```
!pip install --upgrade keras
```

```
Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-packages (2.13.1)
Collecting keras
  Downloading keras-2.14.0-py3-none-any.whl (1.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 11.4 MB/s eta 0:00:00
Installing collected packages: keras
  Attempting uninstall: keras
    Found existing installation: keras 2.13.1
    Uninstalling keras-2.13.1:
      Successfully uninstalled keras-2.13.1
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the following dependency conflicts.
tensorflow 2.13.0 requires keras<2.14,>=2.13.1, but you have keras 2.14.0 which is incompatible.
Successfully installed keras-2.14.0
WARNING: The following packages were previously imported in this runtime:
  [keras]
You must restart the runtime in order to use newly installed versions.
```

RESTART RUNTIME

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from pandas import datetime
import math
from math import sqrt
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
```

```
<ipython-input-37-6250386a3f42>:4: FutureWarning: The pandas.datetime class is deprecated and will be removed from pandas in a future version. Import from datetime module instead.
  from pandas import datetime
```

```
def get_stock_data(normalized=0):
    url = "/content/RL_DAILY_DATASET (1).csv"
    col_names = ['Date','Open','High','Low','Close','Volume','Adj Close']
    stocks = pd.read_csv(url, header=0, names=col_names)
    df = pd.DataFrame(stocks)
    date_split = df['Date'].str.split('-').str
    df['Year'], df['Month'], df['Day'] = date_split
    df["Volume"] = df["Volume"] / 10000
    return df
```

```
df = get_stock_data(0)
df.head()
```

```
<ipython-input-6-00ca52aac5c7>:7: FutureWarning: Columnar iteration over characters will be deprecated in future releases.
  df['Year'], df['Month'], df['Day'] = date_split
```

	Date	Open	High	Low	Close	Volume	Adj Close	Year	Month	Day
0	2013-01-01	418.037415	419.325226	415.610443	416.402924	0.038789	3152667.0	2013	01	01
1	2013-01-02	418.037415	423.981079	417.319244	419.993866	0.039123	6203434.0	2013	01	02

```
# Assuming you have a DataFrame named 'df'
columns_to_drop = ['Date', 'Low', 'Adj Close', 'Year', 'Month', 'Day']
df = df.drop(columns=columns_to_drop)

# Now 'df' will contain the DataFrame with the specified columns dropped

nan_value_index = []
High = df.High.isnull()
for i in range(0, len(High)):
    if High[i] == 1:
        nan_value_index.append(i)
        df['High'][i] = 0
Open = df.Open.isnull()
for i in range(0, len(Open)):
    if Open[i] == 1:
        nan_value_index.append(i)
        df['Open'][i] = 0
Volume = df.Volume.isnull()
for i in range(0, len(Volume)):
    if Volume[i] == 1:
        nan_value_index.append(i)
        df['Volume'][i] = 0
Close = df.Close.isnull()
for i in range(0, len(Close)):
    if Close[i] == 1:
        nan_value_index.append(i)
        df['Close'][i] = 0
```

```
X = df[['High','Open', 'Volume']]
y = df[['Close']]
factor = 0.70
length = X.shape[0]
total_for_train = int(length*factor)
X_train = X[:total_for_train]
y_train = y[:total_for_train]
X_test = X[total_for_train:]
y_test = y[total_for_train:]
```

```
print("X_train", X_train.shape)
print("y_train", y_train.shape)
print("X_test", X_test.shape)
print("y_test", y_test.shape)
```

```
X_train (1821, 3)
y_train (1821, 1)
```

```
X_test (781, 3)
y_test (781, 1)
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

def regression_model():
  model = Sequential()
  model.add(Dense(units=64, kernel_initializer='uniform', activation='relu', input_dim=3))
  model.add(Dense(units=32, kernel_initializer='uniform', activation='relu'))
  model.add(Dense(units=1))#kernel_initializer='uniform', activation='linear'))
  model.compile(optimizer='adam', loss='mse', metrics=[])
  return model

model = regression_model()
model.fit(X_train, y_train, batch_size=16, epochs=50, validation_split=0.33)
```

```
77/77 [==============================] - 0s 4ms/step - loss: 37.6739 - val_loss: 253.1469
Epoch 43/50
77/77 [==============================] - 0s 4ms/step - loss: 35.8862 - val_loss: 297.9766
Epoch 44/50
77/77 [==============================] - 0s 4ms/step - loss: 33.6368 - val_loss: 292.7625
Epoch 45/50
77/77 [==============================] - 0s 5ms/step - loss: 34.5380 - val_loss: 247.1014
Epoch 46/50
77/77 [==============================] - 0s 5ms/step - loss: 33.4811 - val_loss: 250.3091
Epoch 47/50
77/77 [==============================] - 0s 5ms/step - loss: 33.8534 - val_loss: 245.1804
Epoch 48/50
77/77 [==============================] - 0s 5ms/step - loss: 34.5249 - val_loss: 262.5407
Epoch 49/50
77/77 [==============================] - 0s 4ms/step - loss: 33.6269 - val_loss: 271.0528
Epoch 50/50
77/77 [==============================] - 0s 4ms/step - loss: 36.4653 - val_loss: 244.5139
<keras.src.callbacks.History at 0x7e3a18233520>
```

```
from sklearn.metrics import r2_score, mean_squared_error

# Make predictions on the testing dataset
predictions = model.predict(X_test)

# Calculate R-squared (coefficient of determination)
r_squared = r2_score(y_test, predictions)

# Calculate mean squared error (MSE)
mse = mean_squared_error(y_test, predictions)

# Print R-squared and MSE
print(f"R-squared (R2): {r_squared}")
print(f"Mean Squared Error (MSE): {mse}")
```

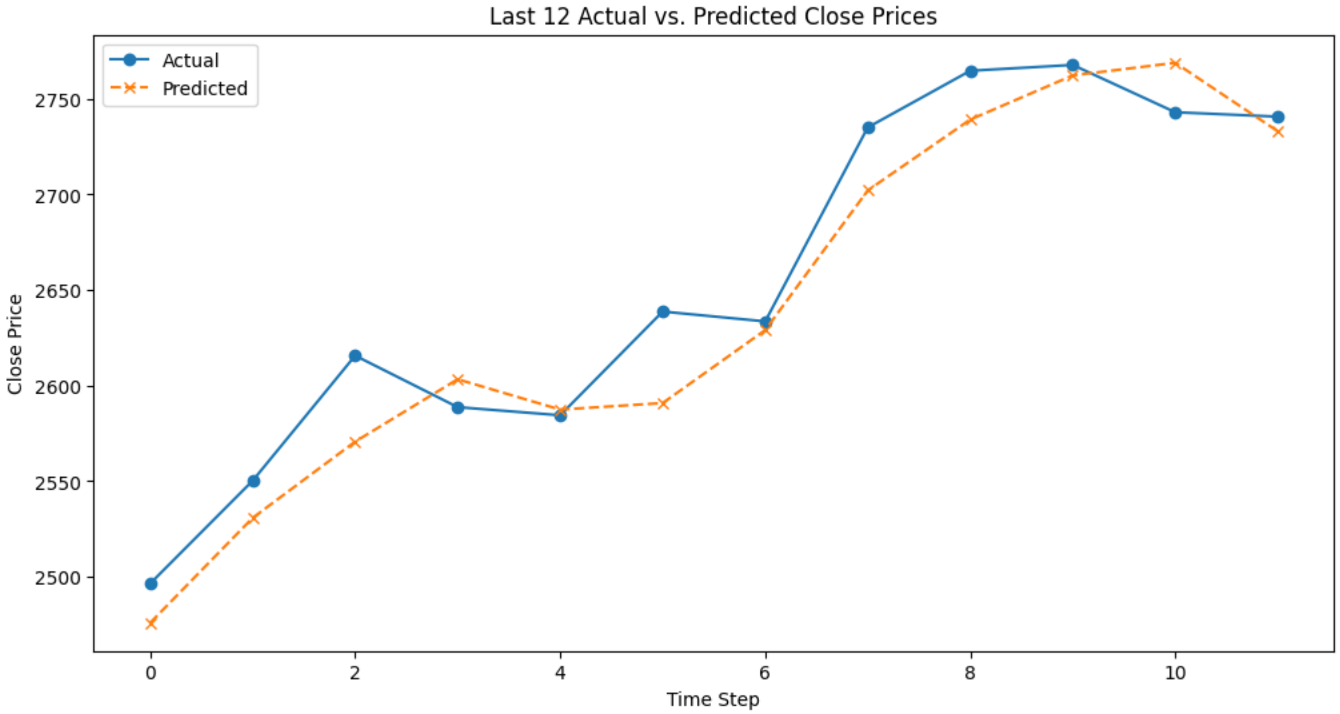
```
25/25 [==============================] - 0s 1ms/step
R-squared (R2): 0.9903157585817445
Mean Squared Error (MSE): 702.5469506902991
```

```
# Assuming you have trained your model and obtained predictions in a variable 'predictions'

# Last 12 actual values
actual_values = y_test[-12:]  # Assuming y_test contains your actual 'Close' values

# last 12 predicted values
predicted_values = predictions[-12:]

# Print the actual values
print("Actual Values:")
print(actual_values)

# Print the predicted values
print("Predicted Values:")
print(predicted_values)
```

```
Actual Values:
            Close
```

```
2590  2496.449951
2591  2550.250000
2592  2615.699951
2593  2588.750000
2594  2584.500000
2595  2638.750000
2596  2633.600098
2597  2735.050049
2598  2764.699951
2599  2767.750000
2600  2743.000000
2601  2740.699951
Predicted Values:
[[2475.8757]
 [2530.8108]
 [2570.6936]
 [2603.337 ]
 [2587.4731]
 [2590.8894]
 [2629.054 ]
 [2702.0986]
 [2739.1794]
 [2762.2776]
 [2768.8596]
 [2733.0322]]
```

```
import matplotlib.pyplot as plt

# Assuming you have actual_values and predicted_values as defined in your code

# Create a range of indices for the x-axis (assuming one data point per time step)
x = range(len(actual_values))

# Plot the last 12 actual and predicted values
plt.figure(figsize=(12, 6))  # Adjust the figure size as needed
plt.plot(x, actual_values, label='Actual', marker='o', linestyle='-')
plt.plot(x, predicted_values, label='Predicted', marker='x', linestyle='--')

# Add labels and a legend
plt.xlabel('Time Step')
plt.ylabel('Close Price')
plt.legend()

# Show the plot
plt.title('Last 12 Actual vs. Predicted Close Prices')
plt.show()
```


Last 12 Actual vs. Predicted Close Prices
Actual
Predicted
Close Price
Time Step
2500
2550
2600
2650
2700
2750
0
2
4
6
8
10