_historical_klines('BTCUSDT', Client.KLINE_INTERVAL_1DAY, '1 Jan 2021')
```

In [15]:
```
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 [16]:
```
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 [17]:
```
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 [18]:
```
btc_ohlcv_daily = btc_daily_df.iloc[:,0:6]
btc_ohlcv_daily = btc_ohlcv_daily.set_index('Open Time')
btc_ohlcv_daily
```

Out[18]:

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	56306.03	6358.684250

286 rows × 5 columns

In [19]:
```
btc_ohlcv_daily.to_csv("Resources/daily_btc_ohclv_2021.csv")
```

In [20]:
```
"""
Pulling historical minute data
"""
```

Out[20]:
```
'\nPulling historical minute data \n'
```

In [21]:
```
historical_minute = client.get_historical_klines('BTCUSDC', Client.KLINE_INTERVAL_1MINUTE, '5 day ago UTC')
```

In [22]:
```
hist_min = pd.DataFrame(historical_minute)
```

In [23]:
```
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 [24]:
```
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 [25]:
```
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 [26]:
```
```

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

Out[26]:

Open Time	Open	High	Low	Close	Volume
2021-10-08 04:26:00	53898.06	53935.92	53896.56	53932.45	0.03221
2021-10-08 04:27:00	53913.09	53924.25	53899.24	53913.47	0.05638
2021-10-08 04:28:00	53913.84	53919.86	53913.84	53919.86	0.00325
2021-10-08 04:29:00	53919.47	53920.97	53911.90	53912.13	0.01066
2021-10-08 04:30:00	53920.97	53920.97	53905.24	53907.93	0.07439
...	...	...	...	...	...
2021-10-13 04:21:00	56338.92	56338.92	56324.39	56336.55	0.13028
2021-10-13 04:22:00	56319.66	56323.03	56300.00	56300.00	0.49797
2021-10-13 04:23:00	56303.46	56326.07	56303.46	56311.46	0.39256
2021-10-13 04:24:00	56311.46	56311.46	56311.46	56311.46	0.00000
2021-10-13 04:25:00	56302.60	56312.69	56300.00	56300.00	0.93517

7200 rows × 5 columns

In [27]:
```
btc_ohlcv_minute.to_csv("Resources/minute_btc_ohclv_2021.csv")
```

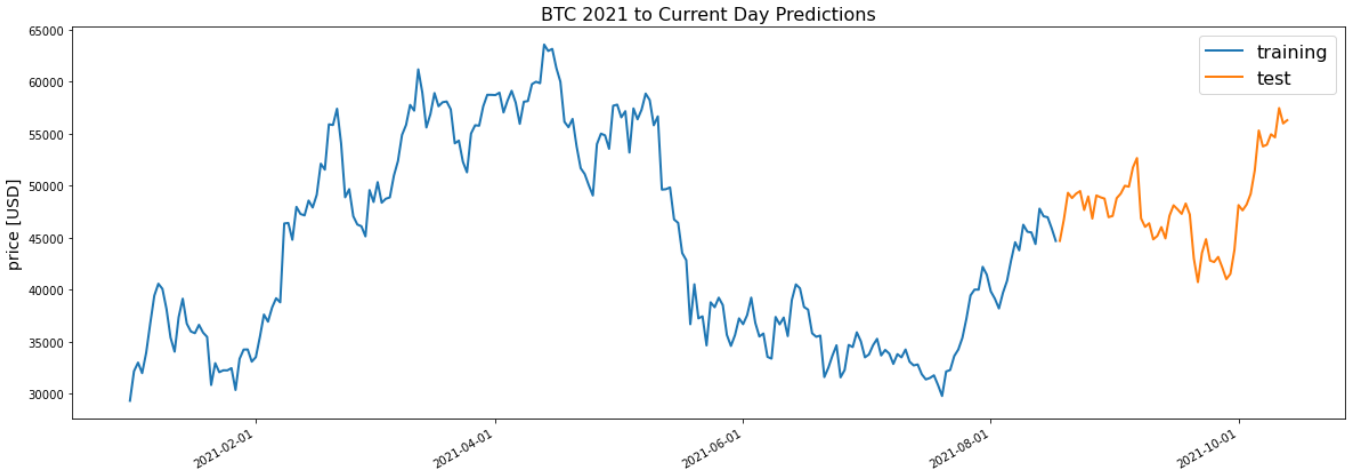
In [28]:
```
"""
Next, we will be using the daily data for our Recurrent Neural Network. We are using Recurrent Neural Network for its use of time series and sequential data
RNN specializes in using information from prior inputs and uses it to influence current inputs and outputs, and the cycle repeats.
"""
```

Out[28]:
```
'\nNext, we will be using the daily data for our Recurrent Neural Network. We are using Recurrent Neural Network for its use of time series and sequential d
ata. \nRNN specializes in using information from prior inputs and uses it to influence current inputs and outputs, and the cycle repeats. \n'
```

In [29]:
```
btc_df = pd.read_csv(Path("Resources/daily_btc_ohclv_2021.csv"),
                     index_col= "Open Time")
target_col = 'Close'
```

In [30]:
```
# 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 [31]:
```
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 [32]:
```
"""
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[32]:
```
'\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 [33]:
```
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 [34]:
```
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 [35]:
```
# J.Guanzon Comment: We want to use the data from Jan-Jun 2021 and use the rest of the data to train and predict the rest of the data.
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 [36]:
```
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 [37]:
```
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 [38]:
```
np.random.seed(50)
window_len = 10
test_size = 0.2
zero_base = True
lstm_neurons = 100
epochs = 20
batch_size = 32
loss = 'mse'
dropout = 0.2
optimizer = 'adam'
```

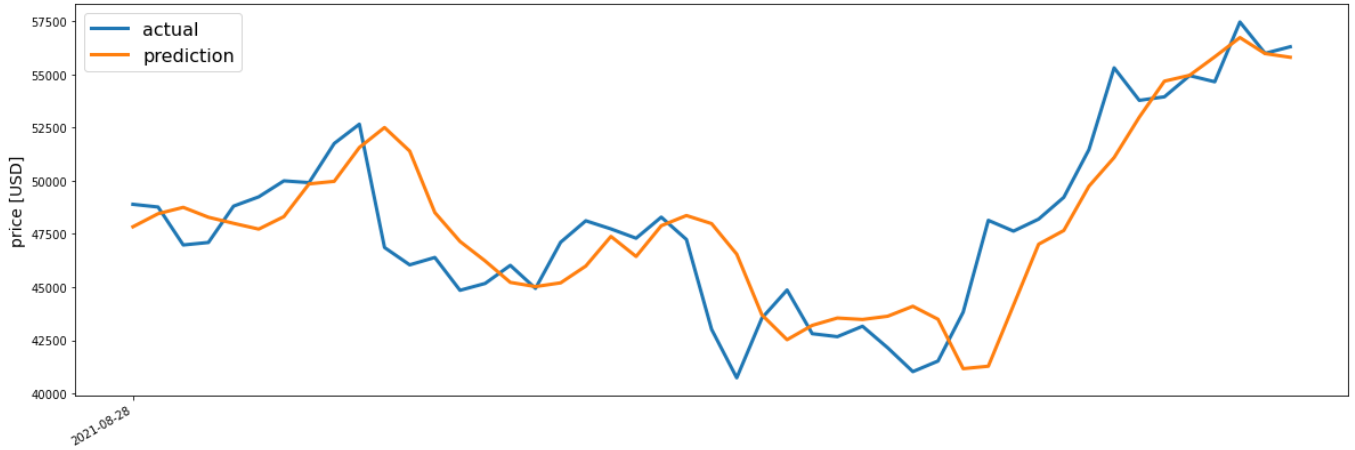
In [39]:
```
train, test, X_train, X_test, y_train, y_test = prepare_data(
    btc_df, target_col, window_len=window_len, zero_base=zero_base, test_size=test_size)
model = build_lstm_model(
    X_train, output_size=1, neurons=lstm_neurons, dropout=dropout, loss=loss,
    optimizer=optimizer)
history = model.fit(
    X_train, y_train, epochs=epochs, batch_size=batch_size, verbose=1, shuffle=True)
```

```
Epoch 1/20
7/7 [==============================] - 1s 7ms/step - loss: 0.0196
Epoch 2/20
7/7 [==============================] - 0s 4ms/step - loss: 0.0099
Epoch 3/20
7/7 [==============================] - 0s 4ms/step - loss: 0.0084
Epoch 4/20
7/7 [==============================] - 0s 4ms/step - loss: 0.0071
Epoch 5/20
7/7 [==============================] - 0s 4ms/step - loss: 0.0061
Epoch 6/20
7/7 [==============================] - 0s 4ms/step - loss: 0.0059
Epoch 7/20
7/7 [==============================] - 0s 4ms/step - loss: 0.0053
Epoch 8/20
7/7 [==============================] - 0s 4ms/step - loss: 0.0052
Epoch 9/20
7/7 [==============================] - 0s 4ms/step - loss: 0.0053
Epoch 10/20
7/7 [==============================] - 0s 4ms/step - loss: 0.0044
Epoch 11/20
7/7 [==============================] - 0s 4ms/step - loss: 0.0041
Epoch 12/20
7/7 [==============================] - 0s 4ms/step - loss: 0.0044
Epoch 13/20
7/7 [==============================] - 0s 4ms/step - loss: 0.0037
Epoch 14/20
7/7 [==============================] - 0s 4ms/step - loss: 0.0036
Epoch 15/20
7/7 [==============================] - 0s 4ms/step - loss: 0.0034
Epoch 16/20
7/7 [==============================] - 0s 4ms/step - loss: 0.0036
Epoch 17/20
7/7 [==============================] - 0s 4ms/step - loss: 0.0033
Epoch 18/20
7/7 [==============================] - 0s 4ms/step - loss: 0.0035
Epoch 19/20
7/7 [==============================] - 0s 4ms/step - loss: 0.0035
Epoch 20/20
7/7 [==============================] - 0s 4ms/step - loss: 0.0030
```

In [40]:
```
targets = test[target_col][window_len:]
preds = model.predict(X_test).squeeze()
mean_absolute_error(preds, y_test)
```

Out[40]:
```
0.038214562599121475
```

In [41]:
```
# 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)
```


actual
prediction
price [USD]
57500
55000
52500
50000
47500
45000
42500
40000
2021-08-28

In [ ]: