Hands-On

Hands-On ini digunakan pada kegiatan Microcredential Associate Data Scientist 2021

Tugas Mandiri Pertemuan 14

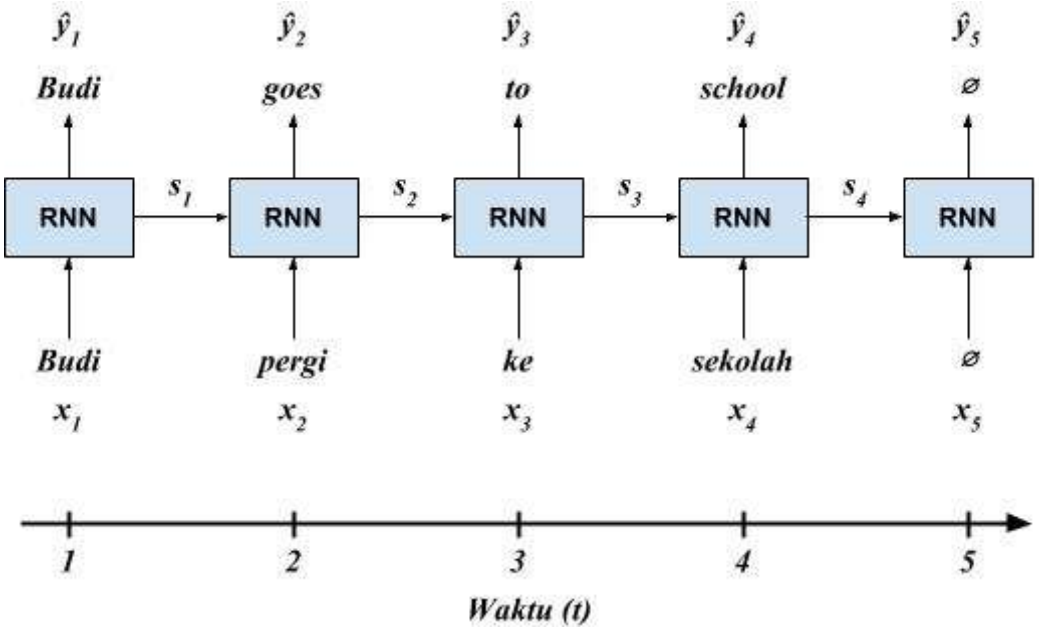
Pertemuan 14 (empatbelas) pada Microcredential Associate Data Scientist 2021 menyampaikan materi mengenai Membangun Model (RNN dan LSTM). silakan Anda kerjakan Latihan 1 s/d 5. Output yang anda lihat merupakan panduan yang dapat Anda ikuti dalam penulisan code :)

RNN

Jaringan saraf berulang atau recurrent neural network (RNN) adalah jenis arsitektur jaringan saraf tiruan yang pemrosesannya dipanggil berulang-ulang untuk memroses masukan yang biasanya adalah data sekuensial. RNN masuk dalam kategori deep learning karena data diproses melalui banyak lapis (layer). RNN telah mengalami kemajuan yang pesat dan telah merevolusi bidang-bidang seperti pemrosesan bahasa alami (NLP), pengenalan suara, sintesa musik, pemrosesan data finansial seri waktu, analisa deret DNA, analisa video, dan sebagainya.

RNN memroses input secara sekuensial, sampel per sampel. Dalam tiap pemrosesan, output yang dihasilkan tidak hanya merupakan fungsi dari sampel itu saja, tapi juga berdasarkan state internal yang merupakan hasil dari pemrosesan sampel-sampel sebelumnya (atau setelahnya, pada bidirectional RNN).

Berikut adalah ilustrasi bagaimana RNN bekerja. Misalnya kita membuat RNN untuk menerjemahkan bahasa Indonesia ke bahasa Inggris



Ilustrasi di atas kelihatan rumit, tapi sebenarnya cukup mudah dipahami.

- sumbu horizontal adalah waktu, direpresentasikan dengan simbol t. Dapat kita bayangkan pemrosesan berjalan dari kiri ke kanan. Selanjutnya kita sebut t adalah langkah waktu (time step).
- Keseluruhan input adalah kalimat, dalam hal ini:

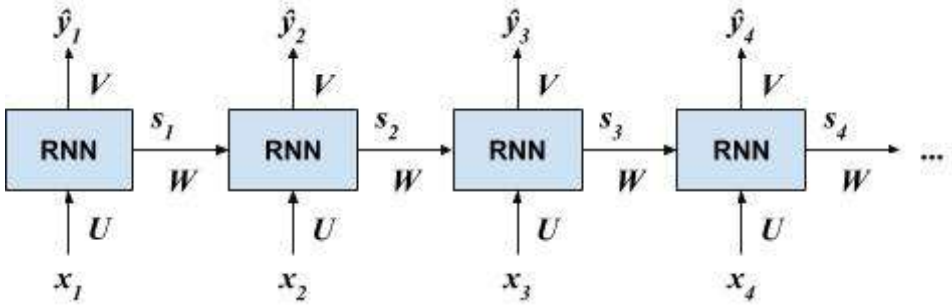
```
Budi pergi ke sekolah.
```

- Pemrosean input oleh RNN adalah kata demi kata. Input kata-kata ini disimbolkan dengan x1 ,x2 , … ,x5 , atau secara umum xt.
- Output adalah kalimat, dalam hal ini:

```
Budi goes to school.
```

- RNN memberikan output kata demi kata, dan ini kita simbolkan dengan ŷ1, ŷ2, …, ŷ5 , atau secara umum ŷt .
- Dalam tiap pemrosesan, RNN akan menyimpan state internal yaitu st, yang diberikan dari satu langkah waktu ke langkah waktu berikutnya. Inilah “memori” dari RNN.

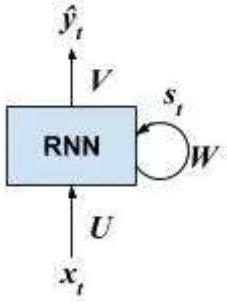
Dengan contoh di atas, kita bisa generalisasikan arsitektur RNN sebagai berikut:



Tambahan yang tidak terdapat di diagram sebelumnya adalah U, V, dan W, yang merupakan parameter-parameter yang dimiliki RNN. Kita akan bahas pemakaian parameter-parameter ini nanti.

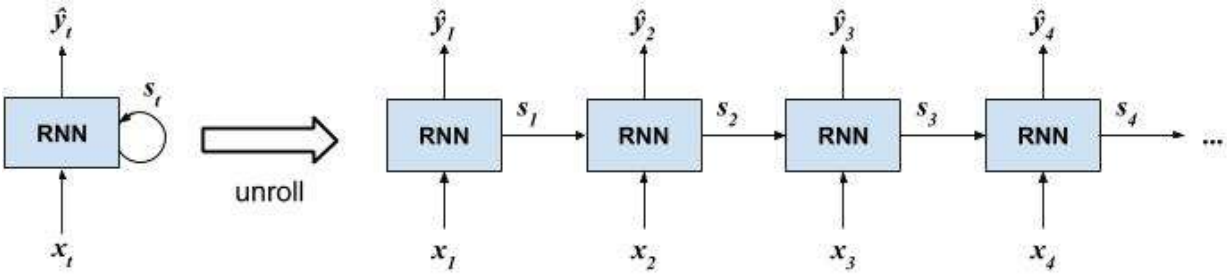
Penting untuk dipahami bahwa walaupun ada empat kotak RNN di gambar di atas, empat kotak itu mencerminkan satu modul RNN yang sama (satu instans model dengan parameter-parameter U, V, dan W yang sama). Penggambaran di atas hanya agar aspek sekuensialnya lebih tergambar.

Alternatif representasinya adalah seperti ini, agar lebih jelas bahwa hanya ada satu modul RNN:



Inilah sebabnya kenapa arsitektur ini disebut RNN. Kata recurrent (berulang) dalam RNN timbul karena RNN melakukan perhitungan yang sama secara berulang-ulang atas input yang kita berikan.

Sering juga kedua ilustrasi di atas digabungkan jadi satu sbb:



Sesuai dengan gambar di atas, ilustrasi di sebelah kanan adalah penjabaran (unrolled) dari versi berulang di sebelah kiri.

Latihan (1)

Melakukan import library yang dibutuhkan

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler

from keras.models import Sequential, load_model
from keras.layers import LSTM, Dense, Dropout
import os
```

Load Dataset

In [6]:
```
#Panggil file (load file bernama Stock.csv) dan simpan dalam dataframe
df = pd.read_csv('Stock.csv')
```

In [7]:
```
# tampilkan 5 baris data
df.head()
```

Out[7]:

	Date	Open	High	Low	Close	Volume	Name
0	2006-01-03	56.45	56.66	55.46	56.53	3716500	UTX
1	2006-01-04	56.80	56.80	55.84	56.19	3114500	UTX
2	2006-01-05	56.30	56.49	55.63	55.98	3118900	UTX
3	2006-01-06	56.45	56.67	56.10	56.16	2874300	UTX
4	2006-01-09	56.37	56.90	56.16	56.80	2467200	UTX

In [10]:
```
df
```

Out[10]:

	Date	Open	High	Low	Close	Volume	Name
0	2006-01-03	56.45	56.66	55.46	56.53	3716500	UTX
1	2006-01-04	56.80	56.80	55.84	56.19	3114500	UTX
2	2006-01-05	56.30	56.49	55.63	55.98	3118900	UTX
3	2006-01-06	56.45	56.67	56.10	56.16	2874300	UTX
4	2006-01-09	56.37	56.90	56.16	56.80	2467200	UTX
...	...	...	...	...	...	...	...
3015	2017-12-22	127.52	127.60	126.95	127.23	1604455	UTX
3016	2017-12-26	127.45	127.94	126.99	127.14	1053109	UTX
3017	2017-12-27	127.46	127.61	126.92	127.58	1810839	UTX
3018	2017-12-28	127.73	128.17	127.29	128.12	1412983	UTX
3019	2017-12-29	128.32	128.49	127.57	127.57	1806925	UTX

3020 rows × 7 columns

Review Data

In [11]:
```
# Melihat Informasi lebih detail mengenai struktur DataFrame dapat dilihat menggu
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3020 entries, 0 to 3019
Data columns (total 7 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Date    3020 non-null   object 
 1   Open    3019 non-null   float64
 2   High    3020 non-null   float64
 3   Low     3020 non-null   float64
 4   Close   3020 non-null   float64
 5   Volume  3020 non-null   int64  
 6   Name    3020 non-null   object 
dtypes: float64(4), int64(1), object(2)
memory usage: 165.3+ KB
```

In [12]:
```
df.tail()
```

Out[12]:

	Date	Open	High	Low	Close	Volume	Name
3015	2017-12-22	127.52	127.60	126.95	127.23	1604455	UTX
3016	2017-12-26	127.45	127.94	126.99	127.14	1053109	UTX
3017	2017-12-27	127.46	127.61	126.92	127.58	1810839	UTX
3018	2017-12-28	127.73	128.17	127.29	128.12	1412983	UTX
3019	2017-12-29	128.32	128.49	127.57	127.57	1806925	UTX

In [13]:
```
df.shape
```

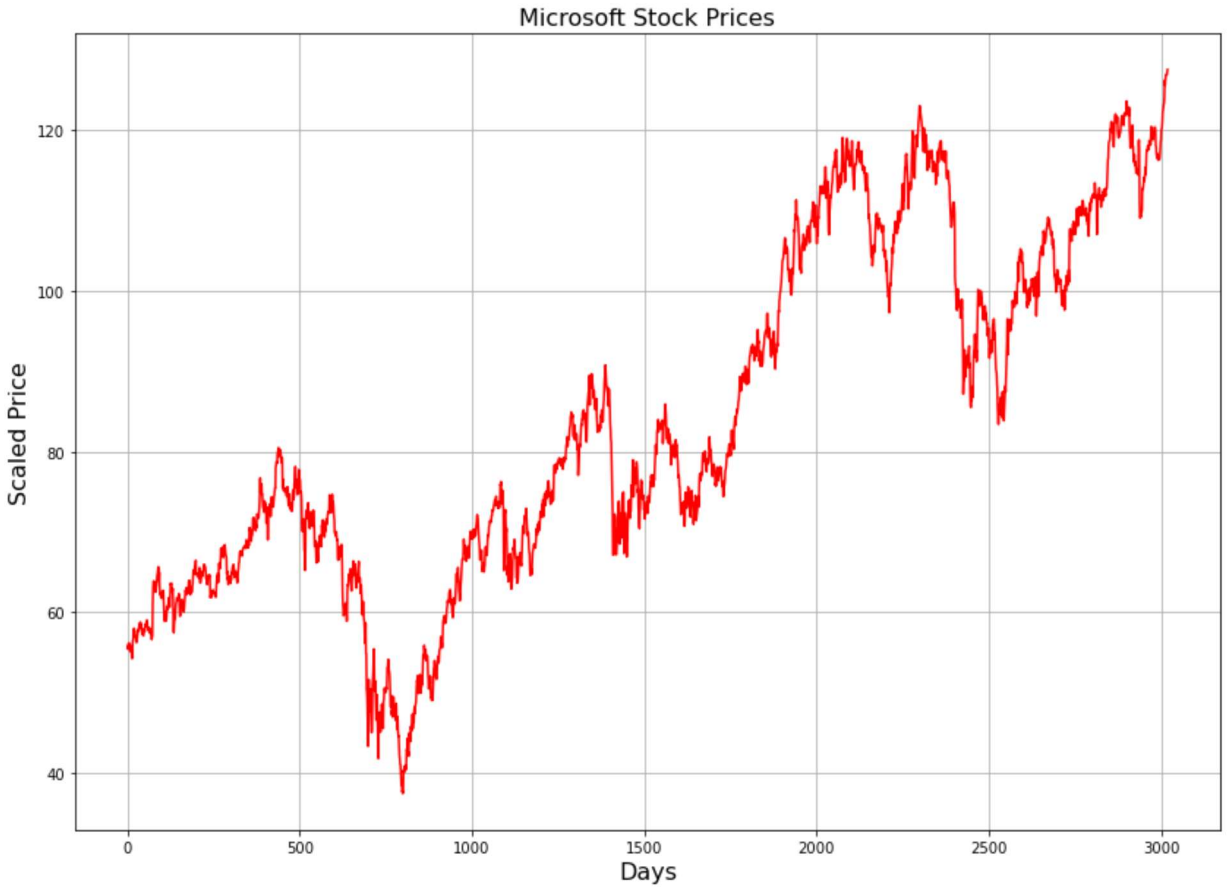
Out[13]:
```
(3020, 7)
```

In [14]:
```
# Kolom 'low' yang akan kita gunakan dalam membangun model
# Slice kolom 'low'

Low_data = df.iloc[:,3:4].values
```

In [15]:
```
# cek output low_data
Low_data
```

Out[15]:
```
array([[ 55.46],
       [ 55.84],
       [ 55.63],
       ...,
       [126.92],
       [127.29],
       [127.57]])
```

In [16]:
```
# Visualizing low_data

plt.figure(figsize=(14,10))
plt.plot(Low_data,c="red")
plt.title("Microsoft Stock Prices",fontsize=16)
plt.xlabel("Days",fontsize=16)
plt.ylabel("Scaled Price",fontsize=16)
plt.grid()
plt.show()
```



Latihan (2)

Data Preprocessing

In [20]:
```
df = df['Open'].values
```

In [22]:
```
df
```

Out[22]:
```
array([ 56.45,  56.8 ,  56.3 , ..., 127.46, 127.73, 128.32])
```

In [23]:
```
df = df.reshape(-1, 1)
print(df.shape)
```

```
(3020, 1)
```

In [24]:
```
df
```

Out[24]:
```
array([[ 56.45],
       [ 56.8 ],
       [ 56.3 ],
       ...,
       [127.46],
       [127.73],
       [128.32]])
```

In [25]:
```
df.ndim
```

Out[25]:
```
2
```

In [26]:
```
df.shape
```

Out[26]:
```
(3020, 1)
```

In [27]:
```
dataset_train = np.array(df[:int(df.shape[0]*0.8)])
dataset_test = np.array(df[int(df.shape[0]*0.8)-50:])
print(dataset_train.shape)
print(dataset_test.shape)
```

```
(2416, 1)
(654, 1)
```

In [28]:
```
scaler = MinMaxScaler(feature_range=(0,1))
dataset_train = scaler.fit_transform(dataset_train)
dataset_train[:5]
```

Out[28]:
```
array([[0.21463528],
       [0.21871359],
       [0.21288744],
       [0.21463528],
       [0.2137031 ]])
```

In [29]:
```
dataset_test = scaler.transform(dataset_test)
dataset_test[:5]
```

Out[29]:
```
array([[0.92612445],
       [0.9228618 ],
       [0.91703566],
       [0.93218364],
       [0.92985318]])
```

In [30]:
```
def create_dataset(df):
    x = []
    y = []
    for i in range(50, df.shape[0]):
        x.append(df[i-50:i, 0])
        y.append(df[i, 0])
    x = np.array(x)
    y = np.array(y)
    return x,y
```

splitting the dataset into training and testing set

In [31]:
```
x_train, y_train = create_dataset(dataset_train)
x_train[:2]
```

Out[31]:
```
array([[0.21463528, 0.21871359, 0.21288744, 0.21463528, 0.2137031 ,
        0.21451876, 0.21754836, 0.20647868, 0.20414822, 0.20764391,
        0.21113959, 0.20531345, 0.2091587 , 0.19785598, 0.2065952 ,
        0.21463528, 0.22652062, 0.23549289, 0.23898858, 0.23269634,
        0.23409462, 0.2350268 , 0.22989979, 0.22512235, 0.22349103,
        0.21859706, 0.21917968, 0.22022838, 0.22279189, 0.22500583,
        0.23176416, 0.23386157, 0.23619203, 0.23560942, 0.2436495 ,
        0.24295036, 0.23677464, 0.24295036, 0.24085295, 0.23852249,
        0.23852249, 0.23176416, 0.23153111, 0.22861804, 0.23153111,
        0.22721976, 0.22220928, 0.23211373, 0.23327896, 0.23339548],
       [0.21871359, 0.21288744, 0.21463528, 0.2137031 , 0.21451876,
        0.21754836, 0.20647868, 0.20414822, 0.20764391, 0.21113959,
        0.20531345, 0.2091587 , 0.19785598, 0.2065952 , 0.21463528,
        0.22652062, 0.23549289, 0.23898858, 0.23269634, 0.23409462,
        0.2350268 , 0.22989979, 0.22512235, 0.22349103, 0.21859706,
        0.21917968, 0.22022838, 0.22279189, 0.22500583, 0.23176416,
        0.23386157, 0.23619203, 0.23560942, 0.2436495 , 0.24295036,
        0.23677464, 0.24295036, 0.24085295, 0.23852249, 0.23852249,
        0.23176416, 0.23153111, 0.22861804, 0.23153111, 0.22721976,
        0.22220928, 0.23211373, 0.23327896, 0.23339548, 0.23071545]])
```

In [32]:
```
y_train
```

Out[32]:
```
array([0.23071545, 0.2406199 , 0.24027033, ..., 0.72174318, 0.71032393,
       0.70717781])
```

In [33]:
```
x_train.shape
```

Out[33]:
```
(2366, 50)
```

In [34]:
```
y_train.shape
```

Out[34]:
```
(2366,)
```

In [35]:
```
x_test, y_test = create_dataset(dataset_test)
x_test[:1]
```

Out[35]:
```
array([[0.92612445, 0.9228618 , 0.91703566, 0.93218364, 0.92985318,
        0.91540433, 0.91482172, 0.91447215, 0.92600792, 0.93171755,
        0.92554183, 0.89932417, 0.89116756, 0.90013983, 0.88860405,
        0.90386856, 0.9015381 , 0.89885808, 0.8933815 , 0.88802144,
        0.87904917, 0.870543  , 0.86471685, 0.86226987, 0.85411326,
        0.81950594, 0.83325565, 0.82416686, 0.83861571, 0.84304358,
        0.85248194, 0.85609415, 0.85609415, 0.86564903, 0.8532976 ,
        0.84991843, 0.77278024, 0.74842694, 0.74749476, 0.73374505,
        0.70787695, 0.71079003, 0.71148916, 0.71486833, 0.73281286,
        0.72570496, 0.72244232, 0.72174318, 0.71032393, 0.70717781]])
```

In [36]:
```
x_test.shape
```

Out[36]:
```
(604, 50)
```

In [37]:
```
y_test.shape
```

Out[37]:
```
(604,)
```

Model Creation LSTM & RNN

In [38]:
```
# Reshape features for LSTM Layer
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
```

In [39]:
```
x_train.shape
```

Out[39]:
```
(2366, 50, 1)
```

In [40]:
```
x_train.ndim
```

Out[40]:
```
3
```

In [41]:
```
x_train.shape[1]
```

Out[41]:
```
50
```

In [42]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM , Dense
```

In [43]:
```
model = Sequential()
model.add(LSTM(units=96, return_sequences=True, input_shape=(x_train.shape[1], 1)
model.add(Dropout(0.2))
model.add(LSTM(units=96))
model.add(Dropout(0.2))
model.add(Dense(units=1))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 50, 96)            37632

 dropout (Dropout)           (None, 50, 96)            0

 lstm_1 (LSTM)               (None, 96)                74112

 dropout_1 (Dropout)         (None, 96)                0

 dense (Dense)               (None, 1)                 97

=================================================================
Total params: 111,841
Trainable params: 111,841
Non-trainable params: 0
_________________________________________________________________
```

In [44]:
```
model.compile(loss='mean_squared_error', optimizer='adam')
```

In [45]:
```
if(not os.path.exists('Stock.csv')):
    model.fit(x_train, y_train, epochs=50, batch_size=32)
    model.save('Stock.csv')
```

In [46]:
```
model
```

Out[46]:
```
<keras.engine.sequential.Sequential at 0x1cc7b0373a0>
```

In [49]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 50, 96)            37632

 dropout (Dropout)           (None, 50, 96)            0

 lstm_1 (LSTM)               (None, 96)                74112

 dropout_1 (Dropout)         (None, 96)                0

 dense (Dense)               (None, 1)                 97

=================================================================
Total params: 111,841
Trainable params: 111,841
Non-trainable params: 0
_________________________________________________________________
```

In [52]:
```
model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=100,batch_size=6
```

```
Epoch 1/100
37/37 [==============================] - 10s 151ms/step - loss: 0.0287 - val
_loss: nan
Epoch 2/100
37/37 [==============================] - 5s 135ms/step - loss: 0.0033 - val_
loss: nan
Epoch 3/100
37/37 [==============================] - 5s 131ms/step - loss: 0.0022 - val_
loss: nan
Epoch 4/100
37/37 [==============================] - 5s 137ms/step - loss: 0.0023 - val_
loss: nan
Epoch 5/100
37/37 [==============================] - 5s 139ms/step - loss: 0.0023 - val_
loss: nan
Epoch 6/100
37/37 [==============================] - 8s 227ms/step - loss: 0.0021 - val_
loss: nan
Epoch 7/100
```

Latihan (5)

Evaluation

R^2 Score dari model RNN 0.9695404783998754

R^2 Score dari model LSTM 0.8531204922506259

Visualisasi Perbandingan Hasil Model prediksi dengan data original

In [56]:
```
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)

fig, ax = plt.subplots(figsize=(8,4))
plt.plot(df, color='red',  label="True Price")
ax.plot(range(len(y_train)+50,len(y_train)+50+len(predictions)),predictions, colo
plt.legend()
```

Out[56]: <matplotlib.legend.Legend at 0x1cc7b460550>

In [57]:
```
y_test_scaled = scaler.inverse_transform(y_test.reshape(-1, 1))

fig, ax = plt.subplots(figsize=(8,4))
ax.plot(y_test_scaled, color='red', label='True Testing Price')
plt.plot(predictions, color='blue', label='Predicted Testing Price')
plt.legend()
```

Out[57]: <matplotlib.legend.Legend at 0x1cc02282550>

In [58]:
```
x = x_test[-1]
num_next_predictions = 100
preds = []
for i in range(num_next_predictions):
    data = np.expand_dims(x, axis=0)
    prediction = model.predict(data)
    prediction = scaler.inverse_transform(prediction)
    preds.append(prediction[0][0])
    x = np.delete(x, 0, axis=0) # delete first
    x = np.vstack([x, prediction]) # add prediction

print(preds)
```

```
[127.24238, 454.64206, 511.3145, 532.5387, 540.4776, 544.56683, 546.816, 548.17
865, 549.04645, 549.6239, 550.02057, 550.3, 550.5006, 550.6467, 550.75433, 550.
83453, 550.8952, 550.94104, 550.97614, 551.0032, 551.02405, 551.0407, 551.0541
4, 551.0649, 551.0738, 551.08136, 551.0881, 551.09424, 551.10004, 551.1059, 55
1.1121, 551.11896, 551.1269, 551.136, 551.1465, 551.15875, 551.1737, 551.19147,
551.21246, 551.2371, 551.2671, 551.3053, 551.3521, 551.40393, 551.4459, 551.468
7, 551.49725, 551.59753, 551.79785, 551.9487, 551.6905, 551.6835, 551.6833, 55
1.6833, 551.6833, 551.68335, 551.6833, 551.6833, 551.68335, 551.6833, 551.6833,
551.6833, 551.6833, 551.6833, 551.6833, 551.6833, 551.6833, 551.68335, 551.6833
5, 551.68335, 551.6833, 551.68335, 551.68335, 551.6833, 551.68335, 551.68335, 5
51.68335, 551.68335, 551.68335, 551.6833, 551.6833, 551.68335, 551.68335, 551.6
8335, 551.68335, 551.68335, 551.6833, 551.6833, 551.68335, 551.68335, 551.6833
5, 551.68335, 551.68335, 551.68335, 551.68335, 551.68335, 551.68335, 551.68335,
551.6833, 551.68335]
```

Berikan Kesimpulan Anda!

Berdasarkan analisis penggunaan metode RNN, LSTM dan perbandingan dengan data original terdapat perbandingan bahwa akurasi data lebih tinggi ketika menggunakan metode RNN dibanding LSTM ataupun Original