4.1.2 LSTM Time Series to Predict Ground Level PM10 with Lagged Variables

Chapter 4.1.2

```
1 from google.colab import drive
2 drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
# Import librries
import pandas as pd
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
import glob #for maps
import plotly.graph_objects as go
import plotly.offline
import matplotlib.pyplot as plt
from datetime import datetime
import warnings
import itertools
import numpy as np
from statsmodels.tsa.stattools import adfuller
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from math import sqrt
warnings.filterwarnings("ignore")
import plotly.express as px
import seaborn as sns
import pandas as pd
import plotly.graph_objects as go
import matplotlib.pyplot as plt
%matplotlib inline


from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Dense, Dropout, LSTM
import keras.models
import tensorflow
from keras.models import Sequential
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dense
```

Data Reading and Additional Processing

```
fin_data = pd.read_csv('/content/gdrive/MyDrive/Sent/final_data_May20_Sept22.
```

```
ar','Month','Day','Hour'])['nitrogendioxide_tropospheric_column','nitrogendi
```

```
fin_data_exp=fin_data_exp.reset_index()
# fin_data_exp=fin_data_exp.drop(columns=['level_0', 'index', 'Year', 'Month
# fin_data_exp.head()
```

```
fin_data_exp=fin_data_exp.drop(columns=['Year', 'Month', 'Day', 'Hour'])
fin_data_exp.head()
```

	nitrogendioxide_tropospheric_column	nitrogendioxide_tropospheric_column
0	0.000008	
1	0.000017	
2	0.000046	
3	0.000033	
4	0.000039	

Feature Engineering for LSTM

Reference:

The model scrip was adopted from the following webpage:

https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/

https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

https://datascience.stackexchange.com/questions/76826/why-are-predictions-from-my-lstm-neural-network-lagging-behind-true-values

```

# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
  n_vars = 1 if type(data) is list else data.shape[1]
  df = DataFrame(data)
  cols, names = list(), list()
  # input sequence (t-n, ... t-1)
  for i in range(n_in, 0, -1):
    cols.append(df.shift(i))
```

```
    names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
  # forecast sequence (t, t+1, ... t+n)
  for i in range(0, n_out):
    cols.append(df.shift(-i))
    if i == 0:
      names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
    else:

      names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
  # put it all together
  agg = concat(cols, axis=1)
  agg.columns = names
  # drop rows with NaN values
  if dropnan:
    agg.dropna(inplace=True)
  return agg

# load dataset

values = fin_data_exp.values
# integer encode direction
encoder = LabelEncoder()


# ensure all data is float
values = values.astype('float32')
# normalize features

#values = lstm_data(values, 4)

scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)


# frame as supervised learning
reframed = series_to_supervised(scaled, 1, 1)
#reframed.drop(reframed.columns[[9,10,11]], axis=1, inplace=True)
#reframed.drop(reframed.columns[[6,7,8,9,10]], axis=1, inplace=True)
#reframed.drop(reframed.columns[[6,7,8,9,10]], axis=1, inplace=True) #window
reframed.head()

```

	var1(t-1)	var2(t-1)	var3(t-1)	var4(t-1)	var5(t-1)	var6(t-1)	var1(t)	var2(t)	va
1	0.155634	0.013603	0.015260	0.427553	0.546352	0.129479	0.164217	0.005729	0.0
2	0.164217	0.005729	0.005855	0.654652	0.739136	0.129479	0.191809	0.036969	0.0
3	0.191809	0.036969	0.033096	0.426778	0.564740	0.201648	0.179365	0.018086	0.0
4	0.179365	0.018086	0.015359	0.500033	0.553890	0.201648	0.185115	0.034446	0.0
5	0.185115	0.034446	0.032485	0.407516	0.745865	0.373201	0.188677	0.039848	0.0

```
1 reframed.shape
```

```
(985, 18)
```

```
1 reframed1 = reframed.iloc[:,:-1]
2 reframed1 = lstm_data(reframed1, 3)
```

```
1 reframed.shape
```

```
(985, 18)
```

```
1 values1.shape
```

```
(985, 18)
```

Model 1 LSTM (window one)

Double-click (or enter) to edit

```
values = reframed.values
#values1 = reframed.values
x, x_test, y, y_test = train_test_split(values[:, :-1],values[:,-1],test_size
x_train, x_val, train_y, val_y = train_test_split(x,y,test_size = 0.2,train_s
# reshape input to be 3D [samples, timesteps, features]

train_X = x_train.reshape((x_train.shape[0], 1, x_train.shape[1]))
val_X = x_val.reshape((x_val.shape[0], 1, x_val.shape[1]))
test_X = x_test.reshape((x_test.shape[0], 1, x_test.shape[1]))
print(train_X.shape, train_y.shape, val_X.shape, val_y.shape, test_X.shape, y
```

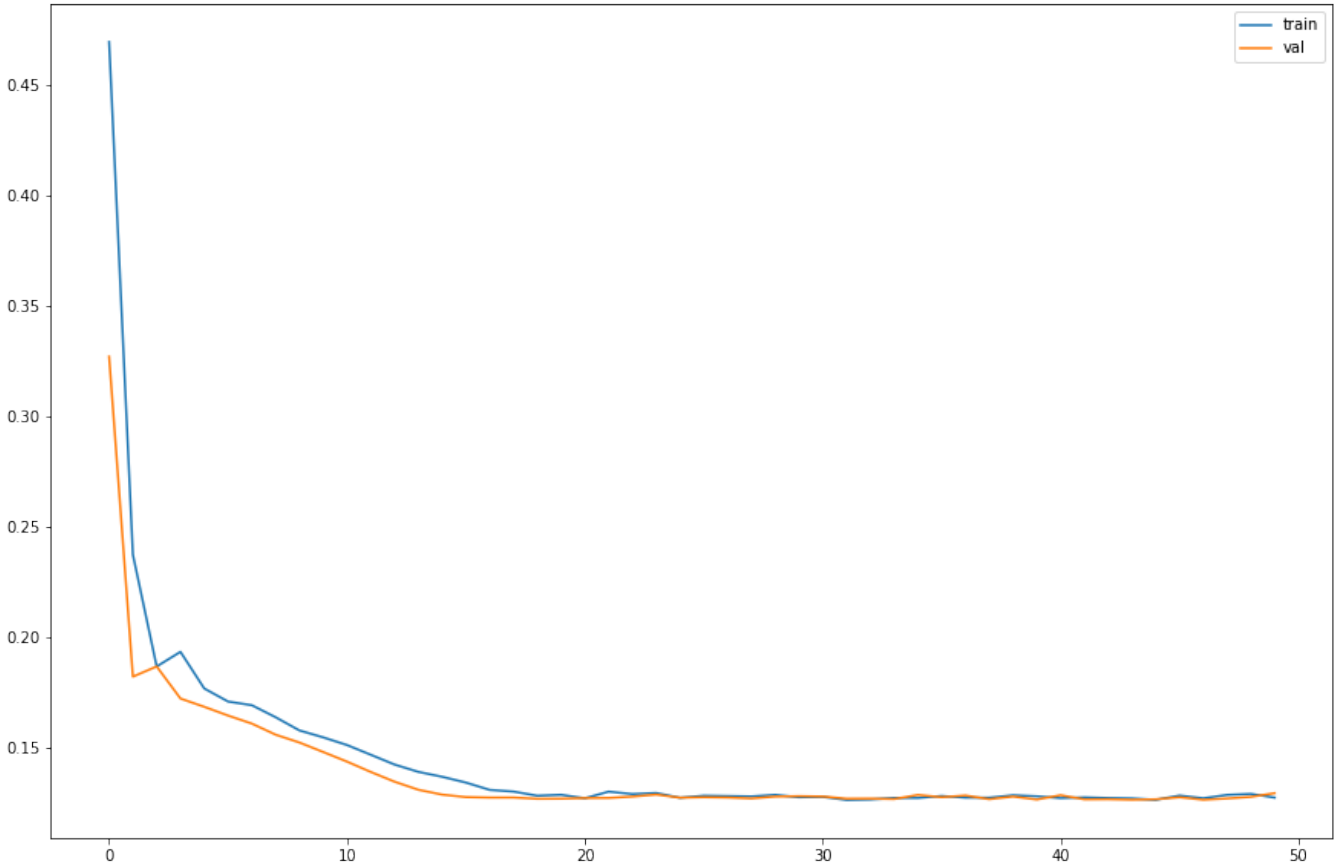
```
(709, 1, 6) (709,) (178, 1, 6) (178,) (99, 1, 6) (99,)
```

```

model = Sequential()
model.add(LSTM(500, input_shape=(train_X.shape[1], train_X.shape[2])))
#model.add(LSTM(1000, input_shape=(train_X.shape[1], train_X.shape[2])))
#model.add(LSTM(2000, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dropout(0.2))
model.add(Dense(1))

model.compile(loss='mae', optimizer='adam') #mean absolute error



history = model.fit(train_X, train_y, epochs=50, batch_size=64, validation_da
# plot history
plt.figure(figsize=(15, 10))

pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='val')
pyplot.legend()
pyplot.show()

y_hat = model.predict(train_X)
y_hat_val = model.predict(val_X)

plt.figure(figsize=(15, 10))
pyplot.plot(y_hat, label='predict', color = 'red')
pyplot.plot(train_y, label='true', color = 'yellow')
pyplot.legend()

pyplot.show()


plt.figure(figsize=(15, 10))
pyplot.plot(y_hat_val, label='predict', color = 'blue')
```

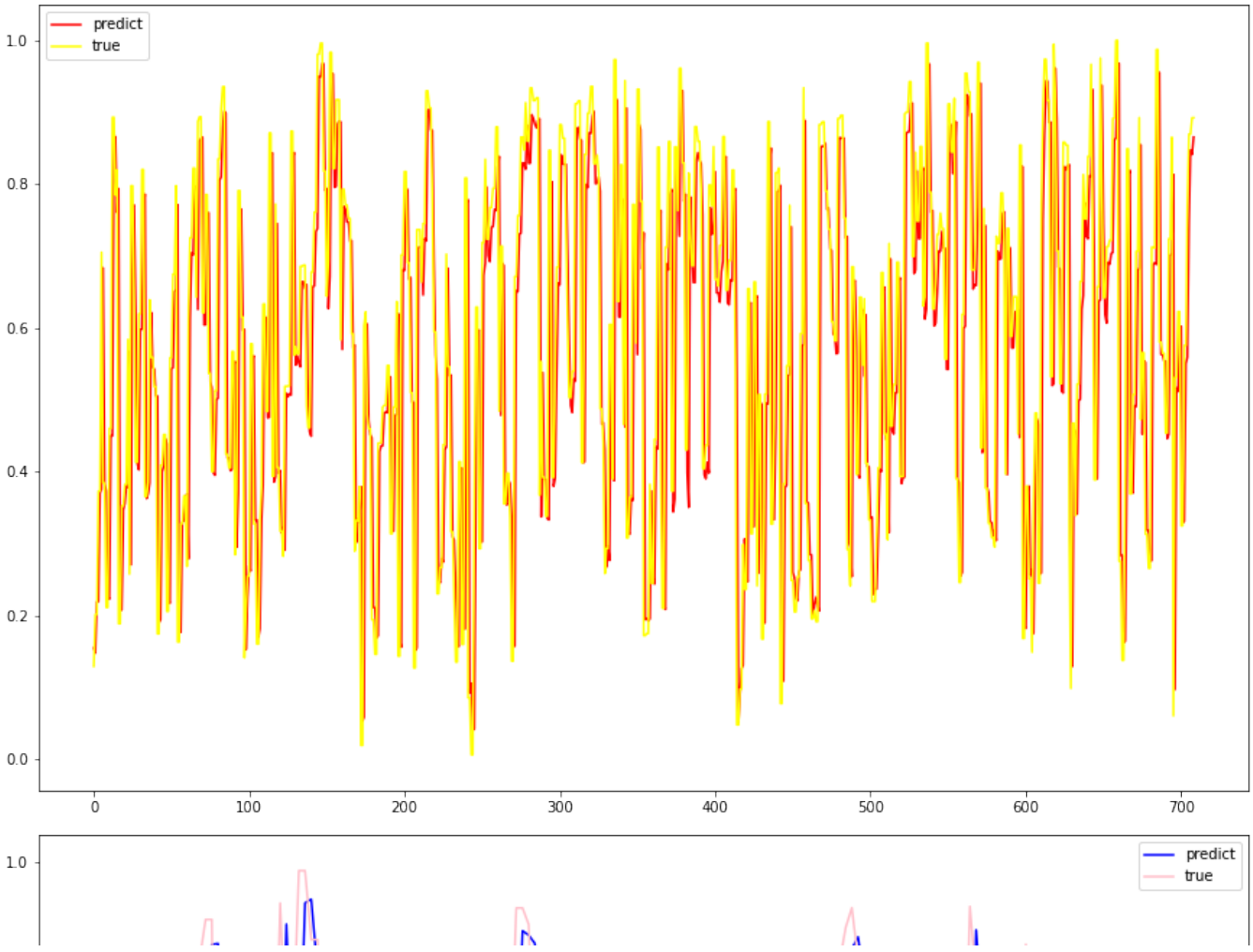
```
pyplot.plot(val_y, label='true', color = 'pink')
pyplot.legend()

pyplot.show()


```

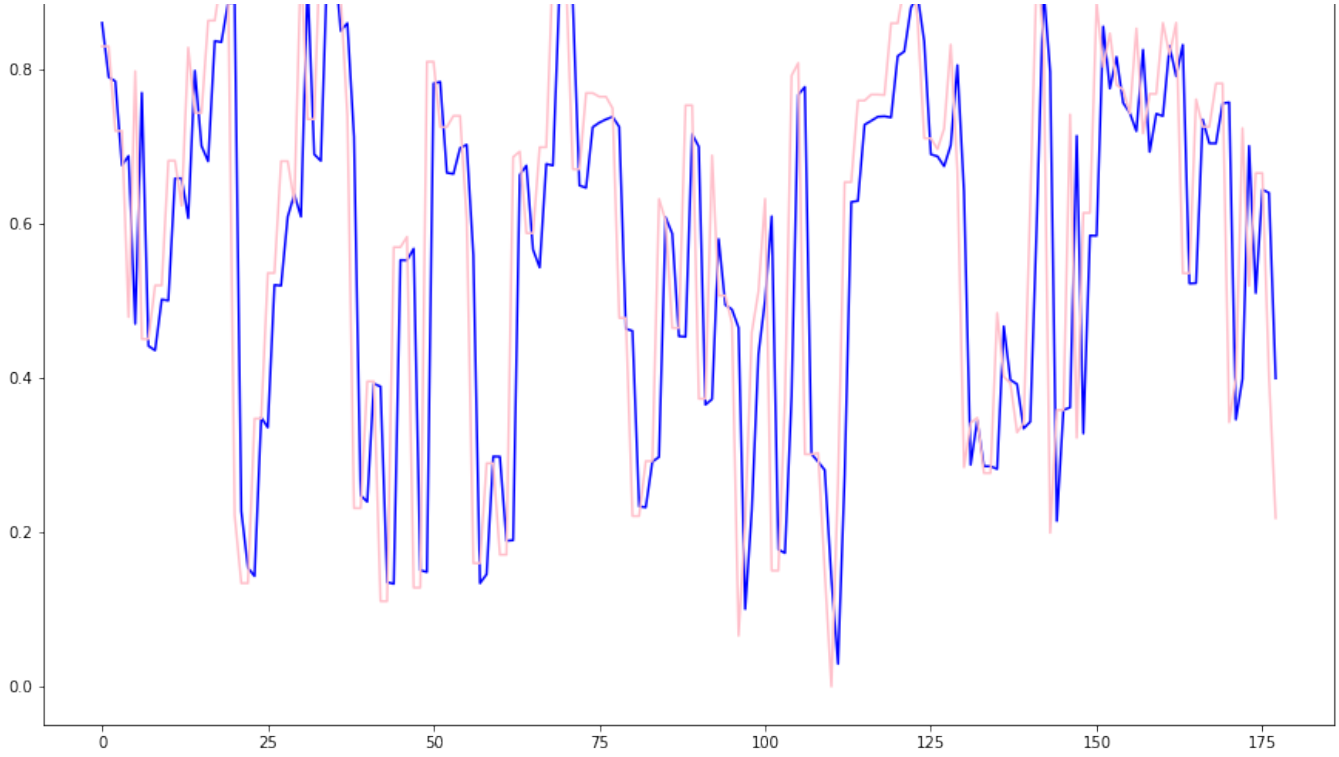
```
Epoch 1/50
12/12 - 3s - loss: 0.4691 - val_loss: 0.3270 - 3s/epoch - 219ms/step
Epoch 2/50
12/12 - 0s - loss: 0.2375 - val_loss: 0.1824 - 246ms/epoch - 20ms/step
Epoch 3/50
12/12 - 0s - loss: 0.1870 - val_loss: 0.1870 - 243ms/epoch - 20ms/step
Epoch 4/50
12/12 - 0s - loss: 0.1936 - val_loss: 0.1725 - 259ms/epoch - 22ms/step
Epoch 5/50
12/12 - 0s - loss: 0.1771 - val_loss: 0.1688 - 239ms/epoch - 20ms/step
Epoch 6/50
12/12 - 0s - loss: 0.1712 - val_loss: 0.1649 - 246ms/epoch - 20ms/step
Epoch 7/50
12/12 - 0s - loss: 0.1695 - val_loss: 0.1612 - 256ms/epoch - 21ms/step
Epoch 8/50
12/12 - 0s - loss: 0.1641 - val_loss: 0.1562 - 255ms/epoch - 21ms/step
Epoch 9/50
12/12 - 0s - loss: 0.1581 - val_loss: 0.1527 - 245ms/epoch - 20ms/step
Epoch 10/50
12/12 - 0s - loss: 0.1550 - val_loss: 0.1484 - 243ms/epoch - 20ms/step
Epoch 11/50
12/12 - 0s - loss: 0.1515 - val_loss: 0.1441 - 241ms/epoch - 20ms/step
Epoch 12/50
12/12 - 0s - loss: 0.1471 - val_loss: 0.1394 - 282ms/epoch - 23ms/step
Epoch 13/50
12/12 - 0s - loss: 0.1427 - val_loss: 0.1350 - 242ms/epoch - 20ms/step
Epoch 14/50
12/12 - 0s - loss: 0.1394 - val_loss: 0.1313 - 260ms/epoch - 22ms/step
Epoch 15/50
12/12 - 0s - loss: 0.1372 - val_loss: 0.1291 - 240ms/epoch - 20ms/step
Epoch 16/50
12/12 - 0s - loss: 0.1346 - val_loss: 0.1280 - 250ms/epoch - 21ms/step
Epoch 17/50
12/12 - 0s - loss: 0.1313 - val_loss: 0.1278 - 244ms/epoch - 20ms/step
Epoch 18/50
12/12 - 0s - loss: 0.1305 - val_loss: 0.1278 - 246ms/epoch - 20ms/step
Epoch 19/50
12/12 - 0s - loss: 0.1286 - val_loss: 0.1273 - 258ms/epoch - 21ms/step
Epoch 20/50
12/12 - 0s - loss: 0.1291 - val_loss: 0.1273 - 253ms/epoch - 21ms/step
Epoch 21/50
12/12 - 0s - loss: 0.1275 - val_loss: 0.1276 - 233ms/epoch - 19ms/step
Epoch 22/50
12/12 - 0s - loss: 0.1305 - val_loss: 0.1276 - 244ms/epoch - 20ms/step
Epoch 23/50
12/12 - 0s - loss: 0.1294 - val_loss: 0.1283 - 247ms/epoch - 21ms/step
```

```
Epoch 24/50
12/12 - 0s - loss: 0.1299 - val_loss: 0.1292 - 252ms/epoch - 21ms/step
Epoch 25/50
12/12 - 0s - loss: 0.1277 - val_loss: 0.1278 - 261ms/epoch - 22ms/step
Epoch 26/50
12/12 - 0s - loss: 0.1287 - val_loss: 0.1280 - 244ms/epoch - 20ms/step
Epoch 27/50
12/12 - 0s - loss: 0.1285 - val_loss: 0.1278 - 252ms/epoch - 21ms/step
Epoch 28/50
12/12 - 0s - loss: 0.1283 - val_loss: 0.1274 - 254ms/epoch - 21ms/step
Epoch 29/50
12/12 - 0s - loss: 0.1290 - val_loss: 0.1283 - 234ms/epoch - 20ms/step
Epoch 30/50
12/12 - 0s - loss: 0.1280 - val_loss: 0.1285 - 244ms/epoch - 20ms/step
Epoch 31/50
12/12 - 0s - loss: 0.1281 - val_loss: 0.1283 - 242ms/epoch - 20ms/step
Epoch 32/50
12/12 - 0s - loss: 0.1267 - val_loss: 0.1273 - 247ms/epoch - 21ms/step
Epoch 33/50
12/12 - 0s - loss: 0.1269 - val_loss: 0.1273 - 247ms/epoch - 21ms/step
Epoch 34/50
12/12 - 0s - loss: 0.1276 - val_loss: 0.1271 - 240ms/epoch - 20ms/step
Epoch 35/50
12/12 - 0s - loss: 0.1276 - val_loss: 0.1290 - 239ms/epoch - 20ms/step
Epoch 36/50
12/12 - 0s - loss: 0.1285 - val_loss: 0.1280 - 251ms/epoch - 21ms/step
Epoch 37/50
12/12 - 0s - loss: 0.1278 - val_loss: 0.1287 - 244ms/epoch - 20ms/step
Epoch 38/50
12/12 - 0s - loss: 0.1278 - val_loss: 0.1271 - 246ms/epoch - 20ms/step
Epoch 39/50
12/12 - 0s - loss: 0.1289 - val_loss: 0.1283 - 247ms/epoch - 21ms/step
Epoch 40/50
12/12 - 0s - loss: 0.1284 - val_loss: 0.1269 - 258ms/epoch - 22ms/step
Epoch 41/50
12/12 - 0s - loss: 0.1275 - val_loss: 0.1289 - 247ms/epoch - 21ms/step
Epoch 42/50
12/12 - 0s - loss: 0.1279 - val_loss: 0.1270 - 245ms/epoch - 20ms/step
Epoch 43/50
12/12 - 0s - loss: 0.1276 - val_loss: 0.1271 - 254ms/epoch - 21ms/step
Epoch 44/50
12/12 - 0s - loss: 0.1274 - val_loss: 0.1268 - 250ms/epoch - 21ms/step
Epoch 45/50
12/12 - 0s - loss: 0.1268 - val_loss: 0.1270 - 253ms/epoch - 21ms/step
Epoch 46/50
12/12 - 0s - loss: 0.1287 - val_loss: 0.1279 - 244ms/epoch - 20ms/step
Epoch 47/50
12/12 - 0s - loss: 0.1275 - val_loss: 0.1268 - 250ms/epoch - 21ms/step
Epoch 48/50
12/12 - 0s - loss: 0.1290 - val_loss: 0.1274 - 264ms/epoch - 22ms/step
Epoch 49/50
12/12 - 0s - loss: 0.1295 - val_loss: 0.1281 - 244ms/epoch - 20ms/step
Epoch 50/50
12/12 - 0s - loss: 0.1278 - val_loss: 0.1298 - 246ms/epoch - 20ms/step
```



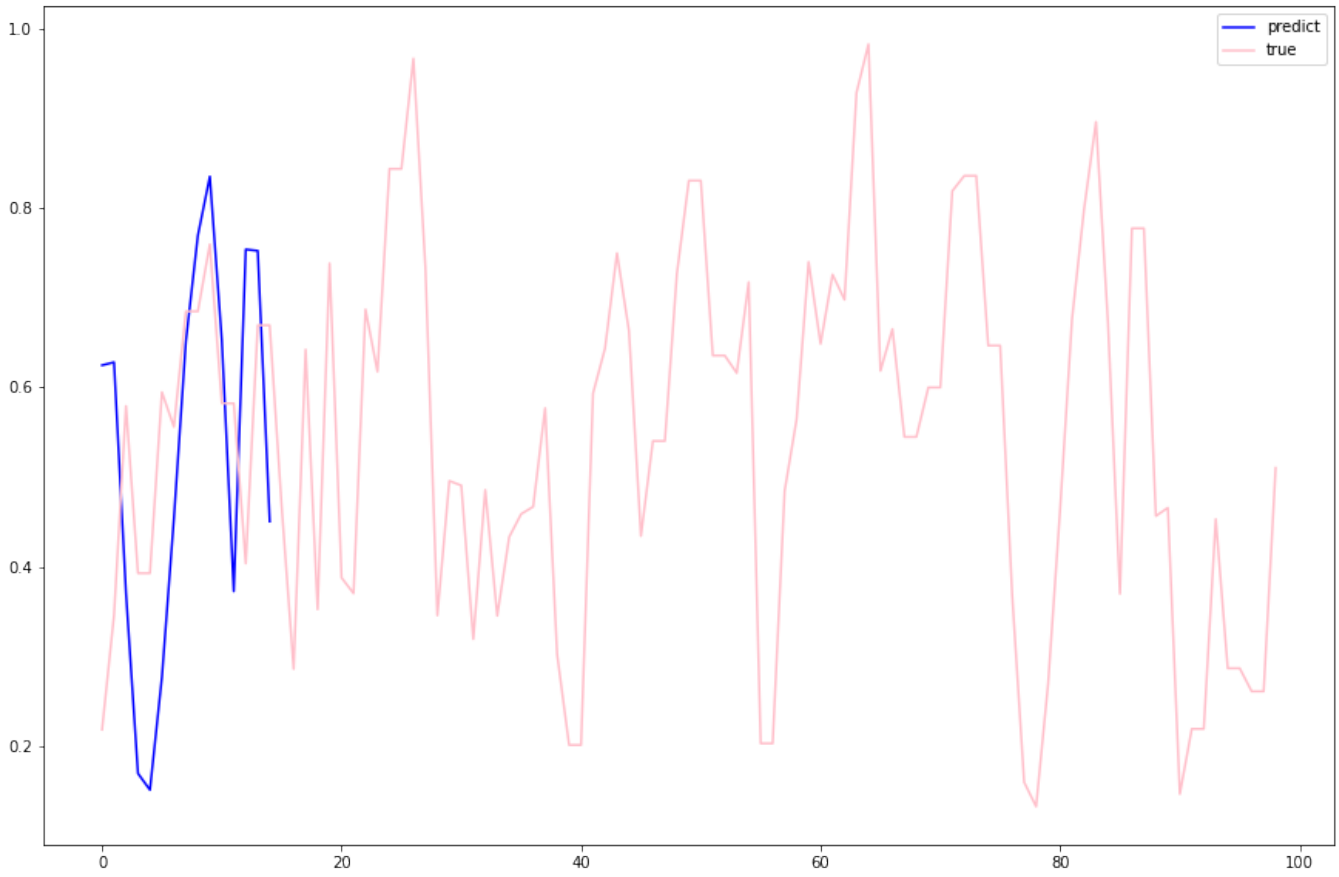

```
23/23 [==============================] - 1s 4ms/step
6/6 [==============================] - 0s 5ms/step
```




0.8
0.6
0.4
0.2
0.0
0
25
50
75
100
125
150
175

```
#Test set

y_hat_test = model.predict(test_X)

plt.figure(figsize=(15, 10))

pyplot.plot(y_hat_test[75:90], label='predict', color = 'blue')
pyplot.plot(y_test, label='true', color = 'pink')
pyplot.legend()

pyplot.show()



print('MSE comparison ----')
print(f"LSTM:\t{mean_squared_error(y_hat_test, y_test):.4f}")
print('R2_score ----')
print(f"LSTMf:\t{r2_score(y_hat_test, y_test):.4f}")

```

```
4/4 [==============================] - 0s 5ms/step
```




```
MSE comparison ----
LSTM:   0.0320
R2_score ----
LSTMf:  0.1574
```