```python
from time import time
import timeit

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import LSTM
from keras.models import Sequential
from sklearn.preprocessing import MinMaxScaler

# Sources Referenced
# https://medium.com/neuronio/predicting-stock-prices-with-lstm-349f5a0974d4
# https://heartbeat.fritz.ai/using-a-keras-long-shortterm-memory-lstm-model-to-predict-stock-prices-a08c9f69aa74
# https://keras.io/getting-started/sequential-model-guide/
# https://www.kdnuggets.com/2018/11/keras-long-short-term-memory-lstm-model-predict-stock-prices.html
# https://stackabuse.com/time-series-analysis-with-lstm-using-pythons-keras-library/
# https://machinelearningmastery.com/make-predictions-long-short-term-memory-models-keras/
# https://stackoverflow.com/questions/48760472/how-to-use-the-keras-model-to-forecast-for-future-dates-or-events

# Requirements:
# Python3.6+
# TensorFlow library
# keras library


class LongShortTermMemory:
    def __init__(self, name, timestep, epoch, batch, output_dim, dropout, data_column, csv_train_file, csv_future,
                 csv_test_file=None):
        self.name = name
        self.timestep = timestep
        self.epoch = epoch
        self.batch = batch
        self.output_dim = output_dim
        self.dropout_percent = dropout / 100
        self.data_column = data_column
        self.csv_train = csv_train_file
        self.csv_test = csv_test_file
        self.csv_future = csv_future
        self.scaler = MinMaxScaler(feature_range=(0, 1))
        self.X_train = None
```

```
43          self.y_train = None
44          self.model = None
45          self.training_data = None
46
47      # array is an np array
48      def max_range(self, array):
49          # determine max size of the data based as a
    multiple of the batch size
50          max_range = int((len(array) - self.timestep) / self
    .batch)
51          max_range = max_range * self.batch + self.timestep
52          return max_range
53
54      # reshape the array
55      def reshape_2d_array(self, array):
56          max_range = self.max_range(array)
57          array_3d = []
58          for i in range(self.timestep, max_range):
59              array_3d.append(array[i - self.timestep:i, 0])
60          array_3d = np.array(array_3d)
61          array_3d = np.reshape(array_3d, (array_3d.shape[0
    ], array_3d.shape[1], 1))
62          return array_3d
63
64      # offset a numpy array for displaying the graph
    correctly
65      def offset_array(self, array, offset):
66          offset_arr = []
67          for i in range(offset):
68              offset_arr.append([None])
69          for i in range(len(array)):
70              offset_arr.append(array[i])
71          offset_arr = np.array(offset_arr)
72          return offset_arr
73
74      # predicts the future!
75      def predict_future(self):
76          # predict based on the last few weeks
77          X_data = self.get_data(self.csv_future)
78          predictions = self.model.predict(X_data)
79          future = [predictions]
80          future = np.array(future)
81          future = self.scaler.inverse_transform(future[0])
    # rescale prices
82          return future
83
84      # use the pandas dataframe as the index
85      @staticmethod
86      def set_date_as_index(data_frame):
87          data_frame['Date'] = pd.to_datetime(data_frame['
```

```
 87 Date'])
 88         data_frame.set_index('Date', inplace=True)
 89         return data_frame
 90 
 91     # get data from a csv file
 92     def get_data(self, csv):
 93         # import the close data
 94         data = pd.read_csv(csv)
 95         # set the index to be the dates
 96         data = self.set_date_as_index(data)
 97         data_set = [[x] for x in data[self.data_column].
    values]
 98         data_set = np.array(data_set)
 99         # scale the data for performance
100         data_set_scaled = self.scaler.fit_transform(
    data_set)
101         # put data into 3d array for LSTM digestion
102         X_data = []
103         # determine max size of the data based as a
    multiple of the batch size
104         max_range = self.max_range(data)
105         for i in range(self.timestep, max_range):
106             X_data.append(data_set_scaled[i - self.
    timestep:i, 0])
107         # convert the data to numpy arrays
108         X_data = np.array(X_data)
109         # reshape the data for the LSTM model
110         X_data = np.reshape(X_data, (X_data.shape[0],
    X_data.shape[1], 1))
111         return X_data
112 
113     def get_training_data(self):
114         # import the close data
115         self.training_data = pd.read_csv(self.csv_train)
116         self.training_data = self.set_date_as_index(self.
    training_data)
117         # get the close values
118         training_set = [[x] for x in self.training_data[
    self.data_column].values]
119         training_set = np.array(training_set)
120         # scale the data for performance
121         training_set_scaled = self.scaler.fit_transform(
    training_set)
122         # put data into 3d array for LSTM digestion
123         X_train = []
124         y_train = []
125         for i in range(self.timestep, len(self.
    training_data)):
126             X_train.append(training_set_scaled[i - self.
    timestep:i, 0])
```

```
127             y_train.append(training_set_scaled[i, 0])
128         # convert the data to numpy arrays
129         X_train = np.array(X_train)
130         self.y_train = np.array(y_train)
131         self.X_train = np.reshape(X_train,
132                                   (X_train.shape[0],
    X_train.shape[1], 1))  # reshape the data for the LSTM
    model
133
134     # Build the LSTM model
135     def build_model(self):
136         model = Sequential()  # initialize the model
137         # add layers to our model
138         model.add(LSTM(units=self.output_dim,
    return_sequences=True, input_shape=(self.X_train.shape[1
    ], 1)))
139         model.add(Dropout(self.dropout_percent))
140         model.add(LSTM(units=self.output_dim,
    return_sequences=True))
141         model.add(Dropout(self.dropout_percent))
142         model.add(LSTM(units=self.output_dim,
    return_sequences=True))
143         model.add(Dropout(self.dropout_percent))
144         model.add(LSTM(units=self.output_dim))
145         model.add(Dropout(self.dropout_percent))
146         model.add(Dense(units=1))  # add layer to specify
    output of 1 layer
147         # compile with the Adam optimizer and computer the
     mean squared error
148         model.compile(optimizer='adam', loss='
    mean_squared_error')
149         # run the model, this may take several minutes
150         model.fit(self.X_train, self.y_train, epochs=self.
    epoch, batch_size=self.batch)
151         self.model = model
152
153     # make a prediction based on csv_test
154     def predict(self):
155         # test the model
156         dataset_test = pd.read_csv(self.csv_test)  #
    import the test set that we will make predictions on
157         self.dataset_test = self.set_date_as_index(
    dataset_test)
158         real_stock_price = [[x] for x in dataset_test[self
    .data_column].values]
159         real_stock_price = np.array(real_stock_price)
160         dataset_total = self.training_data[self.
    data_column]
161         # transform the new dataset for performance
162         inputs = dataset_total.values
```

```
163         inputs = inputs.reshape(-1, 1)
164         inputs = self.scaler.transform(inputs)
165         # Reshape the data to a 3d array
166         X_test = []
167         for i in range(self.timestep, len(dataset_test)):
168             X_test.append(inputs[i - self.timestep:i, 0])
169         X_test = np.array(X_test)
170         X_test = np.reshape(X_test, (X_test.shape[0],
    X_test.shape[1], 1))
171         # make the prediction
172         test_predict_price = self.model.predict(X_test)
173         test_predict_price = self.scaler.inverse_transform
    (test_predict_price)    # rescale prices
174         # # predict the future
175         future = self.predict_future()
176         future = self.offset_array(future, len(
    real_stock_price))
177         # plot the data
178         plt.plot(real_stock_price, color='darkgrey', label
    =f'{self.name} Stock Price')
179         plt.plot(test_predict_price, color='orange', label
    =f'Predicted {self.name} Stock Price')
180         plt.plot(future, color='darkviolet', label=f'
    Predicted {self.name} Future Stock Price')
181         plt.title(f'{self.name} Test Price Prediction')
182         plt.xlabel('Time')
183         plt.ylabel(f'{self.name} Stock Price')
184         plt.legend()
185         # save the plot
186         timestamp = int(time())  # time since epoch
187         plot = plt.gcf()
188         plt.show()
189         plt.draw()
190         plot.savefig(f'plots/{self.name}_{self.epoch}_{
    timestamp}.png', dpi=100)
191
192     def run_lstm(self):
193         self.get_training_data()
194         self.build_model()
195         self.predict()
196
197
198 if __name__ == '__main__':
199     # uncomment one section below to run the model for
    those companies
200     # list of stock files
201     # All arrays below must have companies data in the
    same order
202     # companies = ['Adidas', 'Bitcoin', 'Costco', 'S&P 500
    ']
```

```
203     # train_stocks = ['data/adidas/ADDYY.csv', 'data/
    bitcoin/BTC-USD.csv', 'data/costco/COST.csv', 'data/s&p/^
    GSPC.csv']
204     # test_stocks = ['data/adidas/ADDYY.csv', 'data/
    bitcoin/BTC-USD.csv', 'data/costco/COST.csv', 'data/s&p/^
    GSPC.csv']
205     # future_stocks = ['data/adidas/ADDYY-future.csv', '
    data/bitcoin/BTC-USD-test.csv', 'data/costco/COST-test.csv
    ', 'data/s&p/^GSPC-test.csv']
206
207     # companies = ['ADP', 'Honeywell', 'Medtronic']
208     # train_stocks = ['data/adp/ADP.csv', 'data/honeywell/
    HON.csv', 'data/medtronic/MDT.csv']
209     # test_stocks = ['data/adp/ADP.csv', 'data/honeywell/
    HON.csv', 'data/medtronic/MDT.csv']
210     # future_stocks = ['data/adp/ADP-future.csv', 'data/
    honeywell/HON-future.csv', 'data/medtronic/MDT-future.csv
    ']
211
212     # companies = ['FireEye', 'GoPro', 'Tesla']
213     # train_stocks = ['data/fireeye/FEYE.csv', 'data/gopro
    /GPRO.csv', 'data/tesla/TSLA.csv']
214     # test_stocks = ['data/fireeye/FEYE.csv', 'data/gopro/
    GPRO.csv', 'data/tesla/TSLA.csv']
215     # future_stocks = ['data/fireeye/FEYE-future.csv', '
    data/gopro/GPRO-future.csv', 'data/tesla/TSLA-future.csv']
216
217     companies = ['Tesla']
218     train_stocks = ['data/tesla/TSLA.csv']
219     test_stocks = ['data/tesla/TSLA.csv']
220     future_stocks = ['data/tesla/TSLA-future.csv']
221
222     columns = ['Open', 'High', 'Low', 'Close', 'Adj Close'
    ]
223     start = timeit.default_timer()
224     for i in range(len(train_stocks)):
225         for col in columns:
226             lstm = LongShortTermMemory(name=f'{companies[i
    ]}_{col}',
227                                        timestep=7,
228                                        epoch=100,
229                                        batch=7,
230                                        output_dim=50,
231                                        dropout=20,
232                                        data_column=col,
233                                        csv_train_file=
    train_stocks[i],
234                                        csv_test_file=
    test_stocks[i],
235                                        csv_future=
```

```
235 future_stocks[i])
236             lstm.run_lstm()
237     print(f'Total run time: {timeit.default_timer() -
    start}')
238
```