LAPORAN UTS

KECERDASAN BUATAN

Prediksi Harga Beras Medium di Indonesia Menggunakan Model Recurrent Neural Network (RNN)

NAMA: Fazli Haqqi M Ramadhani

KELAS: D

NIM: 202331233

NAMA DOSEN: Ir. Abdul Haris, S. Kom., M.Kom

1. **Import Library**

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense
from tensorflow.keras.callbacks import EarlyStopping
import datetime, os
```

Penjelasan:

- numpy, pandas: untuk manipulasi data.
- matplotlib.pyplot: untuk visualisasi grafik.
- keras.models, keras.layers: untuk membuat model RNN.
- MinMaxScaler: untuk normalisasi data agar model lebih cepat dan akurat saat belajar.

2. **Load Data**

```
DATA_PATH = "Rata-rata_Harga_Pangan_Bulanan_Tingkat_Produsen_Nasional (Angka April 2025).csv"
COMMODITY = "Beras Medium Tk. Penggilingan (Rp/Kg)"  # ubah sesuai kebutuhan
SEQUENCE_LENGTH = 12  # menggunakan 12 bulan terakhir untuk prediksi bulan berikutnya
EPOCHS = 300
BATCH_SIZE = 16
```

- **DATA_PATH:**
  - Menunjukkan lokasi file CSV yang digunakan sebagai data input
  - File ini berisi data harga pangan bulanan berbagai komoditas nasional sampai bulan April 2025.
- **COMMODITY:**
  - Kalau kita ingin memprediksi komoditas lain, tinggal ubah saja nilainya sesuai nama kolom di CSV.
- **SEQUENCE_LENGTH:**
  - Digunakan untuk menentukan berapa banyak data bulan sebelumnya yang dijadikan input oleh model
  - Artinya, model akan melihat 12 bulan ke belakang untuk memprediksi harga bulan ke-13. Ini penting karena data harga biasanya punya pola tahunan.
- **EPOCHS:**
  - Menentukan berapa kali seluruh data akan dipelajari ulang oleh model selama proses pelatihan
  - Jadi, model akan mengulang proses belajar sebanyak 300 kali agar bisa mengenali pola harga secara lebih akurat.
- **BATCH_SIZE:**
  - Menentukan berapa data yang diproses sekaligus dalam satu waktu saat training

- Ini membantu mempercepat proses pelatihan dan membuat pembelajaran model lebih stabil.

3. Proses Baca & Pembersihan Data

- **Membaca Data dari File CSV**

```
raw_df = pd.read_csv(DATA_PATH, sep=";", skiprows=1)
```

Di sini kita pakai pandas.read_csv() untuk membaca file data harga pangan dari file CSV. Kita lewati baris pertama (header tambahan) dengan skiprows=1 karena biasanya baris pertama hanya keterangan umum, bukan data.

- **Menghapus Kolom Kosong atau Tidak Perlu**

```
raw_df = raw_df.loc[:, ~raw_df.columns.str.contains('^Unnamed')]
```

Beberapa file CSV otomatis menghasilkan kolom Unnamed kalau ada kolom kosong di Excel. Di baris ini, kita bersihkan semua kolom yang tidak punya nama (biasanya kosong atau sisa dari Excel).

- **Membersihkan Nilai Harga (Kolom 'Harga')**

```
raw_df['Harga'] = (raw_df['Harga']
                   .str.replace('Rp', '', regex=False)
                   .str.replace('.', '', regex=False)
                   .str.replace(',', '', regex=False)
                   .replace('-', np.nan)
                   .astype(float))
```

Data harga biasanya masih dalam bentuk string seperti "Rp12.500", atau bahkan ada tanda - jika data kosong. Di langkah ini:

  - Tanda titik (.), koma (,), dan tulisan Rp dihapus.
  - Tanda - diganti jadi NaN (data kosong).
  - Hasil akhirnya dikonversi ke tipe float supaya bisa dihitung oleh model.

- **Mengubah Nama Bulan ke Angka**

```
month_map = {
    'Januari': 1, 'Februari': 2, 'Maret': 3, 'April': 4,
    'Mei': 5, 'Juni': 6, 'Juli': 7, 'Agustus': 8,
    'September': 9, 'Oktober': 10, 'November': 11, 'Desember': 12
}
raw_df['MonthNum'] = raw_df['Bulan'].map(month_map)
```

Bulan awalnya masih dalam format teks seperti "Januari", "Februari", dll. Kita buat kamus (dict) month_map yang mengubah nama-nama bulan jadi angka (1 sampai 12), agar bisa diproses sebagai waktu.

- **Membuat Kolom Tanggal Lengkap**

```
raw_df['Date'] = pd.to_datetime(dict(year=raw_df['Tahun'],
                                     month=raw_df['MonthNum'],
                                     day=1))
```

Dengan menggunakan kolom Tahun dan MonthNum, kita buat kolom baru Date dalam format waktu Python (datetime). Hari diset ke 1 karena data hanya mencatat rata-rata bulanan, jadi kita pakai awal bulan.

- **Memfilter Komoditas yang Dipilih**

```
df = raw_df[raw_df['Komoditas'] == COMMODITY].copy()
df = df.sort_values('Date').reset_index(drop=True)
```

Karena data ini berisi banyak komoditas, kita filter hanya yang sesuai dengan variabel COMMODITY, misalnya beras. Kemudian datanya diurutkan berdasarkan waktu (Date) dan indeksnya direset agar rapi.

- **Menampilkan 5 Data Pertama**

```
df.head()
```

Terakhir, kita tampilkan 5 baris pertama dari data hasil pembersihan, hanya untuk memastikan proses bersih-bersihnya berhasil.

➢ **OUTPUT**

	Komoditas	Tahun	Bulan	Harga	MonthNum	Date
0	Beras Medium Tk. Penggilingan (Rp/Kg)	2019	Januari	9497.0	1.0	2019-01-01
1	Beras Medium Tk. Penggilingan (Rp/Kg)	2019	Februari	9524.0	2.0	2019-02-01
2	Beras Medium Tk. Penggilingan (Rp/Kg)	2019	Maret	9379.0	3.0	2019-03-01
3	Beras Medium Tk. Penggilingan (Rp/Kg)	2019	April	9192.0	4.0	2019-04-01
4	Beras Medium Tk. Penggilingan (Rp/Kg)	2019	Mei	9123.0	5.0	2019-05-01

4. Normalisasi & Sekuens Data

- **Normalisasi Data Harga**

```
scaler = MinMaxScaler(feature_range=(0, 1))
prices_scaled = scaler.fit_transform(df[['Harga']])
```

Di sini kita pakai MinMaxScaler dari sklearn.preprocessing untuk mengubah data harga ke rentang antara 0 sampai 1. Tujuannya supaya data lebih stabil saat dipelajari oleh model RNN, karena model lebih sensitif kalau ada nilai harga yang terlalu besar atau kecil.

- **Fungsi Pembuatan Dataset Sekuensial**

```
def create_sequences(data, seq_length):
```

Kita bikin fungsi create_sequences() untuk membentuk data menjadi urutan data berukuran tetap. Misalnya, jika SEQUENCE_LENGTH = 12, maka setiap input ke model adalah 12 bulan sebelumnya, dan target (label) adalah harga bulan ke-13.

- **Penjelasan Isi Fungsi**

```
for i in range(len(data) - seq_length):
    X.append(data[i:i + seq_length])
    y.append(data[i + seq_length])
```

- X = sekumpulan data input (12 bulan sebelumnya)
- y = data target atau label (harga bulan ke-13)
  Jadi kita sliding window sepanjang 12, lalu geser 1 langkah untuk ambil data berikutnya.

- **Konversi ke Array Numpy**

```
return np.array(X), np.array(y)
```

Kita ubah list Python ke bentuk array NumPy karena nanti TensorFlow atau Keras lebih mudah membaca bentuk array ini.

- **Pemanggilan Fungsi**

```
X, y = create_sequences(prices_scaled, SEQUENCE_LENGTH)
```

Di baris ini, kita buat dataset X dan y yang akan digunakan untuk melatih model RNN. X adalah data input dengan shape (jumlah sampel, panjang urutan, 1 fitur), dan y adalah target-nya.

- **Cek Ukuran Data**

```
print("Shape X:", X.shape, "Shape y:", y.shape)
```

Ini buat memastikan bentuk data sesuai. Hasil Shape X: (64, 12, 1) artinya:

- 64 sampel (data training)
- Setiap sampel berisi 12 data bulan
- 1 fitur (harga)

Sedangkan y punya shape (64, 1) karena kita prediksi 1 nilai untuk tiap input.

➢ **OUTPUT**

```
Shape X: (64, 12, 1) Shape y: (64, 1)
```

5. Membangun & Melatih Model RNN

- **Membangun Arsitektur Model**

```
model = Sequential([
    SimpleRNN(64, activation='tanh', input_shape=(SEQUENCE_LENGTH, 1)),
    Dense(1)
])
```

Di sini kita membangun model RNN menggunakan Sequential dari Keras. Isinya ada dua lapisan (layer):

- SimpleRNN(64): Layer utama dengan 64 unit neuron, cocok untuk data deret waktu (time series). Kita pakai aktivasi tanh karena bisa mengatur nilai output antara -1 dan 1, bagus untuk mempelajari urutan.
- Dense(1): Layer output yang mengeluarkan 1 nilai prediksi (harga bulan berikutnya).

Bentuk input-nya adalah (SEQUENCE_LENGTH, 1), artinya tiap input punya panjang 12 data (12 bulan) dan 1 fitur (harga).

- **Compile Model**

```
model.compile(optimizer='adam', loss='mse')
```

Kita menggunakan:

- Optimizer adam: metode training yang cepat dan stabil.
- Loss function mse: Mean Squared Error, karena kita memprediksi nilai numerik (regresi).

- **EarlyStopping**

```
es = EarlyStopping(monitor='loss', patience=20, restore_best_weights=True)
```

Ini digunakan untuk menghentikan training lebih awal jika loss tidak membaik selama 20 epoch, supaya:

- Training tidak terlalu lama
- Menghindari overfitting
- Memakai bobot model terbaik yang ditemukan selama proses training

- **Melatih Model**

```
history = model.fit(X, y, epochs=EPOCHS, batch_size=BATCH_SIZE, verbose=1, callbacks=[es])
```

Kita melatih model dengan:

- X, y sebagai input dan target
- epochs = 300 → jumlah maksimum pelatihan
- batch_size = 16 → data dibagi jadi kelompok kecil
- verbose = 1 → menampilkan progress tiap epoch
- callbacks = [es] → menggunakan EarlyStopping

6. Prediksi Harga 2 Tahun ke Depan

- **Menentukan Berapa Banyak Bulan Diprediksi**

```
future_steps = 24  # 24 bulan = 2 tahun
```

Kita mau prediksi harga untuk 24 bulan ke depan (artinya 2 tahun).

- **Ambil Data 12 Bulan Terakhir**

```
last_sequence = prices_scaled[-SEQUENCE_LENGTH:]  # 12 bulan terakhir
```

Karena model kita butuh 12 bulan ke belakang untuk memprediksi bulan ke depan, kita ambil 12 data terakhir dari data training.

- **Persiapan Awal**

```
predictions_scaled = []
input_seq = last_sequence.copy()
```

  - **predictions_scaled: menampung hasil prediksi (masih dalam bentuk hasil normalisasi).**
  - **input_seq: urutan data awal yang akan terus diperbarui selama looping prediksi.**

- **Looping Prediksi 24 Kali**

```
for _ in range(future_steps):
    pred_scaled = model.predict(input_seq.reshape(1, SEQUENCE_LENGTH, 1), verbose=0)[0][0]
    predictions_scaled.append(pred_scaled)
    # Perbarui input_seq
    input_seq = np.append(input_seq[1:], [[pred_scaled]], axis=0)
```

Ini bagian utama prediksi:

- Setiap kali, kita bentuk input_seq menjadi shape (1, 12, 1) dan masukkan ke model untuk mendapatkan 1 prediksi ke depan.
- Prediksi disimpan di predictions_scaled.
- Lalu input diperbarui: data lama dibuang, hasil prediksi baru dimasukkan. Jadi input selalu bergerak maju seperti sliding window.

- **Denormalisasi**

```
predictions = scaler.inverse_transform(np.array(predictions_scaled).reshape(-1, 1)).flatten()
```

Karena hasil prediksi tadi masih dalam bentuk data yang dinormalisasi, kita ubah kembali ke skala harga asli menggunakan inverse_transform.

- **Buat Tanggal Prediksi**

```
last_date = df['Date'].iloc[-1]
future_dates = pd.date_range(last_date + pd.DateOffset(months=1),
                             periods=future_steps, freq='MS')
```

- Ambil tanggal terakhir dari data.
- Lalu buat daftar tanggal ke depan selama 24 bulan (frekuensi bulanan 'MS' = Month Start).

- **Gabungkan Hasil ke DataFrame**

```
pred_df = pd.DataFrame({
    'Date': future_dates,
    'Predicted_Price': predictions
})
```

Akhirnya, hasil prediksi dan tanggal disusun jadi DataFrame agar mudah dianalisis atau divisualisasikan.

➢ **OUTPUT**

	Date	Predicted_Price
0	2025-05-01	12424.921875
1	2025-06-01	12353.971680
2	2025-07-01	12395.587891
3	2025-08-01	12404.455078
4	2025-09-01	12408.277344

7. Visualisasi Prediksi Harga

- **Ukuran Grafik**

```
plt.figure(figsize=(12, 6))
```

Kita atur ukuran grafik supaya lebih lebar dan enak dilihat: 12 satuan lebar dan 6 tinggi.

- **Plot Data Aktual**

```
plt.plot(df['Date'], df['Harga'], label='Actual')
```

Ini menampilkan data harga aktual (historis) dari dataset utama. Digambarkan dalam bentuk garis berdasarkan tanggal.

- **Plot Data Prediksi**

```
plt.plot(pred_df['Date'], pred_df['Predicted_Price'], label='Predicted')
```

Ini menampilkan hasil prediksi dari model RNN selama 24 bulan ke depan, juga dalam bentuk garis.

- **Judul dan Label**

```
plt.title(f'Prediksi Harga {COMMODITY}')
plt.xlabel('Tanggal')
plt.ylabel('Harga (Rp)')
```

  - Judul grafik dibuat dinamis sesuai komoditas (misalnya "Beras Medium Tk. Penggilingan").
  - Sumbu X dilabeli "Tanggal", sumbu Y "Harga (Rp)".

- **Legends dan Layout**

```
plt.legend()
plt.tight_layout()
```

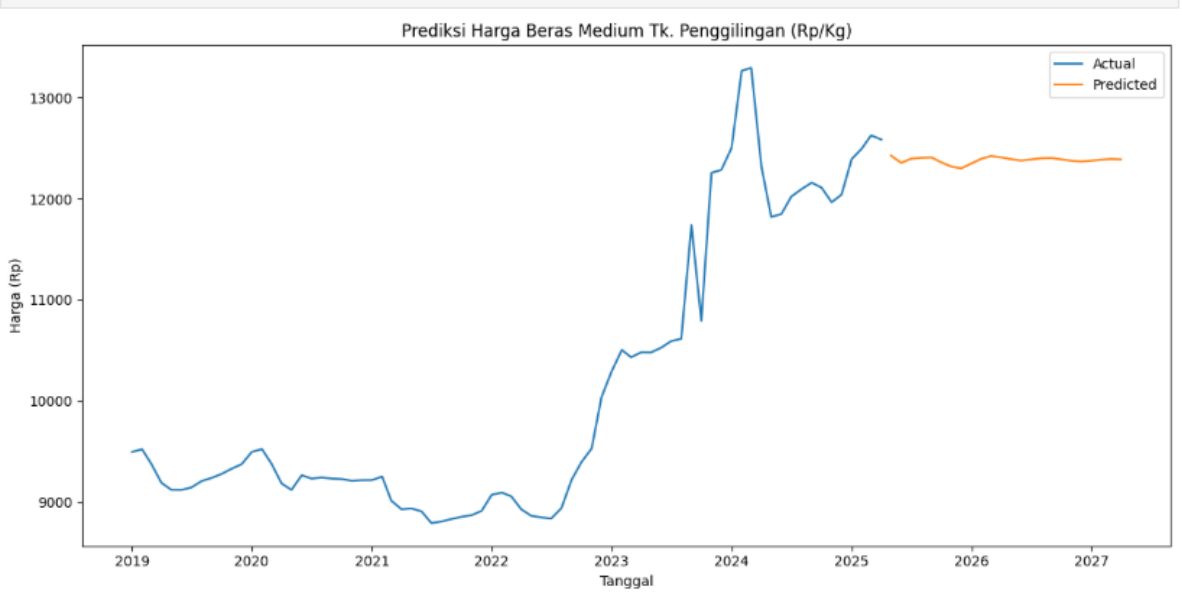
  - legend() menampilkan label untuk garis 'Actual' dan 'Predicted'.
  - tight_layout() merapikan tampilan agar tidak terpotong.

- **Tampilkan Grafik**

```
plt.show()
```

Ini perintah terakhir untuk menampilkan hasil visualisasi.

➢ **OUTPUT GRAFIK**



- **Garis Biru: Data Actual**
  - Ini adalah harga aktual beras dari tahun 2019 hingga sekitar awal 2025.
  - Terlihat bahwa harga beras relatif stabil dari 2019 sampai sekitar pertengahan 2022.
  - Mulai akhir 2022 hingga 2024, terjadi kenaikan harga yang tajam — ini bisa disebabkan oleh faktor eksternal seperti inflasi, gangguan pasokan, atau kebijakan pemerintah.
  - Setelah puncaknya di awal 2024, harga mulai sedikit turun dan stabil menjelang 2025.
- **Garis Oranye: Hasil Prediksi Model**
  - Garis oranye menunjukkan prediksi harga beras untuk **24 bulan ke depan** (2 tahun), mulai dari 2025 sampai 2027.
  - Pola prediksi cenderung **stabil dengan sedikit fluktuasi** kecil di sekitar Rp 12.200–12.400 per kilogram.
  - Ini menandakan bahwa model RNN memperkirakan harga akan **cenderung stabil** dalam 2 tahun ke depan, tidak ada lonjakan ekstrem seperti sebelumnya.

8. **Menyimpan Hasil Prediksi**

- **Penamaan File Otomatis**

```
output_csv = f"prediksi_{COMMODITY.replace(' ', '_').replace('(', '').replace(')', '').replace('/', '')}.csv"
```

Bagian ini membuat nama file CSV secara otomatis berdasarkan nama komoditas, misalnya:

  - kalau COMMODITY = "Beras Medium Tk. Penggilingan (Rp/Kg)", maka akan jadi:
    Prediksi_Beras_Medium_Tk._Penggilingan_RpKg.csv

Tujuannya agar nama file rapi dan aman dipakai di sistem file (tidak ada spasi, tanda kurung, atau slash).

- **Simpan ke CSV**

```
pred_df.to_csv(output_csv, index=False)
```

DataFrame pred_df yang berisi tanggal prediksi dan hasil prediksinya disimpan ke file .csv. index=False artinya kolom indeks tidak ikut disimpan ke dalam file.

- **Konfirmasi**

```
print(f"Prediksi disimpan ke {output_csv}")
```

Ini hanya untuk memberi info ke user bahwa file hasil sudah berhasil dibuat.

9. **Evaluasi Model RNN**

- **Import Library Evaluasi**

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import numpy as np
```

- Digunakan untuk membagi data (train_test_split) dan mengevaluasi performa model seperti akurasi dan confusion matrix.
- matplotlib dan numpy digunakan untuk visualisasi dan manipulasi data numerik**.**

- **Pisahkan Data Training dan Testing**

```
try:
    X_train, X_test, y_train, y_test
except NameError:
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
```

  - Jika sebelumnya belum ada variabel X_train dan X_test, maka program akan otomatis membagi data (X, y) menjadi 80% training dan 20% testing.
  - shuffle=False penting untuk **time series**, karena urutan waktu harus dipertahankan.
- **Prediksi Menggunakan Model**

```
y_pred = model.predict(X_test)
```

  Model yang sudah dilatih akan digunakan untuk memprediksi harga (y_pred) dari data test (X_test).
- **Visualisasi Hasil Prediksi vs Data Asl**

```
plt.plot(y_test, label='Actual')
plt.plot(y_pred, label='Predicted')
```

  - Grafik ini menunjukkan seberapa dekat prediksi model terhadap data asli.
  - Kalau garis *Predicted* dan *Actual* saling menumpuk, berarti modelnya bagus.
- **Confusion Matrix**

```
y_pred_classes = (y_pred > 0.5).astype(int)
y_test_bin = (y_test > 0.5).astype(int)
```

  - Hanya digunakan kalau modelnya klasifikasi.
  - try-except digunakan untuk menghindari error, karena RNN kamu digunakan untuk regresi harga (bukan klasifikasi).
- **Grafik Loss dan Akurasi Selama Pelatihan**

  Loss

```
plt.plot(history.history['loss'], label='Training Loss')
```

  - Loss adalah ukuran seberapa jauh prediksi dari target sesungguhnya.
  - Jika val_loss tersedia, akan dibandingkan juga.

  Accuracy

```
plt.plot(history.history['accuracy'], label='Training Accuracy')
```

  - Ini hanya muncul kalau modelnya klasifikasi.

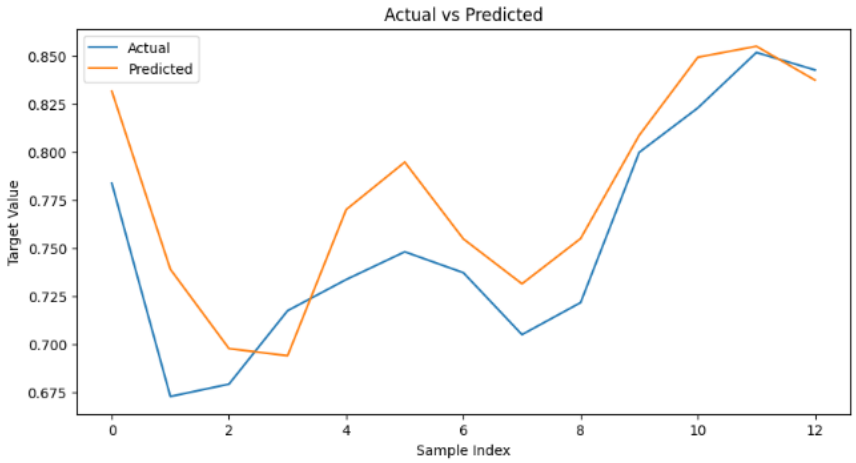
- Untuk regresi biasanya tidak tersedia, jadi bagian ini akan di-skip otomatis.

- **Summary Struktur Model**

```
model.summary()
```

  - Menampilkan detail model: jumlah layer, parameter, dan bentuk input/output.
  - Berguna untuk debugging atau memahami arsitektur model.

➢ **OUTPUT GRAFIK**



Penjelasan Grafik:

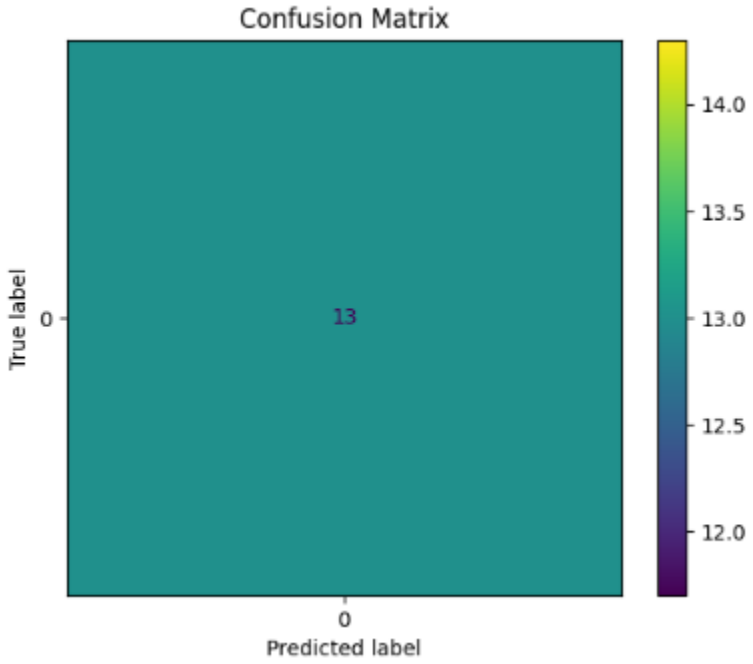
- Grafik ini menunjukkan perbandingan antara nilai aktual (data harga sebenarnya dari data test) dan nilai prediksi (hasil prediksi model RNN) pada data test.
- Sumbu X: Sample Index (indeks data test, mewakili urutan waktu atau bulan dalam data test).
- Sumbu Y: Target Value (nilai harga yang sudah dinormalisasi ke rentang [0,1] menggunakan MinMaxScaler).
- Garis biru: Harga aktual.
- Garis oranye: Harga prediksi.

Analisis Hasil:

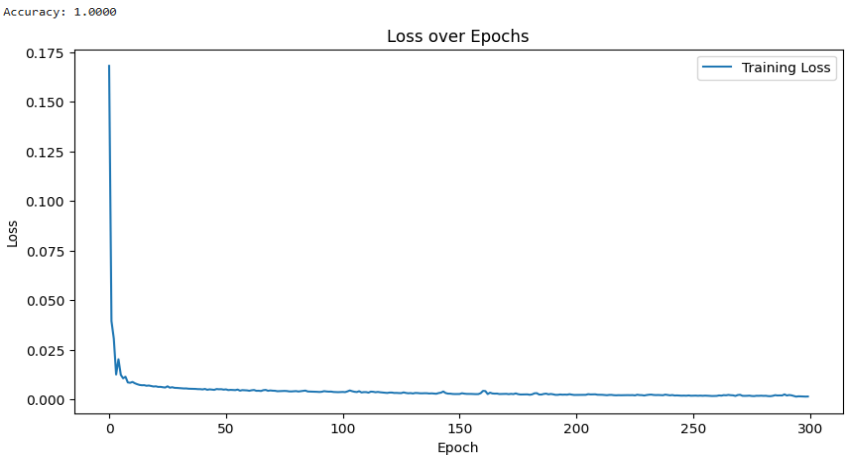
- **Keseluruhan Tren**: Garis prediksi (oranye) mengikuti tren garis aktual (biru) dengan cukup baik, terutama pada indeks 3 hingga 12. Ini menunjukkan bahwa model RNN mampu menangkap pola umum dalam data harga.
- Penyimpangan: Ada beberapa penyimpangan, misalnya pada indeks 0 hingga 2, di mana prediksi awalnya terlalu tinggi dibandingkan data aktual. Penyimpangan ini mungkin disebabkan oleh kurangnya data historis awal yang cukup untuk prediksi akurat pada

langkah pertama.

- Akurasi: Pada indeks 8 hingga 12, prediksi sangat mendekati nilai aktual, menunjukkan bahwa model bekerja lebih baik pada data yang lebih baru dalam rentang test.
- Maksud: Grafik ini menunjukkan bahwa model RNN cukup baik dalam memprediksi harga beras medium, meskipun ada sedikit ketidaksesuaian pada beberapa titik. Model ini dapat digunakan untuk memberikan gambaran tren harga di masa depan, meskipun tidak sepenuhnya sempurna.



Untuk confusion matrix tidak mendukung pada dataset saya



Penjelasan Grafik:

- Grafik ini menunjukkan perkembangan training loss (kerugian pelatihan) selama proses pelatihan model RNN.

- Sumbu X: Epoch (jumlah iterasi pelatihan, hingga 300 epoch).
- Sumbu Y: Loss (nilai Mean Squared Error, metrik yang digunakan untuk mengukur seberapa jauh prediksi model dari data aktual).
- Garis biru: Training loss.

Analisis Hasil:

- Penurunan Loss: Loss mulai dari 0.175 pada epoch awal dan turun drastis hingga di bawah
- 0.025 dalam 50 epoch pertama, kemudian stabil di sekitar 0.005 hingga epoch 300. Ini menunjukkan bahwa model belajar dengan baik dan konvergen (mencapai nilai loss yang rendah dan stabil).
- Early Stopping: Meskipun diatur hingga 300 epoch, model menggunakan EarlyStopping dengan patience=20, yang berarti pelatihan akan berhenti jika loss tidak membaik setelah 20 epoch. Grafik menunjukkan loss stabil setelah ~100 epoch, sehingga pelatihan mungkin berhenti lebih awal.
- Performa Model: Loss yang rendah dan stabil menunjukkan bahwa model RNN telah belajar pola dalam data pelatihan dengan baik. Namun, karena tidak ada validation loss (grafik hanya menampilkan training loss), kita tidak tahu apakah model overfit (terlalu cocok dengan data pelatihan) atau tidak.
- Maksud: Grafik ini menunjukkan bahwa model RNN dilatih dengan sukses, dengan loss yang sangat rendah pada akhir pelatihan. Namun, tanpa validation loss, kita perlu evaluasi lebih lanjut untuk memastikan model tidak overfit dan dapat digeneralisasi ke data baru.

Model: "sequential"

Layer (type)	Output Shape	Param #
simple_rnn (SimpleRNN)	(None, 64)	4,224
dense (Dense)	(None, 1)	65

Total params: 12,869 (50.27 KB)
Trainable params: 4,289 (16.75 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 8,580 (33.52 KB)

Penjelasan: Menampilkan arsitektur model dengan 4,224 parameter pada lapisan RNN dan 65 pada lapisan Dense.