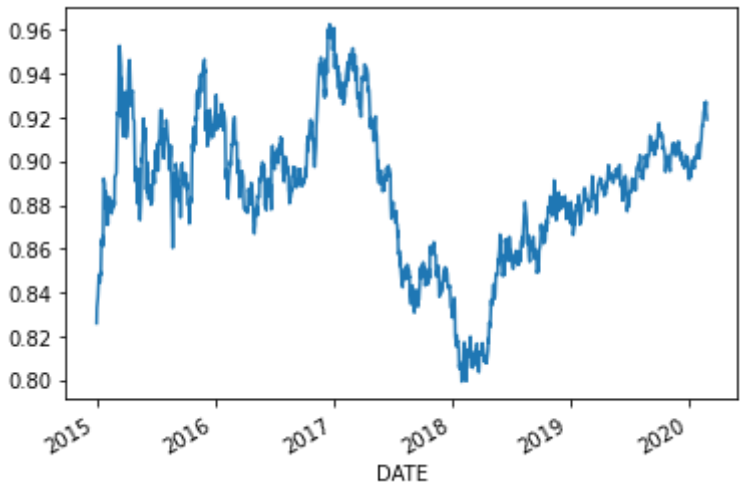
```
from pandas import DataFrame
from pandas import Series
from pandas import concat
from pandas import read_csv
from pandas import datetime
from sklearn.metrics import mean_squared_error
#from sklearn.metrics import mean_absolute_error
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from math import sqrt
from matplotlib import pyplot
#import matplotlib.pyplot as plt
import numpy
import pandas
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: FutureWarning: T
  """
```

```
# date-time parsing function for loading the dataset
def parser(x):
    return datetime.strptime(x, '%m/%d/%y')

# load dataset
series=read_csv('/content/rate.csv',parse_dates=[0],date_parser=parser)
series=pandas.Series(series['currency rate'].values,index=series['DATE'])
#n=len(series)
series.plot()
pyplot.show()
```




```
# create a differenced series
def difference(dataset, interval=1):
    diff = list()
```

```
        for i in range(interval, len(dataset)):
            value = dataset[i] - dataset[i - interval]
            diff.append(value)
        return Series(diff)

    # transform data to be stationary
    raw_values = series.values
    diff_values = difference(raw_values, 1)


    # frame a sequence as a supervised learning problem
    def timeseries_to_supervised(data, lag=1):
        df = DataFrame(data)
        columns = [df.shift(i) for i in range(1, lag+1)]
        columns.append(df)
        df = concat(columns, axis=1)
        df.fillna(0, inplace=True)
        return df

    # transform data to be supervised learning
    supervised = timeseries_to_supervised(diff_values, 1)
    supervised_values = supervised.values


    supervised_values
```

```
     array([[ 0.    ,  0.0071],
            [ 0.0071,  0.0048],
            [ 0.0048,  0.0031],
            ...,
            [ 0.0019, -0.0052],
            [-0.0052, -0.0008],
            [-0.0008, -0.0022]])
```

```
    # split data into train and test-sets
    train, test = supervised_values[0:-365], supervised_values[-365:]
    print(len(train),len(test))
```

```
     978 365
```

```
    # scale train and test data to [-1, 1]
    def scale(train, test):
      # fit scaler
        scaler = MinMaxScaler(feature_range=(-1, 1))
        scaler = scaler.fit(train)
      # transform train
        train = train.reshape(train.shape[0], train.shape[1])
        train_scaled = scaler.transform(train)
      # transform test
        test = test.reshape(test.shape[0], test.shape[1])
        test_scaled = scaler.transform(test)
        return scaler, train_scaled, test_scaled
```

```
# transform the scale of the data
scaler, train_scaled, test_scaled = scale(train, test)


# fit an LSTM network to training data
def fit_lstm(train, batch_size, nb_epoch, neurons):
    X, y = train[:, 0:-1], train[:, -1]
    X = X.reshape(X.shape[0], 1, X.shape[1])
    model = Sequential()
    model.add(LSTM(neurons, batch_input_shape=(batch_size, X.shape[1], X.shape[2]), s
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    for i in range(nb_epoch):
        model.fit(X, y, epochs=1, batch_size=batch_size, verbose=0, shuffle=False)
        model.reset_states()
    return model

# fit the model
lstm_model = fit_lstm(train_scaled, 1, 30, 4)


# forecast the entire training dataset to build up state for forecasting
train_reshaped = train_scaled[:, 0].reshape(len(train_scaled), 1, 1)
lstm_model.predict(train_reshaped, batch_size=1)
```

```
            [0.12875609],
            [0.12763354],
            [0.12797971],
            [0.13423452],
            [0.12973182],
            [0.12996273],
            [0.12894931],
            [0.12996417],
            [0.13192867],
            [0.13064566],
            [0.1297333 ],
            [0.12798706],
            [0.1278232 ],
            [0.12766427],
            [0.12840702],
            [0.12697682],
            [0.12671754],
            [0.12688942],
            [0.13097307],
            [0.12860921],
            [0.12837015],
            [0.12999614],
            [0.12777199],
            [0.1283387 ],
            [0.12981829],
            [0.1282286 ],
            [0.12740754],
            [0.12635665],
            [0.12604117],
            [0.12551877]
```

```
               [0.1259167],
               [0.12792172],
               [0.1278238 ],
               [0.12673596],

               [0.12650543],
               [0.12611307],
               [0.12435926],
               [0.12488362],
               [0.12629257],
               [0.1294515 ],
               [0.13071889],
               [0.13552065],
               [0.13398594],
               [0.13100015],
               [0.13497669],
               [0.1348112 ],
               [0.13320842],
               [0.13215698],
               [0.12985833],
               [0.12804258],
               [0.12841678],
               [0.12654322],
               [0.12879977],
               [0.12761359],
               [0.12656483],
               [0.12858564],
               [0.12809339],
               [0.12869479],
               [0.13154279],
               [0.12887478],
```

```
# invert differenced value
def inverse_difference(history, yhat, interval=1):
    return yhat + history[-interval]

# make a one-step forecast
def forecast_lstm(model, batch_size, X):
    X = X.reshape(1, 1, len(X))
    yhat = model.predict(X, batch_size=batch_size)
    return yhat[0,0]

# inverse scaling for a forecasted value
def invert_scale(scaler, X, value):
    new_row = [x for x in X] + [value]
    array = numpy.array(new_row)
    array = array.reshape(1, len(array))
    inverted = scaler.inverse_transform(array)
    return inverted[0, -1]


# walk-forward validation on the test data
predictions = list()
for i in range(len(test_scaled)):
# make one-step forecast
    X, y = test_scaled[i, 0:-1], test_scaled[i, -1]
```

```
    yhat = forecast_lstm(lstm_model, 1, X)
# invert scaling
    yhat = invert_scale(scaler, X, yhat)
# invert differencing
    yhat = inverse_difference(raw_values, yhat, len(test_scaled)+1-i)
# store forecast
    predictions.append(yhat)
    expected = raw_values[len(train) + i + 1]
    print('day=%d, Predicted=%f, Expected=%f' % (i+1, yhat, expected))

# report performance
rmse = sqrt(mean_squared_error(raw_values[-365:], predictions))
#mae = sqrt(mean_absolute_error(raw_values[-365:], predictions))
print('Test RMSE: %.3f' % rmse)
#print('Test MAE: %.3f' % mae)
# line plot of observed vs predicted
pyplot.plot(raw_values[-365:])
pyplot.plot(predictions)
pyplot.show()
```

```
day=323, Predicted=0.901152, Expected=0.894800
day=324, Predicted=0.894972, Expected=0.893000
day=325, Predicted=0.893149, Expected=0.891900
day=326, Predicted=0.892041, Expected=0.891900
day=327, Predicted=0.892019, Expected=0.895000
day=328, Predicted=0.895077, Expected=0.895900
day=329, Predicted=0.895962, Expected=0.893000
day=330, Predicted=0.893095, Expected=0.896500
day=331, Predicted=0.896547, Expected=0.900400
day=332, Predicted=0.900425, Expected=0.900400
day=333, Predicted=0.900427, Expected=0.899200
day=334, Predicted=0.899238, Expected=0.898100
day=335, Predicted=0.898150, Expected=0.898500
day=336, Predicted=0.898546, Expected=0.896800
day=337, Predicted=0.896874, Expected=0.897900
day=338, Predicted=0.897955, Expected=0.901800
day=339, Predicted=0.901832, Expected=0.901300
day=340, Predicted=0.901344, Expected=0.902200
day=341, Predicted=0.902228, Expected=0.901400
day=342, Predicted=0.901444, Expected=0.904600
day=343, Predicted=0.904619, Expected=0.907100
day=344, Predicted=0.907111, Expected=0.907500
day=345, Predicted=0.907515, Expected=0.907300
day=346, Predicted=0.907322, Expected=0.908200
day=347, Predicted=0.908219, Expected=0.906400
day=348, Predicted=0.906453, Expected=0.901400
day=349, Predicted=0.901536, Expected=0.904200
day=350, Predicted=0.904279, Expected=0.905500
day=351, Predicted=0.905567, Expected=0.909200
day=352, Predicted=0.909235, Expected=0.910700
day=353, Predicted=0.910724, Expected=0.913600
day=354, Predicted=0.913602, Expected=0.916500
day=355, Predicted=0.916491, Expected=0.916100
day=356, Predicted=0.916106, Expected=0.919700
day=357, Predicted=0.919686, Expected=0.922400
day=358, Predicted=0.922385, Expected=0.923200
day=359, Predicted=0.923191, Expected=0.922900
day=360, Predicted=0.922905, Expected=0.926600
day=361, Predicted=0.926590, Expected=0.925400
day=362, Predicted=0.925425, Expected=0.927300
day=363, Predicted=0.927307, Expected=0.922100
day=364, Predicted=0.922212, Expected=0.921300
day=365, Predicted=0.921389, Expected=0.919100
Test RMSE: 0.003
```