EE798 ENDTERM ASSIGNMENT

The Libraries I used are Pandas, Numpy, Scipy, Keras, Json. To Save the model parameters, architecture and weights, the following code.

```
import numpy as np
import json
from keras.models import model_from_json
# ... Rest of the code ...
# Save the trained model architecture
model_architecture = model.to_json()
with open('model_architecture.json', 'w') as json_file:
    json_file.write(model_architecture)
# Save the trained model weights
model.save_weights('model_weights.h5')
# Save the model parameters
parameters = {
    'n_steps_in': n_steps_in,
    'n_steps_out': n_steps_out,
    'n_features': n_features
}
with open('model_parameters.json', 'w') as json_file:
    json.dump(parameters, json_file)
```

Code to load the model is incorporated in predict function.

```
# Load the saved model architecture
with open('model_architecture.json', 'r') as json_file:
    model_architecture = json_file.read()
loaded_model = model_from_json(model_architecture)

# Load the saved model weights
loaded_model.load_weights('model_weights.h5')

# Load the model parameters
with open('model_parameters.json', 'r') as json_file:
    parameters = json.load(json_file)
n_steps_in = parameters['n_steps_in']
n_steps_out = parameters['n_steps_out']
n_features = parameters['n_features']

# Perform evaluation using the loaded model
# ...
```

The code to preprocess the data to convert to multivariate numpy array

```
import pandas as pd
        import numpy as np
        from scipy.interpolate import CubicSpline
        interpolate_columns = data.columns[data.columns != 'Date']

        for column in interpolate_columns:
            column_values = data[column].values
            not_null_indices = ~pd.isnull(column_values)
            indices = pd.Series(range(len(column_values)))
            cs = CubicSpline(indices[not_null_indices], column_values[not_null_indices])
            data[column] = cs(indices)



        from numpy import array
        from numpy import hstack
```

```
        def split_sequences( sequences, n_steps_in, n_steps_out):
                  X, y = list(), list()
                  for i in range(len(sequences)):
                      # find the end of this pattern
                      end_ix = i + n_steps_in
                      out_end_ix = end_ix + n_steps_out-1
                      # check if we are beyond the dataset
                      if out_end_ix > len(sequences):
                          break
                      # gather input and output parts of the pattern
                      seq_x, seq_y = sequences[i:end_ix, :], sequences[end_ix-1:out_end_ix, :]
                      X.append(seq_x)
                      y.append(seq_y)

                  return array(X), array(y)


        # define input sequence
        array_Open  = np.array(data['Open'])
        array_High  = np.array(data['High'])
        array_Low   = np.array(data['Low'])
        array_Close = np.array(data['Close'])
        array_Adj   = np.array(data['Adj Close'])
        array_Volume =  np.array(data['Volume'])

        # convert to [rows, columns] structure
        array_Open = array_Open.reshape((len(array_Open), 1))
        array_High = array_High.reshape((len(array_High), 1))
        array_Low = array_Low.reshape((len(array_Low), 1))
        array_Close = array_Close.reshape((len(array_Close), 1))
        array_Adj = array_Adj.reshape((len(array_Adj), 1))
        array_Volume = array_Volume.reshape((len(array_Volume), 1))

        # horizontally stack columns
        dataset = hstack((array_Open,
                          array_High,
                          array_Low,
                          array_Close,
                          #array_Adj,
                          #array_Volume
                          ))
        n_steps_in, n_steps_out = 7, 2
        # covert into input/output
        X, y = split_sequences(dataset, n_steps_in, n_steps_out)
```

The model uses bidirectional stacked lstm with 200 lstm units and 4 hidden layers

```
from numpy import array
from numpy import hstack
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import RepeatVector
from keras.layers import TimeDistributed
from keras.layers import Bidirectional

n_features = X.shape[2]

model = Sequential()
#model.add(LSTM(200, activation='relu', input_shape=(n_steps_in, n_features)))
```

```
model.add(Bidirectional(LSTM(200, activation='relu'), input_shape=(n_steps_in, n_features)))
model.add(RepeatVector(n_steps_out))
model.add(Bidirectional
                       (LSTM(200, activation='relu', return_sequences=True)))
model.add(TimeDistributed(Dense(n_features)))
model.compile(optimizer='adam', loss='mse')

model.fit(X, y, epochs=500, verbose=0)
# demonstrate prediction
```

End