1.TIME SERIES DATA CLEANING,LOADING AND HANDLING TIMES SERIES DATA AND PRE -PROCESSING

AIM:

To implement programs for time series data cleaning, loading and handling times series data and pre- processing techniques.

PROCEDURE AND CODE:

1. **Import Libraries & Load Data**
   - Import numpy, pandas, tensorflow, and other required libraries.
   - Read the dataset (cleaned_weather.csv) and check for missing values.
2. **Preprocess Data**
   - Extract the temperature ('T' column) and visualize it using .plot().
   - Define a function to create input (X) and output (y) using a sliding window approach.
3. **Prepare Training, Validation, and Test Sets**
   - Split X and y into X_train, y_train, X_val, y_val, and X_test, y_test.
4. **Build LSTM Model**
   - Create a Sequential model with an LSTM layer and Dense layers.
   - Compile the model using Mean Squared Error (MSE) and Adam optimizer**.**
5. **Train the Model**
   - Train the model using model.fit() with validation data and save the best model.
6. **Evaluate & Plot Predictions**
   - Load the saved model and make predictions on train, validation, and test sets.
   - Plot predicted vs. actual values to visualize model performance.

CODE:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
```

```
    for filename in filenames:
        print(os.path.join(dirname, filename))
import pandas as pd
import numpy as np
import tensorflow as tf
     df = pd.read_csv('/content/cleaned_weather.csv',index_col='date')
df.isna().sum()
df.head(5)
df.shape
df.columns
temp=df['T']
temp.plot()
def df_to_X_y(df, window_size):
    X, y = [], []
    df_as_np = df.to_numpy()
    for i in range(len(df_as_np) - window_size):
        row = df_as_np[i:i + window_size]  # Collect a window of size `window_size`
        X.append(row)
        label = df_as_np[i + window_size]  # Label corresponds to the next value
        y.append(label)
    return np.array(X), np.array(y)
WINDOW_SIZE = 5
X, y = df_to_X_y(temp, WINDOW_SIZE)
X.shape, y.shape


X



Y
X_train, y_train = X[:40000], y[:40000]
X_val, y_val = X[40000:45000], y[40000:45000]
X_test, y_test = X[45000:], y[45000:]

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import *
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.losses import MeanSquaredError
```

```
from tensorflow.keras.metrics import RootMeanSquaredError
from tensorflow.keras.optimizers import Adam


X_train.shape, y_train.shape




model1 = Sequential()
model1.add(InputLayer((5,1)))
model1.add(LSTM(64))
model1.add(Dense(8, 'relu'))
model1.add(Dense(1, 'linear'))

model1.summary()
cp = ModelCheckpoint('/kaggle/working/best_model.keras', save_best_only=True)
model1.compile(loss=MeanSquaredError(), optimizer=Adam(learning_rate=0.0001), metrics =
[RootMeanSquaredError()])

model1.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=10, callbacks=[cp])

rom tensorflow.keras.models import load_model
model1 = load_model('/kaggle/working/best_model.keras')

train_predictions = model1.predict(X_train).flatten()
train_results = pd.DataFrame(data={'Train Predictions':train_predictions, 'Actuals':y_train})
train_results
import matplotlib.pyplot as plt

plt.plot(train_results['Train Predictions'])
plt.plot(train_results['Actuals'])

plt.plot(train_results['Train Predictions'][:100])
plt.plot(train_results['Actuals'][:100])

val_predictions = model1.predict(X_val).flatten()
val_results = pd.DataFrame(data={'Val Predictions':val_predictions, 'Actuals':y_val})
```

val_results
plt.plot(val_results['Val Predictions'][:100])
plt.plot(val_results['Actuals'][:100])
test_predictions = model1.predict(X_test).flatten()
test_results = pd.DataFrame(data={'Test Predictions':test_predictions, 'Actuals':y_test})
test_results

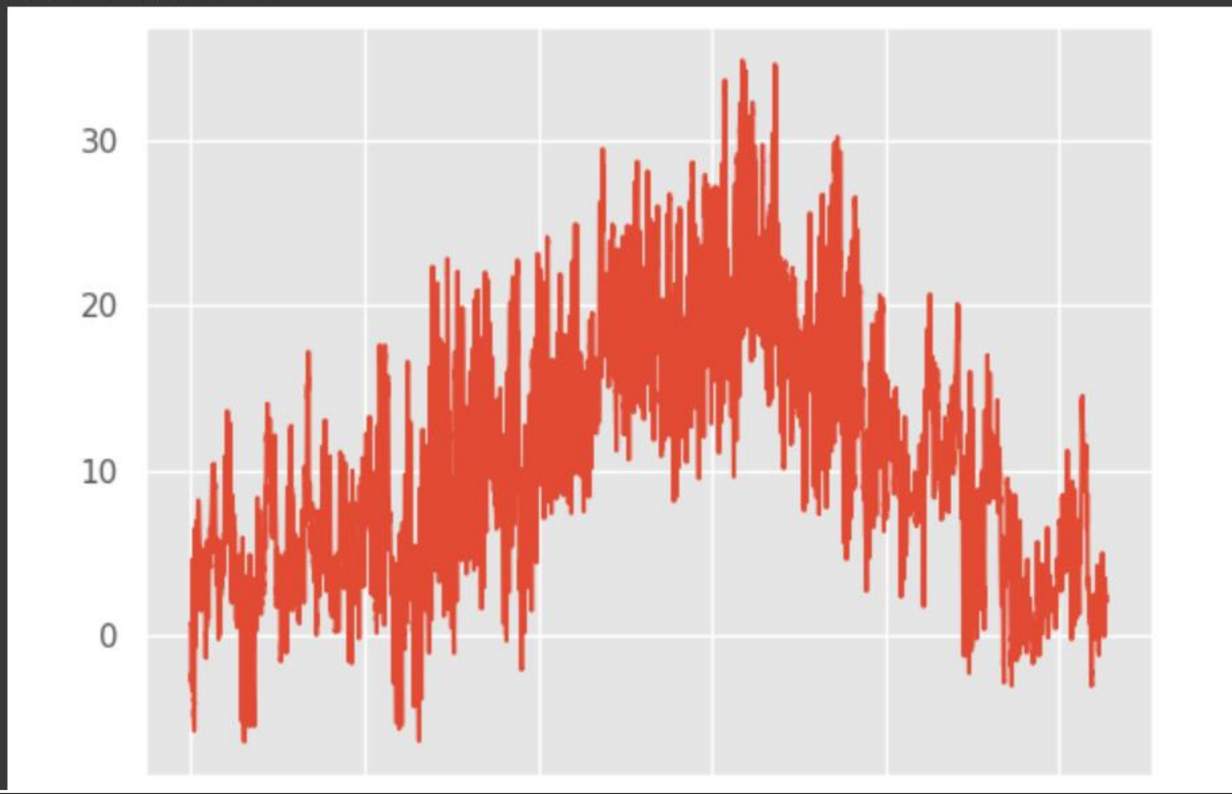
plt.plot(test_results['Test Predictions'][:100])
plt.plot(test_results['Actuals'][:100])

OUTPUT:

date	p	T	Tpot	Tdew	rh	VPmax	VPact	VPdef	sh	H2OC	rho	wv	max. wv	wd	rain	raining	SWDR	PAR	max. PAR	Tlog
2020-01-01 00:10:00	1008.89	0.71	273.18	-1.33	86.1	6.43	5.54	0.89	3.42	5.49	1280.62	1.02	1.60	224.3	0.0	0.0	0.0	0.0	0.0	11.45
2020-01-01 00:20:00	1008.76	0.75	273.22	-1.44	85.2	6.45	5.49	0.95	3.39	5.45	1280.33	0.43	0.84	206.8	0.0	0.0	0.0	0.0	0.0	11.51
2020-01-01 00:30:00	1008.66	0.73	273.21	-1.48	85.1	6.44	5.48	0.96	3.39	5.43	1280.29	0.61	1.48	197.1	0.0	0.0	0.0	0.0	0.0	11.60
2020-01-01 00:40:00	1008.64	0.37	272.86	-1.64	86.3	6.27	5.41	0.86	3.35	5.37	1281.97	1.11	1.48	206.4	0.0	0.0	0.0	0.0	0.0	11.70
2020-01-01 00:50:00	1008.61	0.33	272.82	-1.50	87.4	6.26	5.47	0.79	3.38	5.42	1282.08	0.49	1.40	209.6	0.0	0.0	0.0	0.0	0.0	11.81

```
Index(['p', 'T', 'Tpot', 'Tdew', 'rh', 'VPmax', 'VPact', 'VPdef', 'sh', 'H2OC',
       'rho', 'wv', 'max. wv', 'wd', 'rain', 'raining', 'SWDR', 'PAR',
       'max. PAR', 'Tlog'],
      dtype='object')
```

```
<Axes: xlabel='date'>
```




```
((52691, 5), (52691,))
```

```
X
```

```
array([[0.71, 0.75, 0.73, 0.37, 0.33],
       [0.75, 0.73, 0.37, 0.33, 0.34],
       [0.73, 0.37, 0.33, 0.34, 0.19],
       ...,
       [2.35, 2.32, 2.27, 2.28, 2.13],
       [2.32, 2.27, 2.28, 2.13, 1.99],
       [2.27, 2.28, 2.13, 1.99, 2.07]])
```

```
y
```

```
array([0.34, 0.19, 0.03, ..., 1.99, 2.07, 2.01])
```

```
((40000, 5), (40000,))
```

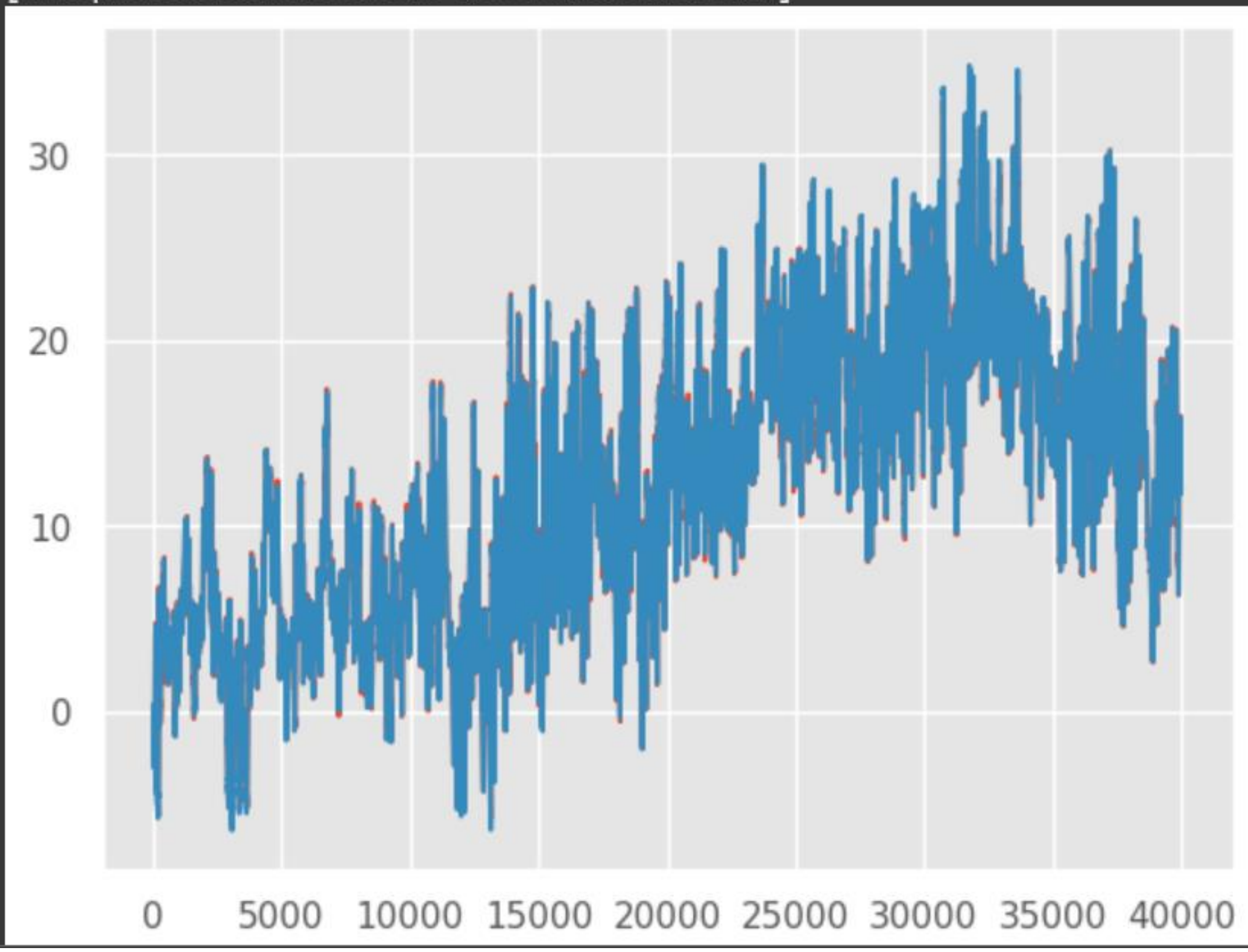
```python
model1 = Sequential()
model1.add(InputLayer((5,1)))
model1.add(LSTM(64))
model1.add(Dense(8, 'relu'))
model1.add(Dense(1, 'linear'))

model1.summary()
```

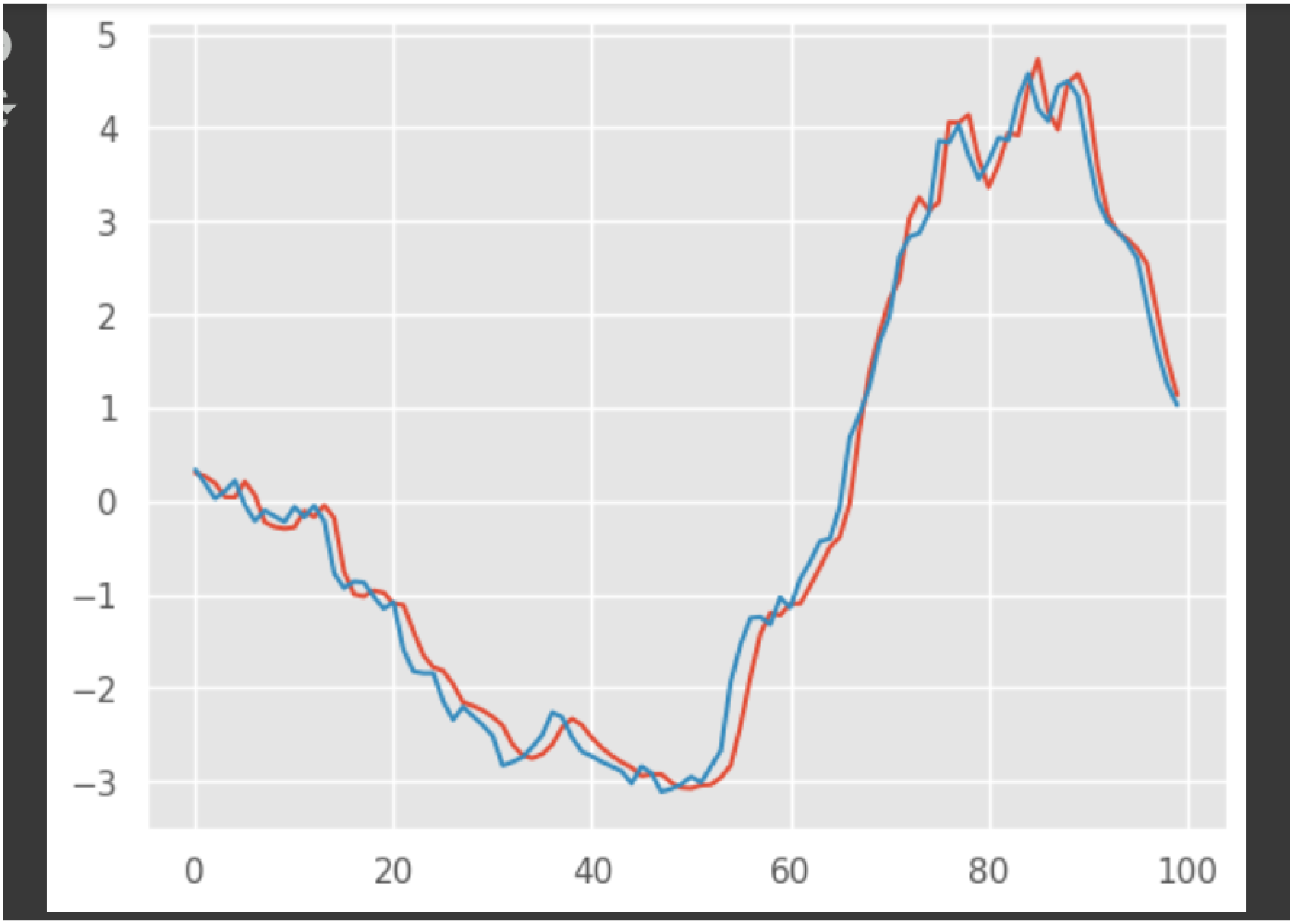
Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 64)	16,896
dense (Dense)	(None, 8)	520
dense_1 (Dense)	(None, 1)	9

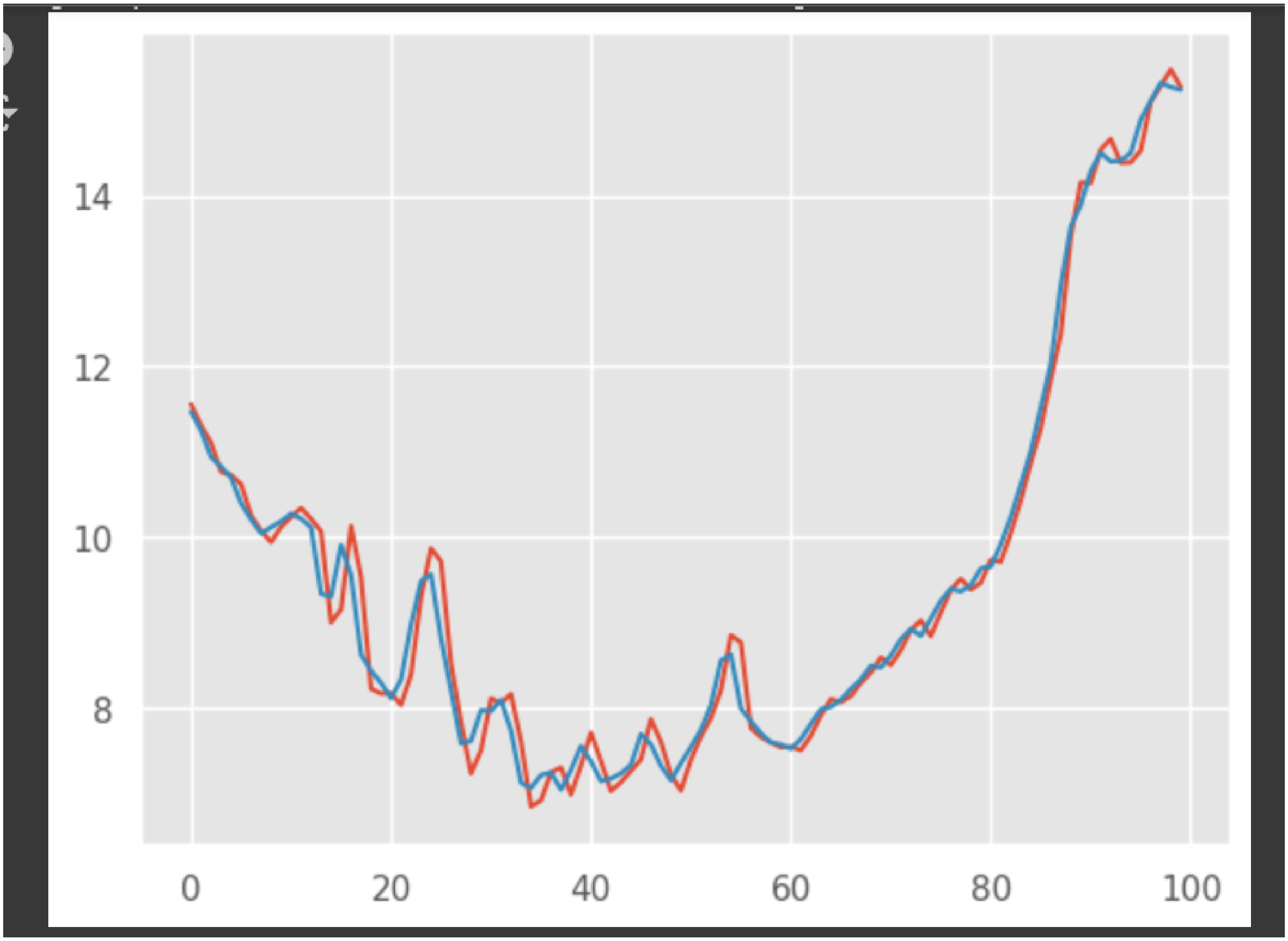
Total params: 17,425 (68.07 KB)
Trainable params: 17,425 (68.07 KB)
Non-trainable params: 0 (0.00 B)

```
[<matplotlib.lines.Line2D at 0x7dbfb9de8410>]
```

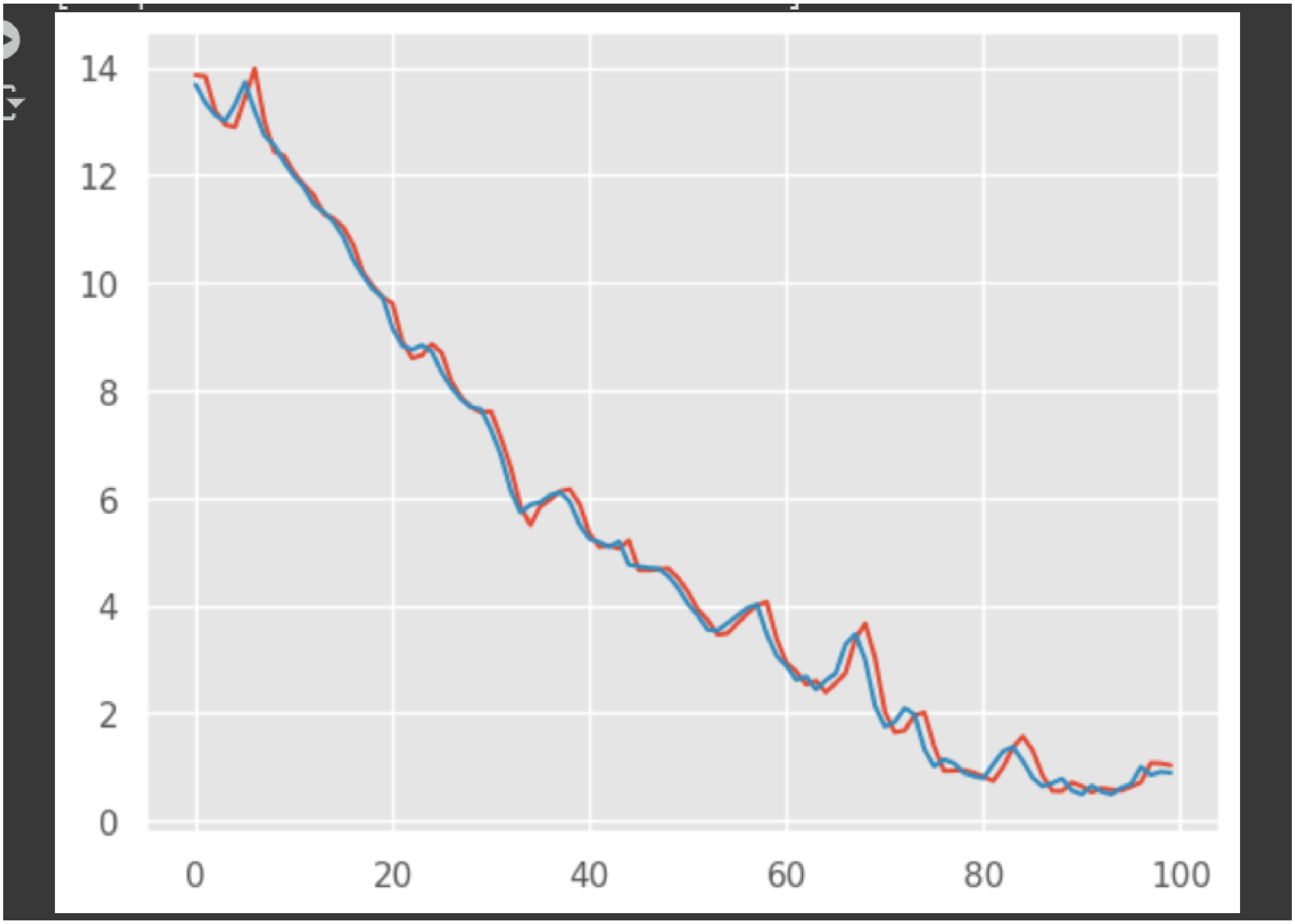




5
4
3
2
1
0
−1
−2
−3
0
20
40
60
80
100



14
12
10
8
0
20
40
60
80
100



RESULT:

The above program has been successfully return and executed.