```python
# -*- coding: utf-8 -*-
"""
Created on Sat Nov  5 11:48:04 2022

@author: Nayeem Badshah, EMP_ID: 191323
"""

import math
import pandas_datareader as web # version should be 0.10.0
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential, load_model
from keras.layers import Dense, LSTM
import matplotlib.pyplot as plt
from openpyxl import load_workbook
from datetime import date, datetime
import scipy.stats
plt.style.use('fivethirtyeight')



scaler = MinMaxScaler(feature_range=(0,1))
def plot_stock(df, stock_name):
    plt.figure(figsize=(16,8))
    plt.title(f"Clos Price History {stock_name}")
    plt.plot(df['Close'])
    plt.xlabel('Date', fontsize=(18))
    plt.ylabel('Clos Price', fontsize=(18))
    plt.show()


def scale_data(dataset):
    scaled_data = scaler.fit_transform(dataset)
    return scaled_data

def filter_data(df,column):
    # create a new dataframe with only close column
    data = df.filter([column])
    return data
```

```
def train_test_split(scaled_data, training_data_len, window_size):
    # <<<<<<<<<<<<<<<<<<<<<<<<<<Training Data>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
    # create the scaled training dataset
    train_data = scaled_data[0:training_data_len,:]

    # split the data into x_train and y_train data sets
    x_train = []
    y_train = []
    p_n = window_size
    for i in range(p_n, len(train_data)):
        x_train.append(train_data[i-p_n:i,0])
        y_train.append(train_data[i, 0])

    # convert the x_train and y_train to numpy arrays
    x_train, y_train = np.array(x_train), np.array(y_train)

    # reshape the data
    x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
    # <<<<<<<<<<<<<<<<<<<<<<<<<<Training Data>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

    # <<<<<<<<<<<<<<<<<<<<<<<<<<Test Data>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
    # create the testting data set
    # a new array containing scaled values from index 1543 to 2003
    test_data = scaled_data[training_data_len - window_size: , :]
    # creat x_test and y_test
    x_test = []
    y_test = scaled_data[training_data_len:,:]
    for i in range(window_size, len(scaled_data) - (training_data_len - window_size)):
        x_test.append(test_data[i-window_size:i,0])

    # convert the data to numpy array
    x_test = np.array(x_test)

    # reshape the data
    x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
    # <<<<<<<<<<<<<<<<<<<<<<<<<<Test Data>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

    return x_train, y_train, x_test, y_test
```

```python
def get_training_data_len(df, training_size):
    # get the number of rows to train the model on
    return math.ceil(len(df) * training_size)


def model_build_train(x_train, y_train, lstm_input_layer, lstm_middle_layer, dense_layer, epochs):
    model = Sequential()
    model.add(LSTM(lstm_input_layer, return_sequences=True, input_shape = (x_train.shape[1],1)))
    model.add(LSTM(lstm_middle_layer, return_sequences=False))
    model.add(Dense(dense_layer))
    model.add(Dense(1))

    # compile the model
    model.compile(optimizer='adam', loss='mean_squared_error') # for Manik root mean squared error

    # train the model
    model.fit(x_train, y_train, batch_size=1, epochs=epochs)
    return model

def model_predict(model, x_test):
    # get the model's predicted price values
    preds = model.predict(x_test)
    return preds

def run_model_iteration(df, hyper_params):
    # filter data with only close column
    data = filter_data(df, 'Close')
    # scale data with Min Max Scaler
    scaled_dataset = scale_data(data.values)
    run_data = []
    models = []
    for index, hyper_param in hyper_params.iterrows():
        trainig_data_size =  hyper_param[0]
        hyper_param = hyper_param[1:].astype(int)
        training_data_len = get_training_data_len(data, trainig_data_size)

        # split data into train test
        x_train, y_train, x_test, y_test = \
            train_test_split(scaled_dataset, training_data_len, hyper_param['window_size'])
```

```
        # model build and train
        model = model_build_train(x_train = x_train,\
                                  y_train = y_train,
                                  lstm_input_layer = hyper_param['lstm_input_layer'],
                                  lstm_middle_layer = hyper_param['lstm_middle_layer'],
                                  dense_layer = hyper_param['dense_layer'],
                                  epochs = hyper_param['epochs'])
        models.append(model)
        y_preds = scaler.inverse_transform(model_predict(model, x_test))

        y_test = scaler.inverse_transform(y_test)

        rmse = get_rmse(y_preds, y_test)
        accuracy = get_accuracy(y_test, y_preds)

        train_data, validation_data = \
            consolidate_train_validation_data(data, y_preds, training_data_len)
        # plot_after_pred(train_data, validation_data)

        # val_data = validation_data.copy()
        run_info = {
            'Run'                : index,
            'trainig_data_len'   : trainig_data_size,
            'window_size'        : hyper_param['window_size'],
            'lstm_input_layer'   : hyper_param['lstm_input_layer'],
            'lstm_middle_layer'  : hyper_param['lstm_middle_layer'],
            'dense_layer'        : hyper_param['dense_layer'],
            'epochs'             : hyper_param['epochs'],
            'RMSE'               : rmse,
            'Accuracy'           : accuracy
        }
        run_data.append(run_info)
        print(run_info)
    return run_data, models

def get_accuracy(real, predict):
    real = np.array(real) + 1
    predict = np.array(predict) + 1
    percentage = 1 - np.sqrt(np.mean(np.square((real - predict) / real)))
    return percentage * 100
```

```python
def get_rmse(y_preds, y_test):
    # get root mean squared error (RMSE)
    rmse = np.sqrt(np.mean(y_preds - y_test)**2)
    return rmse

def df_index_alter(df):
    df = df.sort_values(by=['Date'],ascending=True)
    # setting the index as date
    df.index = df['Date']
    # drop date column as it is now index column
    df = df.drop(['Date'],axis=1)
    return df

def consolidate_train_validation_data(df, preds, training_data_len):
    # plot the data
    train = df[:training_data_len].copy()
    valid = df[training_data_len:].copy()
    valid['Predictions'] = preds
    return train, valid

def plot_after_pred(train, valid, title, stock_name):
    # visulize the data
    plt.figure(figsize=(16,8))
    plt.title(title)
    plt.xlabel('Date', fontsize=18)
    plt.ylabel(f'Closr Price {stock_name}', fontsize=18)
    plt.plot(train['Close'])
    plt.plot(valid[['Close', 'Predictions']])
    plt.legend(['Train', 'Actual', 'Predictions'], loc='lower right')
    plt.show()
    return

def forecast(model, window_size, scaled_dataset, days):
    last_window_size = len(scaled_dataset)-window_size
    last_winow_df = scaled_dataset[last_window_size:]
    last_winow_df = last_winow_df.reshape((1, window_size, 1))

    future_days = days
    future_preds = []
    temp_data = []
```

```python
    for day in range(future_days):
        if day == 0:
            y_hat = model.predict(last_winow_df)
            y_hat_ = scaler.inverse_transform(y_hat)
            future_preds.append(y_hat_[0][0])
            temp_data = list(last_winow_df.flatten())
            temp_data.append(y_hat[0][0])
            temp_data = temp_data[1:]
        else:
            temp_data = np.array(temp_data).reshape((1, window_size, 1))
            y_hat = model.predict(temp_data)
            y_hat_ = scaler.inverse_transform(y_hat)
            future_preds.append(y_hat_[0][0])
            temp_data = list(temp_data.flatten())
            temp_data.append(y_hat[0][0])
            temp_data = temp_data[1:]
    return future_preds


def mean_confidence_interval(data, confidence):
    a = 1.0 * np.array(data)
    n = len(a)
    m, se = np.mean(a), scipy.stats.sem(a)
    h = se * scipy.stats.t.ppf((1 + confidence) / 2., n-1)
    return m, m-h, m+h

def check_forecast(forecasted_data, upper_bound, lower_bound):
    forecast_df = pd.DataFrame({'Forecast':forecasted_data})
    forecast_df['upper_bound'] = upper_bound
    forecast_df['lower_bound'] = lower_bound
    forecast_df['is_in_between'] = forecast_df['Forecast'].between(forecast_df['lower_bound'],\
                                                               forecast_df['upper_bound'],  inclusive='both')
    return forecast_df

def model_evalution_forecasting(model_path, df_path, window_size, training_data_size, future_days, stock_name, ci):
    # get NASDAQ dataset
    df = pd.read_excel(df_path)

    # alter index to date
    df = df_index_alter(df)
```

```python
    # plot the whole data
    plot_stock(df, stock_name)

    model = load_model(model_path)
    # filter data with only close column
    data = filter_data(df, 'Close')

    # scale data with Min Max Scaler
    scaled_dataset = scale_data(data.values)

    # get training data length
    training_data_len = get_training_data_len(data, training_data_size)
    # split data into train test
    x_train, y_train, x_test, y_test =\
        train_test_split(scaled_dataset, training_data_len, window_size)
    t1 = scaler.inverse_transform(model.predict(x_test))
    train, valid = consolidate_train_validation_data(df[['Close']], t1, training_data_len)
    future_close = forecast(model, window_size, scaled_dataset, future_days)

    d = pd.DataFrame(future_close,columns=['Predictions'])
    d.index = pd.date_range(date(2022, 11, 7), date(2022, 11, 16), freq='D')
    valid1 = valid.append(d)
    plot_after_pred(train, valid1, stock_name, stock_name)
    mean, lower_bound, upper_bound = mean_confidence_interval(future_close,confidence=ci)
    forecast_df = check_forecast(future_close, upper_bound, lower_bound)
    forecast_df.index = pd.date_range(date(2022, 11, 7), date(2022, 11, 16), freq='D')

    return df, train, valid, valid1, forecast_df

#####################################################################################################################################

if __name__ == "__main__":


    #<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<Model Training>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>


    nifty_df_path = "PATH_TO_NIFTY_DATA"
    ftse_df_path = "PATH_TO_FTSE_DATA"
    nasdaq_df_path = "PATH_TO_NASDAQ_DATA"
```

```
df_nifty = pd.read_excel(nifty_df_path)
df_ftse = pd.read_excel(ftse_df_path)
df_nasdaq = pd.read_excel(nasdaq_df_path)

# import hyper params
hyper_params_path = "PATH_TO_HYPER_PARAMETERS_DATA"
hyper_params = pd.read_excel(hyper_params_path, "hyper_params")


# train nifty data
run_df_nifty, models_nifty = run_model_iteration(df_nifty, hyper_params)
nifty_best_model_idx = run_df_nifty[run_df_nifty['RMSE'] == run_df_nifty['RMSE'].min()]['Run'].values[0]
# save best model
models_nifty[nifty_best_model_idx].save("SAVE_MODEL_TO_SPECIFIED_PATH")

# train FTSE data
run_df_ftse, models_ftse = run_model_iteration(df_ftse, hyper_params)
ftse_best_model_idx = run_df_ftse[run_df_ftse['RMSE'] == run_df_ftse['RMSE'].min()]['Run'].values[0]
# save best model
models_ftse[ftse_best_model_idx].save("SAVE_MODEL_TO_SPECIFIED_PATH")

# train NASDAQ data
run_df_nasdaq, models_nasdaq = run_model_iteration(df_nasdaq, hyper_params)
nasdaq_best_model_idx = run_df_nasdaq[run_df_nasdaq['RMSE'] == run_df_nasdaq['RMSE'].min()]['Run'].values[0]
# save best model
models_nasdaq[nasdaq_best_model_idx].save("SAVE_MODEL_TO_SPECIFIED_PATH")


#<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<Model Training>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>



#<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<Model Evalution and Forecast>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
nasdaq_model_path = "NASDAQ_MODEL_PATH"

df_nasdaq, train_nasdaq, valid_nasdaq, valid1_nasdaq, forecast_df_nasdaq =\
    model_evalution_forecasting(model_path = nasdaq_model_path,\
                                df_path = nasdaq_df_path,
                                window_size = 70,
```

```
                                training_data_size = 0.8,
                                future_days = 5,
                                stock_name = 'NASDAQ',
                                ci = 0.99949)


ftse_model_path = "FTSE_MODEL_PATH"

df_ftse, train_ftse, valid_ftse, valid1_ftse, forecast_df_ftse =\
    model_evalution_forecasting(model_path = ftse_model_path,\
                                df_path = ftse_df_path,
                                window_size = 60,
                                training_data_size = 0.8,
                                future_days = 5,
                                stock_name = 'FTSE',
                                ci = 0.99949)


nifty_model_path = "NIFTY_MODEL_PATH"

df_nifty, train_nifty, valid_nifty, valid1_nifty, forecast_df_nifty =\
    model_evalution_forecasting(model_path = nifty_model_path,\
                                df_path = nifty_df_path,
                                window_size = 30,
                                training_data_size = 0.8,
                                future_days = 5,
                                stock_name = 'NIFTY50',
                                ci = 0.99949)
#<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<Model Evalution and Forecast>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

#---------------------------Thank You-----------------------------#
# </>
```