| Price | Open | High | Low | Close | Volume |
|---|---|---|---|---|---|
| Ticker | BMRI.JK | BMRI.JK | BMRI.JK | BMRI.JK | BMRI.JK |
| Date | | | | | |
| **2015-02-16** | 1804.317046 | 1811.898210 | 1781.573554 | 1785.364136 | 85716800 |
| **2015-02-17** | 1785.364055 | 1796.735801 | 1766.411146 | 1773.992310 | 28532800 |
| **2015-02-18** | 1819.479290 | 1864.966272 | 1800.526380 | 1808.107544 | 167238000 |
| **2015-02-20** | 1815.688746 | 1849.803984 | 1815.688746 | 1830.851074 | 74989600 |
| **2015-02-23** | 1804.316949 | 1827.060440 | 1792.945204 | 1800.526367 | 118434000 |
| **2015-02-24** | 1789.154622 | 1811.898113 | 1785.364040 | 1800.526367 | 81622800 |
| **2015-02-25** | 1804.317139 | 1808.107721 | 1796.735974 | 1804.317139 | 62958800 |

```
# Setting Random Seed
import random
import numpy as np
import tensorflow as tf

SEED = 42
random.seed(SEED)
np.random.seed(SEED)
tf.random.set_seed(SEED)
```

```
# Import library
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tcn import TCN
```

```
# Load ulang data
df = pd.read_csv('SAHAM-BMRI.JK.csv', skiprows=2, header=0)
df.columns = ['Date', 'Open', 'High', 'Low', 'Close', 'Volume']

# Pastikan baris pertama memang berisi data
df = df[1:]

# Konversi kolom Date ke format datetime
df['Date'] = pd.to_datetime(df['Date'], format='%Y-%m-%d', errors='coerce')

# Jadikan Date sebagai index
df.set_index('Date', inplace=True)
```

+ Kode + Teks

```
df.info(7)
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2471 entries, 2015-02-17 to 2025-02-13
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Open    2471 non-null   float64
 1   High    2471 non-null   float64
 2   Low     2471 non-null   float64
 3   Close   2471 non-null   float64
 4   Volume  2471 non-null   int64  
dtypes: float64(4), int64(1)
memory usage: 115.8 KB
```

```
df.head(7)
```

| Date | Open | High | Low | Close | Volume |
|---|---|---|---|---|---|
| **2015-02-17** | 1785.364055 | 1796.735801 | 1766.411146 | 1773.992310 | 28532800 |
| **2015-02-18** | 1819.479290 | 1864.966272 | 1800.526380 | 1808.107544 | 167238000 |
| **2015-02-20** | 1815.688746 | 1849.803984 | 1815.688746 | 1830.851074 | 74989600 |
| **2015-02-23** | 1804.316949 | 1827.060440 | 1792.945204 | 1800.526367 | 118434000 |
| **2015-02-24** | 1789.154622 | 1811.898113 | 1785.364040 | 1800.526367 | 81622800 |
| **2015-02-25** | 1804.317261 | 1808.107843 | 1796.736096 | 1804.317261 | 62958800 |
| **2015-02-26** | 1789.154622 | 1804.316949 | 1789.154622 | 1800.526367 | 76328800 |

```
# Pilih fitur (gunakan semua atau hanya Close)
features = df[['Open', 'High', 'Low', 'Close', 'Volume']]
```

```
# === 2. Normalisasi Data ===
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(features)
```

```
# === 3. Fungsi untuk Membuat Dataset ===
def create_dataset(data, time_steps=60):
    x, y = [], []
    for i in range(time_steps, len(data)):
        x.append(data[i - time_steps:i])  # Ambil 60 hari sebelumnya sebagai input
        y.append(data[i, 3])  # Prediksi harga 'Close'
    return np.array(x), np.array(y)
```

```
# === 4. Split Data menjadi Training dan Testing ===
train_size = int(len(scaled_data) * 0.8)
train_data, test_data = scaled_data[:train_size], scaled_data[train_size:]
```

```
# Buat dataset untuk training dan testing
x_train, y_train = create_dataset(train_data)
x_test, y_test = create_dataset(test_data)
```

```
# Reshape data agar sesuai dengan input model TCN (samples, time_steps, features)
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], x_train.shape[2]))
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], x_test.shape[2]))
```

```
# === 5. Pembuatan Arsitektur Model TCN ===

model_tcn = Sequential([
    TCN(input_shape=(x_train.shape[1], x_train.shape[2]), nb_filters=64, kernel_size=5, dilations=[1, 2, 4, 8], return_sequences=False),
    Dropout(0.1),
    Dense(50, activation='relu'),
    Dense(1)
])
```

```
/usr/local/lib/python3.11/dist-packages/tcn/tcn.py:268: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. W
  super(TCN, self).__init__(**kwargs)
```

```
# Compile Model
optimizer = Adam(learning_rate=0.0001)
model_tcn.compile(optimizer=optimizer, loss='mean_squared_error')
```

```
# === 6. Training Model ===
history = model_tcn.fit(x_train, y_train, epochs=50, batch_size=32, validation_data=(x_test, y_test))
```

```
Epoch 1/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 10s 92ms/step - loss: 0.0616 - val_loss: 0.0128
Epoch 2/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 8s 64ms/step - loss: 0.0120 - val_loss: 0.0166
Epoch 3/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 5s 85ms/step - loss: 0.0065 - val_loss: 0.0072
Epoch 4/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 4s 64ms/step - loss: 0.0044 - val_loss: 0.0029
Epoch 5/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 4s 64ms/step - loss: 0.0039 - val_loss: 0.0038
Epoch 6/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 5s 86ms/step - loss: 0.0030 - val_loss: 0.0038
Epoch 7/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 9s 66ms/step - loss: 0.0030 - val_loss: 0.0025
```

```
Epoch 8/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 6s 78ms/step - loss: 0.0022 - val_loss: 0.0021
Epoch 9/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 4s 63ms/step - loss: 0.0018 - val_loss: 0.0025
Epoch 10/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 4s 64ms/step - loss: 0.0018 - val_loss: 0.0075
Epoch 11/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 5s 87ms/step - loss: 0.0015 - val_loss: 0.0027
Epoch 12/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 4s 63ms/step - loss: 0.0015 - val_loss: 0.0018
Epoch 13/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 4s 72ms/step - loss: 0.0011 - val_loss: 0.0046
Epoch 14/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 5s 71ms/step - loss: 0.0013 - val_loss: 0.0048
Epoch 15/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 4s 64ms/step - loss: 0.0010 - val_loss: 0.0039
Epoch 16/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 7s 88ms/step - loss: 9.8361e-04 - val_loss: 0.0036
Epoch 17/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 9s 64ms/step - loss: 8.8020e-04 - val_loss: 0.0019
Epoch 18/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 6s 81ms/step - loss: 8.2022e-04 - val_loss: 0.0016
Epoch 19/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 4s 64ms/step - loss: 8.0349e-04 - val_loss: 0.0023
Epoch 20/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 4s 64ms/step - loss: 7.8887e-04 - val_loss: 0.0026
Epoch 21/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 5s 83ms/step - loss: 6.9922e-04 - val_loss: 0.0021
Epoch 22/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 4s 63ms/step - loss: 7.8451e-04 - val_loss: 0.0034
Epoch 23/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 6s 77ms/step - loss: 6.3300e-04 - val_loss: 0.0031
Epoch 24/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 4s 71ms/step - loss: 6.6168e-04 - val_loss: 0.0035
Epoch 25/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 4s 63ms/step - loss: 5.9688e-04 - val_loss: 0.0033
Epoch 26/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 7s 88ms/step - loss: 6.6217e-04 - val_loss: 0.0061
Epoch 27/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 9s 64ms/step - loss: 5.5667e-04 - val_loss: 0.0023
Epoch 28/50
60/60 ━━━━━━━━━━━━━━━━━━━━ 5s 85ms/step - loss: 5.5853e-04 - val_loss: 0.0040
Epoch 29/50
```

```python
# === 7. Prediksi ===
predictions = model_tcn.predict(x_test)
```

```
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step
```

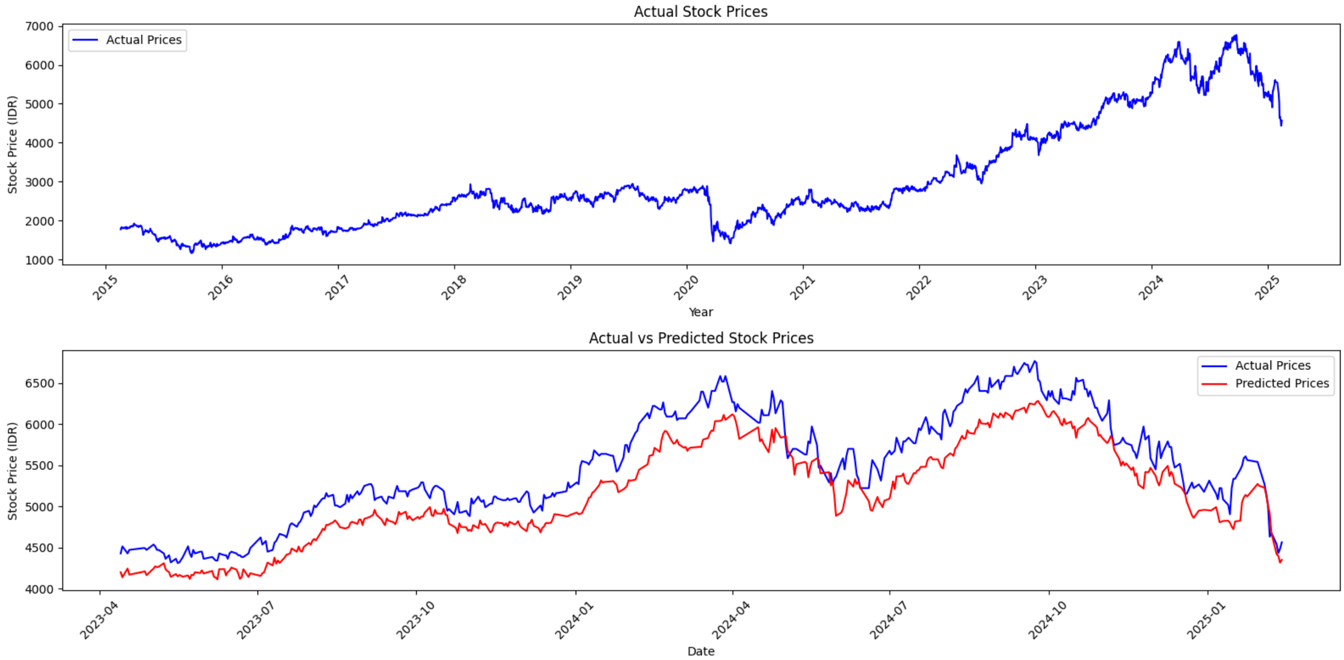
```python
# Denormalisasi hasil prediksi
test_data_partial = test_data[60:]
zero_fill = np.zeros((len(y_test), scaled_data.shape[1]))
zero_fill[:, 3] = predictions.flatten()
predictions_denormalized = scaler.inverse_transform(zero_fill)[:, 3]
y_test_denormalized = scaler.inverse_transform(test_data_partial)[:, 3]
```

```python
# === 8. Evaluasi Model TCN ===
r2 = r2_score(y_test_denormalized, predictions_denormalized)
mse = mean_squared_error(y_test_denormalized, predictions_denormalized)
rmse = np.sqrt(mse)
mape = np.mean(np.abs((y_test_denormalized - predictions_denormalized) / y_test_denormalized)) * 100

print("\nEvaluasi Model LSTM (Dalam Skala IDR):")
print(f"R Squared: {r2:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"Mean Absolute Percentage Error (MAPE): {mape:.2f} %")
```

```
Evaluasi Model TCN (Dalam Skala IDR):
R Squared: 0.7003
Mean Squared Error (MSE): 126005.86
Root Mean Squared Error (RMSE): 354.97
Mean Absolute Error (MAE): 326.13
Mean Absolute Percentage Error (MAPE): 5.88 %
```

```python
# === 9. Visualisasi Hasil ===
import matplotlib.dates as mdates
plt.figure(figsize=(16, 8))

# Grafik 1: Harga Aktual
plt.subplot(2, 1, 1)
plt.plot(df.index, df['Close'], color='blue', label='Actual Prices')
plt.gca().xaxis.set_major_locator(mdates.YearLocator())
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
```

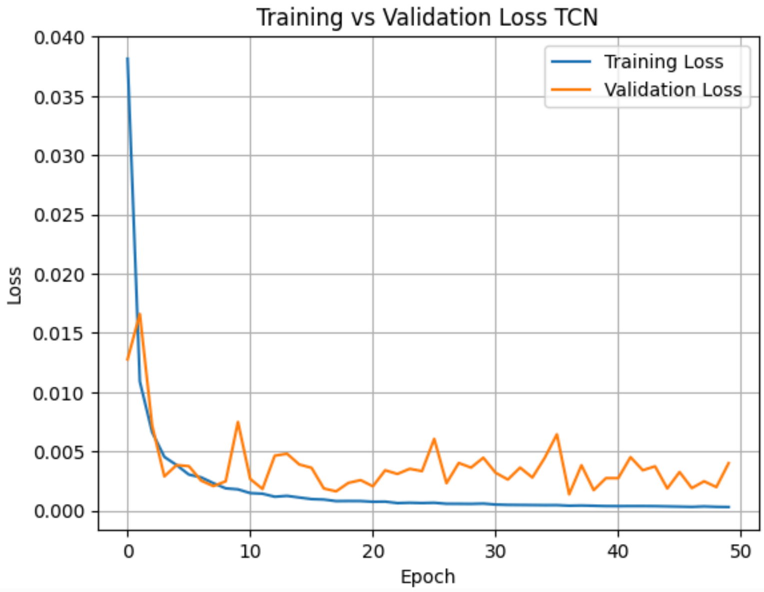
```
plt.title('Actual Stock Prices')
plt.xlabel('Year')
plt.ylabel('Stock Price (IDR)')
plt.legend()
plt.xticks(rotation=45)

# Grafik 2: Prediksi Harga vs Aktual
plt.subplot(2, 1, 2)
plt.plot(df.index[-len(y_test_denormalized):], y_test_denormalized, color='blue', label='Actual Prices')
plt.plot(df.index[-len(y_test_denormalized):], predictions_denormalized, color='red', label='Predicted Prices')
plt.title('Actual vs Predicted Stock Prices')
plt.xlabel('Date')
plt.ylabel('Stock Price (IDR)')
plt.legend()
plt.xticks(rotation=45)

plt.tight_layout()
plt.show()
```




```
# Ambil data terbaru dari test_data
last_60_days = test_data[-60:]  # Ambil 60 hari terakhir sebagai input
last_60_days = np.reshape(last_60_days, (1, last_60_days.shape[0], last_60_days.shape[1]))

# Prediksi harga hari berikutnya
predicted_next_day = model_tcn.predict(last_60_days)

# Denormalisasi hasil prediksi
zero_fill_next_day = np.zeros((1, scaled_data.shape[1]))  # Buat array kosong sesuai jumlah fitur
zero_fill_next_day[:, 3] = predicted_next_day.flatten()  # Masukkan prediksi ke indeks ke-3 ('Close')

# Denormalisasi prediksi harga hari berikutnya
predicted_next_day_price = scaler.inverse_transform(zero_fill_next_day)[:, 3]

# Output hasil prediksi
accuracy = r2 * 100 # Melakukan konversi ke persen
stock_name = df
print(f"Prediksi harga saham di hari berikutnya adalah Rp{predicted_next_day_price[0]:,.2f}, dengan tingkat akurasi sebesar {accuracy:.2
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
Prediksi harga saham di hari berikutnya adalah Rp4,410.99, dengan tingkat akurasi sebesar 70.03%
```

```
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training vs Validation Loss TCN')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
```




```
# Menyimpan model LSTM ke file .h5
model_tcn.save('model_tcn.h5')

import pickle

# Simpan scaler
with open('scaler.pkl', 'wb') as f:
    pickle.dump(scaler, f)

from google.colab import files

files.download('model_tcn.h5')
files.download('scaler.pkl')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
```