ANALISIS MENDALAM: PROYEK PEMODELAN PREDIKSI HARGA TELUR AYAM DI GORONTALO

Pemodelan Prediksi Harga Telur Ayam di Pasar Modern Provinsi Gorontalo Periode Juli 2024-2025 dengan Pendekatan Jaringan Syaraf Tiruan MLP", merupakan sebuah studi kasus yang sangat relevan dalam penerapan kecerdasan buatan (AI) untuk memecahkan masalah di dunia nyata. Dikerjakan oleh M. Rafly Aulia Akbar (NPM: 2210010574), proyek ini bertujuan membangun sebuah model yang mampu memprediksi fluktuasi harga telur, sebuah komoditas penting, dengan menggunakan metode Jaringan Saraf Tiruan (JST) atau *Artificial Neural Network.*

Tahap 1: Pengumpulan dan Persiapan Data

1. Sumber Data: Data historis harga telur ayam di pasar modern Provinsi Gorontalo diperoleh dari situs tepercaya, databoks.katadata.co.id. Pemilihan sumber data yang kredibel adalah langkah awal yang krusial untuk memastikan validitas model yang akan dibangun.
   (“https://databoks.katadata.co.id/ekonomi-makro/statistik/5cce9117b2c50e6/harga-telur-ayam-harian-di-pasar-modern-provinsi-gorontalo”).

2. Pemuatan Data:

```
# 1. Upload file Excel
from google.colab import files
import pandas as pd
import io
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

uploaded = files.upload()

# 2. Ambil nama file Excel dari hasil upload
filename = list(uploaded.keys())[0]

# 3. Baca Excel jadi DataFrame
df = pd.read_excel(io.BytesIO(uploaded[filename]))

# 4. Lihat isi file
df.head()
```

```
Choose Files  No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving harga-telur-ayam-di-pasar-modern-periode-juli-2024-2025 (1).xlsx to harga-telur-ayam-di-pasar-modern-periode-juli-2024-2025 (1) (1).xlsx
```

	Nama Data	Gorontalo
0	2024-07-04	46.25
1	2024-07-05	46.25
2	2024-07-08	46.25
3	2024-07-09	46.25
4	2024-07-10	47.20

Proyek ini dieksekusi menggunakan Google Colaboratory, sebuah platform berbasis cloud yang ideal untuk proyek *machine learning*. Langkah pertama adalah mengunggah dataset yang sudah disiapkan dalam format Excel ke dalam lingkungan Colab. Dengan menggunakan *library Pandas* di Python, file Excel tersebut dibaca dan diubah menjadi sebuah *DataFrame*—sebuah struktur data tabular yang menjadi fondasi utama untuk analisis data di Python. Perintah **df.head()** kemudian dijalankan untuk menampilkan lima baris pertama dari data, sebuah "pemeriksaan kewarasan" (*sanity check*) untuk memastikan data telah dimuat dengan benar dan sesuai format yang diharapkan.

Tahap 2: Pra-pemrosesan Data (*The Secret Sauce*)

Data mentah jarang sekali bisa langsung digunakan untuk melatih model. Di sinilah proses pra-pemrosesan data berperan penting untuk mengubah data menjadi format yang "dimengerti" oleh model Jaringan Saraf Tiruan.

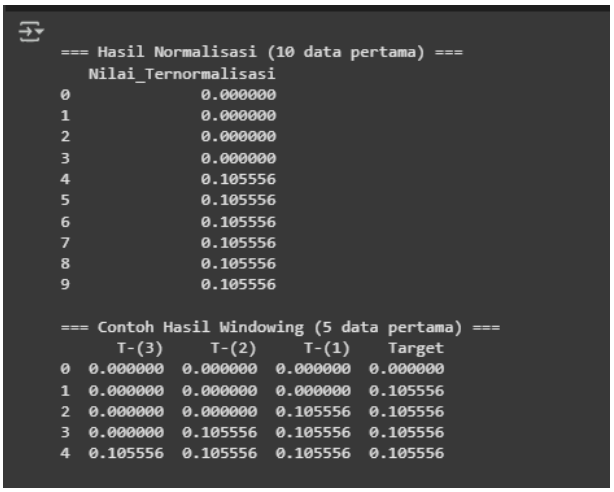
1. Normalisasi Data:

```
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df[['Gorontalo']]) # Kolom 'Gorontalo'

# Buat DataFrame untuk hasil normalisasi
normalized_df = pd.DataFrame(scaled_data,
                             columns=['Nilai_Ternormalisasi'])
print("\n=== Hasil Normalisasi (10 data pertama) ===")
print(normalized_df.head(10))

# 3. Windowing
def windowed_dataset(series, window_size):
    X, y = [], []
    for i in range(len(series) - window_size):
        X.append(series[i:i+window_size])
        y.append(series[i+window_size])
    return np.array(X), np.array(y)

window_size = 3
X, y = windowed_dataset(scaled_data, window_size)

# 4. Tampilkan Hasil Windowing
# Tampilkan 5 window pertama dalam bentuk tabel
window_df = pd.DataFrame(X[:5].reshape(5, window_size), columns=[f'T-({window_size - i})' for i in range(window_size)])
window_df['Target'] = y[:5]

print("\n=== Contoh Hasil Windowing (5 data pertama) ===")
print(window_df)
```

Jaringan saraf tiruan bekerja paling optimal ketika data input berada dalam skala yang seragam. Harga telur yang berfluktuasi dalam ribuan rupiah bisa jadi terlalu besar untuk diproses secara efisien. Oleh karena itu, dilakukan normalisasi menggunakan **MinMaxScaler** dari *library Scikit-learn*. Teknik ini "memampatkan" rentang nilai harga ke dalam skala antara 0 dan 1. Nilai terendah dalam dataset akan menjadi 0, nilai

tertinggi menjadi 1, dan nilai lainnya akan direpresentasikan secara proporsional di antara keduanya. Ini membantu model belajar lebih cepat dan lebih stabil.

2. Windowing (Jendela Data):



```
=== Hasil Normalisasi (10 data pertama) ===
   Nilai_Ternormalisasi
0              0.000000
1              0.000000
2              0.000000
3              0.000000
4              0.105556
5              0.105556
6              0.105556
7              0.105556
8              0.105556
9              0.105556

=== Contoh Hasil Windowing (5 data pertama) ===
      T-(3)     T-(2)     T-(1)    Target
0  0.000000  0.000000  0.000000  0.000000
1  0.000000  0.000000  0.000000  0.105556
2  0.000000  0.000000  0.105556  0.105556
3  0.000000  0.105556  0.105556  0.105556
4  0.105556  0.105556  0.105556  0.105556
```

Ini adalah inti dari pemodelan data deret waktu (*time series*). Model tidak bisa memprediksi harga besok hanya dengan melihat harga hari ini; ia butuh konteks historis. Metode windowing digunakan untuk menyediakan konteks ini. Dalam proyek ini, **window_size** ditetapkan sebesar 3. Artinya, model akan "diajari" untuk melihat data harga dari 3 hari sebelumnya (disebut sebagai fitur atau *features*) untuk menebak harga pada hari berikutnya (disebut sebagai target atau *label*). Proses ini digeser satu per satu sepanjang dataset, sehingga menghasilkan banyak sekali set data latihan. Contohnya, data hari ke-1, 2, dan 3 digunakan untuk memprediksi harga hari ke-4; data hari ke-2, 3, dan 4 digunakan untuk memprediksi harga hari ke-5, dan seterusnya.

Tahap 3: Pembagian Dataset Latih dan Uji

Bagaimana kita tahu jika model yang kita bangun benar-benar pintar atau hanya "menghafal" data? Jawabannya adalah dengan membagi dataset.

Rasio Pembagian:

```
[ ] from sklearn.model_selection import train_test_split

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.40, shuffle=False
    )
```

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

```
    X_train shape: (118, 3, 1)
    X_test shape : (79, 3, 1)
    y_train shape: (118, 1)
    y_test shape : (79, 1)
```

```
[ ] print("\nContoh X_train[0]:", X_train[0].flatten())
    print("Target y_train[0]:", y_train[0][0])
```

```

    Contoh X_train[0]: [0. 0. 0.]
    Target y_train[0]: 0.0
```

```
[ ] print("\nContoh X_train[0]:", X_train[0].flatten())
    print("Target y_train[0]:", y_train[0][0])
```

```

    Contoh X_train[0]: [0. 0. 0.]
    Target y_train[0]: 0.0
```

Pada tahap ini, dataset dibagi menggunakan fungsi **train_test_split**.

- Rasio Pembagian: Parameter **test_size=0.40** digunakan untuk membagi data menjadi 60% data latih (118 data) dan 40% data uji (79 data).
- Menjaga Urutan: **shuffle=False** adalah parameter kunci yang memastikan urutan data tidak diacak. Hal ini sangat penting untuk data deret waktu agar pola kronologisnya tetap terjaga.
- Bukti: Contoh **X_train[0]** yang ditampilkan adalah data paling awal, membuktikan bahwa urutan data berhasil dipertahankan.

```
# Tampilkan 3 data awal dari X_test dan targetnya
test_window_df = pd.DataFrame(X_test[:3].reshape(3, X_test.shape[1]),
                              columns=[f'T-({X_test.shape[1]} - i)' for i
                                       in range(X_test.shape[1])])
test_window_df['Target'] = y_test[:3]
print("\nContoh isi X_test:")
print(test_window_df)
```

```
Contoh isi X_test:
   T-(3 - i)  T-(3 - i)  T-(3 - i)    Target
0   0.811111   0.811111   0.811111  0.811111
1   0.811111   0.811111   0.811111  0.811111
2   0.811111   0.811111   0.811111  0.811111
```

Setelah dataset dibagi, penting untuk memeriksa isi dari data uji. Gambar ini menampilkan tiga contoh pertama dari data uji (*X_test*) beserta targetnya. Tujuannya adalah untuk memverifikasi struktur data yang akan digunakan untuk mengevaluasi performa model. Data inilah yang akan menjadi tolok ukur untuk melihat seberapa akurat prediksi model terhadap data yang belum pernah "dilihat" sebelumnya.

Tahap 4: Arsitektur dan Pelatihan Model JST

Ini adalah tahap di mana "otak" dari sistem prediksi ini dirancang.

1. Arsitektur Model MLP:

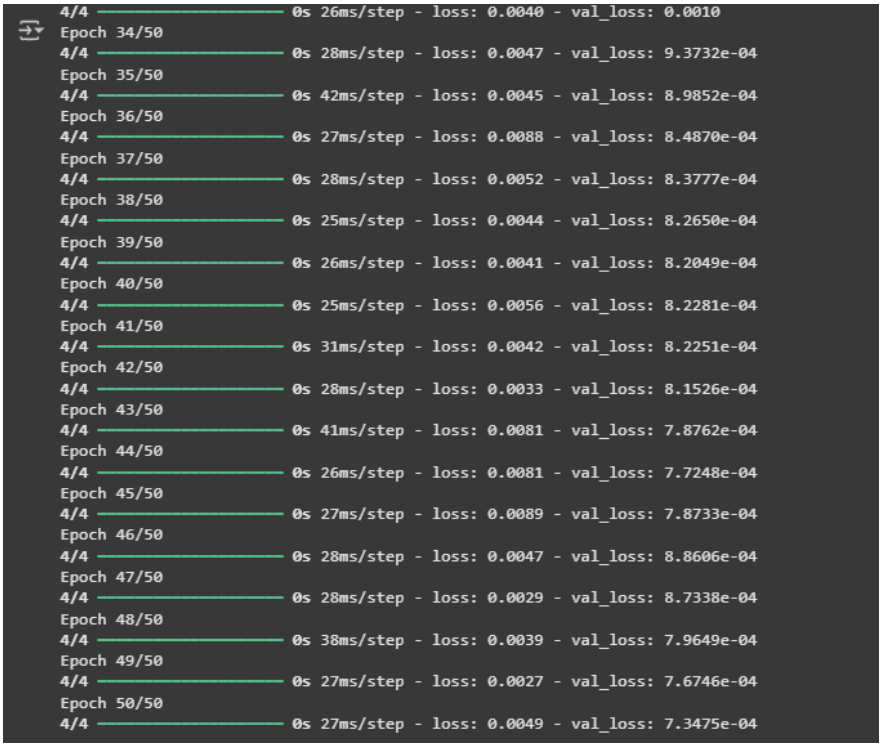
```
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(window_size,)))
model.add(Dense(32, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
history = model.fit(X_train, y_train, epochs=50,
                    validation_data=(X_test, y_test))
```

Model yang digunakan adalah *Multi-Layer Perceptron* (MLP) yang dibangun secara sekuensial.

- *Input Layer*: Lapisan ini menerima input berupa 3 nilai data historis (sesuai **window_size**).
- *Hidden Layer 1*: Terdiri dari 64 neuron dengan fungsi aktivasi *relu*. Lapisan ini bertugas mengenali pola-pola dasar dari data input.
- *Hidden Layer 2*: Terdiri dari 32 neuron dengan fungsi aktivasi *relu*, yang bertugas mengombinasikan pola-pola dari lapisan sebelumnya untuk membentuk pola yang lebih kompleks.

- *Output Layer*: Hanya memiliki 1 neuron, karena tujuannya adalah untuk mengeluarkan satu nilai prediksi, yaitu harga telur pada hari berikutnya.

2. Kompilasi dan Pelatihan:



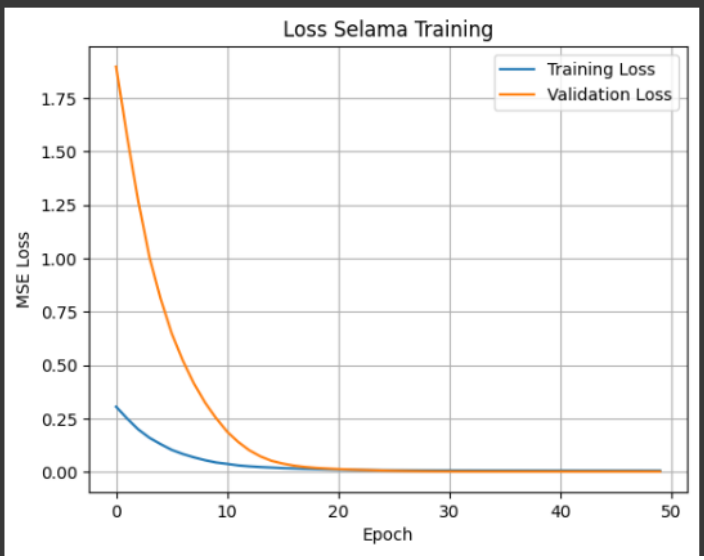
Sebelum dilatih, model "dikompilasi" dengan **optimizer='adam'** yang bertugas memandu model untuk belajar secara efisien, dan **loss='mse'** (*Mean Squared Error*) yang menjadi metrik untuk mengukur tingkat kesalahan prediksi. Model kemudian dilatih selama 50 *epoch*, artinya model melihat keseluruhan data latih sebanyak 50 kali untuk terus memperbaiki kemampuannya.

Tahap 5: Evaluasi Performa Model

Setelah model selesai dilatih, saatnya mengukur kinerjanya.

1. *Grafik Loss*:

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

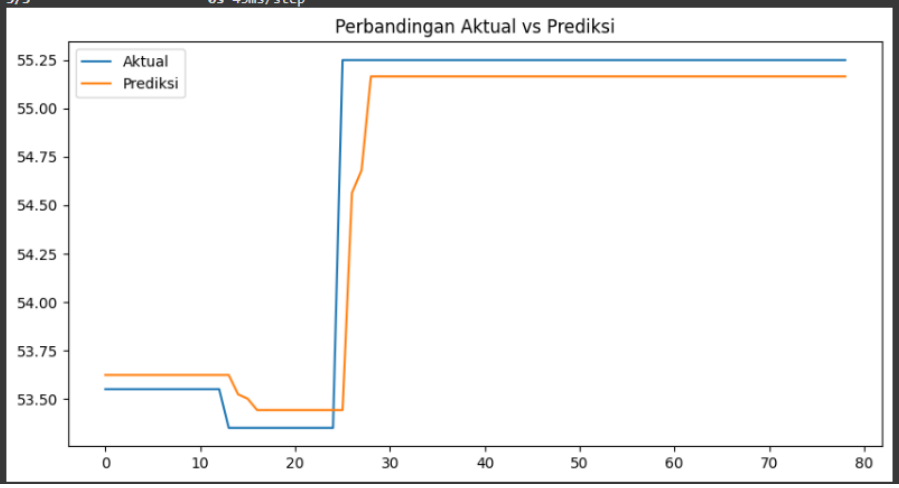


Grafik "Loss Selama Training" menunjukkan tingkat kesalahan model pada data latih (*Training Loss*) dan data uji (*Validation Loss*) di setiap epoch. Hasilnya sangat ideal: kedua kurva menurun secara konsisten dan konvergen mendekati nol. Ini menandakan bahwa model berhasil belajar dengan baik dan tidak mengalami *overfitting* (terlalu menghafal data latih).

2. Grafik Perbandingan Aktual vs. Prediksi:

```
y_pred = model.predict(X_test)
y_pred_inv = scaler.inverse_transform(y_pred)
y_test_inv = scaler.inverse_transform(y_test.reshape(-1, 1))
plt.figure(figsize=(10,5))
plt.plot(y_test_inv, label='Aktual')
plt.plot(y_pred_inv, label='Prediksi')
plt.legend()
plt.title('Perbandingan Aktual vs Prediksi')
plt.show()
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
```



Grafik ini adalah visualisasi paling jelas dari performa model. Garis biru menunjukkan harga telur aktual, sementara garis oranye adalah hasil prediksi model. Terlihat bahwa model mampu mengikuti tren data aktual dengan sangat baik, termasuk saat terjadi lonjakan harga.

3. RMSE (*Root Mean Squared Error*):

```
[ ] rmse = np.sqrt(mean_squared_error(y_test_inv, y_pred_inv))
    print("RMSE:", rmse)
```

```
RMSE: 0.2439562629658546
```

Untuk mendapatkan ukuran kuantitatif dari kesalahan, digunakan metrik RMSE. Nilai ini menunjukkan rata-rata seberapa besar selisih antara harga prediksi dengan harga sebenarnya. Pada pengujian awal dengan **test_size=0.40**, didapatkan nilai RMSE sebesar 0.243.

Tahap 6: Eksperimen dan Hasil Akhir

Tidak berhenti pada satu hasil, dilakukan serangkaian eksperimen untuk menemukan konfigurasi terbaik. Pengujian dilakukan dengan mengubah-ubah proporsi **test_size** sambil mempertahankan jumlah epoch sebanyak 50.

Hasilnya disajikan dalam tabel berikut:

Test_Size	RMSE
0.10	0,027
0.15	0,003
0.20	0,008
0.25	0,015
0.30	0,257
0.35	0,246
0.40	0,243

Dari tabel tersebut, dapat ditarik kesimpulan yang sangat signifikan:

Tingkat error terendah (RMSE) sebesar 0,003 dicapai saat menggunakan **test_size** sebesar 0.15. Ini menyiratkan bahwa dengan memberikan porsi data latih yang lebih besar kepada model (85% data), kemampuannya untuk mempelajari pola dan melakukan prediksi pada data baru menjadi jauh lebih akurat.

Secara keseluruhan, proyek ini berhasil menunjukkan bahwa pendekatan Jaringan Saraf Tiruan MLP sangat efektif untuk memodelkan dan memprediksi data deret waktu seperti harga komoditas. Dari pengumpulan data hingga evaluasi dan eksperimen, setiap langkah telah dilakukan secara sistematis untuk menghasilkan model prediksi yang andal.