LSTM Stock Predictor Using Fear and Greed Index

In this notebook, you will build and train a custom LSTM RNN that uses a 10 day window of Bitcoin fear and greed index values to predict the 11th day closing price.

You will need to:

1. Prepare the data for training and testing
2. Build and train a custom LSTM RNN
3. Evaluate the performance of the model

Data Preparation

In this section, you will need to prepare the training and testing data for the model. The model will use a rolling 10 day window to predict the 11th day closing price.

You will need to:

1. Use the `window_data` function to generate the X and y values for the model.
2. Split the data into 70% training and 30% testing
3. Apply the MinMaxScaler to the X and y values
4. Reshape the X_train and X_test data for the model. Note: The required input format for the LSTM is:

```
reshape((X_train.shape[0], X_train.shape[1], 1))
```

In [1]:

```python
import numpy as np
import pandas as pd
import hvplot.pandas
```

In [2]:

```python
# just for fun: to sketch out approximate run time for this notebook
import time   # for stopwatch and sleep
t1 = time.perf_counter()   # track execution time
```

In [3]:

```python
# Set the random seed for reproducibility
# Note: This is for the homework solution, but it is good practice to comment
from numpy.random import seed
seed(1)
from tensorflow import random
random.set_seed(2)
```

In [4]:

```
# Load the fear and greed sentiment data for Bitcoin
file_btc_sent="C:/Users/CS_Knit_tinK_SC/Documents/GitHub/HW_11_DeepLrn_ML_Inpu
df = pd.read_csv(file_btc_sent, index_col="date", infer_datetime_format=True,
df = df.drop(columns="fng_classification")
df.head()
```

Out[4]:

	fng_value
date	
2019-07-29	19
2019-07-28	16
2019-07-27	47
2019-07-26	24
2019-07-25	42

In [5]:

```
# Load the historical closing prices for Bitcoin
file_btc_hist="C:/Users/CS_Knit_tinK_SC/Documents/GitHub/HW_11_DeepLrn_ML_Inpu
df2 = pd.read_csv(file_btc_hist, index_col="Date", infer_datetime_format=True,
df2 = df2.sort_index()
df2.tail()
```

Out[5]:

```
Date
2019-07-25    9882.429688
2019-07-26    9847.450195
2019-07-27    9478.320313
2019-07-28    9531.769531
2019-07-29    9529.889648
Name: Close, dtype: float64
```

In [6]:

```
# Join the data into a single DataFrame
df = df.join(df2, how="inner")
df.tail()
```

Out[6]:

	fng_value	Close
2019-07-25	42	9882.429688
2019-07-26	24	9847.450195
2019-07-27	47	9478.320313
2019-07-28	16	9531.769531
2019-07-29	19	9529.889648

In [7]:

```
df.head()
```

Out[7]:

	fng_value	Close
2018-02-01	30	9114.719727

	fng_value	Close
2018-02-02	15	8870.820313
2018-02-03	40	9251.269531
2018-02-04	24	8218.049805

In [8]:

```
# This function accepts the column number for the features (X) and the target
# It chunks the data up with a rolling window of Xt-n to predict Xt
# It returns a numpy array of X any y
def window_data(df, window, feature_col_number, target_col_number):
    X = []
    y = []
    for i in range(len(df) - window - 1):
        features = df.iloc[i:(i + window), feature_col_number]
        target = df.iloc[(i + window), target_col_number]
        X.append(features)
        y.append(target)
    return np.array(X), np.array(y).reshape(-1, 1)
```

In [9]:

```
# Predict Closing Prices using a 10 day window of previous fng values
# Then, experiment with window sizes anywhere from 1 to 10 and see how the mod
window_size = 10

# Column index 0 is the 'fng_value' column
# Column index 1 is the `Close` column
feature_column = 0
target_column = 1
X, y = window_data(df, window_size, feature_column, target_column)
```

In [10]:

```
# Use 70% of the data for training and the remainder for testing
split = int(0.7 * len(X))

X_train = X[: split]
X_test = X[split:]

y_train = y[: split]
y_test = y[split:]
split
```

Out[10]: 372

In [11]:

```
from sklearn.preprocessing import MinMaxScaler
# Use the MinMaxScaler to scale data between 0 and 1.


# Create a MinMaxScaler object
scaler = MinMaxScaler()

# Fit the MinMaxScaler object with the features data X
scaler.fit(X_train)

# Scale the features training and testing sets
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

# Fit the MinMaxScaler object with the target data Y
scaler.fit(y_train)

# Scale the target training and testing sets
y_train = scaler.transform(y_train)
y_test = scaler.transform(y_test)
```

In [12]:

```
# Reshape the features for the model
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

# Print some sample data after reshaping the datasets
print (f"X_train sample values:\n{X_train[:3]} \n")
print (f"X_test sample values:\n{X_test[:3]}")
```

```
X_train sample values:
[[[0.33333333]
  [0.10606061]
  [0.48484848]
  [0.24242424]
  [0.04545455]
  [0.        ]
  [0.41538462]
  [0.32307692]
  [0.53846154]
  [0.69230769]]

 [[0.10606061]
  [0.48484848]
  [0.24242424]
  [0.04545455]
  [0.        ]
  [0.42424242]
  [0.32307692]
  [0.53846154]
  [0.69230769]
  [0.33846154]]

 [[0.48484848]
  [0.24242424]
  [0.04545455]
  [0.        ]
```

```
  [0.42424242]
  [0.33333333]
  [0.53846154]
  [0.69230769]
  [0.33846154]
  [0.50769231]]]

X_test sample values:
[[[0.48484848]
  [0.57575758]
  [0.45454545]
  [0.60606061]
  [0.60606061]
  [0.53030303]
  [0.52307692]
  [0.49230769]
  [0.44615385]
  [0.83076923]]

 [[0.57575758]
  [0.45454545]
  [0.60606061]
  [0.60606061]
  [0.53030303]
  [0.53030303]
  [0.49230769]
  [0.44615385]
  [0.83076923]
  [0.86153846]]

 [[0.45454545]
  [0.60606061]
  [0.60606061]
  [0.53030303]
  [0.53030303]
  [0.5       ]
  [0.44615385]
  [0.83076923]
  [0.86153846]
  [0.76923077]]]
```

Build and Train the LSTM RNN

In this section, you will design a custom LSTM RNN and fit (train) it using the training data.

You will need to:

1. Define the model architecture
2. Compile the model
3. Fit the model to the training data

Hints:

You will want to use the same model architecture and random seed for both notebooks. This is necessary to accurately compare the performance of the FNG model vs the closing price model.

In [13]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
```

In [14]:
```
# Build the LSTM RNN model.
model = Sequential()

number_units = 5  # equals the time window
dropout_fraction = 0.2

# Layer 1
model.add(LSTM(
    units=number_units,
    # except for final layer, each time we add a new LSTM layer, we must set 
    # it just lets Keras know to connect each layer
    return_sequences=True,
    input_shape=(X_train.shape[1], 1))
    )
model.add(Dropout(dropout_fraction))
# Layer 2
model.add(LSTM(units=number_units, return_sequences=True))
model.add(Dropout(dropout_fraction))
# Layer 3
model.add(LSTM(units=number_units))
model.add(Dropout(dropout_fraction))
# Output layer
model.add(Dense(1))
```

In [15]:
```
# Compile the model
model.compile(optimizer="adam", loss="mean_squared_error")
```

In [16]:
```
# Summarize the model
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 10, 5)             140

 dropout (Dropout)           (None, 10, 5)             0

 lstm_1 (LSTM)               (None, 10, 5)             220

 dropout_1 (Dropout)         (None, 10, 5)             0

 lstm_2 (LSTM)               (None, 5)                 220

 dropout_2 (Dropout)         (None, 5)                 0

 dense (Dense)               (None, 1)                 6

=================================================================
```

```
Total params: 586
Trainable params: 586
Non-trainable params: 0
```

In [17]:

```
# Train the model
model.fit(X_train, y_train, epochs=10, shuffle=False, batch_size=1, verbose=1)
```

```
Epoch 1/10
372/372 [==============================] - 4s 4ms/step - loss: 0.0854
Epoch 2/10
372/372 [==============================] - 2s 4ms/step - loss: 0.0476
Epoch 3/10
372/372 [==============================] - 2s 4ms/step - loss: 0.0499
Epoch 4/10
372/372 [==============================] - 2s 4ms/step - loss: 0.0481
Epoch 5/10
372/372 [==============================] - 2s 4ms/step - loss: 0.0482A: 1s - l
oss - ETA:
Epoch 6/10
372/372 [==============================] - 2s 4ms/step - loss: 0.0462
Epoch 7/10
372/372 [==============================] - 2s 4ms/step - loss: 0.0496
Epoch 8/10
372/372 [==============================] - 2s 4ms/step - loss: 0.0475
Epoch 9/10
372/372 [==============================] - 2s 4ms/step - loss: 0.0438A: 0s -
Epoch 10/10
372/372 [==============================] - 2s 4ms/step - loss: 0.0456
```

Out[17]:

```
<keras.callbacks.History at 0x241e0d52ac8>
```

Model Performance

In this section, you will evaluate the model using the test data.

You will need to:

1. Evaluate the model using the `X_test` and `y_test` data.
2. Use the X_test data to make predictions
3. Create a DataFrame of Real (y_test) vs predicted values.
4. Plot the Real vs predicted values as a line chart

Hints

Remember to apply the `inverse_transform` function to the predicted and y_test values to recover the actual closing prices.

In [18]:

```
# Evaluate the model
model.evaluate(X_test, y_test)
```

```
5/5 [==============================] - 1s 4ms/step - loss: 0.1495
```

Out[18]: 0.14951150119304657

In [19]:
```
# Make some predictions
predicted = model.predict(X_test)
```

In [20]:
```
# Recover the original prices instead of the scaled version
predicted_prices = scaler.inverse_transform(predicted)
real_prices = scaler.inverse_transform(y_test.reshape(-1, 1))
```

In [21]:
```
print(predicted[0:10])
print(predicted_prices[0:10])
```

```
[[0.19965065]
 [0.2075889 ]
 [0.21591792]
 [0.22720937]
 [0.23535277]
 [0.24335726]
 [0.25167897]
 [0.2564672 ]
 [0.2574811 ]
 [0.2552745 ]]
[[4884.002 ]
 [4949.667 ]
 [5018.5635]
 [5111.966 ]
 [5179.3267]
 [5245.539 ]
 [5314.3755]
 [5353.9834]
 [5362.3706]
 [5344.1177]]
```

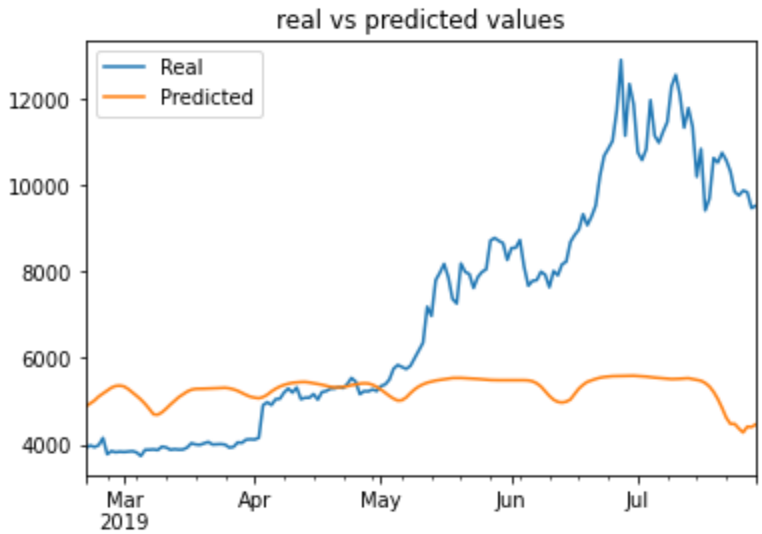
In [22]:
```
# Create a DataFrame of Real and Predicted values
stocks = pd.DataFrame({
    "Real": real_prices.ravel(),
    "Predicted": predicted_prices.ravel()
}, index = df.index[-len(real_prices): ])
stocks.head()
```

Out[22]:

	Real	Predicted
2019-02-20	3924.239990	4884.001953
2019-02-21	3974.050049	4949.666992
2019-02-22	3937.040039	5018.563477
2019-02-23	3983.530029	5111.965820
2019-02-24	4149.089844	5179.326660

In [25]:
```
# Plot the real vs predicted values as a line chart
stocks.plot(title='real vs predicted values')
```

Out[25]: <AxesSubplot:title={'center':'real vs predicted values'}>



In [24]:

```
# time estimate.. just to keep an eye on such aspects a little bit!
t2 = time.perf_counter()
execution_time = t2 - t1
print("execution time: " + str(round(execution_time, 1)) + " seconds")
```

```
execution time: 31.4 seconds
```

In [ ]: