KODE TUGAS AKHIR

Nama: Syifa Frnanda

NIM: 121450094

Prodi:Sains Data

Download data

```
import datetime
from zoneinfo import ZoneInfo

# Buat waktu awal dan akhir tahun 2024 dalam zona WIB
start_wib = datetime.datetime(2024, 1, 1, 0, 0, 0,
tzinfo=ZoneInfo("Asia/Jakarta"))
end_wib = datetime.datetime(2024, 12, 31, 23, 0, 0,
tzinfo=ZoneInfo("Asia/Jakarta"))

# Cetak sebagai datetime
print("Start date WIB\t:", start_wib.strftime('%Y-%m-%d %H:%M:%S %Z'))
print("End date WIB\t:", end_wib.strftime('%Y-%m-%d %H:%M:%S %Z'))

# Konversi ke UTC
start_utc = start_wib.astimezone(ZoneInfo("UTC"))
end_utc = end_wib.astimezone(ZoneInfo("UTC"))

# Cetak hasilnya
print("\nStart date UTC\t:", start_utc.strftime('%Y-%m-%d %H:%M:%S
%Z'))
print("End date UTC\t:", end_utc.strftime('%Y-%m-%d %H:%M:%S %Z'))

# Convert ke UNIX timestamp UTC
print("\nStart timestamp\t=", int(start_utc.timestamp()))
print("End timestamp \t=", int(end_utc.timestamp()))
```

```
Start date WIB   : 2024-01-01 00:00:00 WIB
End date WIB     : 2024-12-31 23:00:00 WIB

Start date UTC   : 2023-12-31 17:00:00 UTC
End date UTC     : 2024-12-31 16:00:00 UTC

Start timestamp  = 1704042000
End timestamp    = 1735660800
```

```
import datetime
from zoneinfo import ZoneInfo

start_ts = int(datetime.datetime(2024, 1, 1, 0, 0, 0,
tzinfo=ZoneInfo("Asia/Jakarta")).timestamp())
end_ts = int(datetime.datetime(2024, 12, 31, 23, 0, 0,
```

```
tzinfo=ZoneInfo("Asia/Jakarta")).timestamp())

print(start_ts, end_ts)
```

```
1704042000 1735660800
```

```
import requests
import pandas as pd
from datetime import datetime
import numpy as np

# Define latitude, longitude, start, and end timestamps
lat = -5.4292
lon = 105.2611
start = start_utc  # 1 Januari 2024 00:00:00
end = end_utc      # 31 Desember 2024 23:00:00
api_key = 'f7dc200381fea57025caff3e960b78cf'

# Construct the URL
url = f"http://api.openweathermap.org/data/2.5/air_pollution/history?
lat={lat}&lon={lon}&start={start}&end={end}&appid={api_key}"
print(url)

# Make a GET request
response = requests.get(url)

# Check if request was successful (status code 200)
if response.status_code == 200:
    data = response.json()  # Parse JSON response

    # Prepare lists to store data
    timestamps = []
    components_data = {'PM10': [], 'PM2.5': [], 'CO': [], 'NO2': [],
'SO2': [], 'O3': []}

    # Iterate over each data point in the response
    for record in data['list']:
        # Append original timestamp format (integer timestamp)
        timestamps.append(record['dt'])

        # Extract air quality components
        components = record['components']
        components_data['PM10'].append(components.get('pm10', None))
        components_data['PM2.5'].append(components.get('pm2_5', None))
        components_data['CO'].append(components.get('co', None))
        components_data['NO2'].append(components.get('no2', None))
        components_data['SO2'].append(components.get('so2', None))
        components_data['O3'].append(components.get('o3', None))

    # Create DataFrame from collected data
    df = pd.DataFrame({
```

```
        'Timestamp': timestamps,  # Keep timestamp in original format
        'PM10': components_data['PM10'],
        'PM2.5': components_data['PM2.5'],
        'CO': components_data['CO'],
        'NO2': components_data['NO2'],
        'SO2': components_data['SO2'],
        'O3': components_data['O3']
    })

    # Create DataFrame with all timestamps from start to end with NaN
values for missing data
    all_dates = range(start, end + 1, 3600)  # Ensure range stops at
'end'
    all_dates_df = pd.DataFrame({'Timestamp': all_dates})

    # Merge the two DataFrames, aligning by timestamps
    df = pd.merge(all_dates_df, df, on='Timestamp', how='left')

    # Save DataFrame to CSV file
    df.to_csv('01-airpol_data_2024_balam.csv', index=False)

    print(df)
else:
    print("Error:", response.status_code)
```

```
http://api.openweathermap.org/data/2.5/air_pollution/history?lat=-
5.4292&lon=105.2611&start=2023-12-31 17:00:00+00:00&end=2024-12-31
16:00:00+00:00&appid=f7dc200381fea57025caff3e960b78cf
Error: 400
```

```
import datetime
from zoneinfo import ZoneInfo
import requests
import pandas as pd

# Buat waktu awal dan akhir tahun 2024 dalam zona WIB
start_wib = datetime.datetime(2024, 1, 1, 0, 0, 0,
tzinfo=ZoneInfo("Asia/Jakarta"))
end_wib = datetime.datetime(2024, 12, 31, 23, 0, 0,
tzinfo=ZoneInfo("Asia/Jakarta"))

# Konversi ke UTC
start_utc = start_wib.astimezone(ZoneInfo("UTC"))
end_utc = end_wib.astimezone(ZoneInfo("UTC"))

# Convert ke UNIX timestamp (UTC)
start = int(start_utc.timestamp())  # 1704042000
end = int(end_utc.timestamp())      # 1735660800

# Cek
```

```
print("Start timestamp =", start)
print("End timestamp   =", end)

# API request
lat = -5.4292
lon = 105.2611
api_key = 'f7dc200381fea57025caff3e960b78cf'
url = f"http://api.openweathermap.org/data/2.5/air_pollution/history?
lat={lat}&lon={lon}&start={start}&end={end}&appid={api_key}"
print("URL:", url)

response = requests.get(url)

if response.status_code == 200:
    data = response.json()
    timestamps = []
    components_data = {'PM10': [], 'PM2.5': [], 'CO': [], 'NO2': [],
'SO2': [], 'O3': []}

    for record in data['list']:
        timestamps.append(record['dt'])
        components = record['components']
        components_data['PM10'].append(components.get('pm10', None))
        components_data['PM2.5'].append(components.get('pm2_5', None))
        components_data['CO'].append(components.get('co', None))
        components_data['NO2'].append(components.get('no2', None))
        components_data['SO2'].append(components.get('so2', None))
        components_data['O3'].append(components.get('o3', None))

    df = pd.DataFrame({
        'Timestamp': timestamps,
        'PM10': components_data['PM10'],
        'PM2.5': components_data['PM2.5'],
        'CO': components_data['CO'],
        'NO2': components_data['NO2'],
        'SO2': components_data['SO2'],
        'O3': components_data['O3']
    })

    # Buat DataFrame lengkap dari jam ke jam
    all_dates = range(start, end + 1, 3600)
    all_dates_df = pd.DataFrame({'Timestamp': all_dates})

    df = pd.merge(all_dates_df, df, on='Timestamp', how='left')

    df.to_csv('01-airpol_data_2024.csv', index=False)
    print(df)
else:
    print("Error:", response.status_code, response.text)
```

```
Start timestamp = 1704042000
End timestamp   = 1735660800
URL: http://api.openweathermap.org/data/2.5/air_pollution/history?
lat=-
5.4292&lon=105.2611&start=1704042000&end=1735660800&appid=f7dc200381fe
a57025caff3e960b78cf
       Timestamp    PM10  PM2.5       CO    NO2    SO2    O3
0     1704042000   87.39  70.58  1468.66  21.42   6.97  0.30
1     1704045600   85.55  70.38  1375.20  15.59   4.53  0.12
2     1704049200   77.89  65.52  1214.98   9.94   2.12  0.36
3     1704052800   63.27  54.75  1054.76   7.37   1.33  0.65
4     1704056400   52.20  46.25   947.95   6.25   1.16  0.73
...          ...     ...    ...      ...    ...    ...   ...
8779  1735646400   78.27  55.65  2456.67  47.30  21.46  0.01
8780  1735650000   98.21  69.76  2670.29  45.93  23.37  0.00
8781  1735653600  120.33  86.30  2937.32  46.61  25.51  0.00
8782  1735657200  130.87  93.58  2964.02  42.16  25.27  0.00
8783  1735660800  122.01  85.65  2590.18  32.90  20.98  0.00

[8784 rows x 7 columns]
```

Data Awal

```
import pandas as pd

# Load the uploaded CSV file to inspect its structure
file_path = '/kaggle/working/01-airpol_data_2024.csv'
data = pd.read_csv(file_path, index_col=None)

# Display the first few rows of the dataset
data
```

```
       Timestamp    PM10  PM2.5       CO    NO2    SO2    O3
0     1704042000   87.39  70.58  1468.66  21.42   6.97  0.30
1     1704045600   85.55  70.38  1375.20  15.59   4.53  0.12
2     1704049200   77.89  65.52  1214.98   9.94   2.12  0.36
3     1704052800   63.27  54.75  1054.76   7.37   1.33  0.65
4     1704056400   52.20  46.25   947.95   6.25   1.16  0.73
...          ...     ...    ...      ...    ...    ...   ...
8779  1735646400   78.27  55.65  2456.67  47.30  21.46  0.01
8780  1735650000   98.21  69.76  2670.29  45.93  23.37  0.00
8781  1735653600  120.33  86.30  2937.32  46.61  25.51  0.00
8782  1735657200  130.87  93.58  2964.02  42.16  25.27  0.00
8783  1735660800  122.01  85.65  2590.18  32.90  20.98  0.00

[8784 rows x 7 columns]
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8784 entries, 0 to 8783
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Timestamp  8784 non-null   int64  
 1   PM10       8664 non-null   float64
 2   PM2.5      8664 non-null   float64
 3   CO         8664 non-null   float64
 4   NO2        8664 non-null   float64
 5   SO2        8664 non-null   float64
 6   O3         8664 non-null   float64
dtypes: float64(6), int64(1)
memory usage: 480.5 KB
```

```
data.describe()
```

```
          Timestamp          PM10         PM2.5            CO  \
count  8.784000e+03  8664.000000  8664.000000  8664.000000   
mean   1.719851e+09    55.885444    45.976274  1179.376776   
std    9.129120e+06   115.962258    36.730055   699.671731   
min    1.704042e+09 -9999.000000     0.590000   233.650000   
25%    1.711947e+09    22.885000    16.440000   594.140000   
50%    1.719851e+09    47.835000    37.600000  1014.710000   
75%    1.727756e+09    82.332500    66.665000  1628.880000   
max    1.735661e+09   299.880000   277.900000  3952.030000   

               NO2  \
count  8664.000000   
mean     13.472380   
std     152.788767   
min   -9999.000000   
25%       6.000000   
50%      11.140000   
75%      21.420000   
max     100.080000   

               SO2           O3  
count  8664.000000  8664.000000  
mean      5.773337    25.011085  
std       4.922607    30.213595  
min       0.400000     0.000000  
25%       2.180000     1.005000  
50%       4.410000    13.590000  
75%       7.510000    37.910000  
max      32.900000   194.550000
```

```
print(data.columns)
```

```
Index(['Timestamp', 'PM10', 'PM2.5', 'CO', 'NO2', 'SO2', 'O3'],
dtype='object')
```

```python
import pandas as pd
from zoneinfo import ZoneInfo

df = data

# Konversi kolom 'Timestamp' menjadi datetime UTC
df['Waktu'] = pd.to_datetime(df['Timestamp'], unit='s', utc=True)

# Konversi ke WIB (Asia/Jakarta)
df['Waktu'] = df['Waktu'].dt.tz_convert('Asia/Jakarta')

# Hapus kolom Timestamp kalau tidak dipakai lagi
df = df.drop(columns=['Timestamp'])

# Pastikan semua kolom selain 'Waktu' adalah float
for col in df.columns:
    if col != 'Waktu':
        df[col] = pd.to_numeric(df[col], errors='coerce')

# Pindahkan 'Waktu' ke depan
cols = ['Waktu'] + [col for col in df.columns if col != 'Waktu']
df = df[cols]

# Cek struktur
df.info()

#df['Waktu'] = df['Waktu'].dt.tz_localize(None)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8784 entries, 0 to 8783
Data columns (total 7 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Waktu   8784 non-null   datetime64[ns, Asia/Jakarta]
 1   PM10    8664 non-null   float64
 2   PM2.5   8664 non-null   float64
 3   CO      8664 non-null   float64
 4   NO2     8664 non-null   float64
 5   SO2     8664 non-null   float64
 6   O3      8664 non-null   float64
dtypes: datetime64[ns, Asia/Jakarta](1), float64(6)
memory usage: 480.5 KB
```

```python
df.head()
```

	Waktu	PM10	PM2.5	CO	NO2	SO2	O3
0	2024-01-01 00:00:00+07:00	87.39	70.58	1468.66	21.42	6.97	0.30
1	2024-01-01 01:00:00+07:00	85.55	70.38	1375.20	15.59	4.53	0.12
2	2024-01-01 02:00:00+07:00	77.89	65.52	1214.98	9.94	2.12	0.36
3	2024-01-01 03:00:00+07:00	63.27	54.75	1054.76	7.37	1.33	0.65
4	2024-01-01 04:00:00+07:00	52.20	46.25	947.95	6.25	1.16	0.73

```
df['Waktu'] = df['Waktu'].dt.tz_localize(None)

# Konversi kolom Waktu menjadi datetime jika belum
df['Waktu'] = pd.to_datetime(df['Waktu'])


df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8784 entries, 0 to 8783
Data columns (total 7 columns):
 #   Column  Non-Null Count  Dtype         
---  ------  --------------  -----         
 0   Waktu   8784 non-null   datetime64[ns]
 1   PM10    8664 non-null   float64       
 2   PM2.5   8664 non-null   float64       
 3   CO      8664 non-null   float64       
 4   NO2     8664 non-null   float64       
 5   SO2     8664 non-null   float64       
 6   O3      8664 non-null   float64       
dtypes: datetime64[ns](1), float64(6)
memory usage: 480.5 KB
```

```
df
```

	Waktu	PM10	PM2.5	CO	NO2	SO2	O3
0	2024-01-01 00:00:00	87.39	70.58	1468.66	21.42	6.97	0.30
1	2024-01-01 01:00:00	85.55	70.38	1375.20	15.59	4.53	0.12
2	2024-01-01 02:00:00	77.89	65.52	1214.98	9.94	2.12	0.36
3	2024-01-01 03:00:00	63.27	54.75	1054.76	7.37	1.33	0.65
4	2024-01-01 04:00:00	52.20	46.25	947.95	6.25	1.16	0.73
...	...	...	...	...	...	...	...
8779	2024-12-31 19:00:00	78.27	55.65	2456.67	47.30	21.46	0.01
8780	2024-12-31 20:00:00	98.21	69.76	2670.29	45.93	23.37	0.00
8781	2024-12-31 21:00:00	120.33	86.30	2937.32	46.61	25.51	0.00
8782	2024-12-31 22:00:00	130.87	93.58	2964.02	42.16	25.27	0.00
8783	2024-12-31 23:00:00	122.01	85.65	2590.18	32.90	20.98	0.00

```
[8784 rows x 7 columns]
```

```
df.to_csv('02-data_date_2024.csv', index=False)

df.columns
```

```
Index(['Waktu', 'PM10', 'PM2.5', 'CO', 'NO2', 'SO2', 'O3'],
dtype='object')
```

```
# Load the uploaded CSV file to inspect its structure
import pandas as pd
file_path = '/content/02-data_date_2024.csv'
```

```
df = pd.read_csv(file_path, index_col=None)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8784 entries, 0 to 8783
Data columns (total 7 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Waktu   8784 non-null   object 
 1   PM10    8664 non-null   float64
 2   PM2.5   8664 non-null   float64
 3   CO      8664 non-null   float64
 4   NO2     8664 non-null   float64
 5   SO2     8664 non-null   float64
 6   O3      8664 non-null   float64
dtypes: float64(6), object(1)
memory usage: 480.5+ KB
```

```
# Konversi kolom Waktu menjadi datetime jika belum
df['Waktu'] = pd.to_datetime(df['Waktu'])
df.info()
```

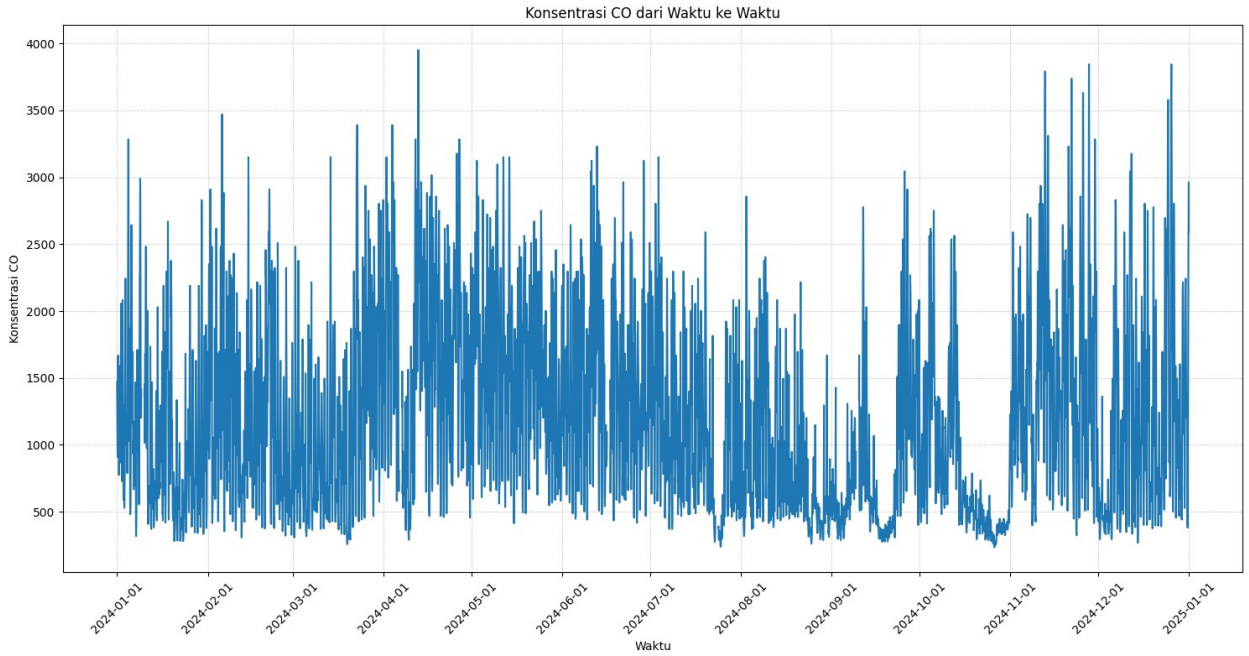
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8784 entries, 0 to 8783
Data columns (total 7 columns):
 #   Column  Non-Null Count  Dtype         
---  ------  --------------  -----         
 0   Waktu   8784 non-null   datetime64[ns]
 1   PM10    8664 non-null   float64       
 2   PM2.5   8664 non-null   float64       
 3   CO      8664 non-null   float64       
 4   NO2     8664 non-null   float64       
 5   SO2     8664 non-null   float64       
 6   O3      8664 non-null   float64       
dtypes: datetime64[ns](1), float64(6)
memory usage: 480.5 KB
```

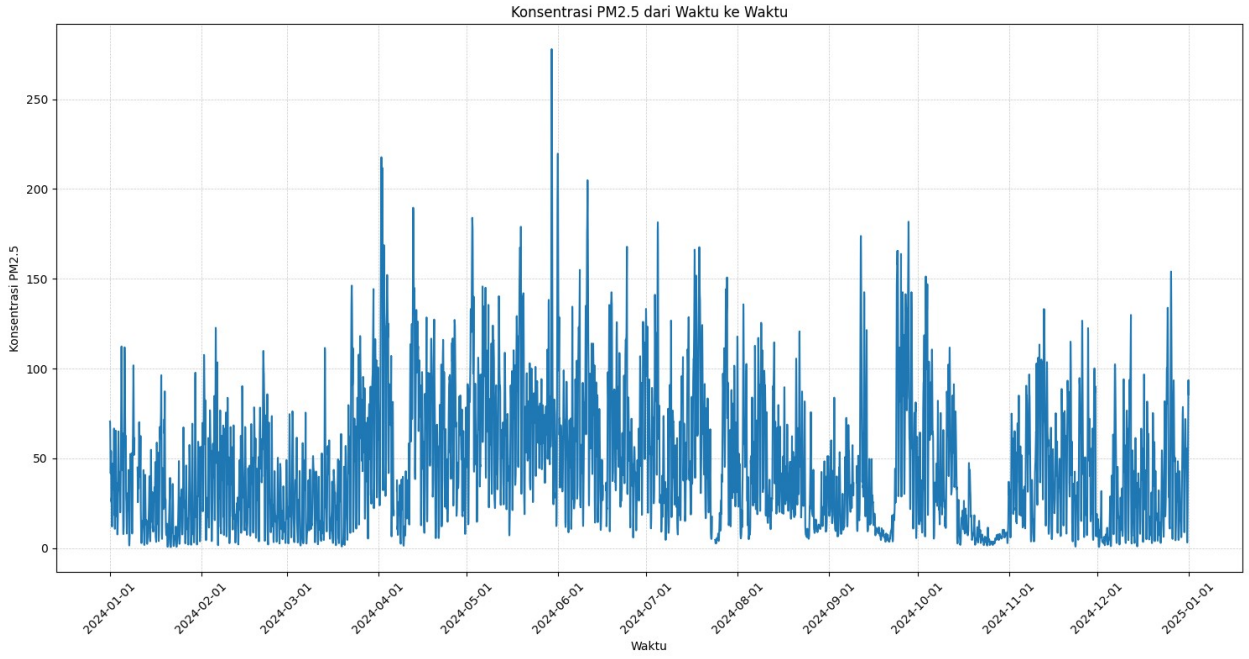
```
# Cek missing value di seluruh dataframe
ada_missing_value = df.isnull().any().any()
print(f"Apakah ada missing value di dataframe: {ada_missing_value}")

# Cek missing value per kolom
missing_value_per_kolom = df.isnull().any()
print("\nMissing value per kolom:")
print(missing_value_per_kolom)

# Cek jumlah missing value per kolom
jumlah_missing_value_per_kolom = df.isnull().sum()
print("\nJumlah missing value per kolom:")
print(jumlah_missing_value_per_kolom)
```

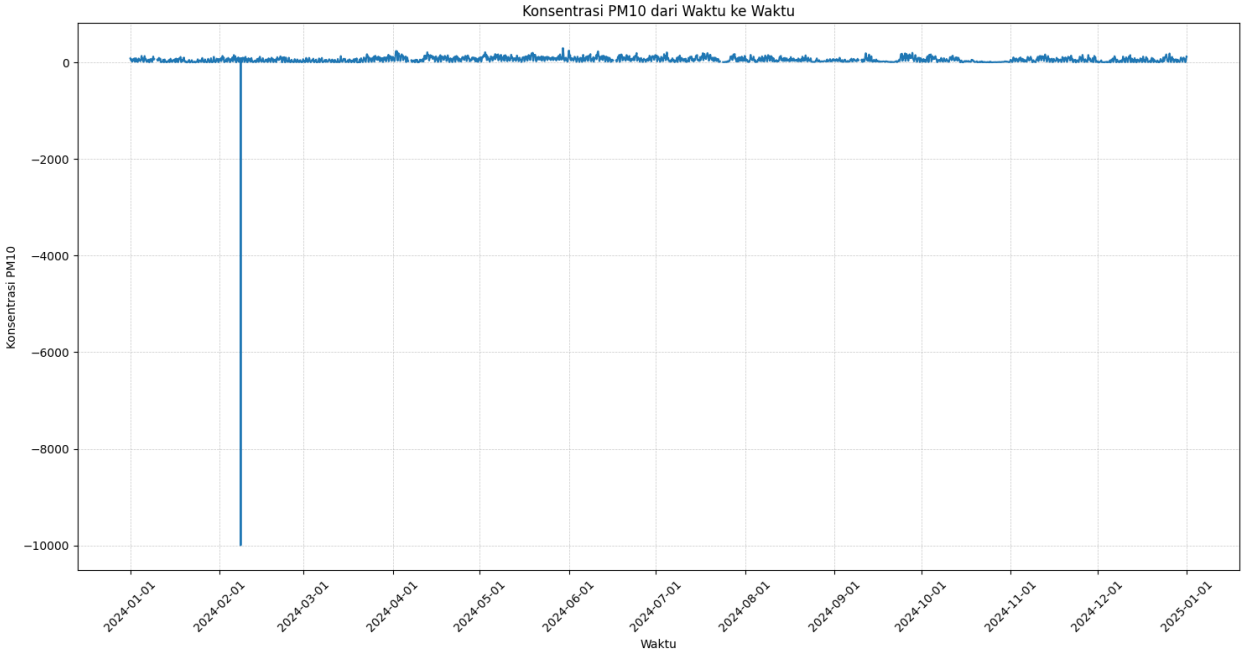
```
Apakah ada missing value di dataframe: True

Missing value per kolom:
Waktu    False
PM10      True
PM2.5     True
CO        True
NO2       True
SO2       True
O3        True
dtype: bool

Jumlah missing value per kolom:
Waktu      0
PM10     120
PM2.5    120
CO       120
NO2      120
SO2      120
O3       120
dtype: int64
```

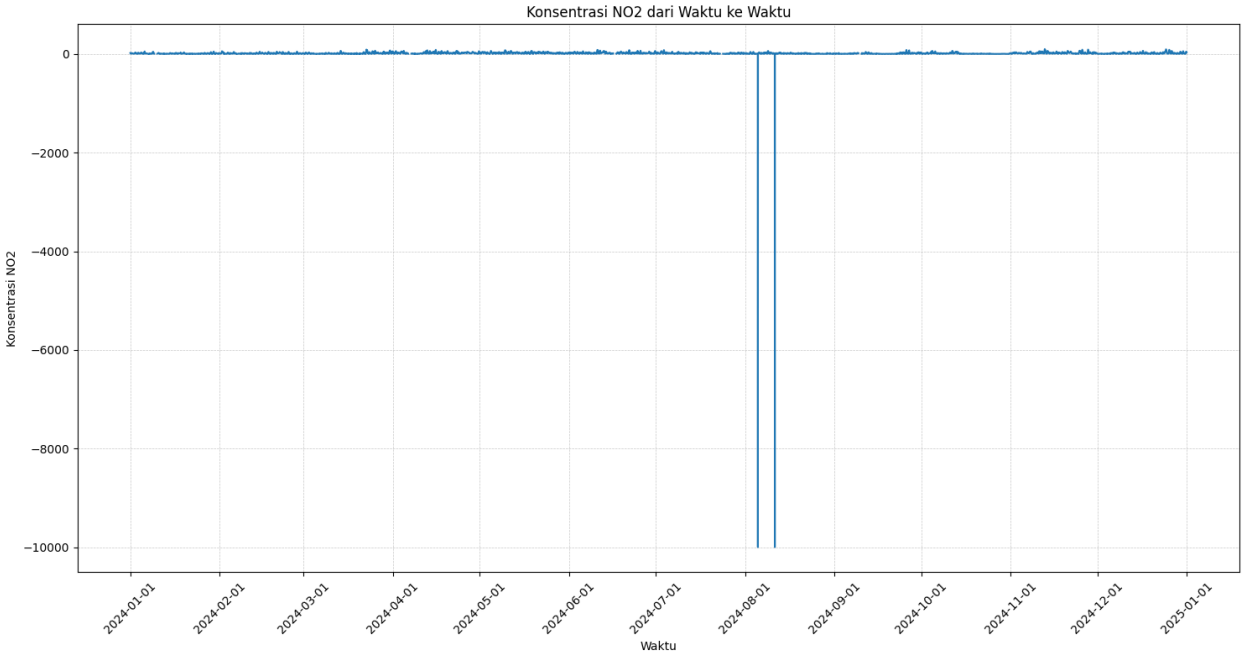
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.dates as mdates

# Tentukan semua kolom pencemar udara (selain waktu dan ISPU jika ada)
columns_to_plot = ['CO', 'PM2.5', 'PM10', 'NO2', 'SO2', 'O3']

# Loop untuk plot setiap kolom
for col in columns_to_plot:
    plt.figure(figsize=(15, 8))
    plt.plot(df['Waktu'], df[col], label=col)

    # Format sumbu x untuk menampilkan tahun-bulan-tanggal
    plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-
%d'))
    plt.gca().xaxis.set_major_locator(mdates.MonthLocator())
    plt.xticks(rotation=45)

    # Set label dan judul
    plt.xlabel('Waktu')
    plt.ylabel(f'Konsentrasi {col}')
    plt.title(f'Konsentrasi {col} dari Waktu ke Waktu')

    # Tambahkan garis grid XY
    plt.grid(True, which='both', linestyle='--', linewidth=0.5,
alpha=0.7)
    #plt.legend()
```

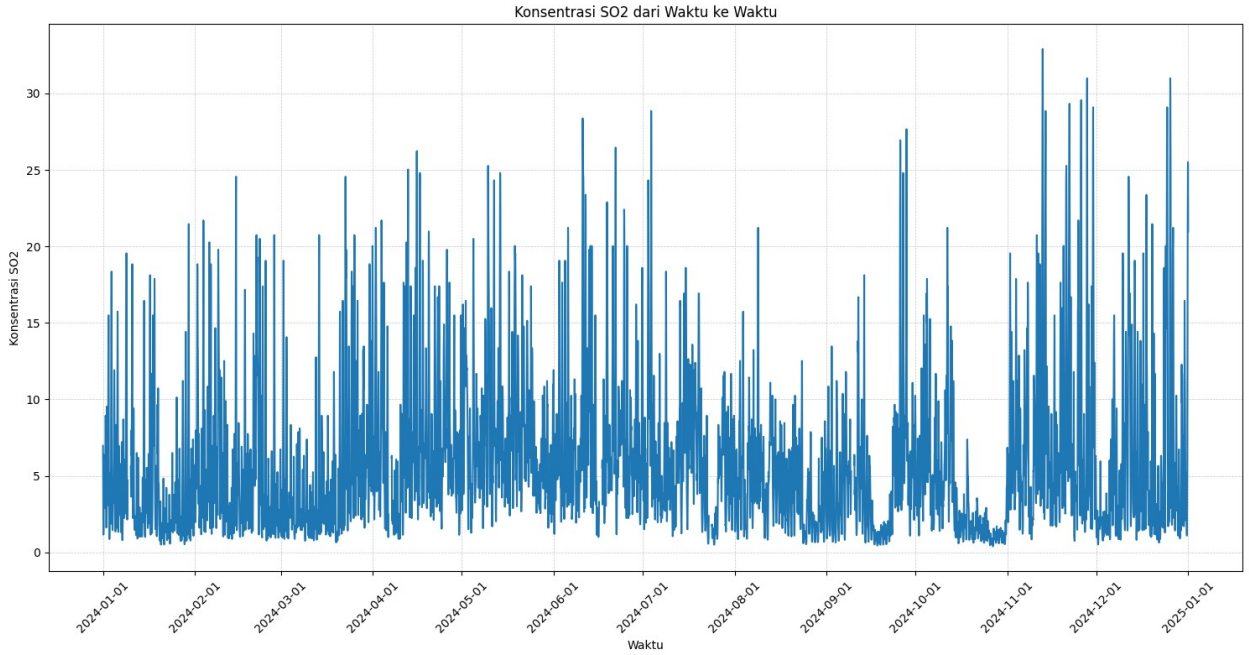
```
    plt.tight_layout()
    plt.show()
```

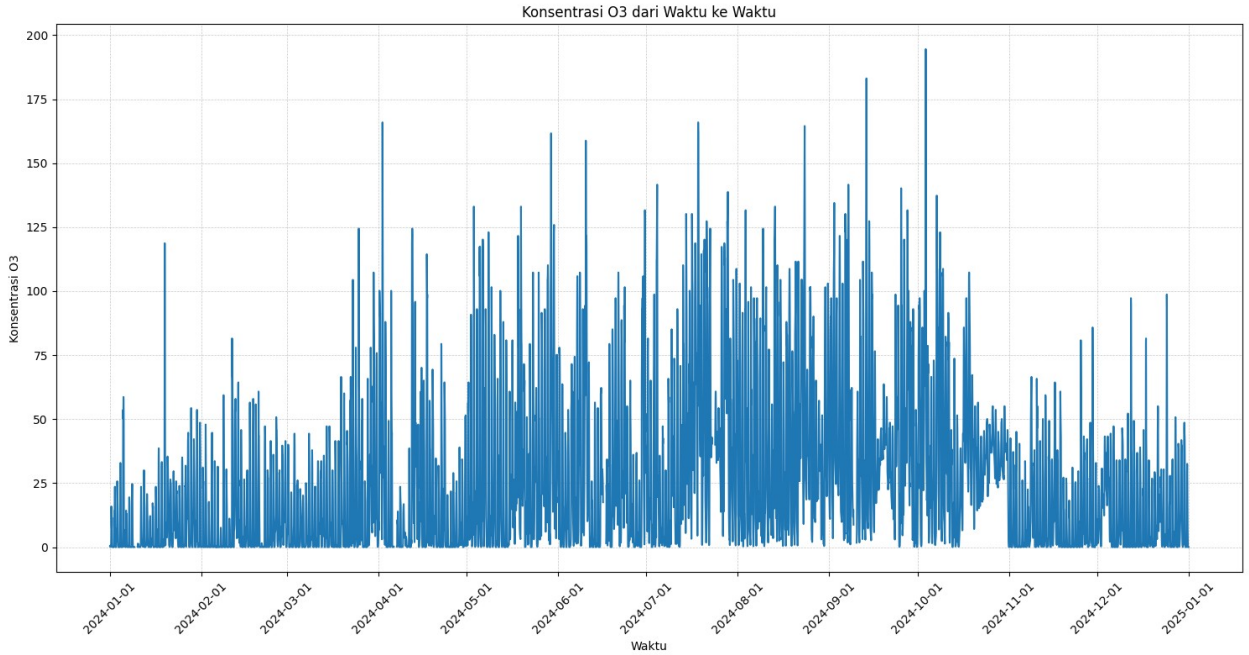





Konsentrasi PM10 dari Waktu ke Waktu
Konsentrasi PM10
0
−2000
−4000
−6000
−8000
−10000
2024-01-01
2024-02-01
2024-03-01
2024-04-01
2024-05-01
2024-06-01
2024-07-01
2024-08-01
2024-09-01
2024-10-01
2024-11-01
2024-12-01
2025-01-01
Waktu

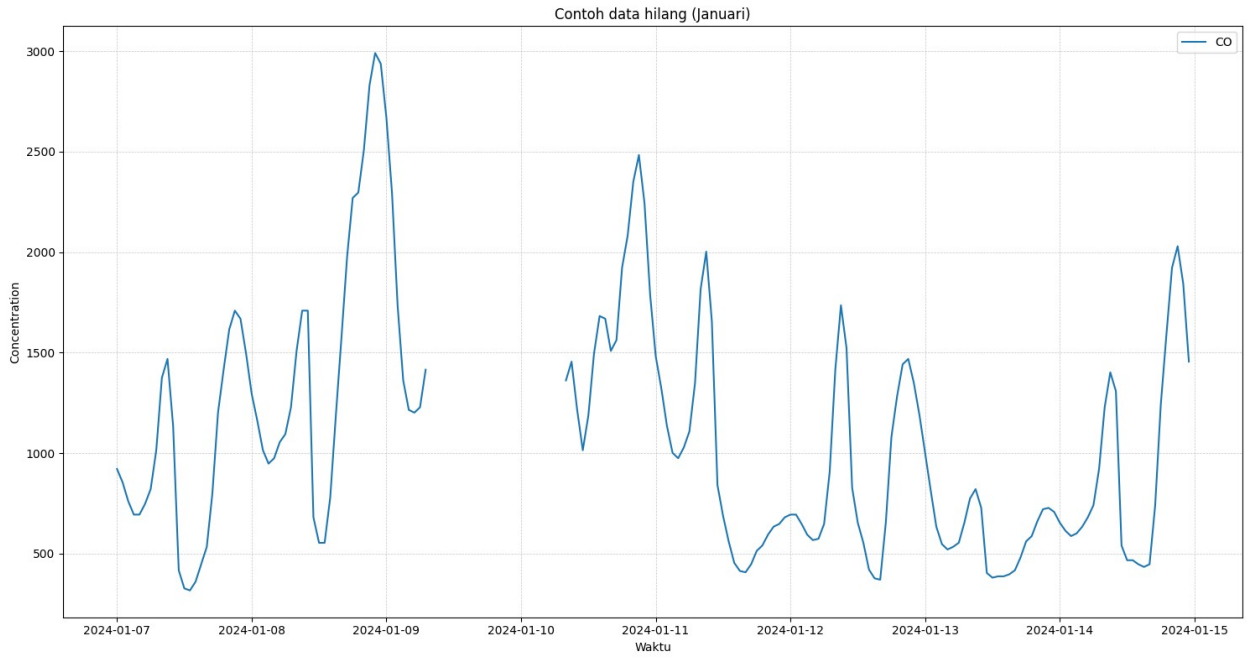

Konsentrasi NO2 dari Waktu ke Waktu
Konsentrasi NO2
0
−2000
−4000
−6000
−8000
−10000
2024-01-01
2024-02-01
2024-03-01
2024-04-01
2024-05-01
2024-06-01
2024-07-01
2024-08-01
2024-09-01
2024-10-01
2024-11-01
2024-12-01
2025-01-01
Waktu



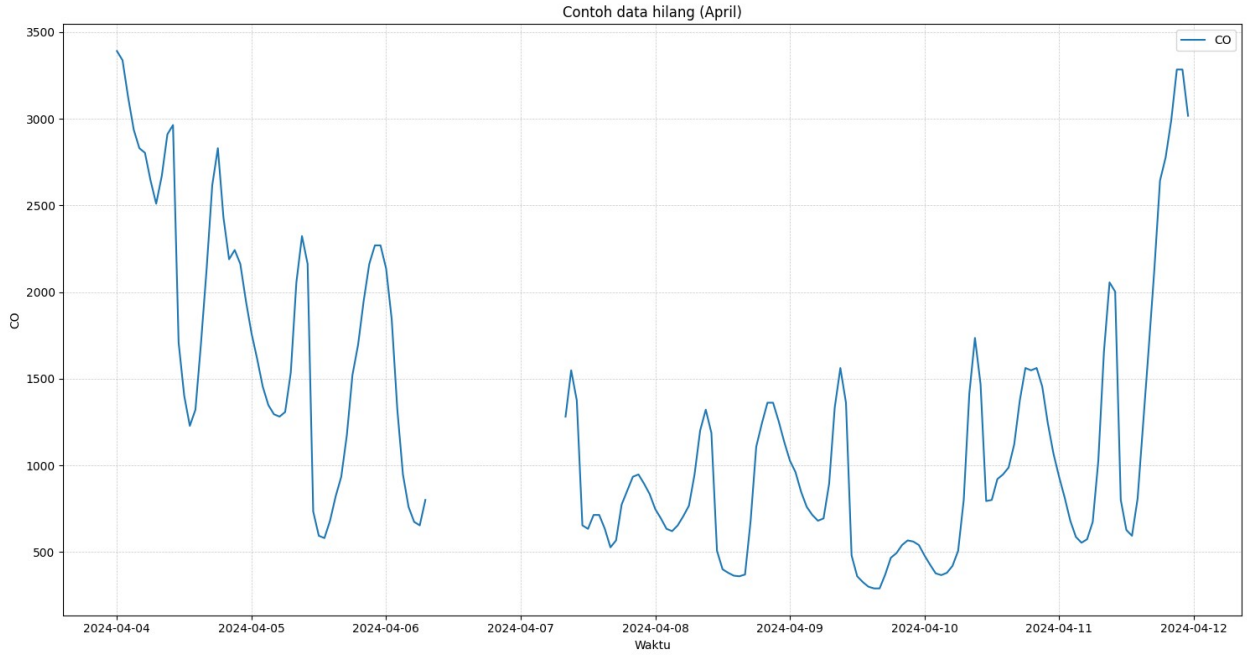



```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Tentukan rentang waktu yang diinginkan
start_time = '2024-01-07 00:00:00'
end_time = '2024-01-14 23:00:00'

# Filter data berdasarkan rentang waktu
```

```
df_filtered = df[(df['Waktu'] >= start_time) & (df['Waktu'] <=
end_time)]

# Tentukan kolom yang ingin diplot
columns_to_plot = ['CO']

# Plot data
plt.figure(figsize=(15, 8))

for col in columns_to_plot:
    # Plot garis untuk kolom yang ada datanya
    plt.plot(df_filtered['Waktu'], df_filtered[col], label=col)

    # Highlight missing values dengan warna lain
    #plt.fill_between(df_filtered['Waktu'], df_filtered[col],
where=df_filtered[col].isnull(), color='red', alpha=0.3,
label=f'Missing {col}' if col == columns_to_plot[0] else "")

# Set labels dan title
plt.xlabel('Waktu')
plt.ylabel('Concentration')
plt.title('Contoh data hilang (Januari)')
plt.legend(loc='upper right')

# Tambahkan garis grid XY
plt.grid(True, which='both', linestyle='--', linewidth=0.5, alpha=0.7)

# Tampilkan plot
plt.tight_layout()
plt.show()
```



```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Konversi kolom Waktu menjadi datetime jika belum
df['Waktu'] = pd.to_datetime(df['Waktu'])

# Tentukan rentang waktu yang diinginkan
start_time = '2024-04-04 00:00:00'
end_time = '2024-04-11 23:00:00'

# Filter data berdasarkan rentang waktu
df_filtered = df[(df['Waktu'] >= start_time) & (df['Waktu'] <=
end_time)]

# Tentukan kolom yang ingin diplot
columns_to_plot = ['CO']

# Plot data
plt.figure(figsize=(15, 8))

for col in columns_to_plot:
    # Plot garis untuk kolom yang ada datanya
    plt.plot(df_filtered['Waktu'], df_filtered[col], label=col)

# Set labels dan title
plt.xlabel('Waktu')
plt.ylabel('CO')
plt.title('Contoh data hilang (April)')
plt.legend(loc='upper right')

# Tambahkan garis grid XY
plt.grid(True, which='both', linestyle='--', linewidth=0.5, alpha=0.7)

# Tampilkan plot
plt.tight_layout()
plt.show()
```




```
# Melihat baris dengan nilai NaN pada kolom manapun
df_missing_before = df[df.isna().any(axis=1)]
df_missing_before
```


```
{"summary":"{\n  \"name\": \"df_missing_before\",\n  \"rows\": 120,\n
\"fields\": [\n    {\n      \"column\": \"Waktu\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"2024-01-09 08:00:00\",\n        \"max\": \"2024-09-10 07:00:00\",\n
\"num_unique_values\": 120,\n        \"samples\": [\n          \"2024-
04-07 04:00:00\",\n          \"2024-04-07 07:00:00\",\n
\"2024-01-09 12:00:00\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"PM10\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": null,\n        \"min\": null,\n
\"max\": null,\n        \"num_unique_values\": 0,\n
\"samples\": [],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PM2.5\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": null,\n        \"min\": null,\n        \"max\": null,\n
\"num_unique_values\": 0,\n        \"samples\": [],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"CO\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": null,\n        \"min\":
null,\n        \"max\": null,\n        \"num_unique_values\": 0,\n
\"samples\": [],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"NO2\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": null,\n        \"min\": null,\n        \"max\": null,\n
\"num_unique_values\": 0,\n        \"samples\": [],\n
```

\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"SO2\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": null,\n   \"min\":
null,\n   \"max\": null,\n   \"num_unique_values\": 0,\n
\"samples\": [],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"O3\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": null,\n   \"min\": null,\n   \"max\": null,\n
\"num_unique_values\": 0,\n   \"samples\": [],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\
n}","type":"dataframe","variable_name":"df_missing_before"}

```
import pandas as pd

# Ambil hanya tanggal (tanpa waktu)
df['Date'] = df['Waktu'].dt.date

# Filter baris yang memiliki NaN di kolom tertentu, misalnya 'CO'
df_nan = df[df['CO'].isna()]

# Mengelompokkan berdasarkan tanggal dan memilih baris yang memiliki
NaN
df_nan_daily = df_nan.groupby('Date').first().reset_index()

# Menampilkan hasil
print(df_nan_daily)
```

	Date	Waktu	PM10	PM2.5	CO	NO2	SO2	O3
0	2024-01-09	2024-01-09 08:00:00	NaN	NaN	NaN	NaN	NaN	NaN
1	2024-01-10	2024-01-10 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN
2	2024-04-06	2024-04-06 08:00:00	NaN	NaN	NaN	NaN	NaN	NaN
3	2024-04-07	2024-04-07 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN
4	2024-06-16	2024-06-16 08:00:00	NaN	NaN	NaN	NaN	NaN	NaN
5	2024-06-17	2024-06-17 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN
6	2024-07-23	2024-07-23 08:00:00	NaN	NaN	NaN	NaN	NaN	NaN
7	2024-07-24	2024-07-24 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN
8	2024-09-09	2024-09-09 08:00:00	NaN	NaN	NaN	NaN	NaN	NaN
9	2024-09-10	2024-09-10 00:00:00	NaN	NaN	NaN	NaN	NaN	NaN

```
import pandas as pd

# Ambil hanya jam (tanpa tanggal)
df['Hour'] = df['Waktu'].dt.hour

# Filter baris yang memiliki NaN di kolom tertentu, misalnya 'CO'
df_nan = df[df['CO'].isna()]

# Mengelompokkan berdasarkan jam dan menghitung total missing value di
setiap jam
df_nan_hourly = df_nan.groupby('Hour').size().reset_index(name='Total
```

```
Missing')

# Menampilkan hasil
print(df_nan_hourly)
```

```
    Hour  Total Missing
0      0              5
1      1              5
2      2              5
3      3              5
4      4              5
5      5              5
6      6              5
7      7              5
8      8              5
9      9              5
10    10              5
11    11              5
12    12              5
13    13              5
14    14              5
15    15              5
16    16              5
17    17              5
18    18              5
19    19              5
20    20              5
21    21              5
22    22              5
23    23              5
```

```
df.columns
```

```
Index(['Waktu', 'PM10', 'PM2.5', 'CO', 'NO2', 'SO2', 'O3', 'Date',
'Hour'], dtype='object')
```

```
df.head()
```

```
                Waktu   PM10  PM2.5       CO    NO2   SO2    O3
Date  \
0 2024-01-01 00:00:00  87.39  70.58  1468.66  21.42  6.97  0.30  2024-
01-01
1 2024-01-01 01:00:00  85.55  70.38  1375.20  15.59  4.53  0.12  2024-
01-01
2 2024-01-01 02:00:00  77.89  65.52  1214.98   9.94  2.12  0.36  2024-
01-01
3 2024-01-01 03:00:00  63.27  54.75  1054.76   7.37  1.33  0.65  2024-
01-01
4 2024-01-01 04:00:00  52.20  46.25   947.95   6.25  1.16  0.73  2024-
01-01
```

```
   Hour
0     0
1     1
2     2
3     3
4     4
```

```
df = df.drop(columns=['Date', 'Hour'])
```

```
df.columns
```

```
Index(['Waktu', 'PM10', 'PM2.5', 'CO', 'NO2', 'SO2', 'O3'],
dtype='object')
```

```
import pandas as pd

# Fungsi untuk menghitung persentase missing value
def calculate_missing_percentage(df):
    missing_percentage = (df.isnull().sum() / len(df)) * 100
    return missing_percentage

# Contoh DataFrame
# df = pd.read_csv('your_dataset.csv')

# Hitung persentase missing value
missing_percentage = calculate_missing_percentage(df)

# Tampilkan hasil
print("Persentase Missing Value per Kolom:")
print(missing_percentage)
```

```
Persentase Missing Value per Kolom:
Waktu    0.00000
PM10     1.36612
PM2.5    1.36612
CO       1.36612
NO2      1.36612
SO2      1.36612
O3       1.36612
dtype: float64
```

```
# Remove rows with any missing values
#df = df.dropna()

# Display the shape of the DataFrame after removing missing values
df.shape
```

```
(8784, 7)
```

```
df.describe()
```

{"summary":"{\n \"name\": \"df\",\n \"rows\": 8,\n \"fields\": [\n{\n   \"column\": \"Waktu\",\n   \"properties\": {\n\"dtype\": \"date\",\n   \"min\": \"1970-01-01\n00:00:00.000008784\",\n   \"max\": \"2024-12-31 23:00:00\",\n\"num_unique_values\": 6,\n   \"samples\": [\n\"8784\",\n   \"2024-07-01 23:30:00\",\n   \"2024-12-31\n23:00:00\"\n   ],\n   \"semantic_type\": \"\",\n\"description\": \"\"\n   }\n   },\n   {\n   \"column\":\"PM10\",\n   \"properties\": {\n   \"dtype\": \"number\",\n\"std\": 5001.3988682233785,\n   \"min\": -9999.0,\n\"max\": 8664.0,\n   \"num_unique_values\": 8,\n\"samples\": [\n   55.88544436749769,\n   82.3325,\n8664.0\n   ],\n   \"semantic_type\": \"\",\n\"description\": \"\"\n   }\n   },\n   {\n   \"column\":\"PM2.5\",\n   \"properties\": {\n   \"dtype\": \"number\",\n\"std\": 3040.107541298734,\n   \"min\": 0.59,\n   \"max\":8664.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n45.976274238227155,\n   66.66499999999999,\n   8664.0\n],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n}\n   },\n   {\n   \"column\": \"CO\",\n   \"properties\": {\n\"dtype\": \"number\",\n   \"std\": 2836.123723388778,\n\"min\": 233.65,\n   \"max\": 8664.0,\n\"num_unique_values\": 8,\n   \"samples\": [\n1179.3767763157894,\n   1628.88,\n   8664.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n}\n   },\n   {\n   \"column\": \"NO2\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 4999.234511103058,\n\"min\": -9999.0,\n   \"max\": 8664.0,\n\"num_unique_values\": 8,\n   \"samples\": [\n13.472379963065558,\n   21.42,\n   8664.0\n   ],\n\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"SO2\",\n   \"properties\": {\n\"dtype\": \"number\",\n   \"std\": 3060.269526598038,\n\"min\": 0.4,\n   \"max\": 8664.0,\n\"num_unique_values\": 8,\n   \"samples\": [\n5.773336795937211,\n   7.51,\n   8664.0\n   ],\n\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"O3\",\n   \"properties\": {\n\"dtype\": \"number\",\n   \"std\": 3048.5742703050555,\n\"min\": 0.0,\n   \"max\": 8664.0,\n\"num_unique_values\": 8,\n   \"samples\": [\n25.011084949215142,\n   37.91,\n   8664.0\n   ],\n\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
# Menampilkan baris yang terdeteksi duplikat (seluruh kolom)
df_duplicated = df[df.duplicated()]
print(df_duplicated)
```

```
Empty DataFrame
Columns: [Waktu, PM10, PM2.5, CO, NO2, SO2, O3]
Index: []

# Cek nilai negatif di semua kolom numerik
negative_values = (df.select_dtypes(include=['number']) < 0).sum()

# Tampilkan hasil
print("Jumlah nilai negatif per kolom:")
print(negative_values)

# Jika ingin melihat baris dengan nilai negatif
negative_rows = df[(df.select_dtypes(include=['number']) <
0).any(axis=1)]
print("Baris dengan nilai negatif:")
negative_rows

Jumlah nilai negatif per kolom:
PM10     1
PM2.5    0
CO       0
NO2      2
SO2      0
O3       0
dtype: int64
Baris dengan nilai negatif:

{"summary":"{\n  \"name\": \"negative_rows\",\n  \"rows\": 3,\n
\"fields\": [\n    {\n      \"column\": \"Waktu\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"2024-02-08 06:00:00\",\n        \"max\": \"2024-08-11 07:00:00\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n          \"2024-
02-08 06:00:00\",\n          \"2024-08-05 10:00:00\",\n
\"2024-08-11 07:00:00\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"PM10\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 5798.641449273557,\n        \"min\": -
9999.0,\n        \"max\": 62.16,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          -9999.0,\n          62.16,\n          26.83\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PM2.5\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 16.273617708835776,\n        \"min\": 17.05,\n
\"max\": 49.02,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          38.32,\n          49.02,\n          17.05\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"CO\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 358.02749410252466,\n
\"min\": 607.49,\n        \"max\": 1321.79,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          921.25,\n
```

1321.79,\n   607.49\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"NO2\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 5774.21283272736,\n   \"min\": -9999.0,\n   \"max\": 2.23,\n   \"num_unique_values\": 2,\n   \"samples\": [\n   -9999.0,\n   2.23\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"SO2\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 2.400062499186219,\n   \"min\": 1.97,\n   \"max\": 6.2,\n   \"num_unique_values\": 3,\n   \"samples\": [\n   1.97,\n   6.2\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"O3\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 26.926339025818816,\n   \"min\": 0.0,\n   \"max\": 47.92,\n   \"num_unique_values\": 3,\n   \"samples\": [\n   0.0,\n   47.92\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"negative_rows"}

```
import numpy as np

# Mengganti nilai negatif dengan NaN untuk semua kolom numerik
df[df.select_dtypes(include=['number']) < 0] = np.nan

# Hitung jumlah nilai negatif yang telah diubah menjadi NaN
negative_values = (df.select_dtypes(include=['number']) < 0).sum()

# Tampilkan hasil
print("Jumlah nilai negatif per kolom setelah diubah menjadi NaN:")
print(negative_values)

# Jika ingin melihat baris yang sebelumnya memiliki nilai negatif
negative_rows = df[df.isna().any(axis=1)]
print("Baris yang memiliki nilai NaN:")
negative_rows
```

```
Jumlah nilai negatif per kolom setelah diubah menjadi NaN:
PM10     0
PM2.5    0
CO       0
NO2      0
SO2      0
O3       0
dtype: int64
Baris yang memiliki nilai NaN:
```

{"summary":"{\n \"name\": \"negative_rows\",\n \"rows\": 123,\n \"fields\": [\n   {\n   \"column\": \"Waktu\",\n   \"properties\": {\n   \"dtype\": \"date\",\n   \"min\":

\"2024-01-09 08:00:00\",\n   \"max\": \"2024-09-10 07:00:00\",\n
\"num_unique_values\": 123,\n   \"samples\": [\n   \"2024-
01-10 02:00:00\",\n   \"2024-04-07 04:00:00\",\n
\"2024-04-07 06:00:00\"\n   ],\n   \"semantic_type\": \"\",\
n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"PM10\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 24.98208257932072,\n   \"min\":
26.83,\n   \"max\": 62.16,\n   \"num_unique_values\": 2,\n
\"samples\": [\n   26.83,\n   62.16\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"PM2.5\",\n   \"properties\": {\
n   \"dtype\": \"number\",\n   \"std\": 16.273617708835776,\
n   \"min\": 17.05,\n   \"max\": 49.02,\n
\"num_unique_values\": 3,\n   \"samples\": [\n   38.32,\n
49.02\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"CO\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 358.02749410252466,\n   \"min\": 607.49,\n
\"max\": 1321.79,\n   \"num_unique_values\": 3,\n
\"samples\": [\n   921.25,\n   1321.79\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"NO2\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": null,\n   \"min\":
2.23,\n   \"max\": 2.23,\n   \"num_unique_values\": 1,\n
\"samples\": [\n   2.23\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"SO2\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 2.400062499186219,\n   \"min\":
1.97,\n   \"max\": 6.2,\n   \"num_unique_values\": 3,\n
\"samples\": [\n   1.97\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"O3\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 26.926339025818816,\n   \"min\":
0.0,\n   \"max\": 47.92,\n   \"num_unique_values\": 3,\n
\"samples\": [\n   0.0\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"negative_rows"}

```
# Cek jumlah missing value per kolom
jumlah_missing_value_per_kolom = df.isnull().sum()
print("\nJumlah missing value per kolom:")
print(jumlah_missing_value_per_kolom)
```

```
Jumlah missing value per kolom:
Waktu      0
PM10     121
PM2.5    120
CO       120
NO2      122
```

```
SO2      120
O3       120
dtype: int64
```

```
import pandas as pd

# Fungsi untuk menghitung persentase missing value
def calculate_missing_percentage(df):
    missing_percentage = (df.isnull().sum() / len(df)) * 100
    return missing_percentage

# Contoh DataFrame
# df = pd.read_csv('your_dataset.csv')

# Hitung persentase missing value
missing_percentage = calculate_missing_percentage(df)

# Tampilkan hasil
print("Persentase Missing and Negative Value per Kolom:")
print(missing_percentage)
```

```
Persentase Missing and Negative Value per Kolom:
Waktu    0.000000
PM10     1.377505
PM2.5    1.366120
CO       1.366120
NO2      1.388889
SO2      1.366120
O3       1.366120
dtype: float64
```

```
print(df.columns)
df.head()
```

```
Index(['Waktu', 'PM10', 'PM2.5', 'CO', 'NO2', 'SO2', 'O3'],
dtype='object')
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 8784,\n  \"fields\":
[\n    {\n      \"column\": \"Waktu\",\n      \"properties\": {\n
\"dtype\": \"date\",\n        \"min\": \"2024-01-01 00:00:00\",\n
\"max\": \"2024-12-31 23:00:00\",\n        \"num_unique_values\":
8784,\n        \"samples\": [\n          \"2024-09-23 01:00:00\",\n
\"2024-03-03 00:00:00\",\n          \"2024-09-29 11:00:00\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PM10\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 42.13902917922888,\n        \"min\": 1.02,\n        \"max\":
299.88,\n        \"num_unique_values\": 6269,\n        \"samples\": [\
n          106.74,\n          41.9,\n          6.94\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"PM2.5\",\n      \"properties\": {\
```

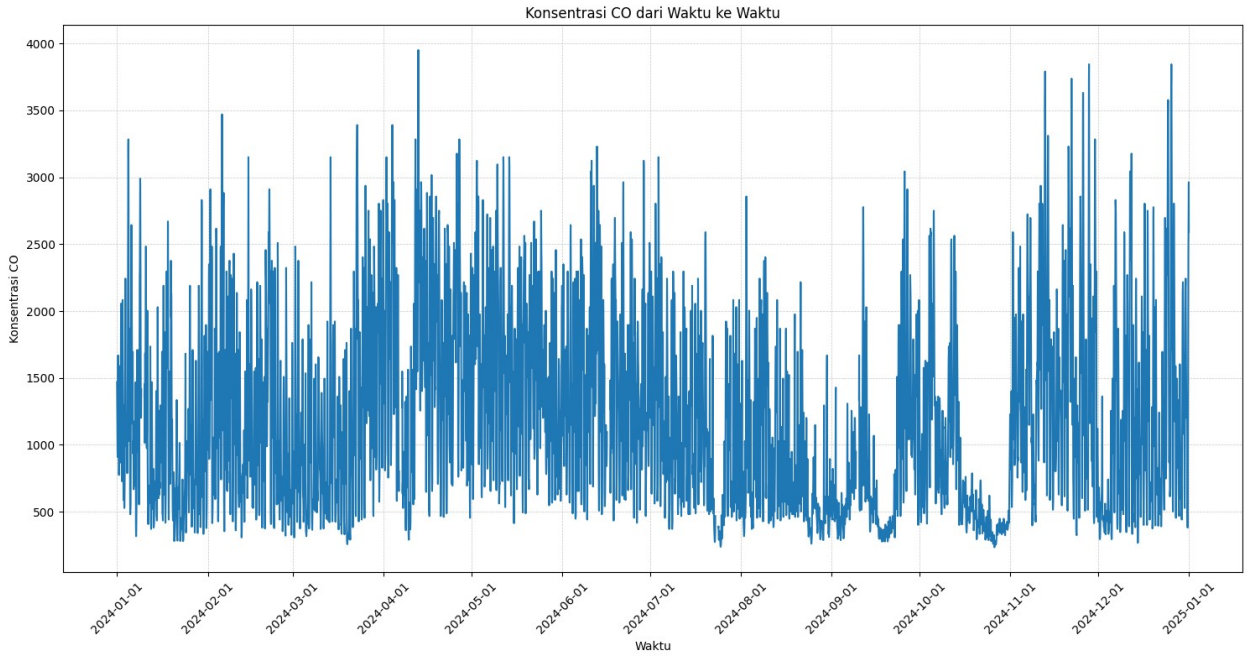
```
n        \"dtype\": \"number\",\n        \"std\": 36.73005477763032,\n
\"min\": 0.59,\n        \"max\": 277.9,\n
\"num_unique_values\": 5880,\n        \"samples\": [\n
91.18,\n          82.85,\n          44.93\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"CO\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 699.6717312282087,\n
\"min\": 233.65,\n        \"max\": 3952.03,\n
\"num_unique_values\": 259,\n        \"samples\": [\n
2910.61,\n          781.06,\n          1668.93\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"NO2\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 13.956821818495603,\n
\"min\": 0.73,\n        \"max\": 100.08,\n
\"num_unique_values\": 424,\n        \"samples\": [\n          20.05,\
n          2.29,\n          11.48\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"SO2\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 4.922606773196687,\n
\"min\": 0.4,\n        \"max\": 32.9,\n        \"num_unique_values\":
373,\n        \"samples\": [\n          23.37,\n          4.77,\n
2.89\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"O3\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 30.213595491664652,\n        \"min\": 0.0,\n        \"max\":
194.55,\n        \"num_unique_values\": 569,\n        \"samples\": [\n
86.55,\n          0.94,\n          27.54\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"df"}
```

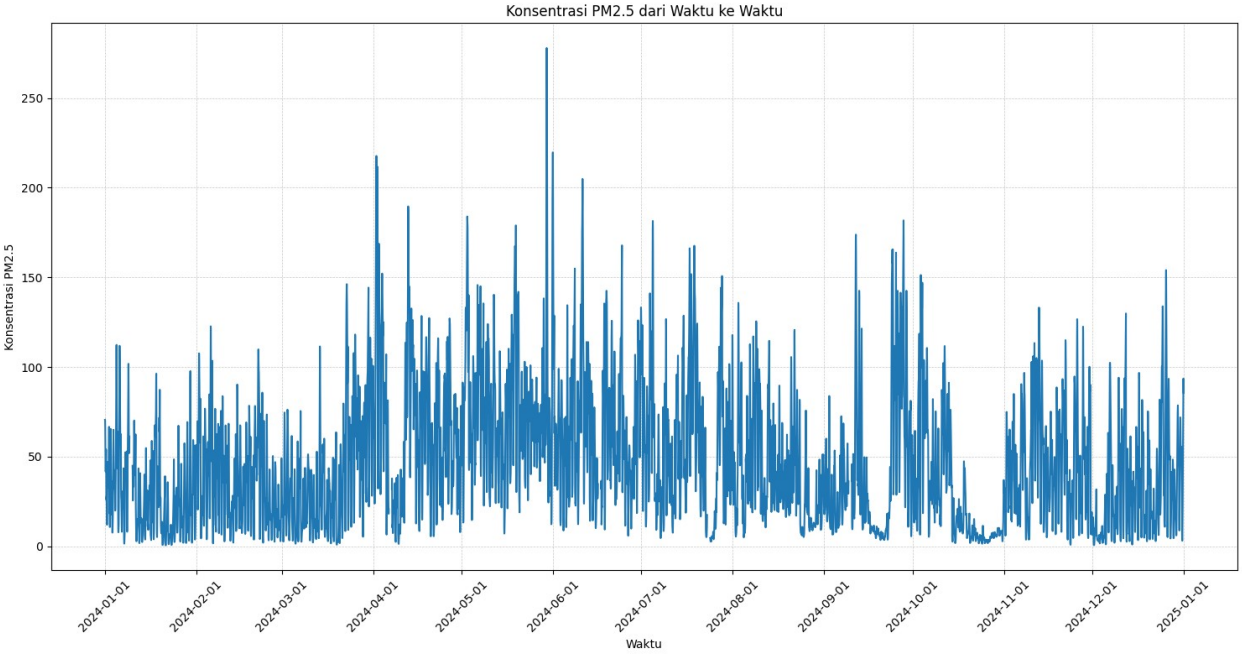

```
df.tail()
```

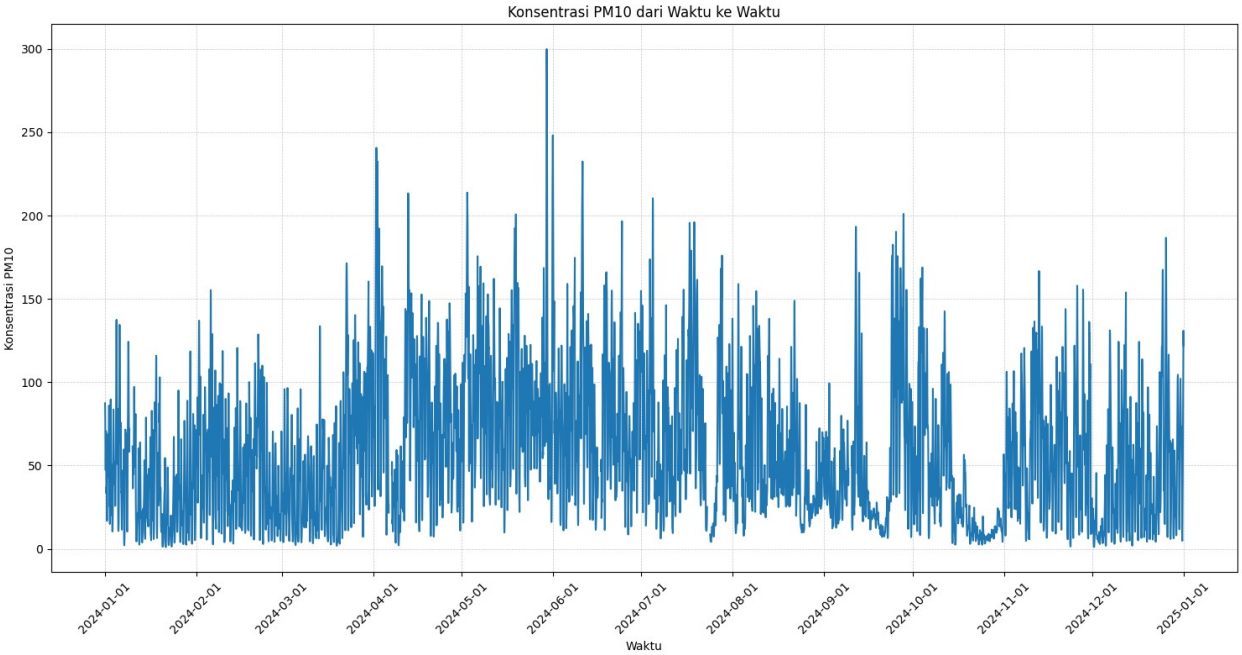
	Waktu	PM10	PM2.5	CO	NO2	SO2	O3
8779	2024-12-31 19:00:00	78.27	55.65	2456.67	47.30	21.46	0.01
8780	2024-12-31 20:00:00	98.21	69.76	2670.29	45.93	23.37	0.00
8781	2024-12-31 21:00:00	120.33	86.30	2937.32	46.61	25.51	0.00
8782	2024-12-31 22:00:00	130.87	93.58	2964.02	42.16	25.27	0.00
8783	2024-12-31 23:00:00	122.01	85.65	2590.18	32.90	20.98	0.00

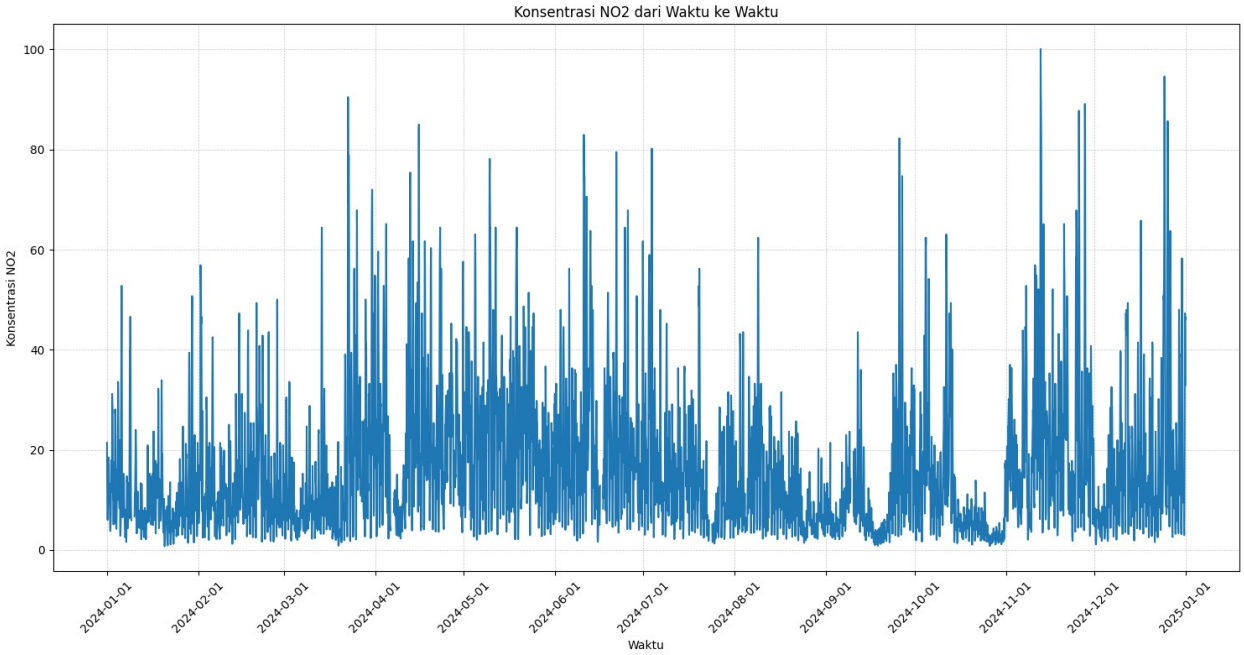
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.dates as mdates

# Tentukan semua kolom pencemar udara (selain waktu dan ISPU jika ada)
columns_to_plot = ['CO', 'PM2.5', 'PM10', 'NO2', 'SO2', 'O3']

# Loop untuk plot setiap kolom
for col in columns_to_plot:
    plt.figure(figsize=(15, 8))
```

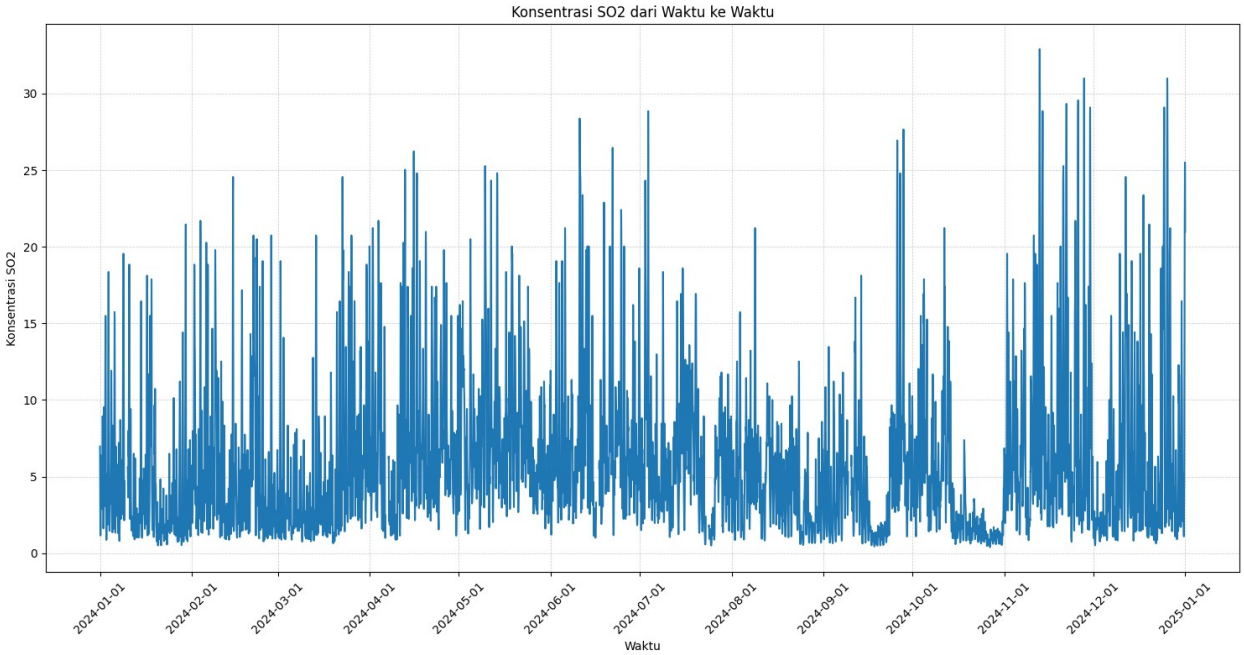
```
    plt.plot(df['Waktu'], df[col], label=col)

    # Format sumbu x untuk menampilkan tahun-bulan-tanggal
    plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-
%d'))
    plt.gca().xaxis.set_major_locator(mdates.MonthLocator())
    plt.xticks(rotation=45)

    # Set label dan judul
    plt.xlabel('Waktu')
    plt.ylabel(f'Konsentrasi {col}')
    plt.title(f'Konsentrasi {col} dari Waktu ke Waktu')

    # Tambahkan garis grid XY
    plt.grid(True, which='both', linestyle='--', linewidth=0.5,
alpha=0.7)
    #plt.legend()
    plt.tight_layout()
    plt.show()
```

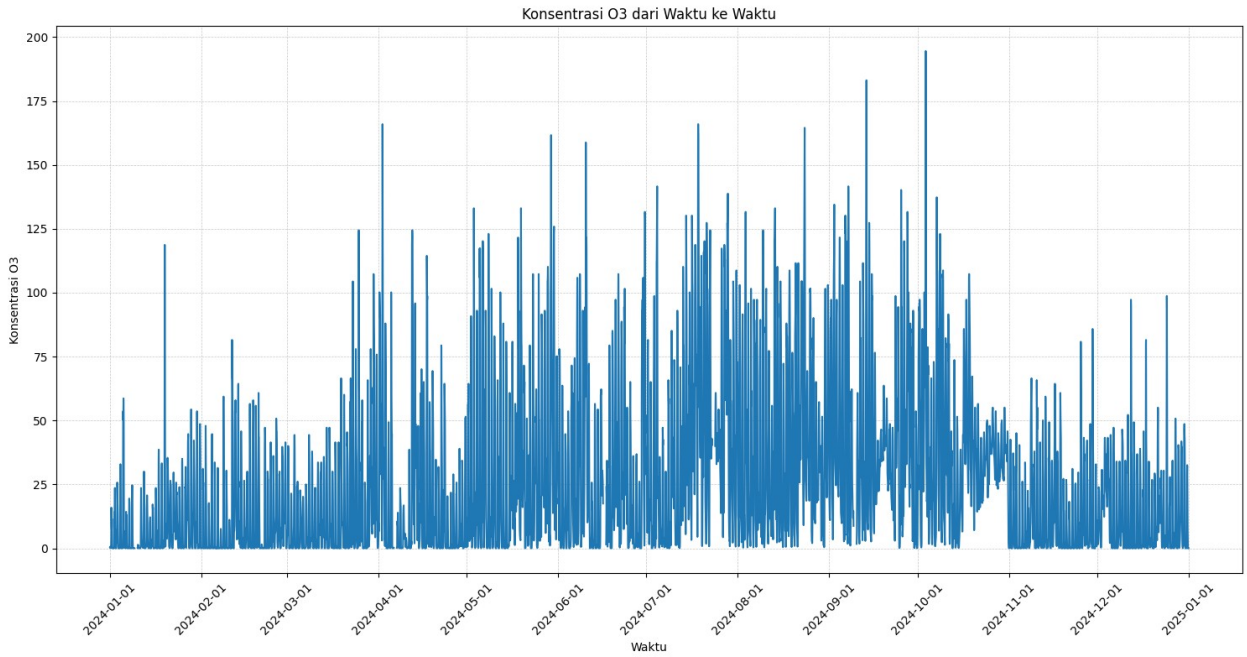



Konsentrasi PM2.5 dari Waktu ke Waktu
Konsentrasi PM2.5
0
50
100
150
200
250
2024-01-01
2024-02-01
2024-03-01
2024-04-01
2024-05-01
2024-06-01
2024-07-01
2024-08-01
2024-09-01
2024-10-01
2024-11-01
2024-12-01
2025-01-01
Waktu


Konsentrasi PM10 dari Waktu ke Waktu
Konsentrasi PM10
0
50
100
150
200
250
300
2024-01-01
2024-02-01
2024-03-01
2024-04-01
2024-05-01
2024-06-01
2024-07-01
2024-08-01
2024-09-01
2024-10-01
2024-11-01
2024-12-01
2025-01-01
Waktu


Konsentrasi NO2 dari Waktu ke Waktu
Konsentrasi NO2
0
20
40
60
80
100
2024-01-01
2024-02-01
2024-03-01
2024-04-01
2024-05-01
2024-06-01
2024-07-01
2024-08-01
2024-09-01
2024-10-01
2024-11-01
2024-12-01
2025-01-01
Waktu


Konsentrasi SO2 dari Waktu ke Waktu
Konsentrasi SO2
0
5
10
15
20
25
30
2024-01-01
2024-02-01
2024-03-01
2024-04-01
2024-05-01
2024-06-01
2024-07-01
2024-08-01
2024-09-01
2024-10-01
2024-11-01
2024-12-01
2025-01-01
Waktu




```
df.to_csv('03-data_date-bersih.csv', index=False)
```

imputasi

data prep

```
import pandas as pd

# Load CSV dan parse kolom 'Waktu' sebagai datetime
file_path = '/kaggle/working/03-data_date-bersih.csv'
df = pd.read_csv(file_path, parse_dates=['Waktu'])

# Display the first few rows of the dataset
df
```

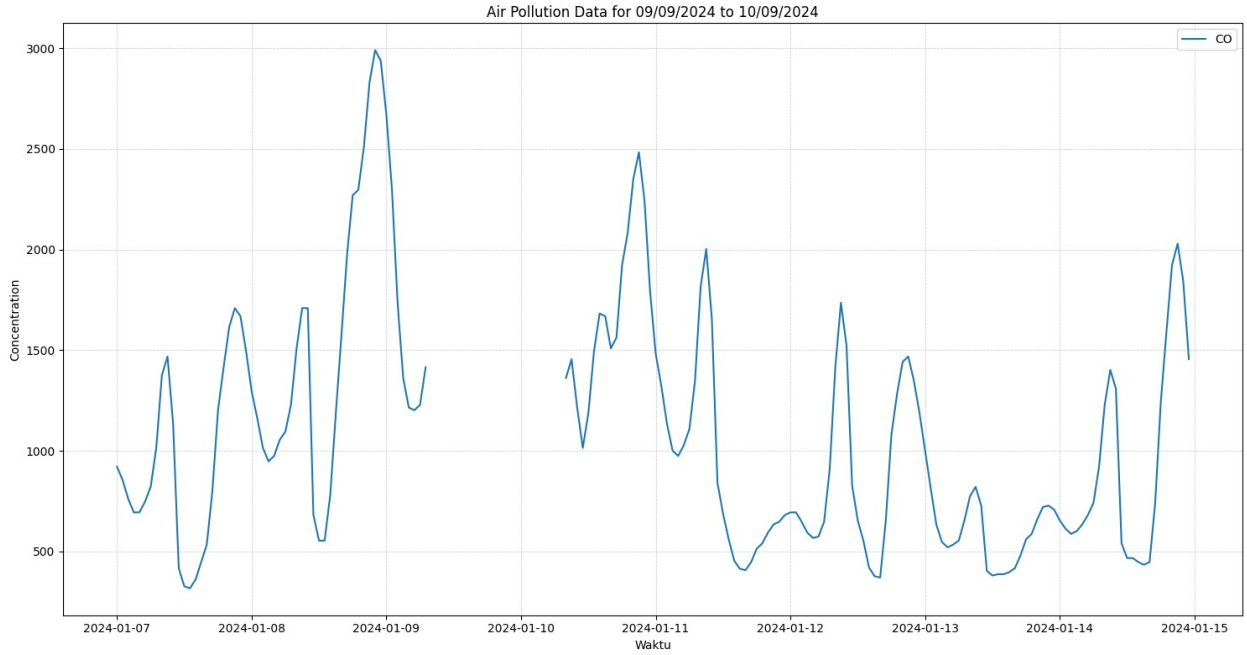
```
                  Waktu    PM10  PM2.5       CO    NO2    SO2    O3
0    2024-01-01 00:00:00   87.39  70.58  1468.66  21.42   6.97  0.30
1    2024-01-01 01:00:00   85.55  70.38  1375.20  15.59   4.53  0.12
2    2024-01-01 02:00:00   77.89  65.52  1214.98   9.94   2.12  0.36
3    2024-01-01 03:00:00   63.27  54.75  1054.76   7.37   1.33  0.65
4    2024-01-01 04:00:00   52.20  46.25   947.95   6.25   1.16  0.73
...                  ...     ...    ...      ...    ...    ...   ...
8779 2024-12-31 19:00:00   78.27  55.65  2456.67  47.30  21.46  0.01
8780 2024-12-31 20:00:00   98.21  69.76  2670.29  45.93  23.37  0.00
8781 2024-12-31 21:00:00  120.33  86.30  2937.32  46.61  25.51  0.00
8782 2024-12-31 22:00:00  130.87  93.58  2964.02  42.16  25.27  0.00
8783 2024-12-31 23:00:00  122.01  85.65  2590.18  32.90  20.98  0.00

[8784 rows x 7 columns]
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8784 entries, 0 to 8783
Data columns (total 7 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Waktu   8784 non-null   datetime64[ns]
 1   PM10    8663 non-null   float64
 2   PM2.5   8664 non-null   float64
 3   CO      8664 non-null   float64
 4   NO2     8662 non-null   float64
 5   SO2     8664 non-null   float64
 6   O3      8664 non-null   float64
dtypes: datetime64[ns](1), float64(6)
memory usage: 480.5 KB
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Konversi kolom Waktu menjadi datetime jika belum
df['Waktu'] = pd.to_datetime(df['Waktu'])

# Tentukan rentang waktu yang diinginkan
start_time = '2024-01-07 00:00:00'
end_time = '2024-01-14 23:00:00'

# Filter data berdasarkan rentang waktu
df_filtered = df[(df['Waktu'] >= start_time) & (df['Waktu'] <=
end_time)]

# Tentukan kolom yang ingin diplot
columns_to_plot = ['CO']

# Plot data
plt.figure(figsize=(15, 8))

for col in columns_to_plot:
    # Plot garis untuk kolom yang ada datanya
    plt.plot(df_filtered['Waktu'], df_filtered[col], label=col)

# Set labels dan title
plt.xlabel('Waktu')
plt.ylabel('Concentration')
plt.title('Air Pollution Data for 09/09/2024 to 10/09/2024')
plt.legend(loc='upper right')

# Tambahkan garis grid XY
plt.grid(True, which='both', linestyle='--', linewidth=0.5, alpha=0.7)
```

```
# Tampilkan plot
plt.tight_layout()
plt.show()
```




```
import pandas as pd

# Ambil hanya jam dari waktu
df['Hour'] = df['Waktu'].dt.hour

# Daftar kolom polutan
polutants = ['PM10', 'PM2.5', 'CO', 'NO2', 'SO2', 'O3']

# Hitung rata-rata tiap jam untuk semua polutan
hourly_avg = df.groupby('Hour')[polutants].mean().reset_index()

# Menampilkan hasil
print("Rata-rata konsentrasi tiap jam:")
print(hourly_avg)
```

```
Rata-rata konsentrasi tiap jam:
    Hour       PM10      PM2.5           CO        NO2        SO2
O3
0      0  77.353823  61.045928  1443.268504  19.749557   6.722355
9.730720
1      1  70.455512  56.704709  1307.969446  17.343463   5.536177
9.970970
2      2  63.109141  51.579917  1138.765125  14.254598   4.225845
10.536704
3      3  58.041440  47.960166  1021.357479  12.123019   3.509889
```

```
10.529114
4      4  55.746565  46.449612   966.730997  11.073850   3.266260
10.476510
5      5  54.371413  45.625402   948.239030  10.617590   3.226094
10.474238
6      6  53.808417  45.020803   957.623573  10.726704   3.387285
9.804626
7      7  59.643767  48.574875  1111.359307  12.666333   4.707479
8.031274
8      8  66.861274  53.179474  1391.952382  15.195457   6.500554
9.985291
9      9  71.496094  56.393352  1566.797091  17.493573   6.789612
16.298809
10    10  69.678726  55.394792  1467.548864  16.973222   5.756953
32.730693
11    11  31.510028  26.165346   703.770471   7.290332   3.664543
55.262133
12    12  26.344183  22.170471   621.793850   5.810831   3.362105
57.643823
13    13  25.335374  21.495734   614.878033   5.822770   3.447812
58.350970
14    14  26.452382  22.326648   653.582105   7.221911   3.748476
55.425014
15    15  29.914266  25.021634   739.165042   9.441662   4.253435
51.558532
16    16  33.329529  27.830111   826.383740  11.851551   4.881967
49.131884
17    17  41.183102  33.972825  1065.147452  19.087479   6.540665
38.721579
18    18  53.575928  43.384460  1405.886343  26.720305   8.447147
24.600886
19    19  65.194044  51.727729  1585.687064  27.741745   9.109861
18.117812
20    20  77.210305  60.550942  1716.936094  27.304515   9.880526
15.589945
21    21  85.521551  66.677812  1765.996011  26.236648  10.065125
13.816122
22    22  88.033102  68.276177  1702.891163  24.180194   9.405125
12.581468
23    23  84.927812  65.901662  1581.313463  21.888033   8.124792
10.896925
```

Mean ToD

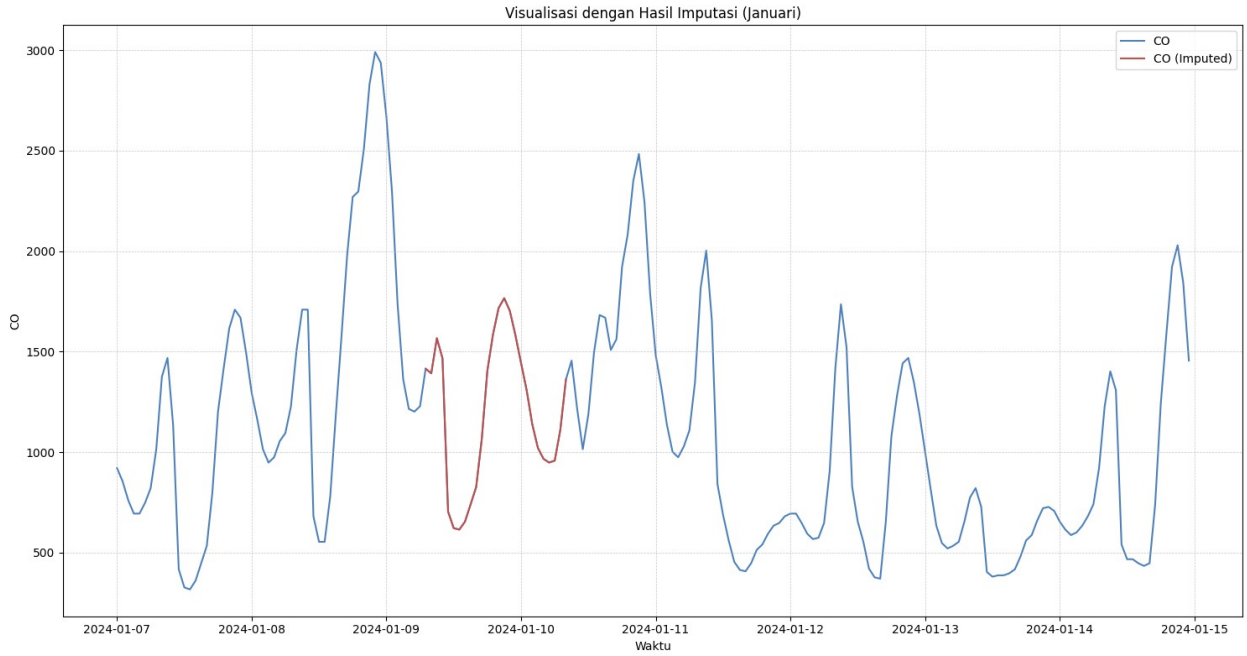
```python
# Asumsikan df sudah memiliki kolom 'hour' sebelumnya

# Daftar kolom yang ingin diimputasi
columns = ['PM10', 'PM2.5', 'CO', 'NO2', 'SO2', 'O3']
```

```
# Proses imputasi Mean Time-of-Day (ToD) berdasarkan kolom 'hour' yang
sudah ada
for col in columns:
    # Hitung mean per jam
    mean_per_hour = df.groupby('Hour')[col].transform('mean')
    # Imputasi hanya nilai NaN
    df[col] = df[col].fillna(mean_per_hour)

# Hapus kolom 'hour' setelah imputasi
df.drop(columns=['Hour'], inplace=True)

# Lihat hasil imputasi
print(df.head())
```

	Waktu	PM10	PM2.5	CO	NO2	SO2	O3
0	2024-01-01 00:00:00	87.39	70.58	1468.66	21.42	6.97	0.30
1	2024-01-01 01:00:00	85.55	70.38	1375.20	15.59	4.53	0.12
2	2024-01-01 02:00:00	77.89	65.52	1214.98	9.94	2.12	0.36
3	2024-01-01 03:00:00	63.27	54.75	1054.76	7.37	1.33	0.65
4	2024-01-01 04:00:00	52.20	46.25	947.95	6.25	1.16	0.73

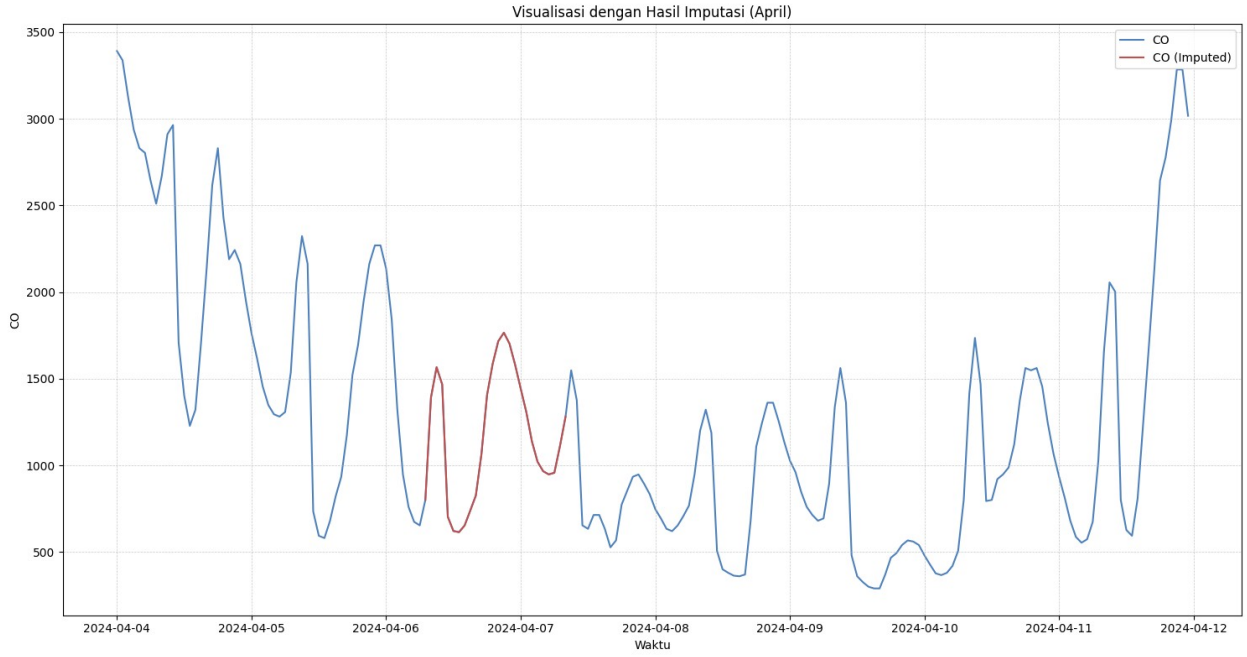
```
import pandas as pd
import matplotlib.pyplot as plt

# Pastikan kolom waktu dalam format datetime
df['Waktu'] = pd.to_datetime(df['Waktu'])

# Filter data dari 7 Jan sampai 14 Jan
start_time = '2024-01-07 00:00:00'
end_time = '2024-01-14 23:00:00'
df_filtered = df[(df['Waktu'] >= start_time) & (df['Waktu'] <=
end_time)]

# Rentang waktu imputasi
highlight_start = pd.to_datetime('2024-01-09 07:00:00')
highlight_end = pd.to_datetime('2024-01-10 08:00:00')

# Plot utama (semua garis biru dulu)
plt.figure(figsize=(15, 8))
plt.plot(df_filtered['Waktu'], df_filtered['CO'], label='CO',
color='#4F81BD', zorder=1)

# Overlay: Plot garis merah di rentang waktu imputasi (di atas garis
biru)
df_highlight = df_filtered[(df_filtered['Waktu'] >= highlight_start) &
(df_filtered['Waktu'] <= highlight_end)]
plt.plot(df_highlight['Waktu'], df_highlight['CO'], label='CO
(Imputed)', color='#C0504D', zorder=2)

# Label dan keterangan
```

```
plt.xlabel('Waktu')
plt.ylabel('CO')
plt.title('Visualisasi dengan Hasil Imputasi (Januari)')
plt.legend()
plt.grid(True, which='both', linestyle='--', linewidth=0.5, alpha=0.7)
plt.tight_layout()
plt.show()
```



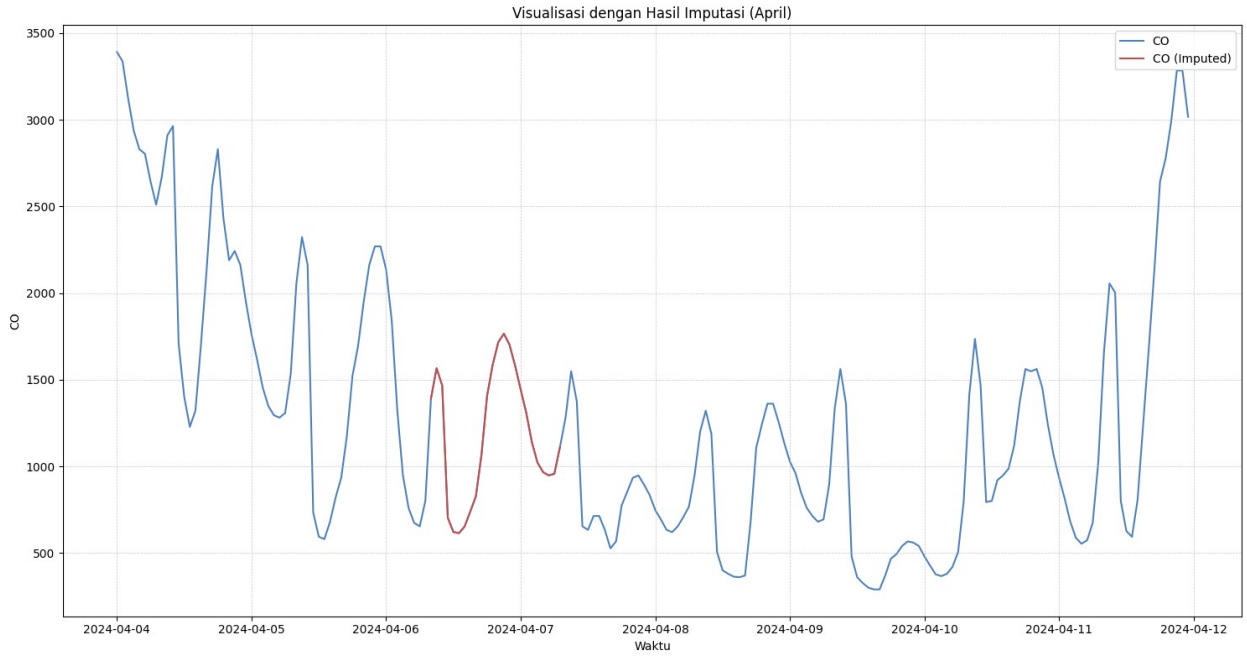

```
import pandas as pd
import matplotlib.pyplot as plt

# Pastikan kolom waktu dalam format datetime
df['Waktu'] = pd.to_datetime(df['Waktu'])

# Tentukan rentang waktu yang diinginkan
start_time = '2024-04-04 00:00:00'
end_time = '2024-04-11 23:00:00'
df_filtered = df[(df['Waktu'] >= start_time) & (df['Waktu'] <=
end_time)]

# Rentang waktu imputasi
highlight_start = pd.to_datetime('2024-04-06 07:00:00')
highlight_end = pd.to_datetime('2024-04-07 08:00:00')

# Plot utama (semua garis biru dulu)
plt.figure(figsize=(15, 8))
plt.plot(df_filtered['Waktu'], df_filtered['CO'], label='CO',
color='#4F81BD', zorder=1)
```

```
# Overlay: Plot garis merah di rentang waktu imputasi (di atas garis biru)
df_highlight = df_filtered[(df_filtered['Waktu'] >= highlight_start) & (df_filtered['Waktu'] <= highlight_end)]
plt.plot(df_highlight['Waktu'], df_highlight['CO'], label='CO (Imputed)', color='#C0504D', zorder=2)

# Label dan keterangan
plt.xlabel('Waktu')
plt.ylabel('CO')
plt.title('Visualisasi dengan Hasil Imputasi (April)')
plt.legend()
plt.grid(True, which='both', linestyle='--', linewidth=0.5, alpha=0.7)
plt.tight_layout()
plt.show()
```




```
import pandas as pd
import matplotlib.pyplot as plt

# Pastikan kolom waktu dalam format datetime
df['Waktu'] = pd.to_datetime(df['Waktu'])

# Tentukan rentang waktu yang diinginkan
start_time = '2024-04-04 00:00:00'
end_time = '2024-04-11 23:00:00'
df_filtered = df[(df['Waktu'] >= start_time) & (df['Waktu'] <= end_time)]

# Rentang waktu imputasi
```

```
highlight_start = pd.to_datetime('2024-04-06 08:00:00')
highlight_end = pd.to_datetime('2024-04-07 07:00:00')

# Plot utama (semua garis biru dulu)
plt.figure(figsize=(15, 8))
plt.plot(df_filtered['Waktu'], df_filtered['CO'], label='CO',
color='#4F81BD', zorder=1)

# Overlay: Plot garis merah di rentang waktu imputasi (di atas garis
biru)
df_highlight = df_filtered[(df_filtered['Waktu'] >= highlight_start) &
(df_filtered['Waktu'] <= highlight_end)]
plt.plot(df_highlight['Waktu'], df_highlight['CO'], label='CO
(Imputed)', color='#C0504D', zorder=2)

# Label dan keterangan
plt.xlabel('Waktu')
plt.ylabel('CO')
plt.title('Visualisasi dengan Hasil Imputasi (April)')
plt.legend()
plt.grid(True, which='both', linestyle='--', linewidth=0.5, alpha=0.7)
plt.tight_layout()
plt.show()
```




```
# Cek jumlah missing value per kolom
jumlah_missing_value_setelah_imputasi = df.isnull().sum()
print("\nJumlah missing value per kolom:")
print(jumlah_missing_value_setelah_imputasi)
```

```
Jumlah missing value per kolom:
Waktu    0
PM10     0
PM2.5    0
CO       0
NO2      0
SO2      0
O3       0
dtype: int64
```

```
# Cek nilai negatif di semua kolom numerik
negative_values = (df.select_dtypes(include=['number']) < 0).sum()

# Tampilkan hasil
print("Jumlah nilai negatif per kolom:")
print(negative_values)

# Jika ingin melihat baris dengan nilai negatif
negative_rows = df[(df.select_dtypes(include=['number']) <
0).any(axis=1)]
print("Baris dengan nilai negatif:")
negative_rows
```

```
Jumlah nilai negatif per kolom:
PM10     0
PM2.5    0
CO       0
NO2      0
SO2      0
O3       0
dtype: int64
Baris dengan nilai negatif:

Empty DataFrame
Columns: [Waktu, PM10, PM2.5, CO, NO2, SO2, O3]
Index: []
```

```
# Tanggal dan waktu yang ingin dicek (tetap sebagai string)
dates_to_check_str = [
    "2024-02-08 06:00:00",
    "2024-08-05 10:00:00",
    "2024-08-11 07:00:00"
]

# --- Solusi yang direkomendasikan ---
# Konversi `dates_to_check_str` menjadi objek datetime
dates_to_check_dt = pd.to_datetime(dates_to_check_str)

# Filter data berdasarkan kolom Waktu
selected_data = df[df['Waktu'].isin(dates_to_check_dt)]
```

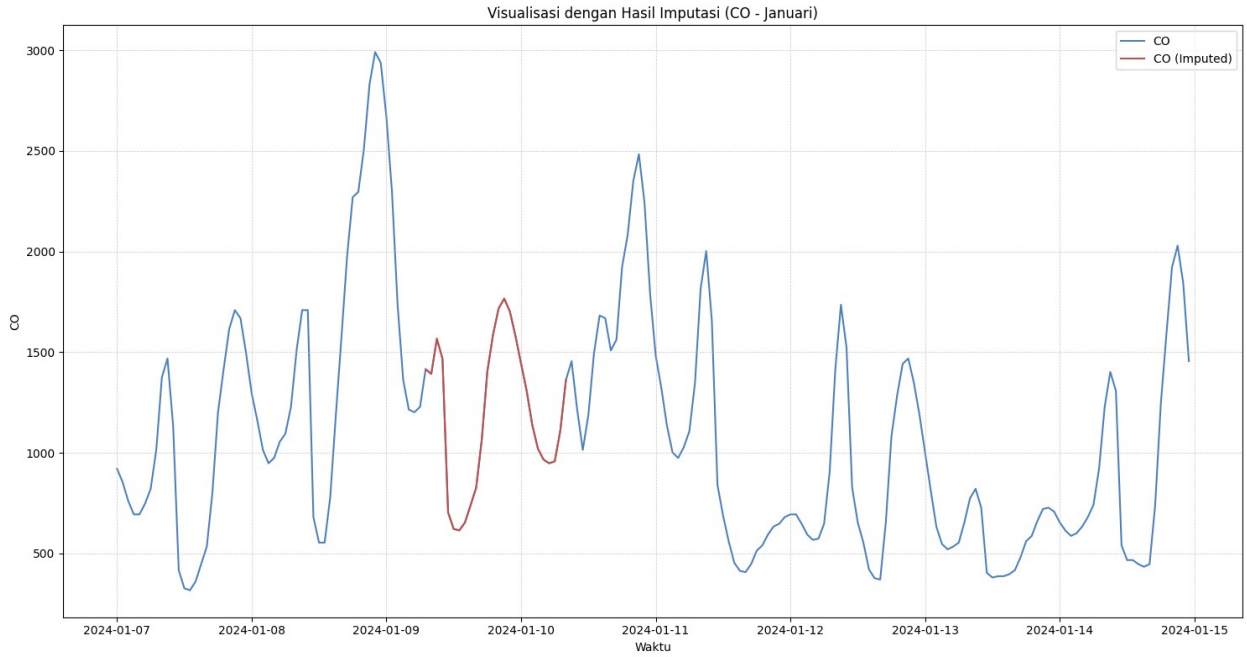
```
selected_data
```

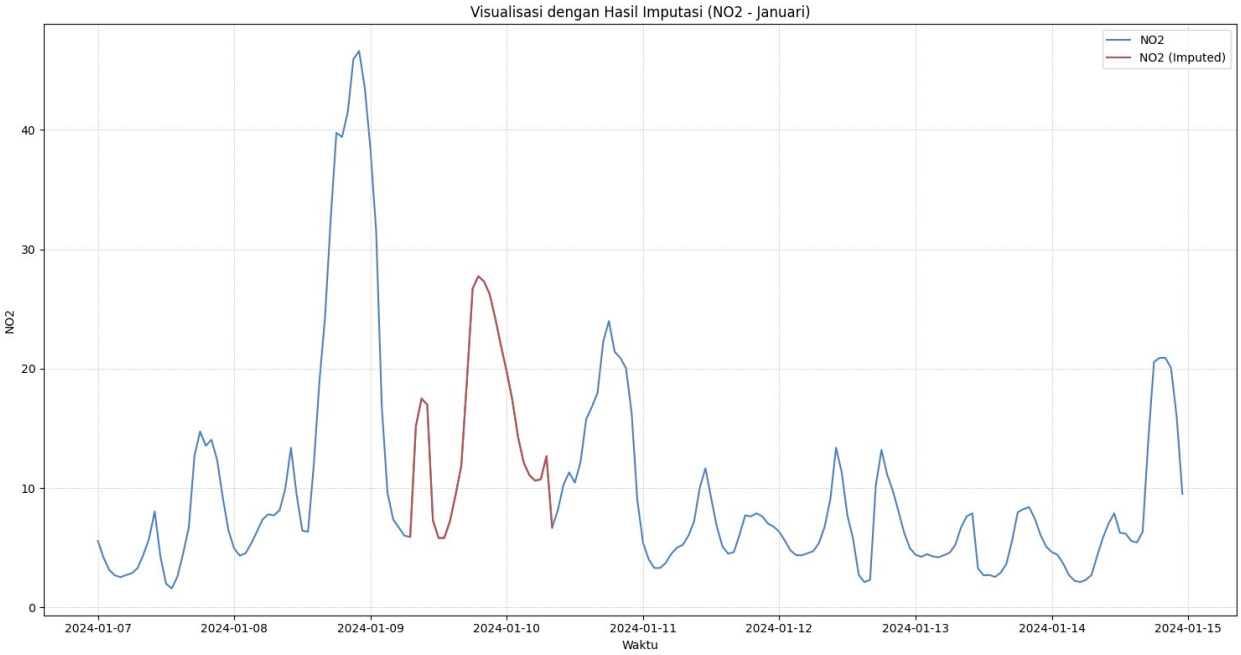
```
                    Waktu       PM10  PM2.5       CO        NO2   SO2
O3
918  2024-02-08 06:00:00  53.808417  38.32   921.25   2.230000  1.97
0.00
5218 2024-08-05 10:00:00  62.160000  49.02  1321.79  16.973222  6.20
47.92
5359 2024-08-11 07:00:00  26.830000  17.05   607.49  12.666333  2.12
2.68
```

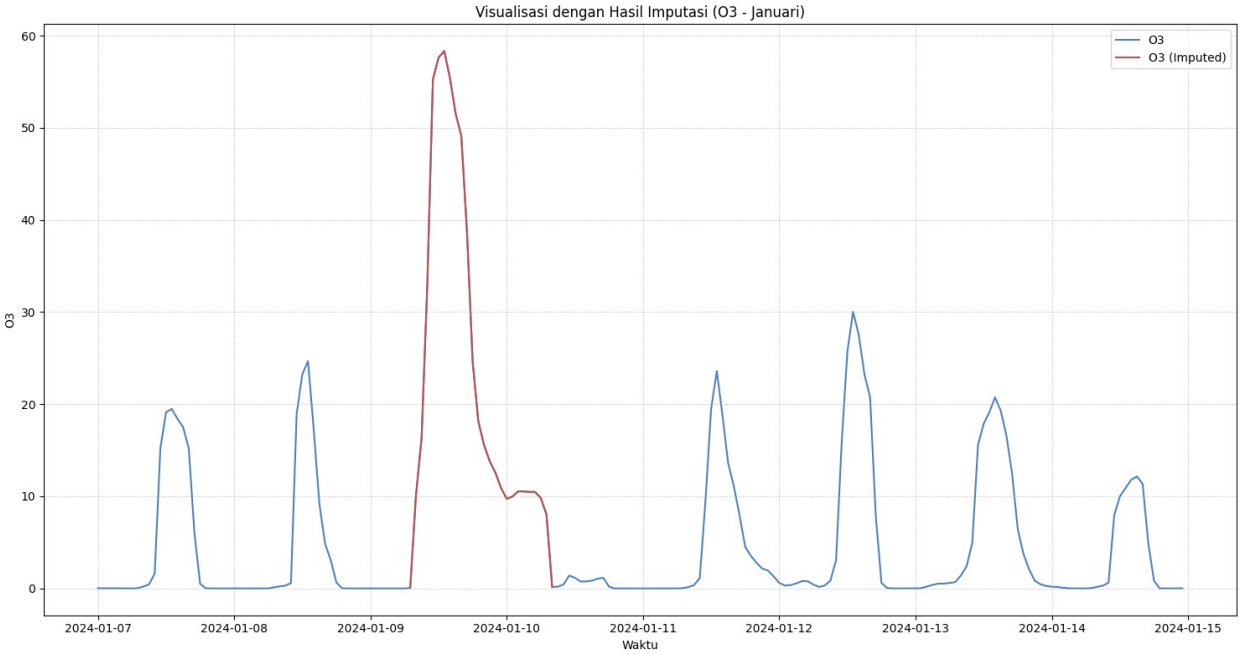
```
import pandas as pd
import matplotlib.pyplot as plt
import os

# Make sure your DataFrame 'df' is loaded.
# If you haven't loaded it yet, uncomment and modify the line below:
# df = pd.read_csv('path/to/your/05-data_ISPU_2024.csv')

# Ensure the 'Waktu' column is in datetime format
df['Waktu'] = pd.to_datetime(df['Waktu'])

# Filter data from January 7th to January 14th
start_time = '2024-01-07 00:00:00'
end_time = '2024-01-14 23:00:00'
df_filtered = df[(df['Waktu'] >= start_time) & (df['Waktu'] <=
end_time)].copy()

# Imputation time range
highlight_start = pd.to_datetime('2024-01-09 07:00:00')
highlight_end = pd.to_datetime('2024-01-10 08:00:00')

# Define the 6 pollutant columns - CORRECTED 'PM25' to 'PM2.5'
pollutant_columns = ['CO', 'NO2', 'O3', 'SO2', 'PM10', 'PM2.5'] # <---
HERE IS THE CHANGE

# Create a directory to save the plots
output_plot_folder = 'pollutant_imputation_plots'
os.makedirs(output_plot_folder, exist_ok=True)

# Loop through each pollutant to generate a separate plot
for pollutant in pollutant_columns:
    plt.figure(figsize=(15, 8))

    # Plot the main line (blue)
    plt.plot(df_filtered['Waktu'], df_filtered[pollutant],
label=pollutant, color='#4F81BD', zorder=1)

    # Overlay: Plot the red line for the imputation range
    df_highlight = df_filtered[(df_filtered['Waktu'] >=
```

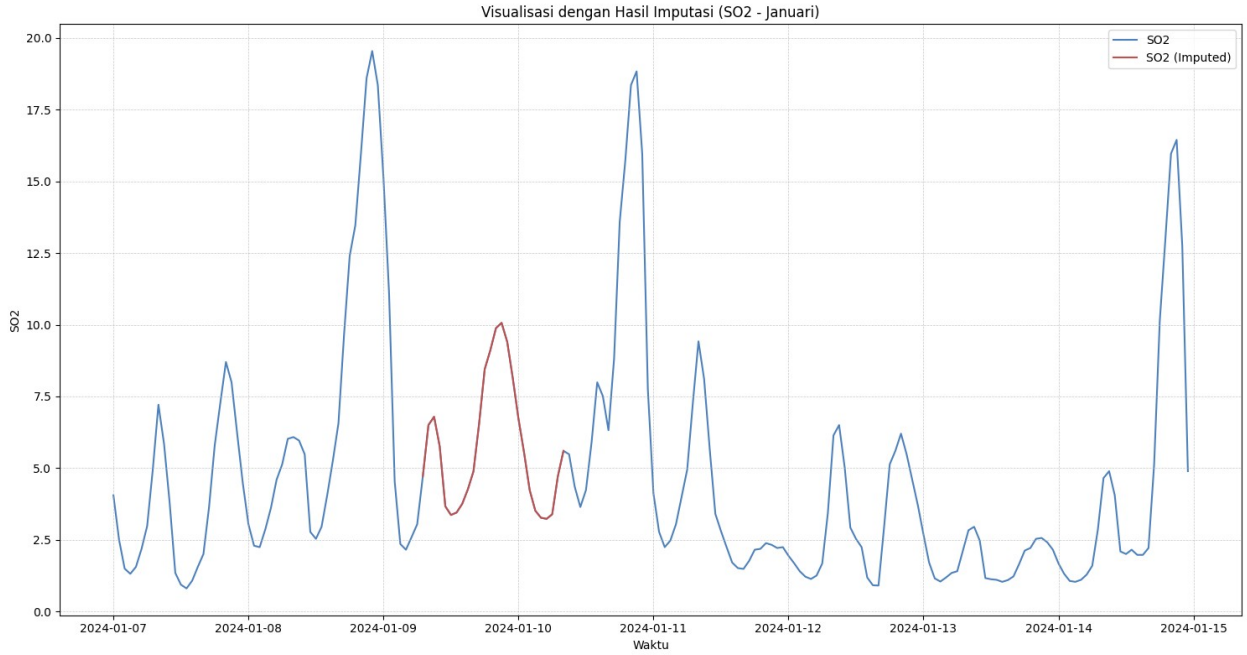
```
highlight_start) & (df_filtered['Waktu'] <= highlight_end)].copy()
    plt.plot(df_highlight['Waktu'], df_highlight[pollutant],
label=f'{pollutant} (Imputed)', color='#C0504D', zorder=2)

    # Labels and title
    plt.xlabel('Waktu')
    plt.ylabel(pollutant)
    plt.title(f'Visualisasi dengan Hasil Imputasi ({pollutant} -
Januari)')
    plt.legend()
    plt.grid(True, which='both', linestyle='--', linewidth=0.5,
alpha=0.7)
    plt.tight_layout()

    # Save the plot and close the figure
    plt.savefig(os.path.join(output_plot_folder,
f'imputation_plot_{pollutant}.png'))


print(f"Visualisasi untuk 6 polutan telah disimpan di folder:
{output_plot_folder}")
```

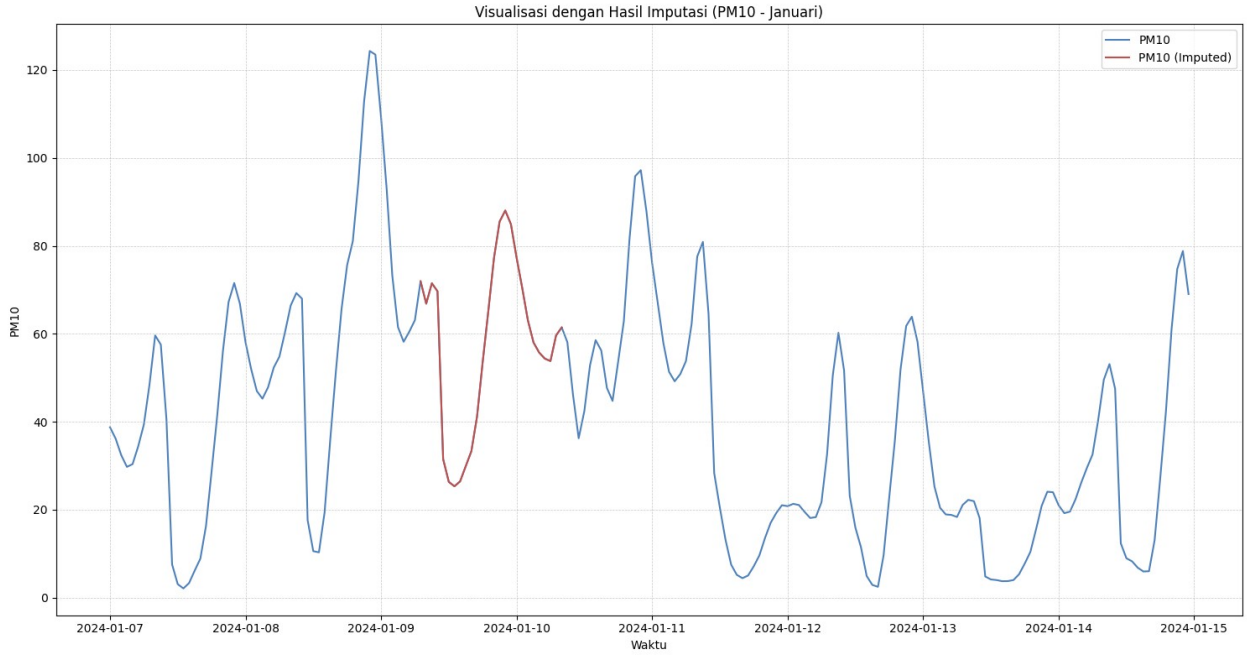
```
Visualisasi untuk 6 polutan telah disimpan di folder:
pollutant_imputation_plots
```

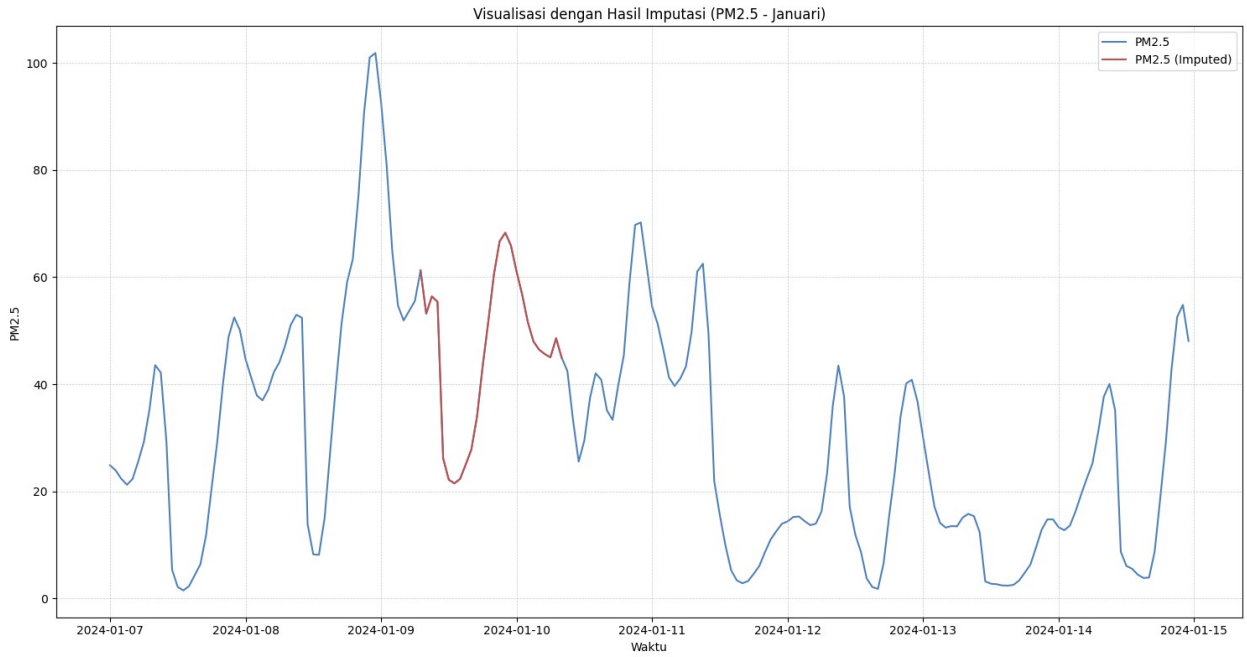



Visualisasi dengan Hasil Imputasi (NO2 - Januari)
NO2
NO2 (Imputed)
NO2
Waktu
0
10
20
30
40
2024-01-07
2024-01-08
2024-01-09
2024-01-10
2024-01-11
2024-01-12
2024-01-13
2024-01-14
2024-01-15


Visualisasi dengan Hasil Imputasi (O3 - Januari)
O3
O3 (Imputed)
O3
Waktu
0
10
20
30
40
50
60
2024-01-07
2024-01-08
2024-01-09
2024-01-10
2024-01-11
2024-01-12
2024-01-13
2024-01-14
2024-01-15


Visualisasi dengan Hasil Imputasi (SO2 - Januari)
SO2
SO2 (Imputed)
SO2
Waktu
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
2024-01-07
2024-01-08
2024-01-09
2024-01-10
2024-01-11
2024-01-12
2024-01-13
2024-01-14
2024-01-15


Visualisasi dengan Hasil Imputasi (PM10 - Januari)
PM10
PM10 (Imputed)
PM10
Waktu
0
20
40
60
80
100
120
2024-01-07
2024-01-08
2024-01-09
2024-01-10
2024-01-11
2024-01-12
2024-01-13
2024-01-14
2024-01-15




```
df.to_csv('04-data_MeanTod_2024.csv', index=False)
```

Hitung ISPU

```
import pandas as pd
df = pd.read_csv('/content/04-data_MeanTod_2024.csv')
```

```
'Timestamp (UTC)', 'CO', 'PM2.5', 'PM10', 'NO2', 'SO2', 'O3'
```

```
('Timestamp (UTC)', 'CO', 'PM2.5', 'PM10', 'NO2', 'SO2', 'O3')
```

```
df.shape
```

```
(8784, 7)
```

```
def calculate_ispu(concentration, c_low, c_high, i_low, i_high):
    return ((concentration - c_low) / (c_high - c_low)) * (i_high -
i_low) + i_low

def calculate_ispu_pm10(concentration):
    if concentration <= 50:
        return calculate_ispu(concentration, 0, 50, 0, 50)
    elif concentration <= 150:
        return calculate_ispu(concentration, 51, 150, 51, 100)
    elif concentration <= 350:
        return calculate_ispu(concentration, 151, 350, 101, 200)
    elif concentration <= 420:
        return calculate_ispu(concentration, 351, 420, 201, 300)
    else:
        return calculate_ispu(concentration, 421, 500, 301, 500)
```

```
def calculate_ispu_pm25(concentration):
    if concentration <= 15.5:
        return calculate_ispu(concentration, 0, 15.5, 0, 50)
    elif concentration <= 55.4:
        return calculate_ispu(concentration, 15.5, 55.4, 51, 100)
    elif concentration <= 150.4:
        return calculate_ispu(concentration, 55.4, 150.4, 101, 200)
    elif concentration <= 250.4:
        return calculate_ispu(concentration, 150.4, 250.4, 201, 300)
    else:
        return calculate_ispu(concentration, 250.4, 500, 301, 500)

def calculate_ispu_co(concentration):
    if concentration <= 4000:
        return calculate_ispu(concentration, 0, 4000, 0, 50)
    elif concentration <= 8000:
        return calculate_ispu(concentration, 4000, 8000, 51, 100)
    elif concentration <= 15000:
        return calculate_ispu(concentration, 8000, 15000, 101, 200)
    elif concentration <= 30000:
        return calculate_ispu(concentration, 15000, 30000, 201, 300)
    else:
        return calculate_ispu(concentration, 30000, 45000, 301, 500)

def calculate_ispu_no2(concentration):
    if concentration <= 80:
        return calculate_ispu(concentration, 0, 80, 0, 50)
    elif concentration <= 200:
        return calculate_ispu(concentration, 80, 200, 51, 100)
    elif concentration <= 1130:
        return calculate_ispu(concentration, 200, 1130, 101, 200)
    elif concentration <= 2260:
        return calculate_ispu(concentration, 1130, 2260, 201, 300)
    else:
        return calculate_ispu(concentration, 2260, 3000, 301, 500)

def calculate_ispu_so2(concentration):
    if concentration <= 52:
        return calculate_ispu(concentration, 0, 52, 0, 50)
    elif concentration <= 180:
        return calculate_ispu(concentration, 53, 180, 51, 100)
    elif concentration <= 400:
        return calculate_ispu(concentration, 181, 400, 101, 200)
    elif concentration <= 800:
        return calculate_ispu(concentration, 401, 800, 201, 300)
    else:
        return calculate_ispu(concentration, 801, 1000, 301, 500)

def calculate_ispu_o3(concentration):
```

```
    if concentration <= 120:
        return calculate_ispu(concentration, 0, 120, 0, 50)
    elif concentration <= 235:
        return calculate_ispu(concentration, 121, 235, 51, 100)
    elif concentration <= 400:
        return calculate_ispu(concentration, 236, 400, 101, 200)
    elif concentration <= 800:
        return calculate_ispu(concentration, 401, 800, 201, 300)
    else:
        return calculate_ispu(concentration, 801, 1000, 301, 500)

def get_air_quality_category(ispu_value):
    if ispu_value <= 50:
        return "Baik"
    elif ispu_value <= 100:
        return "Sedang"
    elif ispu_value <= 200:
        return "Tidak Sehat"
    elif ispu_value <= 300:
        return "Sangat Tidak Sehat"
    else:
        return "Berbahaya"

df['ISPU_PM10'] = df['PM10'].apply(calculate_ispu_pm10)
df['ISPU_PM2.5'] = df['PM2.5'].apply(calculate_ispu_pm25)
df['ISPU_CO'] = df['CO'].apply(calculate_ispu_co)
df['ISPU_NO2'] = df['NO2'].apply(calculate_ispu_no2)
df['ISPU_SO2'] = df['SO2'].apply(calculate_ispu_so2)
df['ISPU_O3'] = df['O3'].apply(calculate_ispu_o3)

df['Nilai ISPU'] = df[
    ['ISPU_PM10', 'ISPU_PM2.5', 'ISPU_CO', 'ISPU_NO2', 'ISPU_SO2',
'ISPU_O3']
].max(axis=1)

df['Kategori'] = df['Nilai ISPU'].apply(get_air_quality_category)

df
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 8784,\n  \"fields\":
[\n    {\n      \"column\": \"Waktu\",\n      \"properties\": {\n
\"dtype\": \"object\",\n        \"num_unique_values\": 8784,\n
\"samples\": [\n          \"2024-09-23 01:00:00\",\n          \"2024-
03-03 00:00:00\",\n          \"2024-09-29 11:00:00\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"PM10\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 41.910576712695416,\n
\"min\": 1.02,\n        \"max\": 299.88,\n
\"num_unique_values\": 6293,\n        \"samples\": [\n
88.59,\n          63.65,\n          114.12\n        ],\n
```

```
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"PM2.5\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 36.519139730560795,\
n        \"min\": 0.59,\n        \"max\": 277.9,\n
\"num_unique_values\": 5904,\n        \"samples\": [\n
58.07,\n          44.84,\n          35.7\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"CO\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 696.2228954127207,\n
\"min\": 233.65,\n        \"max\": 3952.03,\n
\"num_unique_values\": 283,\n        \"samples\": [\n
1668.93,\n          260.35,\n          1455.31\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"NO2\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 13.883107581917832,\n
\"min\": 0.73,\n        \"max\": 100.08,\n
\"num_unique_values\": 448,\n        \"samples\": [\n          2.38,\n
19.54,\n          6.43\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"SO2\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 4.896158084049618,\n        \"min\": 0.4,\n        \"max\":
32.9,\n        \"num_unique_values\": 397,\n        \"samples\": [\n
7.21,\n          9.78,\n          3.81\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"O3\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 30.085010993395432,\n
\"min\": 0.0,\n        \"max\": 194.55,\n
\"num_unique_values\": 593,\n        \"samples\": [\n          2.3,\n
0.67,\n          47.21\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ISPU_PM10\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 27.651930518963916,\n        \"min\":
1.02,\n        \"max\": 175.0659296482412,\n
\"num_unique_values\": 6293,\n        \"samples\": [\n
69.60515151515152,\n          57.26111111111111,\n
82.24121212121213\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ISPU_PM2.5\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 46.14346732590953,\n        \"min\":
1.9032258064516128,\n        \"max\": 322.9250801282051,\n
\"num_unique_values\": 5904,\n        \"samples\": [\n
103.78242105263158,\n          87.03157894736842,\n
75.80701754385966\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ISPU_CO\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 8.702786192659095,\n        \"min\": 2.920625,\n
\"max\": 49.400375000000004,\n        \"num_unique_values\": 283,\n
\"samples\": [\n          20.861625,\n          3.2543750000000005,\n
18.191374999999997\n        ],\n        \"semantic_type\": \"\",\n
```

\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"ISPU_NO2\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 8.668351450767455,\n   \"min\":
0.45625,\n   \"max\": 59.19933333333333,\n
\"num_unique_values\": 448,\n   \"samples\": [\n
1.4874999999999998,\n   12.2125,\n   4.01875\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"ISPU_SO2\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 4.707844311586157,\n   \"min\":
0.38461538461538464,\n   \"max\": 31.634615384615383,\n
\"num_unique_values\": 397,\n   \"samples\": [\n
6.9326923076923075,\n   9.403846153846153,\n
3.6634615384615383\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"ISPU_O3\",\n   \"properties\": {\n   \"dtype\": \"number\",\
n   \"std\": 12.571063154079857,\n   \"min\": 0.0,\n
\"max\": 82.61359649122807,\n   \"num_unique_values\": 593,\n
\"samples\": [\n   0.9583333333333333,\n
0.2791666666666667,\n   19.670833333333334\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Nilai ISPU\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
45.711457767543735,\n   \"min\": 6.033333333333333,\n
\"max\": 322.9250801282051,\n   \"num_unique_values\": 5849,\n
\"samples\": [\n   119.24726315789474,\n
130.32484210526317,\n   53.026315789473685\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Kategori\",\n   \"properties\":
{\n   \"dtype\": \"category\",\n   \"num_unique_values\":
5,\n   \"samples\": [\n   \"Sedang\",\n
\"Berbahaya\",\n   \"Baik\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe","variable_name":"df"}

```
# Hitung jumlah nilai negatif
negative_values = (df.select_dtypes(include=['number']) < 0).sum()

# Tampilkan hasil
print("Jumlah nilai negatif per kolom:")
print(negative_values)

# Jika ingin melihat baris yang sebelumnya memiliki nilai negatif
nan_rows = df[df.isna().any(axis=1)]
print("Baris yang memiliki nilai NaN:")
nan_rows
```

```
Jumlah nilai negatif per kolom:
PM10          0
PM2.5         0
```

```
CO             0
NO2            0
SO2            0
O3             0
ISPU_PM10      0
ISPU_PM2.5     0
ISPU_CO        0
ISPU_NO2       0
ISPU_SO2       0
ISPU_O3        0
Nilai ISPU     0
dtype: int64
Baris yang memiliki nilai NaN:
```

{"repr_error":"Out of range float values are not JSON compliant: nan","type":"dataframe","variable_name":"nan_rows"}

```
# Cek jumlah missing value per kolom
jumlah_missing_value_per_kolom = df.isnull().sum()
print("\nJumlah missing value per kolom:")
print(jumlah_missing_value_per_kolom)
```

```
Jumlah missing value per kolom:
Waktu          0
PM10           0
PM2.5          0
CO             0
NO2            0
SO2            0
O3             0
ISPU_PM10      0
ISPU_PM2.5     0
ISPU_CO        0
ISPU_NO2       0
ISPU_SO2       0
ISPU_O3        0
Nilai ISPU     0
Kategori       0
dtype: int64
```

```
# Replace 'Kategori_Anda' with the actual category you want to display
kategorii = 'Berbahaya'
filtered_df = df[df['Kategori'] == kategorii]

# Display the filtered DataFrame
filtered_df
```

{"summary":"{\n \"name\": \"filtered_df\",\n \"rows\": 4,\n \"fields\": [\n   {\n   \"column\": \"Waktu\",\n \"properties\": {\n   \"dtype\": \"date\",\n   \"min\":

```
\"2024-05-29 20:00:00\",\n        \"max\": \"2024-05-29 23:00:00\",\n
\"num_unique_values\": 4,\n        \"samples\": [\n          \"2024-
05-29 21:00:00\",\n          \"2024-05-29 23:00:00\",\n
\"2024-05-29 20:00:00\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"PM10\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 8.926598362945063,\n        \"min\":
278.4,\n        \"max\": 299.88,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          299.88,\n          278.4,\n          287.51\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PM2.5\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 9.63688227592305,\n        \"min\": 254.44,\n        \"max\":
277.9,\n        \"num_unique_values\": 4,\n        \"samples\": [\n
277.9,\n          254.44,\n          268.03\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"CO\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 186.2851805476396,\n
\"min\": 2056.12,\n        \"max\": 2456.67,\n
\"num_unique_values\": 4,\n        \"samples\": [\n          2429.96,\
n          2056.12,\n          2456.67\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"NO2\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 2.6504009130695683,\n
\"min\": 19.54,\n        \"max\": 25.36,\n
\"num_unique_values\": 4,\n        \"samples\": [\n          22.28,\n
19.54,\n          25.36\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"SO2\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.2005617112013162,\n        \"min\":
9.42,\n        \"max\": 9.89,\n        \"num_unique_values\": 4,\n
\"samples\": [\n          9.54,\n          9.42,\n          9.89\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"O3\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 5.971852727587982,\n
\"min\": 64.37,\n        \"max\": 78.68,\n
\"num_unique_values\": 4,\n        \"samples\": [\n          72.24,\n
64.37,\n          78.68\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"ISPU_PM10\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 4.440870542369649,\n
\"min\": 164.37989949748743,\n        \"max\": 175.0659296482412,\n
\"num_unique_values\": 4,\n        \"samples\": [\n
175.0659296482412,\n          164.37989949748743,\n
168.91201005025124\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ISPU_PM2.5\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 7.683251494025191,\n        \"min\":
304.22099358974356,\n        \"max\": 322.9250801282051,\n
```

\"num_unique_values\": 4,\n   \"samples\": [\n
322.9250801282051,\n   304.22099358974356,\n
315.05596955128203\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"ISPU_CO\",\n   \"properties\": {\n   \"dtype\": \"number\",\
n   \"std\": 2.328564756845494,\n   \"min\": 25.7015,\n
\"max\": 30.708375,\n   \"num_unique_values\": 4,\n
\"samples\": [\n   30.374499999999998,\n   25.7015,\n
30.708375\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"ISPU_NO2\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 1.6565005706684797,\n   \"min\":
12.2125,\n   \"max\": 15.85,\n   \"num_unique_values\": 4,\n
\"samples\": [\n   13.925,\n   12.2125,\n
15.85\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"ISPU_SO2\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 0.1928477992320343,\n   \"min\":
9.057692307692308,\n   \"max\": 9.509615384615385,\n
\"num_unique_values\": 4,\n   \"samples\": [\n
9.173076923076923,\n   9.057692307692308,\n
9.509615384615385\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"ISPU_O3\",\n   \"properties\": {\n   \"dtype\": \"number\",\
n   \"std\": 2.4882719698283275,\n   \"min\":
26.820833333333333,\n   \"max\": 32.78333333333334,\n
\"num_unique_values\": 4,\n   \"samples\": [\n
30.099999999999998,\n   26.820833333333333,\n
32.78333333333334\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Nilai ISPU\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 7.683251494025191,\n   \"min\":
304.22099358974356,\n   \"max\": 322.9250801282051,\n
\"num_unique_values\": 4,\n   \"samples\": [\n
322.9250801282051,\n   304.22099358974356,\n
315.05596955128203\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Kategori\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 1,\n   \"samples\":
[\n   \"Berbahaya\"\n   ],\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"filtered_df"}

```
# Backup kolom ISPU ke DataFrame baru
df_ispu = df[['Waktu', 'ISPU_PM2.5', 'ISPU_PM10', 'ISPU_CO',
'ISPU_NO2', 'ISPU_SO2', 'ISPU_O3', 'Nilai ISPU', 'Kategori']].copy()

# Hapus kolom ISPU dari DataFrame utama
df = df.drop(columns=['ISPU_PM2.5', 'ISPU_PM10', 'ISPU_CO',
'ISPU_NO2', 'ISPU_SO2', 'ISPU_O3'])
```

```
# Tampilkan DataFrame baru yang berisi kolom ISPU
print(df_ispu)
```

```
                  Waktu  ISPU_PM2.5  ISPU_PM10    ISPU_CO  ISPU_NO2
\
0     2024-01-01 00:00:00  116.819158  69.011212  18.358250  13.38750

1     2024-01-01 01:00:00  116.610737  68.100505  17.190000   9.74375

2     2024-01-01 02:00:00  111.546105  64.309192  15.187250   6.21250

3     2024-01-01 03:00:00   99.201754  57.073030  13.184500   4.60625

4     2024-01-01 04:00:00   88.763158  51.593939  11.849375   3.90625

...                   ...         ...        ...        ...       ...

8779  2024-12-31 19:00:00  101.260526  64.497273  30.708375  29.56250

8780  2024-12-31 20:00:00  115.964632  74.366566  33.378625  28.70625

8781  2024-12-31 21:00:00  133.201053  85.314848  36.716500  29.13125

8782  2024-12-31 22:00:00  140.787579  90.531616  37.050250  26.35000

8783  2024-12-31 23:00:00  132.523684  86.146364  32.377250  20.56250


       ISPU_SO2   ISPU_O3  Nilai ISPU     Kategori
0      6.701923  0.125000  116.819158  Tidak Sehat
1      4.355769  0.050000  116.610737  Tidak Sehat
2      2.038462  0.150000  111.546105  Tidak Sehat
3      1.278846  0.270833   99.201754       Sedang
4      1.115385  0.304167   88.763158       Sedang
...         ...       ...         ...          ...
8779  20.634615  0.004167  101.260526  Tidak Sehat
8780  22.471154  0.000000  115.964632  Tidak Sehat
8781  24.528846  0.000000  133.201053  Tidak Sehat
8782  24.298077  0.000000  140.787579  Tidak Sehat
8783  20.173077  0.000000  132.523684  Tidak Sehat

[8784 rows x 9 columns]
```

```
# Tentukan kolom ISPU polutan
ispu_columns = ['ISPU_PM10', 'ISPU_PM2.5', 'ISPU_CO', 'ISPU_NO2',
'ISPU_SO2', 'ISPU_O3']

# Buat kolom baru: dari ISPU mana nilai maksimum berasal
df_ispu['ISPU_Dominan'] = df_ispu[ispu_columns].idxmax(axis=1)
df_ispu.head()
```

```
{"summary":"{\n  \"name\": \"df_ispu\",\n  \"rows\": 8784,\n
\"fields\": [\n    {\n      \"column\": \"Waktu\",\n
\"properties\": {\n        \"dtype\": \"object\",\n
\"num_unique_values\": 8784,\n        \"samples\": [\n
\"2024-09-23 01:00:00\",\n          \"2024-03-03 00:00:00\",\n
\"2024-09-29 11:00:00\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"ISPU_PM2.5\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 46.14346732590953,\n
\"min\": 1.9032258064516128,\n        \"max\": 322.9250801282051,\n
\"num_unique_values\": 5904,\n        \"samples\": [\n
103.78242105263158,\n          87.03157894736842,\n
75.80701754385966\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ISPU_PM10\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 27.651930518963916,\n        \"min\":
1.02,\n        \"max\": 175.0659296482412,\n
\"num_unique_values\": 6293,\n        \"samples\": [\n
69.60515151515152,\n          57.26111111111111,\n
82.24121212121213\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ISPU_CO\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 8.702786192659095,\n        \"min\": 2.920625,\n
\"max\": 49.400375000000004,\n        \"num_unique_values\": 283,\n
\"samples\": [\n          20.861625,\n          3.2543750000000005,\n
18.191374999999997\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ISPU_NO2\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 8.668351450767455,\n        \"min\":
0.45625,\n        \"max\": 59.19933333333333,\n
\"num_unique_values\": 448,\n        \"samples\": [\n
1.4874999999999998,\n          12.2125,\n          4.01875\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ISPU_SO2\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 4.707844311586157,\n        \"min\":
0.38461538461538464,\n        \"max\": 31.634615384615383,\n
\"num_unique_values\": 397,\n        \"samples\": [\n
6.9326923076923075,\n          9.403846153846153,\n
3.6634615384615383\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ISPU_O3\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 12.571063154079857,\n        \"min\": 0.0,\n
\"max\": 82.61359649122807,\n        \"num_unique_values\": 593,\n
\"samples\": [\n          0.9583333333333333,\n
0.2791666666666667,\n          19.670833333333334\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Nilai ISPU\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
45.711457767543735,\n        \"min\": 6.033333333333333,\n
```

\"max\": 322.9250801282051,\n   \"num_unique_values\": 5849,\n
\"samples\": [\n   119.24726315789474,\n
130.32484210526317,\n   53.026315789473685\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Kategori\",\n   \"properties\":
{\n   \"dtype\": \"category\",\n   \"num_unique_values\":
5,\n   \"samples\": [\n   \"Sedang\",\n
\"Berbahaya\",\n   \"Baik\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"ISPU_Dominan\",\n
\"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n   \"samples\": [\n
\"ISPU_O3\",\n   \"ISPU_PM2.5\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe","variable_name":"df_ispu"}

```
print(df_ispu['ISPU_Dominan'].value_counts())
```

```
ISPU_Dominan
ISPU_PM2.5    8358
ISPU_O3        426
Name: count, dtype: int64
```

```
# Replace 'Kategori_Anda' with the actual category you want to display
kate = 'ISPU_O3'
filtered_dfo3 = df_ispu[df_ispu['ISPU_Dominan'] == kate]

# Display the filtered DataFrame
filtered_dfo3
```

{"repr_error":"0","type":"dataframe","variable_name":"filtered_dfo3"}

```
# Tampilkan kategori ISPU hanya dari data dominan O3
df_ispu[df_ispu['ISPU_Dominan'] == 'ISPU_O3']
['Kategori'].value_counts()
```

```
Kategori
Baik    426
Name: count, dtype: int64
```

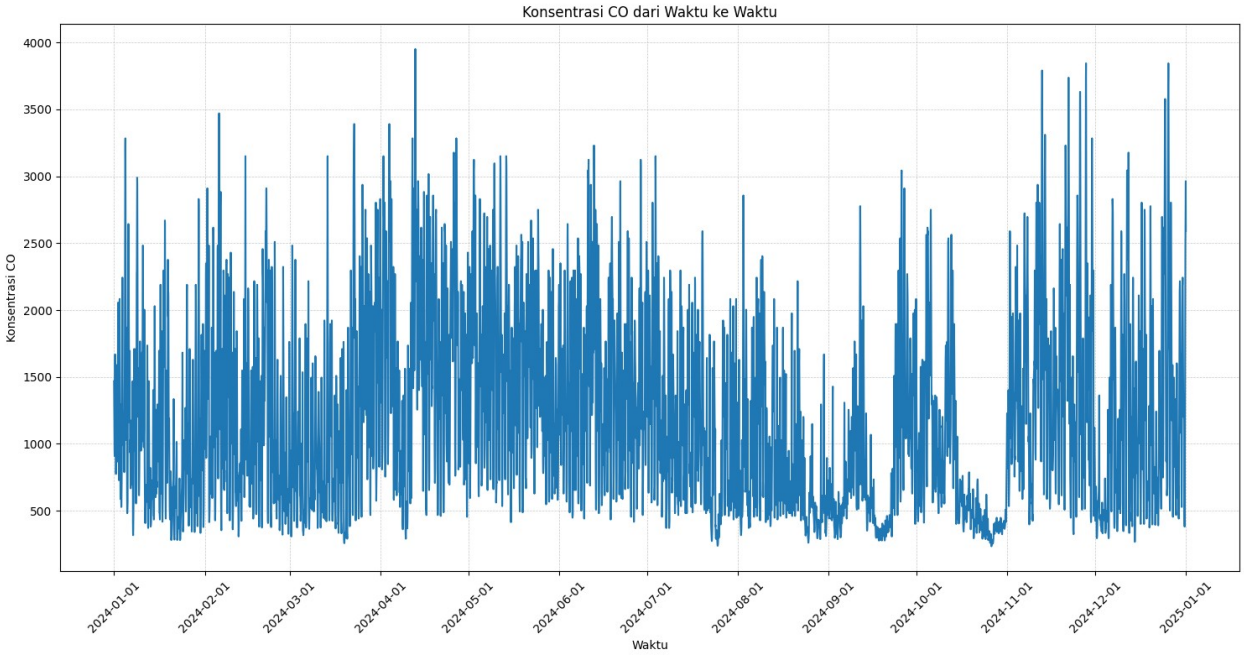
```
df
```

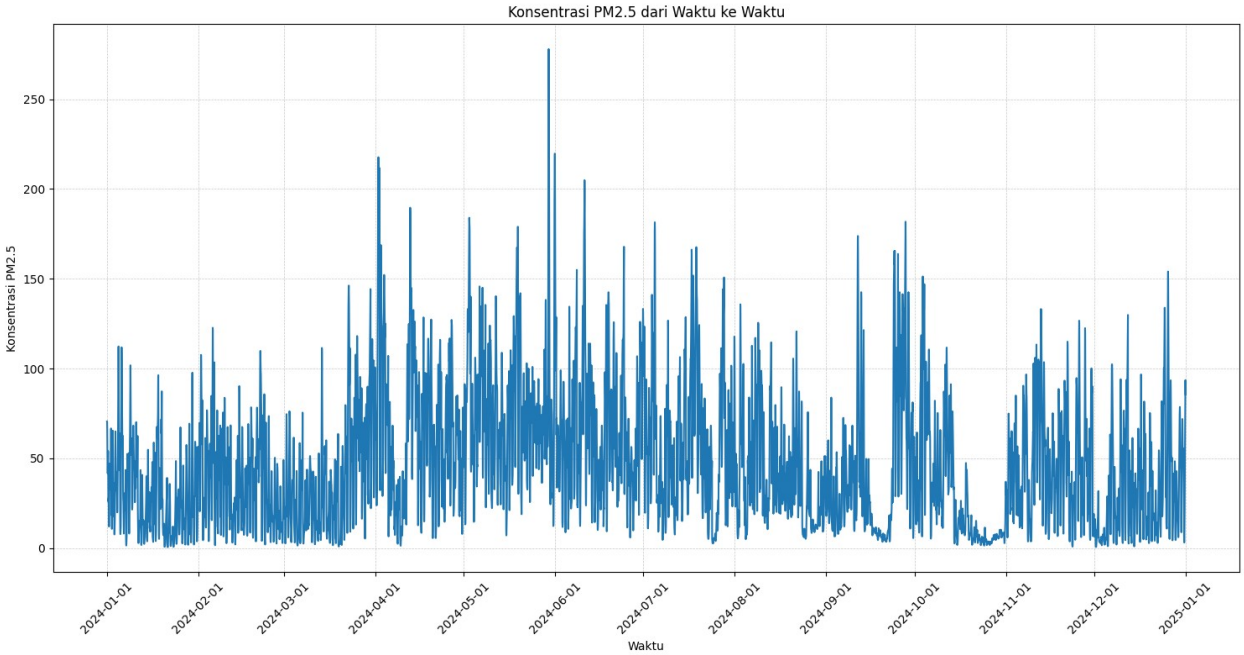
{"summary":"{\n \"name\": \"df\",\n \"rows\": 8784,\n \"fields\":
[\n   {\n   \"column\": \"Waktu\",\n   \"properties\": {\n
\"dtype\": \"date\",\n   \"min\": \"2024-01-01 00:00:00\",\n
\"max\": \"2024-12-31 23:00:00\",\n   \"num_unique_values\":
8784,\n   \"samples\": [\n   \"2024-09-23 01:00:00\",\n
\"2024-03-03 00:00:00\",\n   \"2024-09-29 11:00:00\"\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"PM10\",\n   \"properties\": {\n   \"dtype\": \"number\",\n

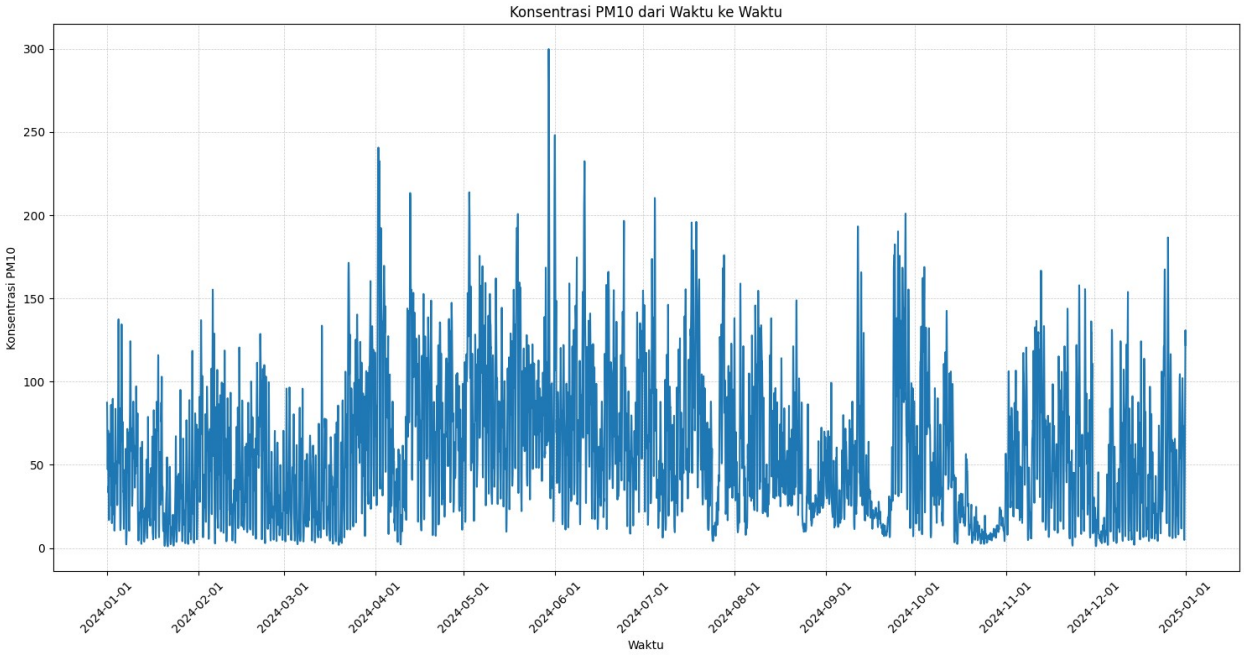
\"std\": 41.910576712695416,\n   \"min\": 1.02,\n   \"max\":
299.88,\n   \"num_unique_values\": 6293,\n   \"samples\": [\
n   88.59,\n   63.65,\n   114.12\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"PM2.5\",\n   \"properties\": {\
n   \"dtype\": \"number\",\n   \"std\": 36.519139730560795,\
n   \"min\": 0.59,\n   \"max\": 277.9,\n
\"num_unique_values\": 5904,\n   \"samples\": [\n
58.07,\n   44.84,\n   35.7\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"CO\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 696.2228954127207,\n
\"min\": 233.65,\n   \"max\": 3952.03,\n
\"num_unique_values\": 283,\n   \"samples\": [\n
1668.93,\n   260.35,\n   1455.31\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"NO2\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 13.883107581917832,\n
\"min\": 0.73,\n   \"max\": 100.08,\n
\"num_unique_values\": 448,\n   \"samples\": [\n   2.38,\n
19.54,\n   6.43\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"SO2\",\n   \"properties\": {\n   \"dtype\": \"number\",\n
\"std\": 4.896158084049618,\n   \"min\": 0.4,\n   \"max\":
32.9,\n   \"num_unique_values\": 397,\n   \"samples\": [\n
7.21,\n   9.78,\n   3.81\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"O3\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 30.085010993395432,\n
\"min\": 0.0,\n   \"max\": 194.55,\n
\"num_unique_values\": 593,\n   \"samples\": [\n   2.3,\n
0.67,\n   47.21\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Nilai ISPU\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 45.711457767543735,\n   \"min\":
6.033333333333333,\n   \"max\": 322.9250801282051,\n
\"num_unique_values\": 5849,\n   \"samples\": [\n
119.24726315789474,\n   130.32484210526317,\n
53.026315789473685\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Kategori\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 5,\n   \"samples\":
[\n   \"Sedang\",\n   \"Berbahaya\",\n
\"Baik\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"df"}

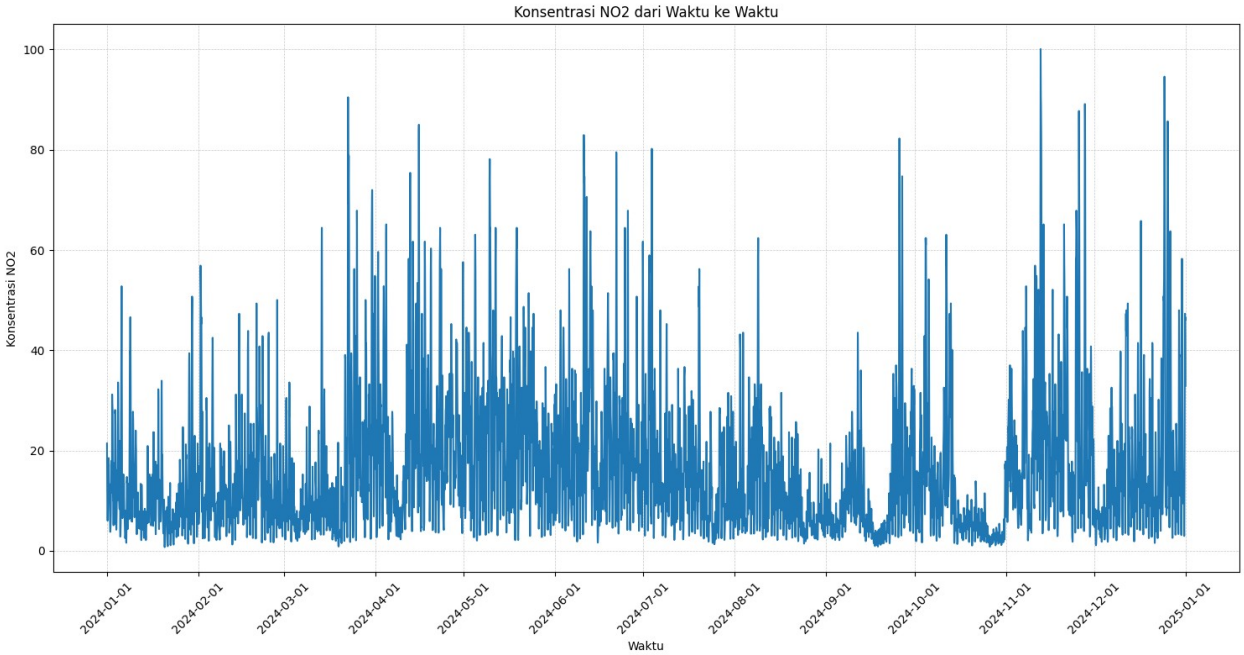
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

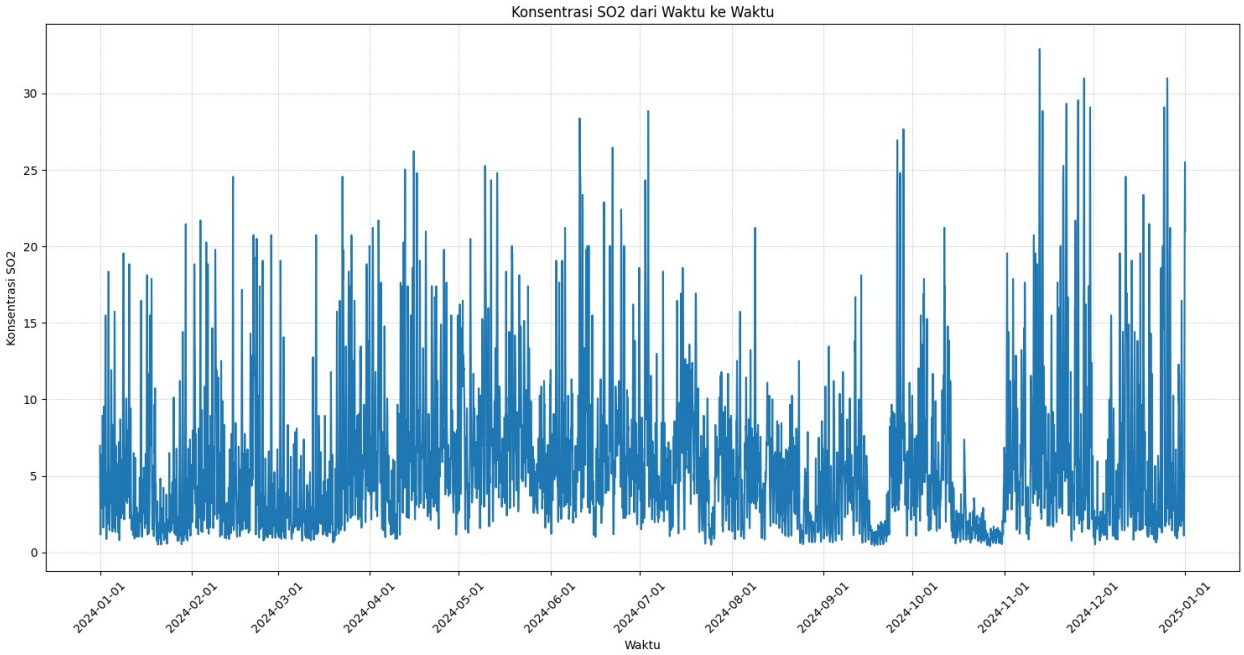
```
import matplotlib.dates as mdates

# Pastikan kolom Waktu dalam format datetime
df['Waktu'] = pd.to_datetime(df['Waktu'])

# Tentukan semua kolom pencemar udara (selain waktu dan ISPU jika ada)
columns_to_plot = ['CO', 'PM2.5', 'PM10', 'NO2', 'SO2', 'O3', 'Nilai
ISPU']

# Loop untuk plot setiap kolom
for col in columns_to_plot:
    plt.figure(figsize=(15, 8))
    plt.plot(df['Waktu'], df[col], label=col)

    # Format sumbu x untuk menampilkan tahun-bulan-tanggal
    plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-
%d'))
    plt.gca().xaxis.set_major_locator(mdates.MonthLocator())
    plt.xticks(rotation=45)

    # Set label dan judul
    plt.xlabel('Waktu')
    plt.ylabel(f'Konsentrasi {col}')
    plt.title(f'Konsentrasi {col} dari Waktu ke Waktu')

    # Tambahkan garis grid XY
    plt.grid(True, which='both', linestyle='--', linewidth=0.5,
alpha=0.7)
    #plt.legend()
    plt.tight_layout()
    plt.show()
```

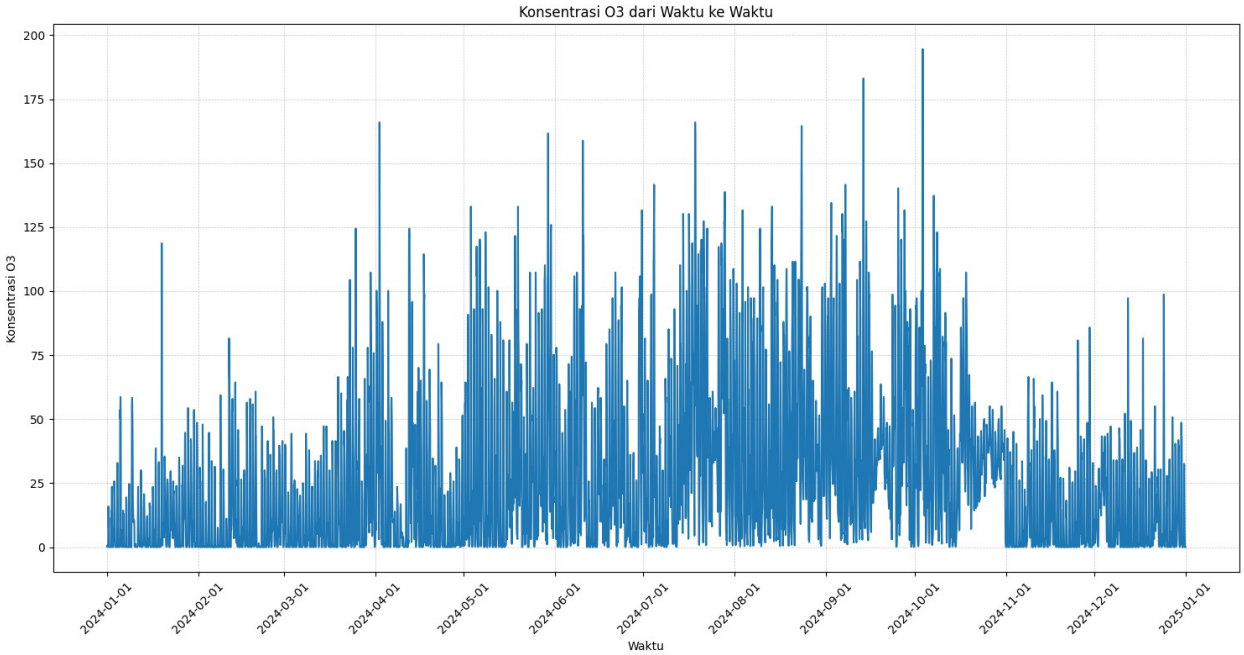

Konsentrasi CO dari Waktu ke Waktu
Konsentrasi CO
4000
3500
3000
2500
2000
1500
1000
500
2024-01-01
2024-02-01
2024-03-01
2024-04-01
2024-05-01
2024-06-01
2024-07-01
2024-08-01
2024-09-01
2024-10-01
2024-11-01
2024-12-01
2025-01-01
Waktu

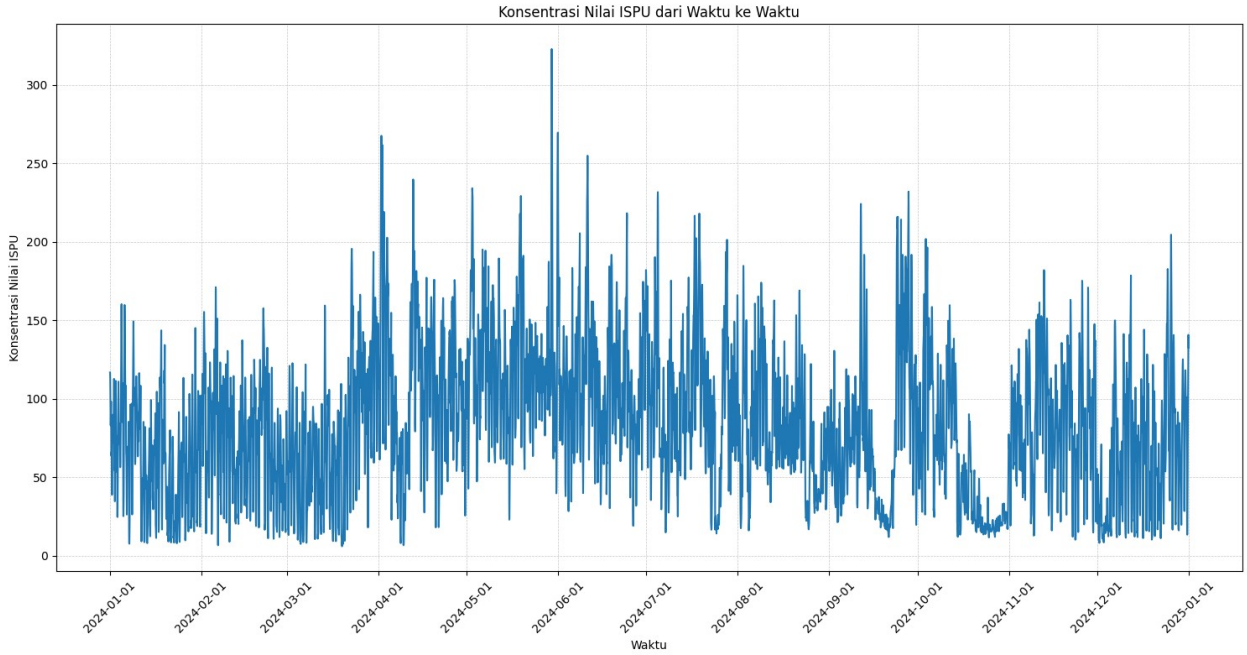

Konsentrasi PM2.5 dari Waktu ke Waktu
Konsentrasi PM2.5
250
200
150
100
50
0
2024-01-01
2024-02-01
2024-03-01
2024-04-01
2024-05-01
2024-06-01
2024-07-01
2024-08-01
2024-09-01
2024-10-01
2024-11-01
2024-12-01
2025-01-01
Waktu


Konsentrasi PM10 dari Waktu ke Waktu
Konsentrasi PM10
300
250
200
150
100
50
0
2024-01-01
2024-02-01
2024-03-01
2024-04-01
2024-05-01
2024-06-01
2024-07-01
2024-08-01
2024-09-01
2024-10-01
2024-11-01
2024-12-01
2025-01-01
Waktu


Konsentrasi NO2 dari Waktu ke Waktu
Konsentrasi NO2
100
80
60
40
20
0
2024-01-01
2024-02-01
2024-03-01
2024-04-01
2024-05-01
2024-06-01
2024-07-01
2024-08-01
2024-09-01
2024-10-01
2024-11-01
2024-12-01
2025-01-01
Waktu


Konsentrasi SO2 dari Waktu ke Waktu
Konsentrasi SO2
0
5
10
15
20
25
30
2024-01-01
2024-02-01
2024-03-01
2024-04-01
2024-05-01
2024-06-01
2024-07-01
2024-08-01
2024-09-01
2024-10-01
2024-11-01
2024-12-01
2025-01-01
Waktu


Konsentrasi O3 dari Waktu ke Waktu
Konsentrasi O3
0
25
50
75
100
125
150
175
200
2024-01-01
2024-02-01
2024-03-01
2024-04-01
2024-05-01
2024-06-01
2024-07-01
2024-08-01
2024-09-01
2024-10-01
2024-11-01
2024-12-01
2025-01-01
Waktu




```
df.columns
```

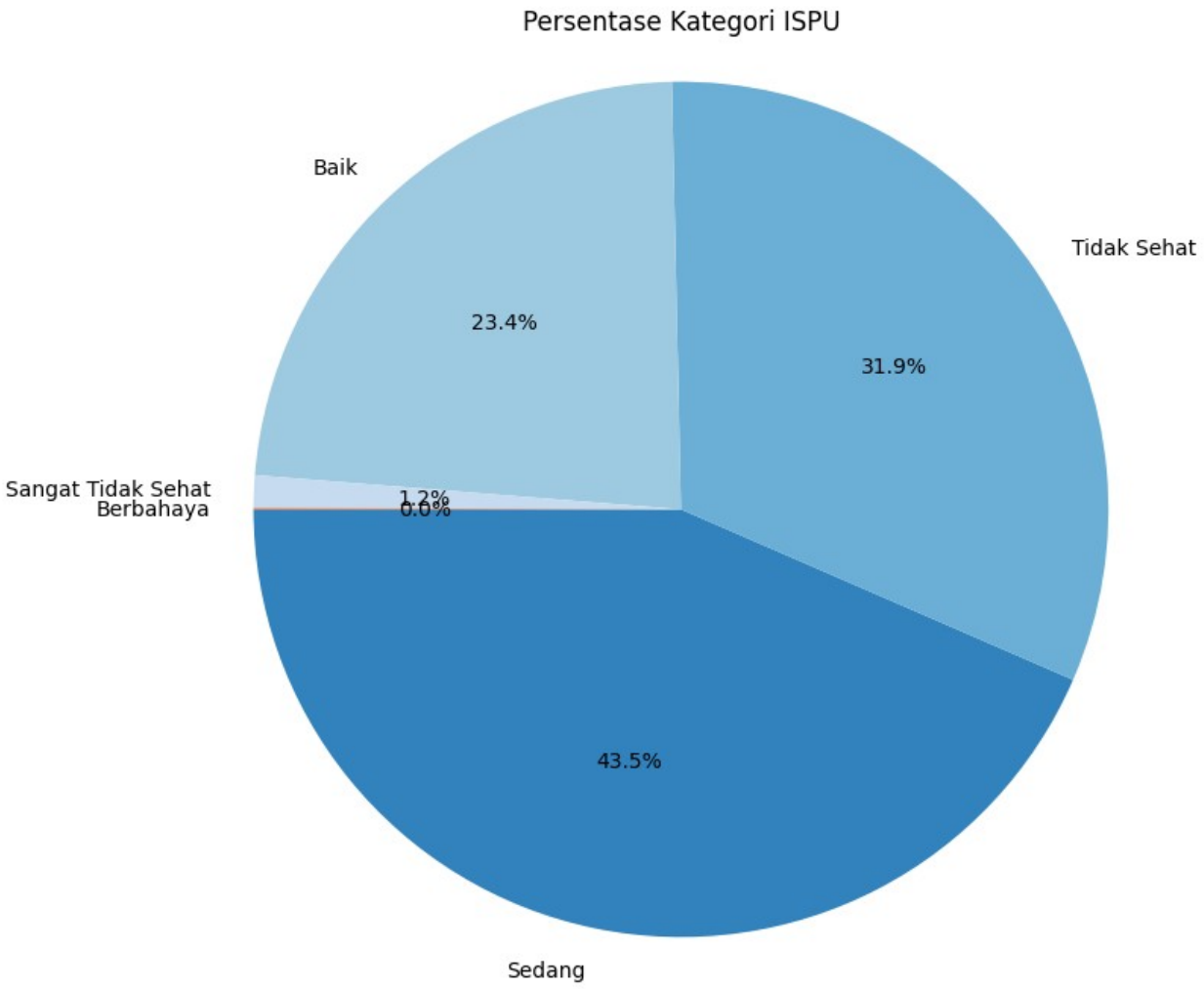
```
Index(['Waktu', 'PM10', 'PM2.5', 'CO', 'NO2', 'SO2', 'O3', 'Nilai
ISPU',
       'Kategori'],
      dtype='object')
```

```
# Melihat nilai unik dan total masing-masing
unique_values = df['Kategori'].value_counts()
print(f"Jumlah nilai unik di kolom 'Kategori': {unique_values}")
```

```
Jumlah nilai unik di kolom 'Kategori': Kategori
Sedang                3819
Tidak Sehat           2799
Baik                  2054
Sangat Tidak Sehat     108
Berbahaya                4
Name: count, dtype: int64
```

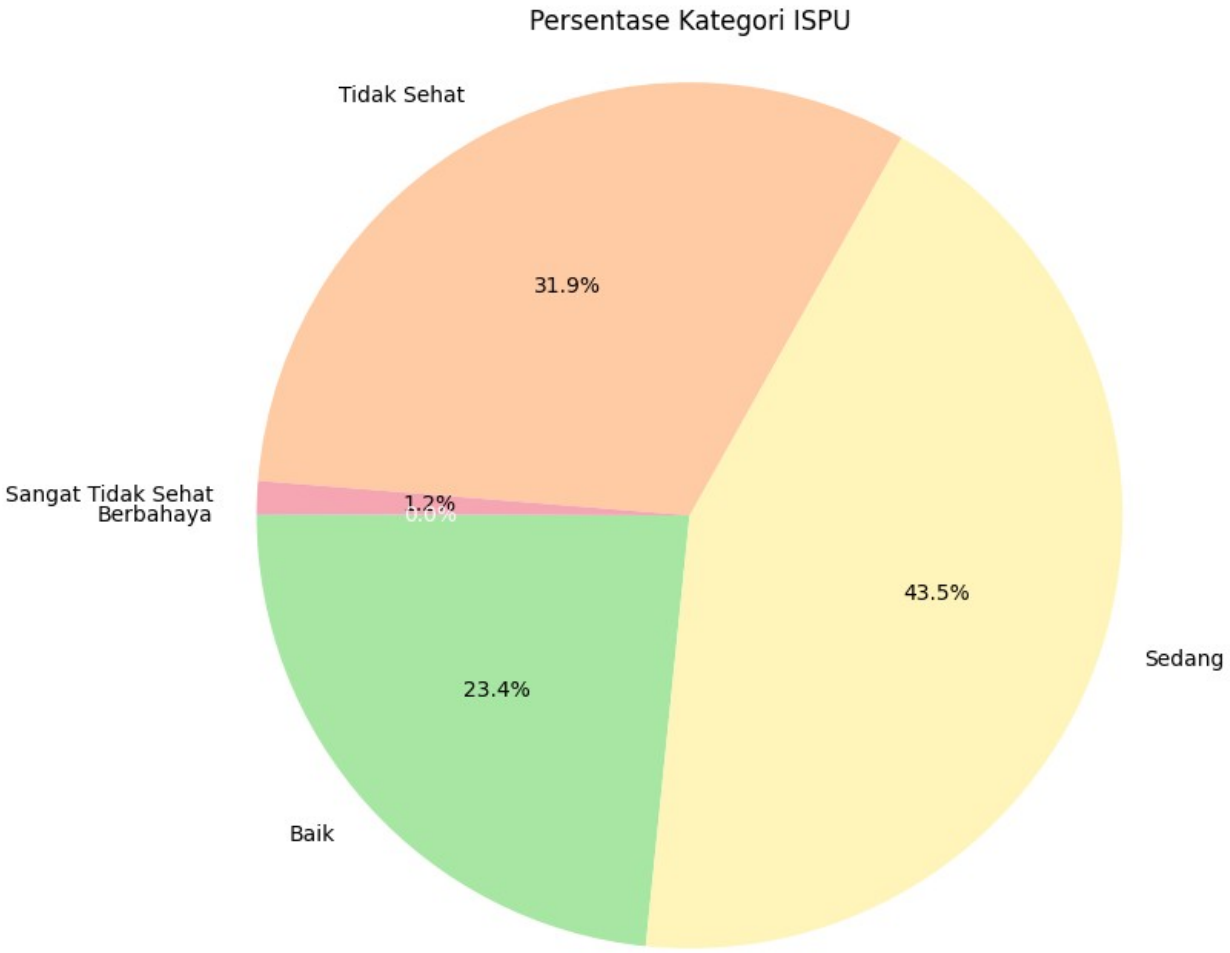
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Misalkan kolom kategori adalah 'Kategori'
# Menghitung distribusi kategori
kategori_counts = df['Kategori'].value_counts()

# Menentukan sudut start angle, misalnya 90 derajat
start_angle = 180  # Anda bisa mengganti nilai ini untuk menyesuaikan
```

```
# Membuat pie chart
plt.figure(figsize=(8, 8))
plt.pie(kategori_counts, labels=kategori_counts.index, autopct='%1.1f%
%', startangle=start_angle, colors=sns.color_palette('tab20c',
len(kategori_counts)))
plt.title('Persentase Kategori ISPU')
plt.axis('equal')  # Menjaga agar pie chart berbentuk bulat
plt.show()
```



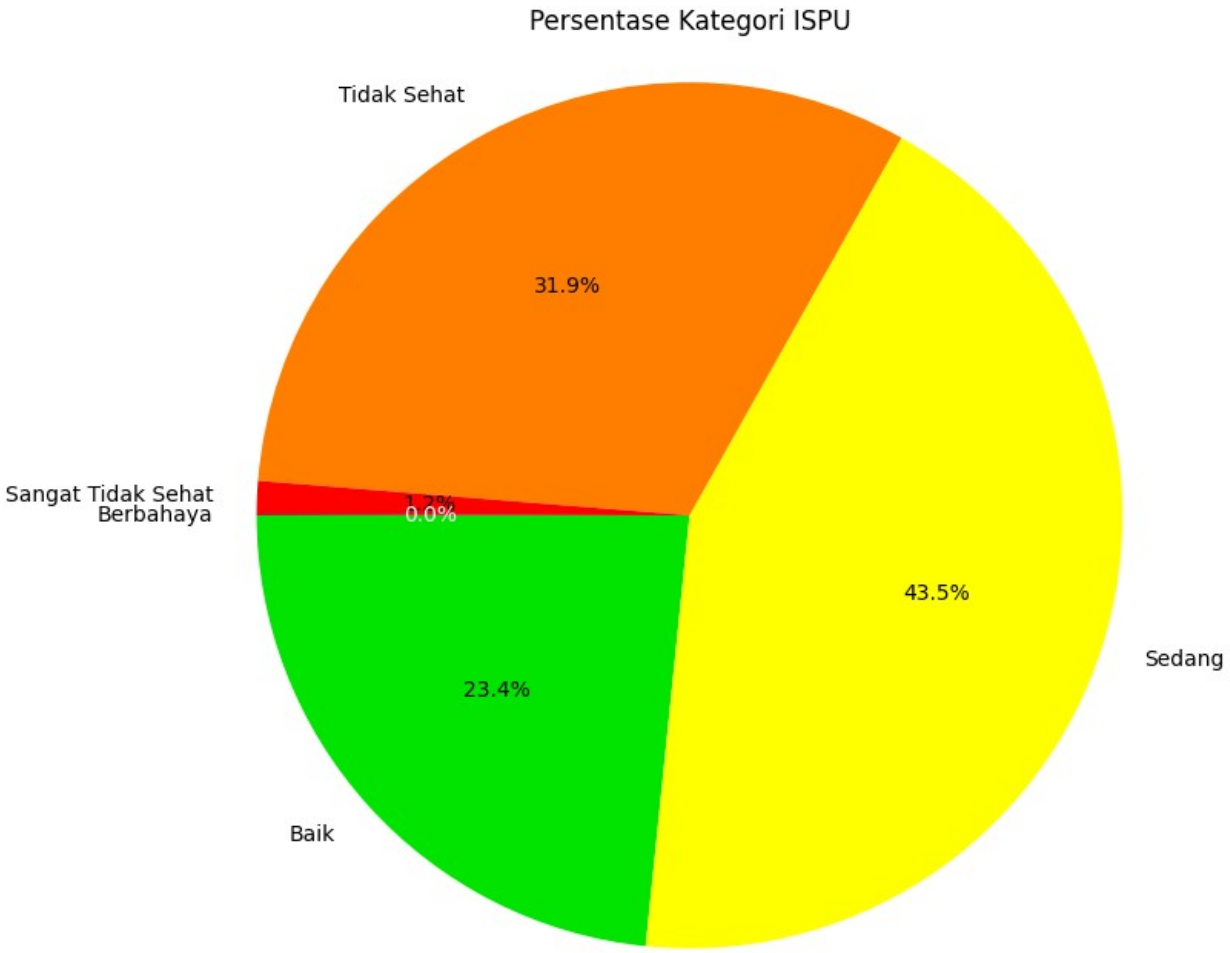

```
import pandas as pd
import matplotlib.pyplot as plt

# Misalnya kolom kategori bernama 'Kategori'
kategori_counts = df['Kategori'].value_counts()

# Urutkan agar warnanya konsisten sesuai urutan kategori ISPU
ordered_labels = ['Baik', 'Sedang', 'Tidak Sehat', 'Sangat Tidak
Sehat', 'Berbahaya']
```

```
kategori_counts = kategori_counts.reindex([k for k in ordered_labels
if k in kategori_counts.index])

# Warna pastel sesuai kategori ISPU
pastel_colors = {
    'Baik': '#a8e6a3',             # pastel green
    'Sedang': '#fff5ba',           # pastel yellow
    'Tidak Sehat': '#ffcba4',      # pastel orange
    'Sangat Tidak Sehat': '#f6a6b2',# pastel red
    'Berbahaya': '#c0c0c0'         # light gray instead of pure black
}

# Ambil warna berdasarkan kategori yang ada di pie
colors = [pastel_colors[kat] for kat in kategori_counts.index]

# Plot pie chart
plt.figure(figsize=(8, 8))
wedges, texts, autotexts = plt.pie(
    kategori_counts,
    labels=kategori_counts.index,
    autopct='%1.1f%%',
    startangle=180,
    colors=colors,
    textprops={'color': 'black'}  # pastikan semua teks jelas
)

# Untuk segmen "Berbahaya", ubah teks persennya jadi putih agar
kontras
for i, label in enumerate(kategori_counts.index):
    if label == 'Berbahaya':
        autotexts[i].set_color('white')

plt.title('Persentase Kategori ISPU')
plt.axis('equal')
plt.show()
```




```
import pandas as pd
import matplotlib.pyplot as plt

# Misalnya kolom kategori bernama 'Kategori'
kategori_counts = df['Kategori'].value_counts()

# Urutkan agar warnanya konsisten sesuai urutan kategori ISPU
ordered_labels = ['Baik', 'Sedang', 'Tidak Sehat', 'Sangat Tidak
Sehat', 'Berbahaya']
kategori_counts = kategori_counts.reindex([k for k in ordered_labels
if k in kategori_counts.index])

# Warna ISPU standar (lebih tegas, tidak pastel)
ispu_colors = {
    'Baik': '#00e400',         # Hijau
    'Sedang': '#ffff00',       # Kuning
    'Tidak Sehat': '#ff7e00',  # Oranye
    'Sangat Tidak Sehat': '#ff0000', # Merah
    'Berbahaya': '#7e0023'     # Hitam keunguan gelap
```

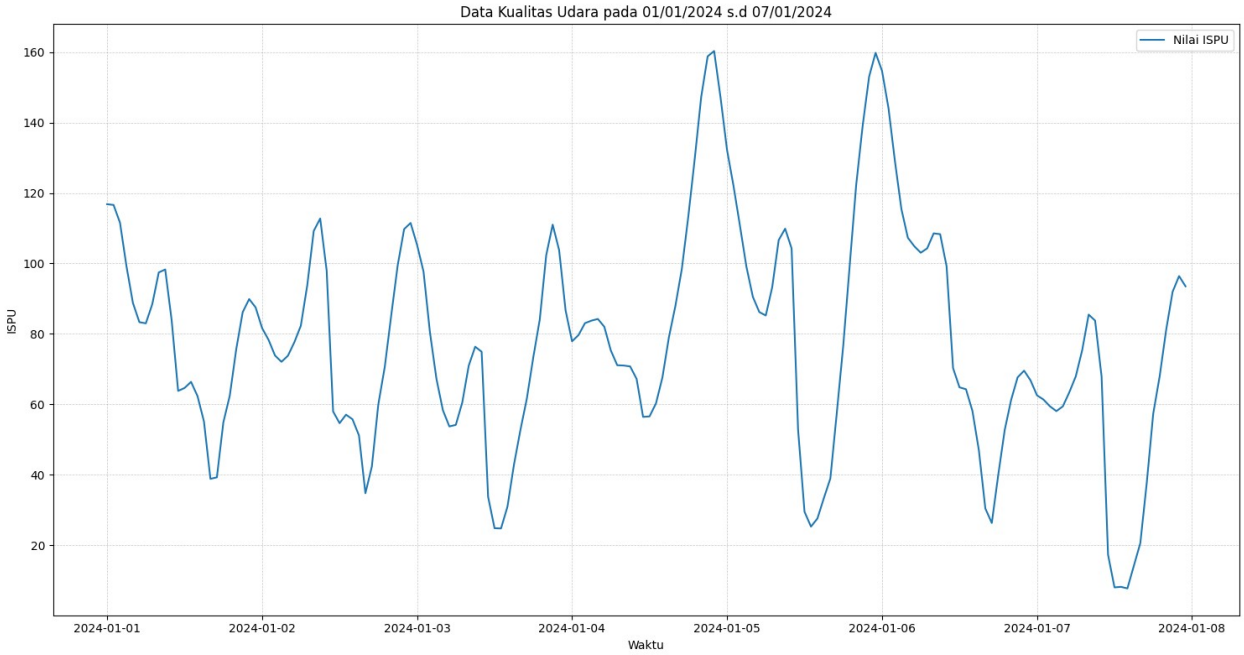
```
}

# Ambil warna berdasarkan kategori yang ada di pie
colors = [ispu_colors[kat] for kat in kategori_counts.index]

# Plot pie chart
plt.figure(figsize=(8, 8))
wedges, texts, autotexts = plt.pie(
    kategori_counts,
    labels=kategori_counts.index,
    autopct='%1.1f%%',
    startangle=180,
    colors=colors,
    textprops={'color': 'black'}
)

# Bikin teks persentase pada warna gelap (misal 'Berbahaya') jadi
putih
for i, label in enumerate(kategori_counts.index):
    if label == 'Berbahaya' or ispu_colors[label] == '#7e0023':
        autotexts[i].set_color('white')

plt.title('Persentase Kategori ISPU')
plt.axis('equal')
plt.show()
```




```
# Replace 'Kategori_Anda' with the actual category you want to display
kategorii = 'Berbahaya'
filtered_df = df[df['Kategori'] == kategorii]

# Display the filtered DataFrame
filtered_df
```

```
                    Waktu    PM10   PM2.5       CO    NO2   SO2     O3
\
3596 2024-05-29 20:00:00  287.51  268.03  2456.67  25.36  9.89  78.68

3597 2024-05-29 21:00:00  299.88  277.90  2429.96  22.28  9.54  72.24

3598 2024-05-29 22:00:00  291.56  268.01  2243.04  20.05  9.66  69.38

3599 2024-05-29 23:00:00  278.40  254.44  2056.12  19.54  9.42  64.37


      Nilai ISPU   Kategori
3596  315.055970  Berbahaya
```

```
3597  322.925080  Berbahaya
3598  315.040024  Berbahaya
3599  304.220994  Berbahaya

df.to_csv('05-data_ISPU_2024.csv', index=False)

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Konversi kolom Waktu menjadi datetime jika belum
df['Waktu'] = pd.to_datetime(df['Waktu'])

# Tentukan rentang waktu yang diinginkan
start_time = '2024-01-1 00:00:00'
end_time = '2024-01-7 23:00:00'

# Filter data berdasarkan rentang waktu
df_filtered = df[(df['Waktu'] >= start_time) & (df['Waktu'] <=
end_time)]

# Tentukan kolom yang ingin diplot
columns_to_plot = ['Nilai ISPU']

# Plot data
plt.figure(figsize=(15, 8))

for col in columns_to_plot:
    plt.plot(df_filtered['Waktu'], df_filtered[col], label=col)

# Tambahkan grid (kotak-kotak latar)
plt.grid(True, linestyle='--', alpha=0.6)

# Set labels dan title
plt.xlabel('Waktu')
plt.ylabel('ISPU')
plt.title('Data Kualitas Udara pada 01/01/2024 s.d 07/01/2024')
plt.legend(loc='upper right')

# Tambahkan garis grid XY
plt.grid(True, which='both', linestyle='--', linewidth=0.5, alpha=0.7)

# Tampilkan plot
plt.tight_layout()
plt.show()
```



data clean

```
import pandas as pd


df = pd.read_csv('/kaggle/working/05-data_ISPU_2024.csv')
df.head()
```

```
                  Waktu   PM10  PM2.5       CO    NO2   SO2    O3
Nilai ISPU  \
0  2024-01-01 00:00:00  87.39  70.58  1468.66  21.42  6.97  0.30
116.819158
1  2024-01-01 01:00:00  85.55  70.38  1375.20  15.59  4.53  0.12
116.610737
2  2024-01-01 02:00:00  77.89  65.52  1214.98   9.94  2.12  0.36
111.546105
3  2024-01-01 03:00:00  63.27  54.75  1054.76   7.37  1.33  0.65
99.201754
4  2024-01-01 04:00:00  52.20  46.25   947.95   6.25  1.16  0.73
88.763158

      Kategori
0  Tidak Sehat
1  Tidak Sehat
2  Tidak Sehat
3       Sedang
4       Sedang
```

```
datas = df.drop(columns=['Waktu','Kategori'])
datas.head()
```

```
    PM10  PM2.5       CO    NO2   SO2    O3  Nilai ISPU
0  87.39  70.58  1468.66  21.42  6.97  0.30  116.819158
1  85.55  70.38  1375.20  15.59  4.53  0.12  116.610737
2  77.89  65.52  1214.98   9.94  2.12  0.36  111.546105
3  63.27  54.75  1054.76   7.37  1.33  0.65   99.201754
4  52.20  46.25   947.95   6.25  1.16  0.73   88.763158
```

```
datas.describe()
```

```
              PM10        PM2.5           CO          NO2           SO2
\
count  8784.000000  8784.000000  8784.000000  8784.000000  8784.000000

mean     57.045741    45.976274  1179.376776    15.783973     5.773337

std      41.910577    36.519140   696.222895    13.883108     4.896158

min       1.020000     0.590000   233.650000     0.730000     0.400000

25%      23.205000    16.717500   594.140000     6.000000     2.210000

50%      48.105000    37.900000  1014.710000    11.140000     4.410000

75%      82.112500    66.335000  1628.880000    21.420000     7.510000

max     299.880000   277.900000  3952.030000   100.080000    32.900000


                O3   Nilai ISPU
count  8784.000000  8784.000000
mean     25.011085    83.287904
std      30.085011    45.711458
min       0.000000     6.033333
25%       1.080000    52.495175
50%      13.590000    78.508772
75%      38.270000   112.395421
max     194.550000   322.925080
```

```
datas.to_csv('06-data_fix_2024.csv', index=False)
```

Model hyperFix

yg terbaru norm

```
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
```

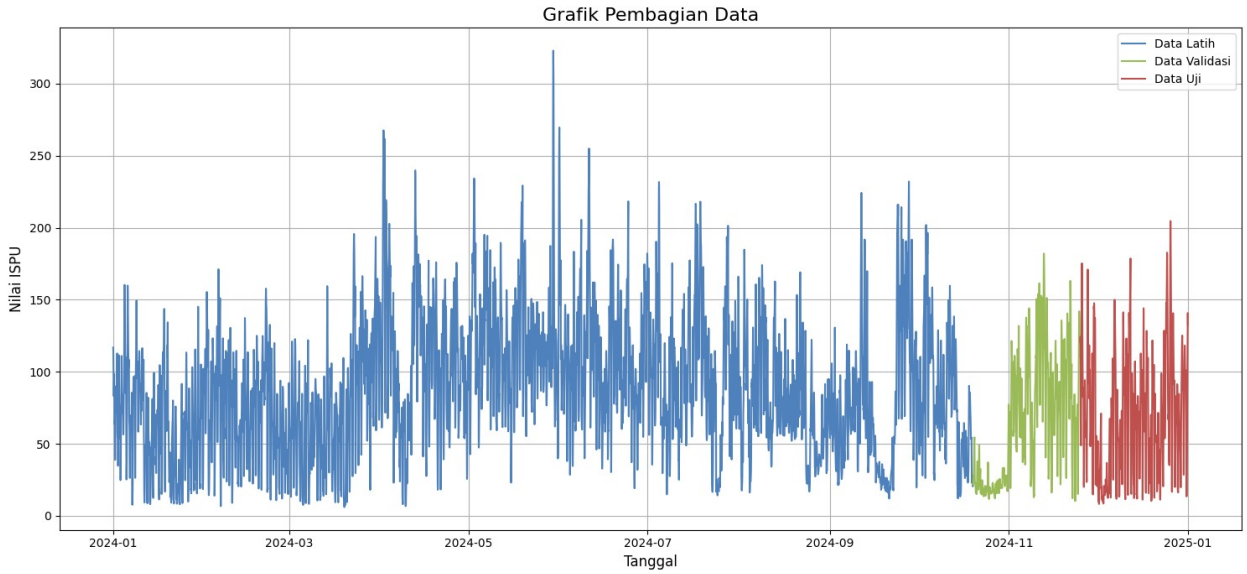
```
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import GRU, Dense, Input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras import backend as K
from tensorflow.keras.losses import MeanSquaredError
from math import sqrt
import os

# ==== Custom Metric ====
# This definition is crucial for both training and loading the model
def root_mean_squared_error(y_true, y_pred):
    return K.sqrt(K.mean(K.square(y_pred - y_true)))

# ==== Load & Preprocess Data ====
data = pd.read_csv('/kaggle/input/datasetwib/05-data_ISPU_2024.csv')
data['Waktu'] = pd.to_datetime(data['Waktu'])
data = data.drop(columns=['Kategori'])

# Define ALL columns as target columns for prediction (assuming all
except 'Waktu' are targets)
# For 7 inputs and 7 outputs, the target columns are the same as input
features
target_columns = data.drop(columns=['Waktu']).columns.tolist()

# Ensure all target columns exist in the dataframe
for col in target_columns:
    if col not in data.columns:
        raise ValueError(f"Target column '{col}' not found in the
dataset. Please check the column names.")

timestamps = data['Waktu'].values
# Separate input features and the target features
# When inputs = outputs, input_features_df and target_feature_df will
be the same
input_features_df = data.drop(columns=['Waktu'])
target_feature_df = data.drop(columns=['Waktu']) # Target is now all 7
features

input_values = input_features_df.values
target_values = target_feature_df.values
feature_names = input_features_df.columns.tolist() # This is also the
target feature names

# ==== Split Data (menggunakan n_train, n_val, n_test seperti yang
diminta) ====
n_total = len(data) # Menggunakan panjang data asli untuk pembagian
n_train = int(n_total * 0.80)
n_val = int(n_total * 0.10)
n_test = n_total - n_train - n_val # Sisa data untuk test
```

```
# Splitting input features
train_input_data = input_values[:n_train]
val_input_data = input_values[n_train : n_train + n_val]
test_input_data = input_values[n_train + n_val : ]

# Splitting target features
train_target_data = target_values[:n_train]
val_target_data = target_values[n_train : n_train + n_val]
test_target_data = target_values[n_train + n_val : ]

train_time = timestamps[:n_train]
val_time = timestamps[n_train : n_train + n_val]
test_time = timestamps[n_train + n_val : ]

# ==== Plot Data Division ====
print("\n Grafik Pembagian Data ")
plt.figure(figsize=(15, 7))
plt.plot(data['Waktu'].iloc[:n_train], data['Nilai
ISPU'].iloc[:n_train], '#4F81BD', label='Data Latih')
# Untuk set validasi dan uji, plot dari titik terakhir set sebelumnya
untuk memastikan kontinuitas
plt.plot(data['Waktu'].iloc[n_train-1 : n_train + n_val], data['Nilai
ISPU'].iloc[n_train-1 : n_train + n_val], '#9BBB59', label='Data
Validasi')
plt.plot(data['Waktu'].iloc[n_train + n_val - 1 : ], data['Nilai
ISPU'].iloc[n_train + n_val - 1 : ], '#C0504D', label='Data Uji')

plt.title('Grafik Pembagian Data', fontsize=16)
plt.xlabel('Tanggal', fontsize=12)
plt.ylabel('Nilai ISPU', fontsize=12)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

print("\n Data Setelah Normalisasi")

# ==== Normalisasi (using separate scalers for input features and
target features) ====
scaler_features = MinMaxScaler() # For input features
scaler_target = MinMaxScaler()   # For target features

# Fit and transform training data
X_train_scaled = scaler_features.fit_transform(train_input_data)
y_train_scaled = scaler_target.fit_transform(train_target_data)

# Transform validation data
X_val_scaled = scaler_features.transform(val_input_data)
y_val_scaled = scaler_target.transform(val_target_data)
```

```
# Transform test data
X_test_scaled = scaler_features.transform(test_input_data)
y_test_scaled = scaler_target.transform(test_target_data)



# Optional: Create DataFrames for scaled data (for inspection)
train_scaled_df = pd.DataFrame(X_train_scaled, columns=feature_names)
# For multi-target, we can't just add 'Nilai ISPU Normalisasi' as a
single column.
# If you want to see the scaled target values, you'd inspect
y_train_scaled directly.
print("\nContoh Data Latih Setelah Normalisasi (Fitur Input):")
print(train_scaled_df.head())
print("\nContoh Data Latih Target Setelah Normalisasi (Semua Fitur
Target):")
print(pd.DataFrame(y_train_scaled, columns=target_columns).head())
```

```
2025-07-16 08:59:23.307005: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to
register cuFFT factory: Attempting to register factory for plugin
cuFFT when one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are
written to STDERR
E0000 00:00:1752656363.659092      35 cuda_dnn.cc:8310] Unable to
register cuDNN factory: Attempting to register factory for plugin
cuDNN when one has already been registered
E0000 00:00:1752656363.761681      35 cuda_blas.cc:1418] Unable to
register cuBLAS factory: Attempting to register factory for plugin
cuBLAS when one has already been registered
```

Grafik Pembagian Data




```
 Data Setelah Normalisasi

Contoh Data Latih Setelah Normalisasi (Fitur Input):
       PM10     PM2.5        CO       NO2       SO2        O3  Nilai
ISPU
0  0.288998  0.252389  0.331536  0.230529  0.229577  0.001542
0.349601
1  0.282841  0.251668  0.306379  0.165571  0.143662  0.000617
0.348944
2  0.257211  0.234142  0.263252  0.102618  0.058803  0.001850
0.332962
3  0.208292  0.195305  0.220124  0.073983  0.030986  0.003341
0.294007
4  0.171251  0.164653  0.191373  0.061504  0.025000  0.003752
0.261067

Contoh Data Latih Target Setelah Normalisasi (Semua Fitur Target):
       PM10     PM2.5        CO       NO2       SO2        O3  Nilai
ISPU
0  0.288998  0.252389  0.331536  0.230529  0.229577  0.001542
0.349601
1  0.282841  0.251668  0.306379  0.165571  0.143662  0.000617
0.348944
2  0.257211  0.234142  0.263252  0.102618  0.058803  0.001850
0.332962
3  0.208292  0.195305  0.220124  0.073983  0.030986  0.003341
0.294007
4  0.171251  0.164653  0.191373  0.061504  0.025000  0.003752
0.261067
```

```python
3# ==== Sequence Maker ====
def create_sequences(input_data, target_data, lookback):
    """
    Creates sequences for time series prediction.
    X contains the input features for 'lookback' steps.
    y contains ALL target features at the next step.
    """
    X, y = [], []
    for i in range(len(input_data) - lookback):
        X.append(input_data[i:i + lookback])
        y.append(target_data[i + lookback]) # y now appends all target
features
    return np.array(X), np.array(y)

# ==== Hyperparameters ====
lookbacks = [12, 24]
batch_sizes = [16, 32, 64]
learning_rates = [0.001, 0.0001]
fixed_gru_architecture = [128, 64, 32] # Hidden units for GRU layers
(keeping your previous values)
epochs = 200

results = []                           # menyimpan ringkasan metrik dari
SEMUA model yang dilatih.
all_histories = {}                     # menyimpan history (loss, rmse,
dll per epoch) dari SEMUA model.

best_overall_val_rmse = float('inf') # nyari model terbaik berdasarkan
Val RMSE terkecil.
best_model = None                      # menyimpan model GRU terbaik
(model Keras) berdasarkan val RMSE terkecil.
best_model_config = {}                 # menyimpan konfigurasi model
terbaik aja, seperti: lb, bs, lr
best_model_test_pred_orig = None     # menyimpan prediksi hasil model
terbaik pada data test, dalam bentuk nilai asli (sudah inverse
transform).
best_model_y_test_orig = None        # menyimpan nilai asli ground
truth (y_test), juga sudah inverse transform.
best_model_lookback = None           # Menyimpan berapa lookback yang
digunakan oleh model terbaik

# ==== Training Loop ====
model_counter = 0
for lookback in lookbacks:
    # Create sequences using the scaled input and target data
    X_train, y_train = create_sequences(X_train_scaled,
y_train_scaled, lookback)
    X_val, y_val = create_sequences(X_val_scaled, y_val_scaled,
lookback)
```

```
    for batch_size in batch_sizes:
        for lr in learning_rates:
            model_counter += 1
            model_name = f"Model GRU {model_counter} (LB={lookback},
GRU={fixed_gru_architecture}, Batch={batch_size}, LR={lr})"
            print(f'\nTraining {model_name}')

            model = Sequential()
            # Input shape: (lookback, number_of_input_features)
            model.add(Input(shape=(lookback, X_train.shape[2])))
            model.add(GRU(fixed_gru_architecture[0],
return_sequences=True)) # First GRU layer
            model.add(GRU(fixed_gru_architecture[1],
return_sequences=True))  # Second GRU layer
            model.add(GRU(fixed_gru_architecture[2]))
# Third GRU layer (no return sequences)
            model.add(Dense(len(target_columns))) # Output Dense layer
now predicts ALL target_columns

            model.compile(optimizer=Adam(learning_rate=lr),
                          loss='mse',
                          metrics=['mse', root_mean_squared_error])

            early_stop = EarlyStopping(monitor='val_loss',
patience=10, restore_best_weights=True)

            history = model.fit(X_train, y_train,
                                validation_data=(X_val, y_val), #
validation digunakan di sini
                                epochs=epochs,
                                batch_size=batch_size,
                                callbacks=[early_stop],
                                verbose=0) # Set verbose to 0 to
suppress per-epoch output during training

            all_histories[model_name] = history.history
            best_epoch_idx =
np.argmin(history.history['val_root_mean_squared_error'])
            all_histories[model_name]['best_epoch_idx'] =
best_epoch_idx
            all_histories[model_name]['best_val_rmse_keras_metric'] =
history.history['val_root_mean_squared_error'][best_epoch_idx]
            all_histories[model_name]
['best_val_mse_at_best_rmse_epoch'] = history.history['val_mse']
[best_epoch_idx]

            print(f"\nPer-Epoch MSE & RMSE for {model_name}")
            print(f"{'Epoch':^5} | {'Train MSE':^12} | {'Val MSE':^12}
| {'Train RMSE':^12} | {'Val RMSE':^12}")
            print("-" * 61)
```

```
            for epoch_num, (mse, val_mse, rmse, val_rmse) in
enumerate(zip(
                history.history['mse'],
                history.history['val_mse'],
                history.history['root_mean_squared_error'],
                history.history['val_root_mean_squared_error']
            ), 1):
                print(f"{epoch_num:5d} | {mse:12.4f} | {val_mse:12.4f}
| {rmse:12.4f} | {val_rmse:12.4f}")

            best_epoch = best_epoch_idx + 1
            best_val_rmse_keras_metric =
history.history['val_root_mean_squared_error'][best_epoch_idx]
            print(f"\nEpoch Terbaik: {best_epoch} (Val RMSE:
{best_val_rmse_keras_metric:.4f})")

            # Create test sequences for evaluation
            X_test, y_test = create_sequences(X_test_scaled,
y_test_scaled, lookback)

            # Make predictions on scaled data
            y_train_pred = model.predict(X_train)
            y_val_pred = model.predict(X_val)
            y_test_pred = model.predict(X_test)

            # Inverse transform actual and predicted values using the
target_scaler
            y_train_orig = scaler_target.inverse_transform(y_train)
            y_val_orig = scaler_target.inverse_transform(y_val)
            y_test_orig = scaler_target.inverse_transform(y_test)

            y_train_pred_orig =
scaler_target.inverse_transform(y_train_pred)
            y_val_pred_orig =
scaler_target.inverse_transform(y_val_pred)
            y_test_pred_orig =
scaler_target.inverse_transform(y_test_pred)

            # Calculate overall RMSE and MSE (these are now for ALL
target features)
            train_rmse = sqrt(mean_squared_error(y_train_orig,
y_train_pred_orig))
            val_rmse = sqrt(mean_squared_error(y_val_orig,
y_val_pred_orig))
            test_rmse = sqrt(mean_squared_error(y_test_orig,
y_test_pred_orig))

            train_mse = mean_squared_error(y_train_orig,
y_train_pred_orig)
            val_mse = mean_squared_error(y_val_orig, y_val_pred_orig)
```

```
            test_mse = mean_squared_error(y_test_orig,
y_test_pred_orig)

            print(f"\nFinal Metrics:")
            print(f"Train MSE:    {train_mse:.4f} | Train RMSE:
{train_rmse:.4f}")
            print(f"Val    MSE:    {val_mse:.4f} | Val    RMSE:
{val_rmse:.4f}")
            print(f"Test   MSE:    {test_mse:.4f} | Test   RMSE:
{test_rmse:.4f}")

            results.append({
                'Model': model_name,
                'Lookback': lookback,
                'GRU Layers': fixed_gru_architecture,
                'Batch Size': batch_size,
                'Learning Rate': lr,
                'Train RMSE': train_rmse,
                'Val RMSE': val_rmse,
                'Test RMSE': test_rmse,
                'Train MSE': train_mse,
                'Val MSE': val_mse,
                'Test MSE': test_mse,
                'Best Epoch': best_epoch
            })

            # Check if current model is the best based on validation
RMSE
            if val_rmse < best_overall_val_rmse:
                best_overall_val_rmse = val_rmse
                best_model = model
                best_model_config = {
                    'Lookback': lookback,
                    'GRU Layers': fixed_gru_architecture,
                    'Batch Size': batch_size,
                    'Learning Rate': lr,
                    'Best Epoch': best_epoch,
                    'Val RMSE': val_rmse,
                    'Val MSE': val_mse,
                    'Target Features': target_columns # Now includes
all target features
                }
                best_model_test_pred_orig = y_test_pred_orig
                best_model_y_test_orig = y_test_orig
                best_model_lookback = lookback

# ---
## Save Metrics Results
#---
results_df = pd.DataFrame(results).sort_values(by='Val RMSE')
```

```
results_df.to_csv('hasil_gru3layer_multi_target_pat10.csv',
index=False)
print("\nHasil metrik semua model disimpan di
'hasil_gru3layer_multi_target_pat10.csv'")

# ---
## Visualize Loss Curves (MSE and RMSE)
#---
print("\nMenghasilkan visualisasi kurva loss untuk semua model...")
output_loss_folder = 'loss_plots_multi_target_pat10'
os.makedirs(output_loss_folder, exist_ok=True)

for model_name, hist_data in all_histories.items():
    # --- Plot MSE Loss Curve ---
    plt.figure(figsize=(15, 8))
    plt.plot(hist_data['loss'], label='Train Loss (MSE)')
    plt.plot(hist_data['val_loss'], label='Val Loss (MSE)')

    best_epoch_idx = hist_data['best_epoch_idx']
    best_val_mse_at_best_rmse_epoch =
hist_data['best_val_mse_at_best_rmse_epoch']

    plt.scatter(best_epoch_idx, hist_data['val_loss'][best_epoch_idx],
color='red', s=50, zorder=5,
                label=f'Best Val Epoch: {best_epoch_idx + 1} (Val MSE:
{best_val_mse_at_best_rmse_epoch:.4f})')

    plt.title(f'Loss (MSE) Curve for {model_name}')
    plt.xlabel('Epoch')
    plt.ylabel('Loss (MSE)')
    plt.ylim(0, 0.02) # Adjusted ylim for better visualization of
scaled loss
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.tight_layout()
    plt.savefig(os.path.join(output_loss_folder,
f'{model_name.replace(" ", "_").replace("(", "").replace(")",
"").replace("[", "").replace("]", "").replace(",", "_").replace(" ",
"")}_loss_mse_curve.png'))
    plt.close()

print(f"Visualisasi kurva loss MSE disimpan di folder:
{output_loss_folder}")

# ---
## Save Best Model
#---
if best_model is not None:
    best_model.save('best_gru_multi_target_model_pat10.h5')
    print("\nModel terbaik disimpan sebagai
```

```
'best_gru_multi_target_model_pat10.h5'")
    print("Konfigurasi model terbaik:")
    for k, v in best_model_config.items():
        print(f"{k}: {v}")

    # ---
    ## Visualize Best Model Predictions for each target feature
    # ---
    print(f"\nMenghasilkan visualisasi prediksi model terbaik untuk
setiap fitur target di data test...")
    output_pred_folder = 'best_model_predictions_multi_target_pat10'
    os.makedirs(output_pred_folder, exist_ok=True)

    start_index_test = best_model_lookback
    test_time_plot_for_best_model = test_time[start_index_test :
start_index_test + len(best_model_y_test_orig)]

    for i, feature_name in enumerate(target_columns):
        plt.figure(figsize=(15, 8))
        plt.plot(test_time_plot_for_best_model,
best_model_y_test_orig[:, i], label='Asli')
        plt.plot(test_time_plot_for_best_model,
best_model_test_pred_orig[:, i], label='Prediksi', alpha=0.7)
        plt.title(f'Prediksi vs Asli ({feature_name}) - Model
Terbaik')
        plt.xlabel('Waktu')
        plt.ylabel('Nilai')
        plt.legend()
        plt.tight_layout()
        plt.savefig(os.path.join(output_pred_folder,
f'best_model_pred_{feature_name.replace(" ", "_")}.png'))
        #plt.close()
    print(f"Visualisasi prediksi model terbaik disimpan di folder:
{output_pred_folder}")

    # ---
    ## Save Test Prediction Results CSV
    #---
    print(f"\nMenyimpan hasil prediksi test dan data asli test model
terbaik untuk semua fitur target ke CSV...")
    df_results_multi = pd.DataFrame({'Waktu':
test_time_plot_for_best_model})
    for i, feature_name in enumerate(target_columns):
        df_results_multi[f'Asli_{feature_name}'] =
best_model_y_test_orig[:, i]
        df_results_multi[f'Prediksi_{feature_name}'] =
best_model_test_pred_orig[:, i]

df_results_multi.to_csv('prediksi_asli_test_model_terbaik_multi_target
_pat10.csv', index=False)
```

```
    print("File
'prediksi_asli_test_model_terbaik_multi_target_pat10.csv' telah
dibuat.")
else:
    print("\nTidak ada model yang terlatih. Pastikan data dan
konfigurasi sudah benar.")

print("\n--- Proses Selesai ---")
print("\nHasil Akhir Semua Model:")
print(results_df)

#
==============================================================================
==============================
# START OF: Code to Load Model and Display Weights/Biases
#
==============================================================================
==============================

# --- Configuration for Weights/Biases Display ---
model_path = 'best_gru_multi_target_model_pat10.h5'

# --- Display Weights and Biases ---
print("\n" + "="*70)
print("  DISPLAYING WEIGHTS AND BIASES OF THE BEST GRU MODEL (MULTI
TARGET)")
print("  (Showing approximately 50 values per dimension)")
print("="*70)

if os.path.exists(model_path):
    try:
        # It's important to pass custom_objects when loading models
that use custom metrics
        loaded_model = load_model(model_path, custom_objects={
            'root_mean_squared_error': root_mean_squared_error,
            'mse': MeanSquaredError() # MSE is not strictly custom,
but good practice to include if explicitly used
        })

        print(f"\nSuccessfully loaded the model from: {model_path}")
        print("\n--- Model Summary ---")
        loaded_model.summary()

        print("\n--- Detailed Weights and Biases for Each Layer ---")
        for i, layer in enumerate(loaded_model.layers):
            weights_and_biases = layer.get_weights()

            if weights_and_biases:
                print(f"\n" + "="*50)
                print(f"  LAYER {i+1}: {layer.name.upper()} (Type:
```

```
{layer.__class__.__name__})")
                print("="*50)

                for j, wb_tensor in enumerate(weights_and_biases):
                    param_type = ""
                    if layer.name.startswith('gru'):
                        if j == 0: param_type = "Kernel Weights (Input
to Gates)"
                        elif j == 1: param_type = "Recurrent Kernel
Weights (Hidden to Gates)"
                        elif j == 2: param_type = "Bias"
                    elif layer.name.startswith('dense'):
                        if j == 0: param_type = "Weights"
                        elif j == 1: param_type = "Biases"
                    else:
                        param_type = f"Parameter {j}"

                    print(f"  > {param_type} (Shape:
{wb_tensor.shape}):")

                    if wb_tensor.ndim == 1:
                        display_len = min(len(wb_tensor), 50)
                        if display_len < len(wb_tensor):
                            print(f"    [ {', '.join(f'{x:.4f}' for x
in wb_tensor[:display_len])} ... (truncated) ]")
                        else:
                            print(f"    {wb_tensor}")
                        print(f"    Min Value:
{np.min(wb_tensor):.6f}, Max Value: {np.max(wb_tensor):.6f}")

                    elif wb_tensor.ndim == 2:
                        rows_to_display = min(wb_tensor.shape[0], 50)
                        cols_to_display = min(wb_tensor.shape[1], 50)

                        print(f"    Sub-matrix (first
{rows_to_display}x{cols_to_display}):")

print(np.array2string(wb_tensor[:rows_to_display, :cols_to_display],
precision=4, separator=', ', suppress_small=True))

                        if rows_to_display < wb_tensor.shape[0] or
cols_to_display < wb_tensor.shape[1]:
                            print("    ... (truncated)")

                        print(f"    Min Value:
{np.min(wb_tensor):.6f}, Max Value: {np.max(wb_tensor):.6f}")
                    else:
                        print(f"    {np.array2string(wb_tensor,
precision=4, separator=', ', suppress_small=True)}")
                        print(f"    Min Value:
```

```
{np.min(wb_tensor):.6f}, Max Value: {np.max(wb_tensor):.6f}")
                    print("-" * 50)

            else:
                print(f"\nLayer {i+1}: {layer.name.upper()} (Type:
{layer.__class__.__name__}) - Does not have trainable weights.")

    except Exception as e:
        print(f"\nError loading or inspecting the model: {e}")
        print("Please ensure:")
        print(f"1. The model file '{model_path}' exists in the current
directory.")
        print("2. The 'root_mean_squared_error' custom metric is
defined identically to how it was saved.")
else:
    print(f"\nError: Model file '{model_path}' not found.")
    print("Please make sure you have trained a model and saved it as
'best_gru_multi_target_model_pat10.h5' in the same directory.")

print("\n" + "="*70)
print("  WEIGHTS AND BIASES DISPLAY COMPLETE")
print("="*70)
```

```
Training Model GRU 1 (LB=12, GRU=[128, 64, 32], Batch=16, LR=0.001)
```

```
I0000 00:00:1752656437.421323      35 gpu_device.cc:2022] Created
device /job:localhost/replica:0/task:0/device:GPU:0 with 13942 MB
memory:  -> device: 0, name: Tesla T4, pci bus id: 0000:00:04.0,
compute capability: 7.5
I0000 00:00:1752656437.422013      35 gpu_device.cc:2022] Created
device /job:localhost/replica:0/task:0/device:GPU:1 with 13942 MB
memory:  -> device: 1, name: Tesla T4, pci bus id: 0000:00:05.0,
compute capability: 7.5
I0000 00:00:1752656443.341865     103 cuda_dnn.cc:529] Loaded cuDNN
version 90300
```

```
Per-Epoch MSE & RMSE for Model GRU 1 (LB=12, GRU=[128, 64, 32],
Batch=16, LR=0.001)
Epoch |  Train MSE   |   Val MSE    |  Train RMSE  |   Val RMSE
------------------------------------------------------------
    1 |       0.0057 |       0.0015 |       0.0684 |       0.0354
    2 |       0.0019 |       0.0010 |       0.0423 |       0.0302
    3 |       0.0016 |       0.0009 |       0.0383 |       0.0271
    4 |       0.0014 |       0.0008 |       0.0353 |       0.0263
    5 |       0.0012 |       0.0008 |       0.0336 |       0.0263
    6 |       0.0011 |       0.0008 |       0.0323 |       0.0249
    7 |       0.0010 |       0.0007 |       0.0311 |       0.0242
    8 |       0.0010 |       0.0007 |       0.0300 |       0.0235
```

```
    9 |       0.0009 |       0.0006 |       0.0293 |       0.0230
   10 |       0.0009 |       0.0006 |       0.0285 |       0.0220
   11 |       0.0008 |       0.0006 |       0.0280 |       0.0219
   12 |       0.0008 |       0.0007 |       0.0273 |       0.0236
   13 |       0.0008 |       0.0006 |       0.0270 |       0.0224
   14 |       0.0008 |       0.0006 |       0.0267 |       0.0220
   15 |       0.0007 |       0.0006 |       0.0262 |       0.0221
   16 |       0.0007 |       0.0007 |       0.0260 |       0.0232
   17 |       0.0007 |       0.0006 |       0.0261 |       0.0220
   18 |       0.0007 |       0.0006 |       0.0254 |       0.0227
   19 |       0.0007 |       0.0005 |       0.0256 |       0.0206
   20 |       0.0007 |       0.0006 |       0.0251 |       0.0231
   21 |       0.0006 |       0.0006 |       0.0245 |       0.0218
   22 |       0.0007 |       0.0006 |       0.0247 |       0.0216
   23 |       0.0006 |       0.0006 |       0.0242 |       0.0216
   24 |       0.0006 |       0.0006 |       0.0239 |       0.0222
   25 |       0.0006 |       0.0006 |       0.0238 |       0.0209
   26 |       0.0006 |       0.0005 |       0.0233 |       0.0203
   27 |       0.0006 |       0.0006 |       0.0231 |       0.0212
   28 |       0.0006 |       0.0006 |       0.0231 |       0.0223
   29 |       0.0006 |       0.0005 |       0.0229 |       0.0208

Epoch Terbaik: 26 (Val RMSE: 0.0203)
220/220 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step

Final Metrics:
Train MSE:    1677.6220 | Train RMSE: 40.9588
Val    MSE:    1340.1701 | Val    RMSE: 36.6083
Test   MSE:    2075.6530 | Test   RMSE: 45.5593

Training Model GRU 2 (LB=12, GRU=[128, 64, 32], Batch=16, LR=0.0001)

Per-Epoch MSE & RMSE for Model GRU 2 (LB=12, GRU=[128, 64, 32],
Batch=16, LR=0.0001)
Epoch |  Train MSE   |   Val MSE    |  Train RMSE  |   Val RMSE
------------------------------------------------------------
    1 |       0.0158 |       0.0085 |       0.1214 |       0.0828
    2 |       0.0070 |       0.0034 |       0.0825 |       0.0548
    3 |       0.0035 |       0.0020 |       0.0581 |       0.0412
    4 |       0.0026 |       0.0015 |       0.0496 |       0.0360
    5 |       0.0023 |       0.0014 |       0.0462 |       0.0339
    6 |       0.0021 |       0.0013 |       0.0443 |       0.0326
    7 |       0.0020 |       0.0012 |       0.0426 |       0.0319
    8 |       0.0018 |       0.0011 |       0.0414 |       0.0310
    9 |       0.0018 |       0.0010 |       0.0403 |       0.0292
   10 |       0.0017 |       0.0010 |       0.0397 |       0.0294
   11 |       0.0017 |       0.0010 |       0.0393 |       0.0281
   12 |       0.0017 |       0.0009 |       0.0392 |       0.0279
```

13	0.0016	0.0009	0.0384	0.0273
14	0.0016	0.0009	0.0381	0.0276
15	0.0015	0.0009	0.0377	0.0269
16	0.0015	0.0010	0.0374	0.0289
17	0.0015	0.0009	0.0371	0.0275
18	0.0015	0.0010	0.0366	0.0277
19	0.0014	0.0008	0.0365	0.0263
20	0.0014	0.0008	0.0362	0.0255
21	0.0014	0.0008	0.0360	0.0258
22	0.0014	0.0008	0.0358	0.0252
23	0.0013	0.0009	0.0351	0.0282
24	0.0013	0.0009	0.0349	0.0272
25	0.0013	0.0008	0.0346	0.0255
26	0.0013	0.0008	0.0344	0.0250
27	0.0013	0.0008	0.0342	0.0254
28	0.0013	0.0008	0.0339	0.0249
29	0.0012	0.0009	0.0337	0.0265
30	0.0012	0.0007	0.0337	0.0247
31	0.0012	0.0008	0.0336	0.0252
32	0.0012	0.0007	0.0333	0.0241
33	0.0012	0.0007	0.0331	0.0251
34	0.0012	0.0008	0.0330	0.0255
35	0.0012	0.0008	0.0328	0.0246
36	0.0012	0.0007	0.0329	0.0242
37	0.0012	0.0007	0.0325	0.0240
38	0.0011	0.0008	0.0324	0.0250
39	0.0011	0.0009	0.0322	0.0265
40	0.0011	0.0007	0.0320	0.0241
41	0.0011	0.0007	0.0320	0.0242
42	0.0011	0.0007	0.0317	0.0240
43	0.0011	0.0007	0.0319	0.0239
44	0.0011	0.0008	0.0316	0.0254
45	0.0011	0.0008	0.0313	0.0243
46	0.0011	0.0007	0.0312	0.0243
47	0.0011	0.0007	0.0311	0.0244
48	0.0010	0.0007	0.0309	0.0233
49	0.0010	0.0007	0.0308	0.0236
50	0.0010	0.0007	0.0306	0.0231
51	0.0010	0.0007	0.0306	0.0245
52	0.0010	0.0008	0.0303	0.0246
53	0.0010	0.0007	0.0303	0.0245
54	0.0010	0.0007	0.0301	0.0243
55	0.0010	0.0007	0.0301	0.0239
56	0.0010	0.0007	0.0300	0.0239
57	0.0010	0.0007	0.0298	0.0230
58	0.0009	0.0007	0.0296	0.0242
59	0.0009	0.0007	0.0296	0.0234
60	0.0009	0.0007	0.0294	0.0240
61	0.0009	0.0007	0.0292	0.0234

```
   62 |       0.0009 |       0.0007 |       0.0291 |       0.0234
   63 |       0.0009 |       0.0007 |       0.0290 |       0.0234
   64 |       0.0009 |       0.0006 |       0.0288 |       0.0227
   65 |       0.0009 |       0.0007 |       0.0288 |       0.0235
   66 |       0.0009 |       0.0007 |       0.0286 |       0.0230
   67 |       0.0009 |       0.0007 |       0.0287 |       0.0234
   68 |       0.0009 |       0.0006 |       0.0284 |       0.0223
   69 |       0.0009 |       0.0007 |       0.0283 |       0.0227
   70 |       0.0009 |       0.0006 |       0.0284 |       0.0230
   71 |       0.0009 |       0.0006 |       0.0280 |       0.0223
   72 |       0.0008 |       0.0006 |       0.0279 |       0.0229
   73 |       0.0008 |       0.0006 |       0.0279 |       0.0221
   74 |       0.0008 |       0.0006 |       0.0277 |       0.0226
   75 |       0.0008 |       0.0007 |       0.0277 |       0.0228
   76 |       0.0008 |       0.0006 |       0.0275 |       0.0222
   77 |       0.0008 |       0.0006 |       0.0274 |       0.0225
   78 |       0.0008 |       0.0006 |       0.0275 |       0.0230
   79 |       0.0008 |       0.0006 |       0.0273 |       0.0218
   80 |       0.0008 |       0.0006 |       0.0271 |       0.0222
   81 |       0.0008 |       0.0007 |       0.0273 |       0.0229
   82 |       0.0008 |       0.0006 |       0.0270 |       0.0221
   83 |       0.0008 |       0.0006 |       0.0269 |       0.0224
   84 |       0.0008 |       0.0006 |       0.0269 |       0.0223
   85 |       0.0008 |       0.0006 |       0.0268 |       0.0230
   86 |       0.0008 |       0.0006 |       0.0268 |       0.0218
   87 |       0.0008 |       0.0006 |       0.0268 |       0.0223
   88 |       0.0008 |       0.0006 |       0.0266 |       0.0217
   89 |       0.0008 |       0.0006 |       0.0266 |       0.0223

Epoch Terbaik: 88 (Val RMSE: 0.0217)
220/220 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step

Final Metrics:
Train MSE:    2090.6723 | Train RMSE: 45.7239
Val    MSE:    1609.0098 | Val    RMSE: 40.1125
Test   MSE:    2528.9937 | Test   RMSE: 50.2891

Training Model GRU 3 (LB=12, GRU=[128, 64, 32], Batch=32, LR=0.001)

Per-Epoch MSE & RMSE for Model GRU 3 (LB=12, GRU=[128, 64, 32],
Batch=32, LR=0.001)
Epoch |  Train MSE   |   Val MSE    |  Train RMSE  |   Val RMSE
------------------------------------------------------------
    1 |       0.0065 |       0.0017 |       0.0758 |       0.0383
    2 |       0.0022 |       0.0012 |       0.0459 |       0.0325
    3 |       0.0017 |       0.0010 |       0.0408 |       0.0286
    4 |       0.0016 |       0.0008 |       0.0387 |       0.0273
    5 |       0.0014 |       0.0009 |       0.0363 |       0.0280
```

```
    6 |       0.0013 |       0.0010 |       0.0351 |       0.0286
    7 |       0.0012 |       0.0008 |       0.0347 |       0.0269
    8 |       0.0012 |       0.0007 |       0.0335 |       0.0250
    9 |       0.0011 |       0.0007 |       0.0324 |       0.0241
   10 |       0.0010 |       0.0011 |       0.0317 |       0.0298
   11 |       0.0010 |       0.0007 |       0.0312 |       0.0247
   12 |       0.0009 |       0.0007 |       0.0300 |       0.0254
   13 |       0.0009 |       0.0006 |       0.0295 |       0.0243
   14 |       0.0009 |       0.0006 |       0.0288 |       0.0231
   15 |       0.0008 |       0.0008 |       0.0282 |       0.0280
   16 |       0.0008 |       0.0007 |       0.0280 |       0.0250
   17 |       0.0008 |       0.0006 |       0.0278 |       0.0238
   18 |       0.0008 |       0.0007 |       0.0276 |       0.0246
   19 |       0.0008 |       0.0006 |       0.0272 |       0.0236
   20 |       0.0007 |       0.0006 |       0.0266 |       0.0223
   21 |       0.0007 |       0.0006 |       0.0266 |       0.0230
   22 |       0.0007 |       0.0006 |       0.0263 |       0.0218
   23 |       0.0007 |       0.0006 |       0.0259 |       0.0223
   24 |       0.0007 |       0.0006 |       0.0259 |       0.0228
   25 |       0.0007 |       0.0006 |       0.0255 |       0.0223
   26 |       0.0007 |       0.0007 |       0.0254 |       0.0234
   27 |       0.0007 |       0.0005 |       0.0251 |       0.0206
   28 |       0.0006 |       0.0006 |       0.0246 |       0.0222
   29 |       0.0006 |       0.0006 |       0.0246 |       0.0233
   30 |       0.0006 |       0.0006 |       0.0248 |       0.0229
   31 |       0.0006 |       0.0006 |       0.0243 |       0.0228
   32 |       0.0006 |       0.0005 |       0.0241 |       0.0211
   33 |       0.0006 |       0.0006 |       0.0238 |       0.0229
   34 |       0.0006 |       0.0005 |       0.0242 |       0.0210
   35 |       0.0006 |       0.0006 |       0.0240 |       0.0215
   36 |       0.0006 |       0.0006 |       0.0235 |       0.0230
   37 |       0.0006 |       0.0005 |       0.0231 |       0.0212

Epoch Terbaik: 27 (Val RMSE: 0.0206)
220/220 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step

Final Metrics:
Train MSE:    1390.3542 | Train RMSE: 37.2875
Val    MSE:    1284.2975 | Val    RMSE: 35.8371
Test   MSE:    1721.6989 | Test   RMSE: 41.4934

Training Model GRU 4 (LB=12, GRU=[128, 64, 32], Batch=32, LR=0.0001)

Per-Epoch MSE & RMSE for Model GRU 4 (LB=12, GRU=[128, 64, 32],
Batch=32, LR=0.0001)
Epoch |  Train MSE   |   Val MSE    |  Train RMSE  |   Val RMSE
------------------------------------------------------------
    1 |       0.0171 |       0.0102 |       0.1274 |       0.0919
```

2	0.0101	0.0067	0.0997	0.0761
3	0.0067	0.0038	0.0812	0.0571
4	0.0044	0.0025	0.0658	0.0466
5	0.0033	0.0019	0.0569	0.0408
6	0.0028	0.0017	0.0521	0.0382
7	0.0025	0.0017	0.0494	0.0394
8	0.0023	0.0015	0.0478	0.0357
9	0.0022	0.0015	0.0462	0.0366
10	0.0021	0.0013	0.0450	0.0342
11	0.0020	0.0013	0.0441	0.0325
12	0.0019	0.0012	0.0433	0.0322
13	0.0019	0.0013	0.0425	0.0338
14	0.0018	0.0012	0.0419	0.0326
15	0.0018	0.0013	0.0414	0.0338
16	0.0017	0.0010	0.0406	0.0298
17	0.0017	0.0014	0.0402	0.0361
18	0.0017	0.0010	0.0401	0.0295
19	0.0016	0.0010	0.0394	0.0292
20	0.0016	0.0010	0.0394	0.0293
21	0.0016	0.0009	0.0389	0.0281
22	0.0016	0.0010	0.0387	0.0302
23	0.0016	0.0009	0.0386	0.0275
24	0.0015	0.0009	0.0384	0.0277
25	0.0015	0.0009	0.0383	0.0274
26	0.0015	0.0009	0.0382	0.0277
27	0.0015	0.0009	0.0378	0.0270
28	0.0015	0.0009	0.0377	0.0275
29	0.0015	0.0009	0.0378	0.0269
30	0.0015	0.0008	0.0375	0.0265
31	0.0015	0.0009	0.0379	0.0275
32	0.0014	0.0009	0.0370	0.0270
33	0.0014	0.0009	0.0372	0.0271
34	0.0015	0.0008	0.0372	0.0266
35	0.0014	0.0008	0.0370	0.0265
36	0.0014	0.0008	0.0370	0.0262
37	0.0014	0.0009	0.0368	0.0275
38	0.0014	0.0008	0.0367	0.0261
39	0.0014	0.0008	0.0366	0.0258
40	0.0014	0.0008	0.0364	0.0270
41	0.0014	0.0008	0.0362	0.0259
42	0.0014	0.0008	0.0363	0.0265
43	0.0014	0.0008	0.0362	0.0260
44	0.0014	0.0008	0.0361	0.0261
45	0.0013	0.0008	0.0357	0.0271
46	0.0013	0.0008	0.0358	0.0256
47	0.0013	0.0009	0.0357	0.0283
48	0.0013	0.0008	0.0355	0.0265
49	0.0013	0.0008	0.0358	0.0259
50	0.0013	0.0008	0.0353	0.0269

51	0.0013	0.0008	0.0352	0.0272
52	0.0013	0.0008	0.0350	0.0255
53	0.0013	0.0008	0.0350	0.0258
54	0.0013	0.0008	0.0353	0.0257
55	0.0013	0.0008	0.0346	0.0260
56	0.0013	0.0008	0.0349	0.0261
57	0.0013	0.0008	0.0347	0.0261
58	0.0013	0.0009	0.0346	0.0278
59	0.0012	0.0008	0.0343	0.0258
60	0.0012	0.0007	0.0343	0.0249
61	0.0012	0.0008	0.0341	0.0252
62	0.0012	0.0008	0.0340	0.0252
63	0.0012	0.0007	0.0339	0.0248
64	0.0012	0.0008	0.0339	0.0255
65	0.0012	0.0008	0.0337	0.0258
66	0.0012	0.0007	0.0335	0.0244
67	0.0012	0.0007	0.0333	0.0241
68	0.0012	0.0007	0.0334	0.0247
69	0.0011	0.0007	0.0331	0.0251
70	0.0011	0.0007	0.0331	0.0251
71	0.0011	0.0007	0.0330	0.0244
72	0.0011	0.0007	0.0326	0.0251
73	0.0011	0.0007	0.0326	0.0242
74	0.0011	0.0007	0.0325	0.0255
75	0.0011	0.0007	0.0324	0.0250
76	0.0011	0.0007	0.0321	0.0246
77	0.0011	0.0007	0.0321	0.0249
78	0.0011	0.0008	0.0320	0.0257
79	0.0011	0.0007	0.0319	0.0246
80	0.0011	0.0007	0.0318	0.0254
81	0.0011	0.0007	0.0318	0.0249
82	0.0010	0.0007	0.0316	0.0241
83	0.0010	0.0008	0.0315	0.0258
84	0.0010	0.0007	0.0314	0.0242
85	0.0010	0.0007	0.0312	0.0238
86	0.0010	0.0007	0.0310	0.0246
87	0.0010	0.0007	0.0311	0.0244
88	0.0010	0.0007	0.0311	0.0246
89	0.0010	0.0007	0.0307	0.0246
90	0.0010	0.0007	0.0307	0.0236
91	0.0010	0.0007	0.0306	0.0248
92	0.0010	0.0007	0.0306	0.0241
93	0.0010	0.0007	0.0304	0.0252
94	0.0010	0.0007	0.0304	0.0245
95	0.0010	0.0007	0.0303	0.0245
96	0.0010	0.0008	0.0304	0.0272
97	0.0010	0.0007	0.0302	0.0237
98	0.0009	0.0007	0.0300	0.0244
99	0.0009	0.0008	0.0299	0.0264

100	0.0009	0.0007	0.0299	0.0232
101	0.0009	0.0007	0.0301	0.0247
102	0.0009	0.0007	0.0298	0.0243
103	0.0009	0.0007	0.0296	0.0246
104	0.0009	0.0007	0.0295	0.0243
105	0.0009	0.0007	0.0294	0.0239
106	0.0009	0.0007	0.0293	0.0238
107	0.0009	0.0007	0.0292	0.0245
108	0.0009	0.0006	0.0293	0.0236
109	0.0009	0.0006	0.0292	0.0236
110	0.0009	0.0007	0.0290	0.0244
111	0.0009	0.0007	0.0290	0.0253
112	0.0009	0.0007	0.0290	0.0238
113	0.0009	0.0006	0.0287	0.0234
114	0.0009	0.0007	0.0286	0.0245
115	0.0009	0.0006	0.0287	0.0230
116	0.0009	0.0006	0.0285	0.0232
117	0.0009	0.0006	0.0285	0.0233
118	0.0008	0.0007	0.0285	0.0244
119	0.0008	0.0007	0.0283	0.0242
120	0.0008	0.0006	0.0283	0.0228
121	0.0008	0.0006	0.0281	0.0228
122	0.0008	0.0006	0.0283	0.0229
123	0.0008	0.0006	0.0281	0.0230
124	0.0008	0.0006	0.0280	0.0227
125	0.0008	0.0006	0.0279	0.0233
126	0.0008	0.0007	0.0278	0.0238
127	0.0008	0.0007	0.0278	0.0244
128	0.0008	0.0006	0.0279	0.0224
129	0.0008	0.0006	0.0275	0.0235
130	0.0008	0.0006	0.0277	0.0228
131	0.0008	0.0006	0.0277	0.0232
132	0.0008	0.0006	0.0275	0.0229
133	0.0008	0.0006	0.0274	0.0224
134	0.0008	0.0006	0.0272	0.0228
135	0.0008	0.0006	0.0273	0.0227
136	0.0008	0.0007	0.0272	0.0246
137	0.0008	0.0006	0.0273	0.0230
138	0.0008	0.0006	0.0271	0.0224
139	0.0008	0.0006	0.0271	0.0229
140	0.0008	0.0006	0.0270	0.0235
141	0.0008	0.0007	0.0271	0.0234
142	0.0008	0.0006	0.0268	0.0233
143	0.0008	0.0006	0.0268	0.0223
144	0.0008	0.0006	0.0269	0.0224
145	0.0007	0.0007	0.0268	0.0239
146	0.0007	0.0006	0.0267	0.0225
147	0.0007	0.0006	0.0267	0.0226
148	0.0007	0.0006	0.0266	0.0228

```
  149 |       0.0007 |       0.0006 |       0.0266 |       0.0222
  150 |       0.0007 |       0.0006 |       0.0266 |       0.0224
  151 |       0.0007 |       0.0006 |       0.0264 |       0.0229
  152 |       0.0007 |       0.0006 |       0.0265 |       0.0220
  153 |       0.0007 |       0.0006 |       0.0265 |       0.0222
  154 |       0.0007 |       0.0006 |       0.0263 |       0.0227
  155 |       0.0007 |       0.0006 |       0.0263 |       0.0234
  156 |       0.0007 |       0.0006 |       0.0262 |       0.0227
  157 |       0.0007 |       0.0006 |       0.0262 |       0.0226
  158 |       0.0007 |       0.0006 |       0.0262 |       0.0226
  159 |       0.0007 |       0.0006 |       0.0261 |       0.0234
  160 |       0.0007 |       0.0006 |       0.0261 |       0.0224
  161 |       0.0007 |       0.0006 |       0.0261 |       0.0224
  162 |       0.0007 |       0.0006 |       0.0260 |       0.0228

Epoch Terbaik: 152 (Val RMSE: 0.0220)
220/220 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step

Final Metrics:
Train MSE:    1840.9783 | Train RMSE: 42.9066
Val    MSE:    1482.4595 | Val    RMSE: 38.5027
Test   MSE:    2362.3048 | Test   RMSE: 48.6035

Training Model GRU 5 (LB=12, GRU=[128, 64, 32], Batch=64, LR=0.001)

Per-Epoch MSE & RMSE for Model GRU 5 (LB=12, GRU=[128, 64, 32],
Batch=64, LR=0.001)
Epoch |  Train MSE   |   Val MSE    |  Train RMSE  |   Val RMSE
------------------------------------------------------------
    1 |       0.0116 |       0.0033 |       0.1036 |       0.0557
    2 |       0.0032 |       0.0016 |       0.0558 |       0.0381
    3 |       0.0022 |       0.0012 |       0.0461 |       0.0336
    4 |       0.0018 |       0.0016 |       0.0426 |       0.0368
    5 |       0.0017 |       0.0009 |       0.0410 |       0.0289
    6 |       0.0015 |       0.0009 |       0.0382 |       0.0273
    7 |       0.0014 |       0.0010 |       0.0374 |       0.0294
    8 |       0.0014 |       0.0008 |       0.0365 |       0.0276
    9 |       0.0013 |       0.0008 |       0.0355 |       0.0263
   10 |       0.0012 |       0.0009 |       0.0348 |       0.0282
   11 |       0.0012 |       0.0007 |       0.0338 |       0.0254
   12 |       0.0011 |       0.0008 |       0.0334 |       0.0264
   13 |       0.0011 |       0.0007 |       0.0326 |       0.0245
   14 |       0.0011 |       0.0008 |       0.0324 |       0.0258
   15 |       0.0010 |       0.0008 |       0.0317 |       0.0257
   16 |       0.0010 |       0.0007 |       0.0311 |       0.0249
   17 |       0.0009 |       0.0007 |       0.0305 |       0.0238
   18 |       0.0009 |       0.0007 |       0.0300 |       0.0246
   19 |       0.0009 |       0.0007 |       0.0296 |       0.0236
```

```
  20 |        0.0009 |        0.0006 |        0.0293 |        0.0236
  21 |        0.0008 |        0.0007 |        0.0287 |        0.0247
  22 |        0.0008 |        0.0006 |        0.0287 |        0.0234
  23 |        0.0008 |        0.0007 |        0.0284 |        0.0244
  24 |        0.0008 |        0.0006 |        0.0277 |        0.0233
  25 |        0.0008 |        0.0006 |        0.0275 |        0.0223
  26 |        0.0008 |        0.0006 |        0.0272 |        0.0226
  27 |        0.0007 |        0.0006 |        0.0268 |        0.0224
  28 |        0.0007 |        0.0006 |        0.0266 |        0.0222
  29 |        0.0007 |        0.0006 |        0.0268 |        0.0226
  30 |        0.0007 |        0.0006 |        0.0259 |        0.0231
  31 |        0.0007 |        0.0006 |        0.0261 |        0.0222
  32 |        0.0007 |        0.0006 |        0.0259 |        0.0228
  33 |        0.0007 |        0.0006 |        0.0259 |        0.0225
  34 |        0.0007 |        0.0006 |        0.0262 |        0.0224
  35 |        0.0007 |        0.0006 |        0.0257 |        0.0223
  36 |        0.0007 |        0.0005 |        0.0256 |        0.0211
  37 |        0.0007 |        0.0006 |        0.0252 |        0.0222
  38 |        0.0006 |        0.0005 |        0.0251 |        0.0215
  39 |        0.0006 |        0.0005 |        0.0250 |        0.0212
  40 |        0.0006 |        0.0005 |        0.0247 |        0.0216
  41 |        0.0006 |        0.0006 |        0.0247 |        0.0220
  42 |        0.0006 |        0.0006 |        0.0248 |        0.0223
  43 |        0.0006 |        0.0006 |        0.0247 |        0.0223
  44 |        0.0006 |        0.0006 |        0.0243 |        0.0226
  45 |        0.0006 |        0.0005 |        0.0242 |        0.0214
  46 |        0.0006 |        0.0005 |        0.0243 |        0.0208
  47 |        0.0006 |        0.0005 |        0.0244 |        0.0211
  48 |        0.0006 |        0.0006 |        0.0239 |        0.0217
  49 |        0.0006 |        0.0005 |        0.0236 |        0.0215
  50 |        0.0006 |        0.0005 |        0.0239 |        0.0209
  51 |        0.0006 |        0.0006 |        0.0238 |        0.0232
  52 |        0.0006 |        0.0006 |        0.0236 |        0.0227
  53 |        0.0006 |        0.0005 |        0.0234 |        0.0217
  54 |        0.0006 |        0.0005 |        0.0232 |        0.0218
  55 |        0.0006 |        0.0005 |        0.0233 |        0.0212
  56 |        0.0006 |        0.0005 |        0.0232 |        0.0202
  57 |        0.0006 |        0.0006 |        0.0231 |        0.0216
  58 |        0.0006 |        0.0006 |        0.0233 |        0.0217
  59 |        0.0005 |        0.0005 |        0.0229 |        0.0206
  60 |        0.0005 |        0.0005 |        0.0230 |        0.0207
  61 |        0.0005 |        0.0005 |        0.0227 |        0.0209
  62 |        0.0005 |        0.0006 |        0.0230 |        0.0220
  63 |        0.0005 |        0.0005 |        0.0229 |        0.0212
  64 |        0.0005 |        0.0006 |        0.0223 |        0.0216
  65 |        0.0005 |        0.0006 |        0.0226 |        0.0224
  66 |        0.0005 |        0.0005 |        0.0223 |        0.0210

Epoch Terbaik: 56 (Val RMSE: 0.0202)
```

```
220/220 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step

Final Metrics:
Train MSE:    1177.8410 | Train RMSE: 34.3197
Val    MSE:    1315.8076 | Val    RMSE: 36.2741
Test   MSE:    1676.2280 | Test   RMSE: 40.9418

Training Model GRU 6 (LB=12, GRU=[128, 64, 32], Batch=64, LR=0.0001)

Per-Epoch MSE & RMSE for Model GRU 6 (LB=12, GRU=[128, 64, 32],
Batch=64, LR=0.0001)
```

Epoch	Train MSE	Val MSE	Train RMSE	Val RMSE
1	0.0216	0.0125	0.1426	0.1074
2	0.0128	0.0095	0.1126	0.0928
3	0.0106	0.0075	0.1025	0.0833
4	0.0087	0.0058	0.0928	0.0728
5	0.0069	0.0042	0.0825	0.0621
6	0.0053	0.0032	0.0726	0.0533
7	0.0042	0.0026	0.0646	0.0479
8	0.0035	0.0022	0.0591	0.0435
9	0.0031	0.0019	0.0551	0.0412
10	0.0028	0.0020	0.0525	0.0422
11	0.0026	0.0016	0.0507	0.0383
12	0.0024	0.0016	0.0489	0.0379
13	0.0023	0.0015	0.0479	0.0362
14	0.0022	0.0014	0.0467	0.0360
15	0.0021	0.0013	0.0459	0.0343
16	0.0021	0.0013	0.0451	0.0335
17	0.0020	0.0012	0.0443	0.0323
18	0.0020	0.0012	0.0438	0.0327
19	0.0019	0.0012	0.0433	0.0329
20	0.0019	0.0013	0.0427	0.0340
21	0.0018	0.0012	0.0422	0.0320
22	0.0018	0.0012	0.0417	0.0323
23	0.0018	0.0011	0.0415	0.0304
24	0.0017	0.0011	0.0412	0.0312
25	0.0017	0.0011	0.0409	0.0318
26	0.0017	0.0010	0.0406	0.0298
27	0.0017	0.0010	0.0403	0.0293
28	0.0016	0.0010	0.0400	0.0288
29	0.0016	0.0010	0.0397	0.0290
30	0.0016	0.0010	0.0397	0.0289
31	0.0016	0.0010	0.0397	0.0289
32	0.0016	0.0010	0.0393	0.0295
33	0.0016	0.0010	0.0390	0.0303
34	0.0016	0.0009	0.0389	0.0280
35	0.0016	0.0009	0.0390	0.0279

36	0.0015	0.0009	0.0389	0.0283
37	0.0015	0.0009	0.0386	0.0277
38	0.0015	0.0009	0.0384	0.0275
39	0.0015	0.0009	0.0383	0.0285
40	0.0015	0.0009	0.0381	0.0289
41	0.0015	0.0008	0.0382	0.0267
42	0.0015	0.0009	0.0380	0.0270
43	0.0015	0.0009	0.0379	0.0273
44	0.0015	0.0009	0.0378	0.0273
45	0.0015	0.0009	0.0377	0.0268
46	0.0015	0.0009	0.0377	0.0273
47	0.0014	0.0008	0.0376	0.0268
48	0.0014	0.0009	0.0375	0.0274
49	0.0015	0.0010	0.0379	0.0294
50	0.0015	0.0009	0.0378	0.0272
51	0.0014	0.0009	0.0373	0.0281
52	0.0014	0.0008	0.0371	0.0266
53	0.0014	0.0009	0.0373	0.0274
54	0.0014	0.0008	0.0372	0.0264
55	0.0014	0.0008	0.0370	0.0267
56	0.0014	0.0010	0.0370	0.0292
57	0.0014	0.0008	0.0368	0.0264
58	0.0014	0.0009	0.0373	0.0276
59	0.0014	0.0009	0.0369	0.0278
60	0.0014	0.0009	0.0369	0.0274
61	0.0014	0.0008	0.0368	0.0266
62	0.0014	0.0009	0.0364	0.0276
63	0.0014	0.0009	0.0365	0.0276
64	0.0014	0.0009	0.0367	0.0272
65	0.0014	0.0009	0.0364	0.0268
66	0.0014	0.0008	0.0365	0.0267
67	0.0014	0.0008	0.0365	0.0259
68	0.0014	0.0008	0.0363	0.0263
69	0.0013	0.0008	0.0361	0.0270
70	0.0014	0.0008	0.0365	0.0265
71	0.0013	0.0008	0.0362	0.0262
72	0.0013	0.0008	0.0361	0.0267
73	0.0013	0.0008	0.0360	0.0265
74	0.0013	0.0008	0.0359	0.0259
75	0.0013	0.0008	0.0357	0.0264
76	0.0013	0.0008	0.0359	0.0272
77	0.0013	0.0008	0.0355	0.0253
78	0.0013	0.0008	0.0358	0.0256
79	0.0013	0.0008	0.0354	0.0262
80	0.0013	0.0008	0.0355	0.0259
81	0.0013	0.0008	0.0353	0.0263
82	0.0013	0.0009	0.0355	0.0274
83	0.0013	0.0008	0.0353	0.0255
84	0.0013	0.0008	0.0352	0.0256

```
  85 |       0.0013 |       0.0008 |       0.0350 |       0.0256
  86 |       0.0013 |       0.0008 |       0.0349 |       0.0264
  87 |       0.0013 |       0.0008 |       0.0350 |       0.0252
  88 |       0.0012 |       0.0008 |       0.0347 |       0.0267
  89 |       0.0012 |       0.0008 |       0.0349 |       0.0251
  90 |       0.0012 |       0.0008 |       0.0347 |       0.0256
  91 |       0.0013 |       0.0008 |       0.0351 |       0.0259
  92 |       0.0012 |       0.0008 |       0.0345 |       0.0256
  93 |       0.0012 |       0.0008 |       0.0344 |       0.0256
  94 |       0.0012 |       0.0008 |       0.0344 |       0.0266
  95 |       0.0012 |       0.0008 |       0.0343 |       0.0267
  96 |       0.0012 |       0.0007 |       0.0344 |       0.0248
  97 |       0.0012 |       0.0007 |       0.0342 |       0.0250
  98 |       0.0012 |       0.0007 |       0.0342 |       0.0251
  99 |       0.0012 |       0.0008 |       0.0342 |       0.0261
 100 |       0.0012 |       0.0007 |       0.0339 |       0.0249
 101 |       0.0012 |       0.0007 |       0.0337 |       0.0248
 102 |       0.0012 |       0.0007 |       0.0338 |       0.0256
 103 |       0.0012 |       0.0007 |       0.0337 |       0.0251
 104 |       0.0012 |       0.0007 |       0.0336 |       0.0247
 105 |       0.0012 |       0.0008 |       0.0337 |       0.0258
 106 |       0.0012 |       0.0007 |       0.0337 |       0.0245
 107 |       0.0012 |       0.0007 |       0.0339 |       0.0247
 108 |       0.0012 |       0.0007 |       0.0336 |       0.0245
 109 |       0.0012 |       0.0007 |       0.0336 |       0.0249
 110 |       0.0011 |       0.0007 |       0.0333 |       0.0246
 111 |       0.0011 |       0.0007 |       0.0333 |       0.0243
 112 |       0.0011 |       0.0008 |       0.0331 |       0.0257
 113 |       0.0011 |       0.0007 |       0.0330 |       0.0248
 114 |       0.0011 |       0.0007 |       0.0330 |       0.0251
 115 |       0.0011 |       0.0008 |       0.0330 |       0.0259
 116 |       0.0011 |       0.0007 |       0.0329 |       0.0248
 117 |       0.0011 |       0.0007 |       0.0329 |       0.0248
 118 |       0.0011 |       0.0007 |       0.0327 |       0.0246
 119 |       0.0011 |       0.0007 |       0.0327 |       0.0252
 120 |       0.0011 |       0.0007 |       0.0326 |       0.0249
 121 |       0.0011 |       0.0007 |       0.0327 |       0.0246

Epoch Terbaik: 111 (Val RMSE: 0.0243)
220/220 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
28/28 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step

Final Metrics:
Train MSE:    3368.8345 | Train RMSE: 58.0417
Val    MSE:    1931.5461 | Val    RMSE: 43.9494
Test   MSE:    3084.8346 | Test   RMSE: 55.5413

Training Model GRU 7 (LB=24, GRU=[128, 64, 32], Batch=16, LR=0.001)
```

Per-Epoch MSE & RMSE for Model GRU 7 (LB=24, GRU=[128, 64, 32], Batch=16, LR=0.001)

Epoch	Train MSE	Val MSE	Train RMSE	Val RMSE
1	0.0053	0.0016	0.0672	0.0370
2	0.0018	0.0009	0.0416	0.0285
3	0.0015	0.0009	0.0372	0.0274
4	0.0013	0.0008	0.0352	0.0263
5	0.0012	0.0010	0.0340	0.0290
6	0.0011	0.0009	0.0327	0.0258
7	0.0011	0.0008	0.0314	0.0246
8	0.0010	0.0007	0.0299	0.0243
9	0.0009	0.0006	0.0289	0.0226
10	0.0009	0.0007	0.0283	0.0246
11	0.0008	0.0007	0.0281	0.0237
12	0.0008	0.0007	0.0274	0.0238
13	0.0008	0.0006	0.0267	0.0225
14	0.0008	0.0005	0.0268	0.0209
15	0.0007	0.0007	0.0262	0.0233
16	0.0007	0.0005	0.0258	0.0205
17	0.0007	0.0006	0.0257	0.0210
18	0.0007	0.0006	0.0257	0.0208
19	0.0007	0.0006	0.0248	0.0219
20	0.0006	0.0005	0.0244	0.0208
21	0.0006	0.0005	0.0245	0.0203
22	0.0006	0.0006	0.0239	0.0219
23	0.0006	0.0005	0.0240	0.0209
24	0.0006	0.0005	0.0235	0.0206
25	0.0006	0.0005	0.0233	0.0192
26	0.0006	0.0005	0.0230	0.0196
27	0.0006	0.0005	0.0229	0.0204
28	0.0005	0.0006	0.0223	0.0215
29	0.0006	0.0005	0.0225	0.0196
30	0.0005	0.0005	0.0220	0.0192
31	0.0005	0.0005	0.0221	0.0204
32	0.0005	0.0005	0.0218	0.0190
33	0.0005	0.0005	0.0214	0.0187
34	0.0005	0.0005	0.0212	0.0207
35	0.0005	0.0005	0.0211	0.0200
36	0.0005	0.0005	0.0208	0.0198
37	0.0005	0.0005	0.0208	0.0192
38	0.0005	0.0005	0.0205	0.0211
39	0.0005	0.0005	0.0205	0.0186
40	0.0004	0.0005	0.0201	0.0196
41	0.0004	0.0005	0.0203	0.0193
42	0.0004	0.0005	0.0198	0.0196
43	0.0004	0.0005	0.0202	0.0194

Epoch Terbaik: 39 (Val RMSE: 0.0186)

```
219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step

Final Metrics:
Train MSE:    994.1506 | Train RMSE: 31.5302
Val    MSE:    1133.3592 | Val    RMSE: 33.6654
Test   MSE:    1676.3735 | Test   RMSE: 40.9435

Training Model GRU 8 (LB=24, GRU=[128, 64, 32], Batch=16, LR=0.0001)

Per-Epoch MSE & RMSE for Model GRU 8 (LB=24, GRU=[128, 64, 32],
Batch=16, LR=0.0001)
Epoch |  Train MSE   |   Val MSE    |  Train RMSE  |   Val RMSE
------------------------------------------------------------
    1 |       0.0155 |       0.0074 |       0.1190 |       0.0794
    2 |       0.0067 |       0.0032 |       0.0804 |       0.0519
    3 |       0.0038 |       0.0021 |       0.0603 |       0.0411
    4 |       0.0028 |       0.0016 |       0.0513 |       0.0367
    5 |       0.0024 |       0.0017 |       0.0477 |       0.0362
    6 |       0.0022 |       0.0013 |       0.0452 |       0.0323
    7 |       0.0020 |       0.0012 |       0.0433 |       0.0305
    8 |       0.0019 |       0.0011 |       0.0420 |       0.0308
    9 |       0.0018 |       0.0011 |       0.0406 |       0.0292
   10 |       0.0017 |       0.0012 |       0.0398 |       0.0316
   11 |       0.0017 |       0.0011 |       0.0391 |       0.0315
   12 |       0.0016 |       0.0010 |       0.0388 |       0.0279
   13 |       0.0016 |       0.0009 |       0.0383 |       0.0277
   14 |       0.0016 |       0.0009 |       0.0382 |       0.0276
   15 |       0.0015 |       0.0009 |       0.0377 |       0.0273
   16 |       0.0015 |       0.0009 |       0.0374 |       0.0276
   17 |       0.0015 |       0.0009 |       0.0373 |       0.0266
   18 |       0.0015 |       0.0009 |       0.0369 |       0.0270
   19 |       0.0015 |       0.0008 |       0.0366 |       0.0258
   20 |       0.0015 |       0.0009 |       0.0366 |       0.0260
   21 |       0.0014 |       0.0008 |       0.0363 |       0.0260
   22 |       0.0014 |       0.0010 |       0.0362 |       0.0283
   23 |       0.0014 |       0.0008 |       0.0359 |       0.0262
   24 |       0.0014 |       0.0008 |       0.0354 |       0.0257
   25 |       0.0014 |       0.0008 |       0.0356 |       0.0257
   26 |       0.0013 |       0.0008 |       0.0351 |       0.0252
   27 |       0.0013 |       0.0008 |       0.0348 |       0.0261
   28 |       0.0013 |       0.0008 |       0.0346 |       0.0256
   29 |       0.0013 |       0.0008 |       0.0344 |       0.0248
   30 |       0.0013 |       0.0009 |       0.0341 |       0.0265
   31 |       0.0012 |       0.0008 |       0.0339 |       0.0247
   32 |       0.0012 |       0.0008 |       0.0336 |       0.0246
   33 |       0.0012 |       0.0008 |       0.0334 |       0.0246
   34 |       0.0012 |       0.0008 |       0.0333 |       0.0253
   35 |       0.0012 |       0.0008 |       0.0331 |       0.0250
```

36	0.0012	0.0008	0.0329	0.0244
37	0.0011	0.0008	0.0325	0.0250
38	0.0011	0.0007	0.0324	0.0244
39	0.0011	0.0008	0.0322	0.0255
40	0.0011	0.0009	0.0320	0.0270
41	0.0011	0.0007	0.0317	0.0239
42	0.0011	0.0007	0.0316	0.0238
43	0.0011	0.0007	0.0314	0.0238
44	0.0011	0.0007	0.0314	0.0242
45	0.0010	0.0007	0.0310	0.0239
46	0.0010	0.0007	0.0310	0.0235
47	0.0010	0.0007	0.0306	0.0236
48	0.0010	0.0007	0.0306	0.0229
49	0.0010	0.0007	0.0303	0.0241
50	0.0010	0.0007	0.0302	0.0251
51	0.0010	0.0007	0.0303	0.0238
52	0.0010	0.0006	0.0300	0.0224
53	0.0010	0.0006	0.0297	0.0228
54	0.0009	0.0007	0.0296	0.0236
55	0.0009	0.0007	0.0294	0.0231
56	0.0009	0.0007	0.0293	0.0228
57	0.0009	0.0006	0.0289	0.0227
58	0.0009	0.0007	0.0289	0.0236
59	0.0009	0.0007	0.0287	0.0228
60	0.0009	0.0007	0.0284	0.0225
61	0.0009	0.0006	0.0285	0.0222
62	0.0009	0.0006	0.0284	0.0228
63	0.0008	0.0006	0.0280	0.0223
64	0.0008	0.0007	0.0280	0.0237
65	0.0008	0.0006	0.0279	0.0221
66	0.0008	0.0006	0.0277	0.0228
67	0.0008	0.0006	0.0277	0.0225
68	0.0008	0.0006	0.0274	0.0222
69	0.0008	0.0007	0.0273	0.0232
70	0.0008	0.0006	0.0272	0.0225
71	0.0008	0.0007	0.0271	0.0235
72	0.0008	0.0006	0.0270	0.0225
73	0.0008	0.0006	0.0268	0.0222
74	0.0008	0.0007	0.0268	0.0225
75	0.0008	0.0006	0.0267	0.0224
76	0.0008	0.0006	0.0265	0.0221
77	0.0008	0.0006	0.0265	0.0218
78	0.0007	0.0006	0.0262	0.0226
79	0.0007	0.0006	0.0262	0.0226
80	0.0007	0.0006	0.0260	0.0219
81	0.0007	0.0006	0.0260	0.0222
82	0.0007	0.0006	0.0258	0.0223
83	0.0007	0.0006	0.0259	0.0218
84	0.0007	0.0006	0.0257	0.0216

85	0.0007	0.0006	0.0257	0.0215
86	0.0007	0.0006	0.0255	0.0220
87	0.0007	0.0006	0.0255	0.0226
88	0.0007	0.0006	0.0254	0.0216
89	0.0007	0.0006	0.0252	0.0221
90	0.0007	0.0006	0.0253	0.0222
91	0.0007	0.0006	0.0252	0.0217
92	0.0007	0.0006	0.0252	0.0216
93	0.0007	0.0006	0.0249	0.0212
94	0.0007	0.0006	0.0251	0.0210
95	0.0007	0.0006	0.0249	0.0216
96	0.0007	0.0006	0.0248	0.0215
97	0.0007	0.0006	0.0249	0.0208
98	0.0007	0.0006	0.0248	0.0210
99	0.0007	0.0006	0.0247	0.0214
100	0.0007	0.0006	0.0246	0.0211
101	0.0007	0.0006	0.0245	0.0210
102	0.0006	0.0005	0.0244	0.0206
103	0.0006	0.0006	0.0245	0.0209
104	0.0006	0.0005	0.0243	0.0204
105	0.0006	0.0006	0.0243	0.0209
106	0.0006	0.0006	0.0242	0.0210
107	0.0006	0.0006	0.0241	0.0207
108	0.0006	0.0005	0.0241	0.0204
109	0.0006	0.0005	0.0239	0.0204
110	0.0006	0.0006	0.0240	0.0209
111	0.0006	0.0005	0.0238	0.0204
112	0.0006	0.0005	0.0239	0.0202
113	0.0006	0.0005	0.0238	0.0201
114	0.0006	0.0005	0.0236	0.0205
115	0.0006	0.0006	0.0237	0.0213
116	0.0006	0.0005	0.0237	0.0206
117	0.0006	0.0006	0.0236	0.0208
118	0.0006	0.0005	0.0235	0.0204
119	0.0006	0.0005	0.0235	0.0200
120	0.0006	0.0006	0.0234	0.0209
121	0.0006	0.0006	0.0233	0.0207
122	0.0006	0.0005	0.0233	0.0207
123	0.0006	0.0005	0.0233	0.0202
124	0.0006	0.0005	0.0233	0.0205
125	0.0006	0.0005	0.0232	0.0206
126	0.0006	0.0005	0.0231	0.0211
127	0.0006	0.0005	0.0232	0.0203
128	0.0006	0.0006	0.0229	0.0215
129	0.0006	0.0005	0.0229	0.0202

```
Epoch Terbaik: 119 (Val RMSE: 0.0200)
219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
```

```
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step

Final Metrics:
Train MSE:    1462.1190 | Train RMSE: 38.2377
Val    MSE:    1324.0824 | Val    RMSE: 36.3879
Test   MSE:    1851.5098 | Test   RMSE: 43.0292

Training Model GRU 9 (LB=24, GRU=[128, 64, 32], Batch=32, LR=0.001)

Per-Epoch MSE & RMSE for Model GRU 9 (LB=24, GRU=[128, 64, 32],
Batch=32, LR=0.001)
```

Epoch	Train MSE	Val MSE	Train RMSE	Val RMSE
1	0.0071	0.0018	0.0786	0.0391
2	0.0022	0.0012	0.0457	0.0317
3	0.0017	0.0010	0.0410	0.0307
4	0.0015	0.0008	0.0381	0.0273
5	0.0014	0.0008	0.0367	0.0264
6	0.0013	0.0008	0.0352	0.0262
7	0.0012	0.0008	0.0344	0.0270
8	0.0012	0.0007	0.0333	0.0250
9	0.0011	0.0008	0.0321	0.0250
10	0.0010	0.0007	0.0313	0.0249
11	0.0010	0.0008	0.0305	0.0276
12	0.0009	0.0007	0.0301	0.0240
13	0.0009	0.0007	0.0293	0.0239
14	0.0009	0.0006	0.0288	0.0228
15	0.0009	0.0006	0.0287	0.0232
16	0.0008	0.0006	0.0279	0.0233
17	0.0008	0.0007	0.0276	0.0246
18	0.0008	0.0006	0.0273	0.0237
19	0.0008	0.0006	0.0270	0.0223
20	0.0007	0.0006	0.0266	0.0224
21	0.0007	0.0006	0.0268	0.0231
22	0.0007	0.0006	0.0265	0.0218
23	0.0007	0.0006	0.0257	0.0222
24	0.0007	0.0005	0.0257	0.0210
25	0.0007	0.0006	0.0254	0.0227
26	0.0007	0.0006	0.0256	0.0228
27	0.0007	0.0006	0.0249	0.0231
28	0.0006	0.0006	0.0246	0.0225
29	0.0006	0.0005	0.0245	0.0207
30	0.0006	0.0005	0.0241	0.0205
31	0.0006	0.0006	0.0243	0.0228
32	0.0006	0.0007	0.0241	0.0236
33	0.0006	0.0006	0.0238	0.0218
34	0.0006	0.0009	0.0235	0.0262
35	0.0006	0.0005	0.0237	0.0210
36	0.0006	0.0005	0.0229	0.0210
37	0.0006	0.0005	0.0229	0.0211

```
   38 |        0.0006 |        0.0005 |        0.0232 |        0.0204
   39 |        0.0006 |        0.0006 |        0.0231 |        0.0216
   40 |        0.0005 |        0.0005 |        0.0226 |        0.0204
   41 |        0.0005 |        0.0005 |        0.0223 |        0.0202
   42 |        0.0005 |        0.0005 |        0.0226 |        0.0203
   43 |        0.0005 |        0.0005 |        0.0220 |        0.0212
   44 |        0.0005 |        0.0005 |        0.0220 |        0.0201
   45 |        0.0005 |        0.0005 |        0.0217 |        0.0200
   46 |        0.0005 |        0.0005 |        0.0219 |        0.0192
   47 |        0.0005 |        0.0005 |        0.0213 |        0.0195
   48 |        0.0005 |        0.0005 |        0.0216 |        0.0200
   49 |        0.0005 |        0.0005 |        0.0213 |        0.0214
   50 |        0.0005 |        0.0005 |        0.0213 |        0.0194
   51 |        0.0005 |        0.0004 |        0.0212 |        0.0193
   52 |        0.0005 |        0.0005 |        0.0208 |        0.0193
   53 |        0.0005 |        0.0005 |        0.0209 |        0.0191
   54 |        0.0005 |        0.0005 |        0.0210 |        0.0197
   55 |        0.0004 |        0.0004 |        0.0203 |        0.0193
   56 |        0.0004 |        0.0004 |        0.0205 |        0.0195
   57 |        0.0004 |        0.0005 |        0.0204 |        0.0192
   58 |        0.0004 |        0.0004 |        0.0201 |        0.0193
   59 |        0.0004 |        0.0005 |        0.0202 |        0.0195
   60 |        0.0004 |        0.0004 |        0.0198 |        0.0186
   61 |        0.0004 |        0.0004 |        0.0199 |        0.0189
   62 |        0.0004 |        0.0004 |        0.0197 |        0.0192
   63 |        0.0004 |        0.0005 |        0.0197 |        0.0191
   64 |        0.0004 |        0.0004 |        0.0194 |        0.0191
   65 |        0.0004 |        0.0005 |        0.0193 |        0.0205
   66 |        0.0004 |        0.0004 |        0.0196 |        0.0187
   67 |        0.0004 |        0.0006 |        0.0191 |        0.0206
   68 |        0.0004 |        0.0004 |        0.0190 |        0.0183
   69 |        0.0004 |        0.0004 |        0.0190 |        0.0188
   70 |        0.0004 |        0.0005 |        0.0189 |        0.0203
   71 |        0.0004 |        0.0004 |        0.0189 |        0.0193
   72 |        0.0004 |        0.0005 |        0.0187 |        0.0194
   73 |        0.0004 |        0.0004 |        0.0184 |        0.0190
   74 |        0.0004 |        0.0005 |        0.0184 |        0.0198
   75 |        0.0004 |        0.0005 |        0.0185 |        0.0198
   76 |        0.0003 |        0.0004 |        0.0181 |        0.0184
   77 |        0.0003 |        0.0004 |        0.0182 |        0.0186
   78 |        0.0003 |        0.0005 |        0.0181 |        0.0194

Epoch Terbaik: 68 (Val RMSE: 0.0183)
219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step

Final Metrics:
Train MSE:    760.6583 | Train RMSE: 27.5800
```

```
Val    MSE:    1013.5377 | Val    RMSE: 31.8361
Test   MSE:    1391.8374 | Test   RMSE: 37.3073

Training Model GRU 10 (LB=24, GRU=[128, 64, 32], Batch=32, LR=0.0001)

Per-Epoch MSE & RMSE for Model GRU 10 (LB=24, GRU=[128, 64, 32],
Batch=32, LR=0.0001)
```

Epoch	Train MSE	Val MSE	Train RMSE	Val RMSE
1	0.0188	0.0109	0.1334	0.0989
2	0.0110	0.0075	0.1041	0.0817
3	0.0075	0.0043	0.0858	0.0619
4	0.0047	0.0029	0.0682	0.0493
5	0.0034	0.0019	0.0578	0.0402
6	0.0028	0.0017	0.0525	0.0385
7	0.0025	0.0016	0.0497	0.0373
8	0.0024	0.0014	0.0482	0.0352
9	0.0023	0.0014	0.0467	0.0339
10	0.0021	0.0013	0.0452	0.0326
11	0.0020	0.0012	0.0445	0.0326
12	0.0020	0.0012	0.0434	0.0328
13	0.0019	0.0011	0.0427	0.0310
14	0.0018	0.0011	0.0420	0.0304
15	0.0018	0.0011	0.0415	0.0316
16	0.0018	0.0011	0.0412	0.0310
17	0.0017	0.0010	0.0407	0.0292
18	0.0017	0.0010	0.0406	0.0291
19	0.0017	0.0010	0.0400	0.0288
20	0.0016	0.0010	0.0396	0.0299
21	0.0016	0.0009	0.0394	0.0284
22	0.0016	0.0010	0.0394	0.0283
23	0.0016	0.0010	0.0388	0.0288
24	0.0016	0.0009	0.0388	0.0278
25	0.0016	0.0009	0.0386	0.0275
26	0.0015	0.0009	0.0385	0.0284
27	0.0015	0.0009	0.0384	0.0273
28	0.0015	0.0009	0.0384	0.0278
29	0.0015	0.0009	0.0382	0.0270
30	0.0015	0.0011	0.0378	0.0323
31	0.0015	0.0009	0.0378	0.0270
32	0.0015	0.0008	0.0378	0.0264
33	0.0014	0.0009	0.0371	0.0268
34	0.0015	0.0009	0.0373	0.0277
35	0.0014	0.0009	0.0371	0.0274
36	0.0014	0.0008	0.0373	0.0261
37	0.0014	0.0009	0.0368	0.0270
38	0.0014	0.0009	0.0368	0.0273
39	0.0014	0.0008	0.0365	0.0262
40	0.0014	0.0008	0.0365	0.0261

41	0.0014	0.0009	0.0363	0.0281
42	0.0014	0.0008	0.0364	0.0255
43	0.0014	0.0008	0.0360	0.0256
44	0.0014	0.0009	0.0361	0.0272
45	0.0013	0.0008	0.0358	0.0261
46	0.0013	0.0008	0.0355	0.0258
47	0.0013	0.0008	0.0356	0.0266
48	0.0013	0.0008	0.0352	0.0254
49	0.0013	0.0008	0.0352	0.0259
50	0.0013	0.0008	0.0351	0.0251
51	0.0013	0.0008	0.0349	0.0252
52	0.0013	0.0008	0.0346	0.0254
53	0.0013	0.0008	0.0347	0.0256
54	0.0012	0.0008	0.0342	0.0253
55	0.0012	0.0008	0.0343	0.0250
56	0.0012	0.0008	0.0342	0.0251
57	0.0012	0.0007	0.0340	0.0248
58	0.0012	0.0009	0.0337	0.0271
59	0.0012	0.0008	0.0338	0.0257
60	0.0012	0.0007	0.0333	0.0248
61	0.0012	0.0008	0.0333	0.0254
62	0.0011	0.0007	0.0330	0.0243
63	0.0011	0.0007	0.0330	0.0249
64	0.0011	0.0008	0.0329	0.0252
65	0.0011	0.0007	0.0325	0.0246
66	0.0011	0.0007	0.0325	0.0244
67	0.0011	0.0007	0.0323	0.0242
68	0.0011	0.0007	0.0322	0.0249
69	0.0011	0.0007	0.0322	0.0249
70	0.0011	0.0009	0.0319	0.0271
71	0.0011	0.0007	0.0319	0.0242
72	0.0011	0.0007	0.0316	0.0257
73	0.0010	0.0007	0.0316	0.0248
74	0.0010	0.0007	0.0314	0.0244
75	0.0010	0.0007	0.0313	0.0251
76	0.0010	0.0007	0.0313	0.0244
77	0.0010	0.0007	0.0310	0.0251
78	0.0010	0.0007	0.0310	0.0250
79	0.0010	0.0007	0.0310	0.0245
80	0.0010	0.0007	0.0309	0.0250
81	0.0010	0.0007	0.0308	0.0235
82	0.0010	0.0007	0.0308	0.0241
83	0.0010	0.0007	0.0307	0.0240
84	0.0010	0.0007	0.0305	0.0246
85	0.0010	0.0007	0.0304	0.0247
86	0.0010	0.0007	0.0303	0.0242
87	0.0010	0.0007	0.0302	0.0249
88	0.0010	0.0007	0.0302	0.0249
89	0.0009	0.0007	0.0301	0.0235

90	0.0009	0.0007	0.0301	0.0234
91	0.0009	0.0007	0.0299	0.0235
92	0.0009	0.0007	0.0297	0.0247
93	0.0009	0.0007	0.0298	0.0238
94	0.0009	0.0006	0.0298	0.0231
95	0.0009	0.0007	0.0296	0.0233
96	0.0009	0.0006	0.0294	0.0232
97	0.0009	0.0007	0.0294	0.0232
98	0.0009	0.0007	0.0293	0.0243
99	0.0009	0.0006	0.0293	0.0231
100	0.0009	0.0006	0.0292	0.0234
101	0.0009	0.0007	0.0292	0.0236
102	0.0009	0.0006	0.0291	0.0229
103	0.0009	0.0006	0.0290	0.0226
104	0.0009	0.0007	0.0289	0.0237
105	0.0009	0.0007	0.0289	0.0249
106	0.0009	0.0007	0.0289	0.0240
107	0.0009	0.0006	0.0288	0.0229
108	0.0009	0.0006	0.0288	0.0233
109	0.0009	0.0006	0.0285	0.0229
110	0.0009	0.0006	0.0286	0.0232
111	0.0008	0.0006	0.0284	0.0234
112	0.0008	0.0006	0.0284	0.0225
113	0.0008	0.0007	0.0283	0.0234
114	0.0008	0.0006	0.0281	0.0227
115	0.0008	0.0006	0.0282	0.0227
116	0.0008	0.0006	0.0281	0.0232
117	0.0008	0.0006	0.0279	0.0229
118	0.0008	0.0006	0.0280	0.0228
119	0.0008	0.0006	0.0278	0.0226
120	0.0008	0.0006	0.0278	0.0237
121	0.0008	0.0006	0.0278	0.0230
122	0.0008	0.0006	0.0277	0.0224
123	0.0008	0.0007	0.0277	0.0237
124	0.0008	0.0006	0.0276	0.0226
125	0.0008	0.0006	0.0275	0.0222
126	0.0008	0.0006	0.0274	0.0225
127	0.0008	0.0006	0.0274	0.0229
128	0.0008	0.0006	0.0271	0.0226
129	0.0008	0.0006	0.0272	0.0223
130	0.0008	0.0006	0.0272	0.0222
131	0.0008	0.0006	0.0270	0.0221
132	0.0008	0.0006	0.0270	0.0221
133	0.0008	0.0006	0.0268	0.0223
134	0.0008	0.0006	0.0270	0.0231
135	0.0008	0.0006	0.0268	0.0226
136	0.0007	0.0006	0.0267	0.0218
137	0.0008	0.0006	0.0268	0.0220
138	0.0007	0.0007	0.0266	0.0237
139	0.0007	0.0006	0.0266	0.0223

```
  140 |       0.0007 |       0.0006 |       0.0264 |       0.0217
  141 |       0.0007 |       0.0006 |       0.0265 |       0.0227
  142 |       0.0007 |       0.0006 |       0.0263 |       0.0223
  143 |       0.0007 |       0.0006 |       0.0262 |       0.0228
  144 |       0.0007 |       0.0006 |       0.0264 |       0.0223
  145 |       0.0007 |       0.0006 |       0.0262 |       0.0218
  146 |       0.0007 |       0.0006 |       0.0261 |       0.0218

Epoch Terbaik: 140 (Val RMSE: 0.0217)
219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step

Final Metrics:
Train MSE:    1911.3257 | Train RMSE: 43.7187
Val    MSE:    1468.6429 | Val    RMSE: 38.3229
Test   MSE:    2325.1789 | Test   RMSE: 48.2201

Training Model GRU 11 (LB=24, GRU=[128, 64, 32], Batch=64, LR=0.001)

Per-Epoch MSE & RMSE for Model GRU 11 (LB=24, GRU=[128, 64, 32],
Batch=64, LR=0.001)
Epoch |  Train MSE   |   Val MSE    |  Train RMSE  |   Val RMSE
------------------------------------------------------------
    1 |       0.0096 |       0.0028 |       0.0941 |       0.0507
    2 |       0.0028 |       0.0018 |       0.0526 |       0.0400
    3 |       0.0022 |       0.0012 |       0.0464 |       0.0327
    4 |       0.0018 |       0.0011 |       0.0422 |       0.0317
    5 |       0.0016 |       0.0010 |       0.0401 |       0.0302
    6 |       0.0016 |       0.0009 |       0.0390 |       0.0289
    7 |       0.0015 |       0.0009 |       0.0382 |       0.0274
    8 |       0.0014 |       0.0010 |       0.0369 |       0.0302
    9 |       0.0013 |       0.0008 |       0.0355 |       0.0265
   10 |       0.0013 |       0.0008 |       0.0350 |       0.0259
   11 |       0.0012 |       0.0008 |       0.0345 |       0.0258
   12 |       0.0012 |       0.0007 |       0.0338 |       0.0250
   13 |       0.0011 |       0.0007 |       0.0331 |       0.0249
   14 |       0.0011 |       0.0007 |       0.0324 |       0.0245
   15 |       0.0010 |       0.0007 |       0.0317 |       0.0252
   16 |       0.0010 |       0.0007 |       0.0312 |       0.0247
   17 |       0.0010 |       0.0007 |       0.0307 |       0.0244
   18 |       0.0009 |       0.0006 |       0.0300 |       0.0237
   19 |       0.0009 |       0.0007 |       0.0297 |       0.0246
   20 |       0.0009 |       0.0007 |       0.0291 |       0.0246
   21 |       0.0008 |       0.0007 |       0.0286 |       0.0241
   22 |       0.0008 |       0.0007 |       0.0283 |       0.0241
   23 |       0.0008 |       0.0007 |       0.0283 |       0.0255
   24 |       0.0008 |       0.0006 |       0.0278 |       0.0226
   25 |       0.0008 |       0.0006 |       0.0275 |       0.0237
   26 |       0.0007 |       0.0006 |       0.0268 |       0.0226
```

```
   27 |       0.0007 |       0.0006 |       0.0267 |       0.0239
   28 |       0.0007 |       0.0006 |       0.0267 |       0.0229
   29 |       0.0007 |       0.0006 |       0.0264 |       0.0232
   30 |       0.0007 |       0.0005 |       0.0260 |       0.0221
   31 |       0.0007 |       0.0006 |       0.0256 |       0.0237
   32 |       0.0007 |       0.0006 |       0.0259 |       0.0230
   33 |       0.0007 |       0.0006 |       0.0256 |       0.0234
   34 |       0.0007 |       0.0006 |       0.0253 |       0.0227
   35 |       0.0007 |       0.0007 |       0.0255 |       0.0242
   36 |       0.0007 |       0.0006 |       0.0253 |       0.0228
   37 |       0.0006 |       0.0006 |       0.0250 |       0.0225
   38 |       0.0006 |       0.0006 |       0.0251 |       0.0228
   39 |       0.0006 |       0.0005 |       0.0246 |       0.0218
   40 |       0.0006 |       0.0006 |       0.0245 |       0.0222
   41 |       0.0006 |       0.0006 |       0.0241 |       0.0232
   42 |       0.0006 |       0.0006 |       0.0244 |       0.0228
   43 |       0.0006 |       0.0006 |       0.0244 |       0.0216
   44 |       0.0006 |       0.0006 |       0.0242 |       0.0228
   45 |       0.0006 |       0.0005 |       0.0237 |       0.0213
   46 |       0.0006 |       0.0006 |       0.0238 |       0.0231
   47 |       0.0006 |       0.0006 |       0.0236 |       0.0221
   48 |       0.0006 |       0.0005 |       0.0239 |       0.0214
   49 |       0.0006 |       0.0005 |       0.0232 |       0.0214
   50 |       0.0006 |       0.0005 |       0.0237 |       0.0213
   51 |       0.0006 |       0.0005 |       0.0233 |       0.0220
   52 |       0.0006 |       0.0006 |       0.0233 |       0.0222
   53 |       0.0005 |       0.0005 |       0.0229 |       0.0211
   54 |       0.0006 |       0.0005 |       0.0233 |       0.0214
   55 |       0.0005 |       0.0006 |       0.0228 |       0.0223
   56 |       0.0005 |       0.0006 |       0.0228 |       0.0224
   57 |       0.0005 |       0.0005 |       0.0229 |       0.0214
   58 |       0.0005 |       0.0005 |       0.0226 |       0.0216

Epoch Terbaik: 53 (Val RMSE: 0.0211)
219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step

Final Metrics:
Train MSE:    1178.2505 | Train RMSE: 34.3257
Val    MSE:    1379.4846 | Val    RMSE: 37.1414
Test   MSE:    1817.8098 | Test   RMSE: 42.6358

Training Model GRU 12 (LB=24, GRU=[128, 64, 32], Batch=64, LR=0.0001)

Per-Epoch MSE & RMSE for Model GRU 12 (LB=24, GRU=[128, 64, 32],
Batch=64, LR=0.0001)
Epoch |  Train MSE   |   Val MSE    |  Train RMSE  |   Val RMSE
------------------------------------------------------------
    1 |       0.0250 |       0.0147 |       0.1527 |       0.1187
```

2	0.0146	0.0115	0.1201	0.1043
3	0.0119	0.0088	0.1089	0.0916
4	0.0097	0.0065	0.0980	0.0788
5	0.0077	0.0048	0.0872	0.0678
6	0.0059	0.0034	0.0766	0.0559
7	0.0048	0.0028	0.0687	0.0501
8	0.0040	0.0024	0.0627	0.0459
9	0.0034	0.0020	0.0578	0.0421
10	0.0029	0.0018	0.0539	0.0397
11	0.0026	0.0015	0.0509	0.0371
12	0.0024	0.0014	0.0487	0.0359
13	0.0023	0.0013	0.0471	0.0344
14	0.0022	0.0013	0.0460	0.0340
15	0.0021	0.0012	0.0451	0.0330
16	0.0020	0.0012	0.0443	0.0331
17	0.0020	0.0012	0.0441	0.0322
18	0.0019	0.0012	0.0433	0.0330
19	0.0018	0.0011	0.0426	0.0317
20	0.0018	0.0011	0.0422	0.0320
21	0.0018	0.0011	0.0416	0.0307
22	0.0017	0.0010	0.0413	0.0306
23	0.0017	0.0010	0.0408	0.0296
24	0.0017	0.0010	0.0405	0.0297
25	0.0017	0.0010	0.0402	0.0292
26	0.0016	0.0010	0.0400	0.0306
27	0.0016	0.0010	0.0397	0.0294
28	0.0016	0.0010	0.0395	0.0290
29	0.0016	0.0010	0.0395	0.0301
30	0.0016	0.0010	0.0393	0.0291
31	0.0016	0.0009	0.0391	0.0281
32	0.0016	0.0009	0.0392	0.0284
33	0.0015	0.0009	0.0385	0.0281
34	0.0015	0.0010	0.0387	0.0298
35	0.0015	0.0009	0.0385	0.0284
36	0.0015	0.0010	0.0385	0.0298
37	0.0015	0.0009	0.0384	0.0278
38	0.0015	0.0009	0.0383	0.0276
39	0.0015	0.0009	0.0384	0.0277
40	0.0015	0.0009	0.0379	0.0279
41	0.0015	0.0009	0.0378	0.0280
42	0.0015	0.0009	0.0377	0.0282
43	0.0015	0.0009	0.0380	0.0273
44	0.0015	0.0009	0.0376	0.0280
45	0.0015	0.0009	0.0376	0.0286
46	0.0014	0.0009	0.0376	0.0276
47	0.0014	0.0008	0.0371	0.0270
48	0.0015	0.0008	0.0378	0.0266
49	0.0014	0.0008	0.0370	0.0268
50	0.0014	0.0008	0.0371	0.0266

51	0.0014	0.0008	0.0372	0.0271
52	0.0014	0.0008	0.0369	0.0277
53	0.0014	0.0009	0.0369	0.0277
54	0.0014	0.0008	0.0367	0.0268
55	0.0014	0.0008	0.0369	0.0273
56	0.0014	0.0008	0.0365	0.0265
57	0.0014	0.0009	0.0365	0.0272
58	0.0014	0.0008	0.0365	0.0262
59	0.0014	0.0008	0.0366	0.0269
60	0.0014	0.0008	0.0363	0.0264
61	0.0014	0.0008	0.0366	0.0272
62	0.0013	0.0008	0.0361	0.0264
63	0.0013	0.0008	0.0360	0.0265
64	0.0013	0.0008	0.0361	0.0263
65	0.0013	0.0008	0.0360	0.0262
66	0.0013	0.0008	0.0358	0.0265
67	0.0013	0.0008	0.0356	0.0266
68	0.0013	0.0008	0.0358	0.0260
69	0.0013	0.0008	0.0357	0.0262
70	0.0013	0.0008	0.0356	0.0269
71	0.0013	0.0008	0.0355	0.0267
72	0.0013	0.0008	0.0356	0.0275
73	0.0013	0.0008	0.0352	0.0259
74	0.0013	0.0008	0.0350	0.0263
75	0.0013	0.0008	0.0351	0.0266
76	0.0013	0.0008	0.0351	0.0259
77	0.0013	0.0008	0.0352	0.0262
78	0.0012	0.0008	0.0348	0.0261
79	0.0012	0.0008	0.0347	0.0259
80	0.0012	0.0008	0.0347	0.0263
81	0.0012	0.0008	0.0347	0.0268
82	0.0012	0.0008	0.0347	0.0257
83	0.0012	0.0008	0.0344	0.0258
84	0.0012	0.0008	0.0345	0.0261
85	0.0012	0.0008	0.0345	0.0256
86	0.0012	0.0008	0.0343	0.0259
87	0.0012	0.0008	0.0340	0.0258
88	0.0012	0.0007	0.0341	0.0258
89	0.0012	0.0008	0.0340	0.0258
90	0.0012	0.0007	0.0339	0.0252
91	0.0012	0.0008	0.0336	0.0259
92	0.0012	0.0008	0.0336	0.0258
93	0.0012	0.0008	0.0337	0.0256
94	0.0012	0.0008	0.0336	0.0257
95	0.0012	0.0007	0.0337	0.0255
96	0.0011	0.0007	0.0335	0.0253
97	0.0011	0.0008	0.0332	0.0261
98	0.0011	0.0007	0.0332	0.0255
99	0.0011	0.0008	0.0331	0.0257

100	0.0011	0.0007	0.0331	0.0249
101	0.0011	0.0007	0.0331	0.0253
102	0.0011	0.0008	0.0328	0.0260
103	0.0011	0.0007	0.0327	0.0259
104	0.0011	0.0008	0.0326	0.0263
105	0.0011	0.0007	0.0329	0.0250
106	0.0011	0.0007	0.0325	0.0259
107	0.0011	0.0007	0.0327	0.0254
108	0.0011	0.0008	0.0325	0.0258
109	0.0011	0.0007	0.0324	0.0250
110	0.0011	0.0007	0.0325	0.0252
111	0.0011	0.0007	0.0323	0.0256
112	0.0011	0.0007	0.0322	0.0248
113	0.0011	0.0007	0.0321	0.0248
114	0.0011	0.0007	0.0320	0.0252
115	0.0011	0.0007	0.0321	0.0249
116	0.0010	0.0008	0.0320	0.0259
117	0.0010	0.0007	0.0318	0.0251
118	0.0010	0.0007	0.0318	0.0251
119	0.0010	0.0007	0.0317	0.0245
120	0.0010	0.0007	0.0317	0.0258
121	0.0010	0.0007	0.0315	0.0260
122	0.0010	0.0008	0.0316	0.0261
123	0.0010	0.0007	0.0314	0.0254
124	0.0010	0.0007	0.0313	0.0255
125	0.0010	0.0007	0.0311	0.0260
126	0.0010	0.0007	0.0314	0.0253
127	0.0010	0.0007	0.0313	0.0248
128	0.0010	0.0007	0.0311	0.0252
129	0.0010	0.0008	0.0311	0.0276
130	0.0010	0.0007	0.0313	0.0253
131	0.0010	0.0007	0.0313	0.0250
132	0.0010	0.0007	0.0310	0.0243
133	0.0010	0.0007	0.0309	0.0251
134	0.0010	0.0007	0.0310	0.0245
135	0.0010	0.0007	0.0308	0.0248
136	0.0010	0.0008	0.0307	0.0257
137	0.0010	0.0007	0.0305	0.0249
138	0.0010	0.0007	0.0306	0.0248
139	0.0010	0.0007	0.0305	0.0251
140	0.0010	0.0007	0.0306	0.0246
141	0.0009	0.0007	0.0305	0.0245
142	0.0009	0.0007	0.0303	0.0245
143	0.0009	0.0007	0.0303	0.0253
144	0.0009	0.0007	0.0301	0.0245
145	0.0009	0.0007	0.0303	0.0254
146	0.0009	0.0007	0.0303	0.0240
147	0.0009	0.0007	0.0300	0.0253
148	0.0009	0.0007	0.0301	0.0246

```
  149 |       0.0009 |       0.0007 |       0.0301 |       0.0248
  150 |       0.0009 |       0.0007 |       0.0299 |       0.0253
  151 |       0.0009 |       0.0007 |       0.0298 |       0.0242
  152 |       0.0009 |       0.0007 |       0.0297 |       0.0242
  153 |       0.0009 |       0.0007 |       0.0297 |       0.0258
  154 |       0.0009 |       0.0007 |       0.0296 |       0.0246
  155 |       0.0009 |       0.0007 |       0.0294 |       0.0242
  156 |       0.0009 |       0.0007 |       0.0295 |       0.0241

Epoch Terbaik: 146 (Val RMSE: 0.0240)
219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
27/27 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step

Final Metrics:
Train MSE:    2650.5903 | Train RMSE: 51.4839
Val    MSE:    1735.2491 | Val    RMSE: 41.6563
Test   MSE:    2755.7787 | Test   RMSE: 52.4955

Hasil metrik semua model disimpan di
'hasil_gru3layer_multi_target_pat10.csv'

Menghasilkan visualisasi kurva loss untuk semua model...
Visualisasi kurva loss MSE disimpan di folder:
loss_plots_multi_target_pat10

Model terbaik disimpan sebagai 'best_gru_multi_target_model_pat10.h5'
Konfigurasi model terbaik:
Lookback: 24
GRU Layers: [128, 64, 32]
Batch Size: 32
Learning Rate: 0.001
Best Epoch: 68
Val RMSE: 31.8361069319492
Val MSE: 1013.5377045825038
Target Features: ['PM10', 'PM2.5', 'CO', 'NO2', 'SO2', 'O3', 'Nilai
ISPU']

Menghasilkan visualisasi prediksi model terbaik untuk setiap fitur
target di data test...
Visualisasi prediksi model terbaik disimpan di folder:
best_model_predictions_multi_target_pat10

Menyimpan hasil prediksi test dan data asli test model terbaik untuk
semua fitur target ke CSV...
File 'prediksi_asli_test_model_terbaik_multi_target_pat10.csv' telah
dibuat.

--- Proses Selesai ---
```

Hasil Akhir Semua Model:

	Model	Lookback	GRU Layers	Batch Size	Learning Rate	Train RMSE	Val RMSE	Test RMSE	Train MSE	Val MSE	Test MSE	Best Epoch
8	Model GRU 9 (LB=24, GRU=[128, 64, 32], Batch=3...	24	[128, 64, 32]	32	0.0010	27.580034	31.836107	37.307337	760.658276	1013.537705	1391.837379	68
6	Model GRU 7 (LB=24, GRU=[128, 64, 32], Batch=1...	24	[128, 64, 32]	16	0.0010	31.530154	33.665400	40.943540	994.150585	1133.359190	1676.373495	39
2	Model GRU 3 (LB=12, GRU=[128, 64, 32], Batch=3...	12	[128, 64, 32]	32	0.0010	37.287454	35.837097	41.493360	1390.354190	1284.297546	1721.698941	27
4	Model GRU 5 (LB=12, GRU=[128, 64, 32], Batch=6...	12	[128, 64, 32]	64	0.0010	34.319688	36.274062	40.941763	1177.840956	1315.807577	1676.227954	56
7	Model GRU 8 (LB=24, GRU=[128, 64, 32], Batch=1...	24	[128, 64, 32]	16	0.0001	38.237665	36.387943	43.029174	1462.119022	1324.082421	1851.509846	119
0	Model GRU 1 (LB=12, GRU=[128, 64, 32], Batch=1...	12	[128, 64, 32]	16	0.0010	40.958784	36.608334	45.559335	1677.621990	1340.170108	2075.652962	26
10	Model GRU 11 (LB=24, GRU=[128, 64, 32], Batch=...	24	[128, 64, 32]	64	0.0010	34.325653	37.141413	42.635780	1178.250484	1379.484551	1817.809751	53
9	Model GRU 10 (LB=24, GRU=[128, 64, 32], Batch=...	24	[128, 64, 32]	32	0.0001	43.718711	38.322877	48.220109	1911.325656	1468.642912	2325.178927	140
3	Model GRU 4 (LB=12, GRU=[128, 64, 32], Batch=3...	12	[128, 64, 32]	32	0.0001	42.906624	38.502721	48.603547	1840.978348	1482.459528	2362.304771	152
1	Model GRU 2 (LB=12, GRU=[128, 64, 32], Batch=1...	12	[128, 64, 32]	16	0.0001	45.723870	40.112465	50.289101	2090.672272	1609.009821	2528.993656	88
11	Model GRU 12 (LB=24, GRU=[128, 64, 32], Batch=...	24	[128, 64, 32]	64	0.0001	51.483883	41.656321	52.495511	2650.590259	1735.249088	2755.778711	146
5	Model GRU 6 (LB=12, GRU=[128, 64, 32], Batch=6...	12	[128, 64, 32]	64	0.0001	58.041662	43.949359	55.541287	3368.834498	1931.546116	3084.834587	111

==========
DISPLAYING WEIGHTS AND BIASES OF THE BEST GRU MODEL (MULTI TARGET)
(Showing approximately 50 values per dimension)
==========

Successfully loaded the model from:
best_gru_multi_target_model_pat10.h5

```
--- Model Summary ---
Model: "sequential_8"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ gru_24 (GRU)                         │ (None, 24, 128)             │          52,608 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ gru_25 (GRU)                         │ (None, 24, 64)              │          37,248 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ gru_26 (GRU)                         │ (None, 32)                  │           9,408 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_8 (Dense)                      │ (None, 7)                   │             231 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
 Total params: 99,497 (388.66 KB)
 Trainable params: 99,495 (388.65 KB)
 Non-trainable params: 0 (0.00 B)
 Optimizer params: 2 (12.00 B)

--- Detailed Weights and Biases for Each Layer ---

==================================================
  LAYER 1: GRU_24 (Type: GRU)
==================================================
  > Kernel Weights (Input to Gates) (Shape: (7, 384)):
    Sub-matrix (first 7x50):
[[-0.4052, -0.1729, -0.213 ,  0.0453, -0.3292, -0.0899, -0.1206, -
0.2198,
  -0.2488, -0.3922, -0.2527,  0.0545, -0.006 , -0.0405,  0.065 , -
0.0469,
  -0.6084, -0.2539, -0.2173, -0.198 , -0.14  , -0.5475, -0.2816, -
0.0244,
  -0.0784, -0.3364,  0.1659, -0.2158, -0.4486, -0.2611, -0.0241, -
```

```
0.2181,
  -0.257 , -0.0546,  0.2854,  0.3176, -0.0314, -0.2192, -0.5119, -
0.3912,
  -0.2067, -0.4876, -0.4329, -0.4138, -0.0281, -0.4813, -0.2803,
0.0141,
  -0.3843, -0.2569],
 [-0.3532, -0.3461, -0.1359, -0.1266, -0.2278,  0.051 , -0.1479, -
0.1603,
  -0.275 , -0.3912, -0.0537, -0.0652,  0.1986,  0.0615,  0.0792, -
0.0087,
  -0.5155, -0.3126, -0.4764, -0.2547,  0.0163, -0.5164, -0.2258, -
0.1123,
  -0.2308, -0.331 ,  0.1402, -0.1194, -0.2888, -0.4904, -0.1278, -
0.1228,
  -0.1925, -0.07  ,  0.0934,  0.2384, -0.0373, -0.2059, -0.4819, -
0.3006,
  -0.109 , -0.2663, -0.3784, -0.4807, -0.1134, -0.4641, -0.208 ,
0.0108,
  -0.4157, -0.2513],
 [-0.3686,  0.0075, -0.2861, -0.0613,  0.0555, -0.0614,  0.0102, -
0.0632,
  -0.06  , -0.0902, -0.2514,  0.0388, -0.0949,  0.056 , -0.1946, -
0.1478,
  -0.3637, -0.1999, -0.1128, -0.0003, -0.1938, -0.432 ,  0.0032, -
0.1062,
   0.0064, -0.2307,  0.1535, -0.2059, -0.2706, -0.1887, -0.2268, -
0.2807,
  -0.0256, -0.114 ,  0.043 ,  0.5049, -0.2379, -0.6665, -0.0603, -
0.2233,
  -0.0695, -0.5516, -0.4038, -0.0656, -0.0833, -0.3822, -0.3558, -
0.0254,
  -0.1841, -0.2758],
 [-0.5222, -0.3383,  0.462 , -0.1407, -0.0795,  0.0361,  0.3566, -
0.1897,
  -0.2958, -0.1088, -0.2503,  0.0929,  0.358 ,  0.2157, -0.1568, -
0.1388,
  -0.5325,  0.101 , -0.0335, -0.0304, -0.128 , -0.4969,  0.0027,
0.2236,
   0.2459, -0.0959, -0.1749, -0.0928, -0.1822, -0.4057, -0.0162, -
0.254 ,
   0.0213,  0.1367,  0.019 ,  0.5223, -0.4224, -0.4644, -0.278 , -
0.2045,
  -0.2328, -0.6253, -0.1766, -0.5724,  0.1106, -0.5539, -0.0515, -
0.1313,
  -0.1491, -0.0151],
 [-0.2944, -0.3799,  0.6173, -0.0909,  0.3022,  0.2222,  0.3997, -
0.0198,
  -0.0099,  0.2139, -0.4197, -0.0929,  0.2874,  0.2697,  0.0201, -
0.045 ,
```

```
  -0.3051,  0.2578, -0.0341,  0.3113, -0.1881, -0.2882, -0.0012,
0.3046,
  -0.183 , -0.0056, -0.121 , -0.1705,  0.14  , -0.0154, -0.219 , -
0.4373,
   0.0109,  0.4088,  0.0929,  0.5019, -0.227 , -0.2628,  0.183 , -
0.5125,
  -0.1632, -0.6138, -0.2499,  0.2102, -0.2491, -0.4232,  0.0402,
0.0132,
  -0.3299,  0.0528],
 [-0.2046, -0.5984,  0.0686, -0.2798, -0.5101, -0.3074, -0.3001, -
0.2745,
  -0.0616, -0.197 ,  0.0768, -0.2763,  0.3895, -0.4566,  0.8065, -
0.0039,
  -0.3776, -0.3054, -0.8033, -0.3834,  0.0067, -0.4188, -0.1225,
0.4081,
  -0.107 ,  0.2268,  0.1059, -0.1786, -0.0449,  0.1173, -0.0249,
0.0938,
  -0.0619,  0.2253,  0.2138,  0.1315,  0.2248,  0.418 , -0.4341, -
0.3482,
  -0.2018, -0.5462, -0.286 , -0.6051,  0.1335,  0.0061, -0.1151, -
0.1834,
  -0.5789,  0.219 ],
 [-0.4755, -0.345 , -0.2118, -0.0375, -0.2518, -0.0623, -0.0309, -
0.0492,
  -0.1556, -0.4967, -0.1134,  0.1635,  0.0952,  0.1939,  0.1598, -
0.1038,
  -0.6318, -0.2959, -0.4548, -0.3395,  0.0255, -0.6081, -0.0191,
0.0426,
  -0.2178, -0.1322,  0.1977, -0.0927, -0.4164, -0.1618, -0.3105, -0.12
,
  -0.1533,  0.0234,  0.1201,  0.451 ,  0.0426, -0.1299, -0.4256, -
0.2521,
  -0.1511, -0.3092, -0.4379, -0.3605,  0.0901, -0.4925, -0.2099,
0.1472,
  -0.3311, -0.1992]]
    ... (truncated)
    Min Value: -1.175338, Max Value: 0.806453
--------------------------------------------------
  > Recurrent Kernel Weights (Hidden to Gates) (Shape: (128, 384)):
    Sub-matrix (first 50x50):
[[ 0.1073, -0.1581,  0.0565, ...,  0.0235,  0.1222, -0.0415],
 [-0.0651,  0.2548,  0.1442, ..., -0.0001,  0.1439, -0.1265],
 [ 0.0335,  0.2001,  0.5049, ..., -0.1796,  0.3113, -0.1102],
 ...,
 [ 0.0324,  0.1888,  0.3374, ...,  0.1137, -0.1651, -0.0578],
 [ 0.0046, -0.1572, -0.1669, ...,  0.1508, -0.0602,  0.0603],
 [-0.3408,  0.1061,  0.4429, ...,  0.1296, -0.1804,  0.1161]]
    ... (truncated)
    Min Value: -1.479944, Max Value: 1.342552
```

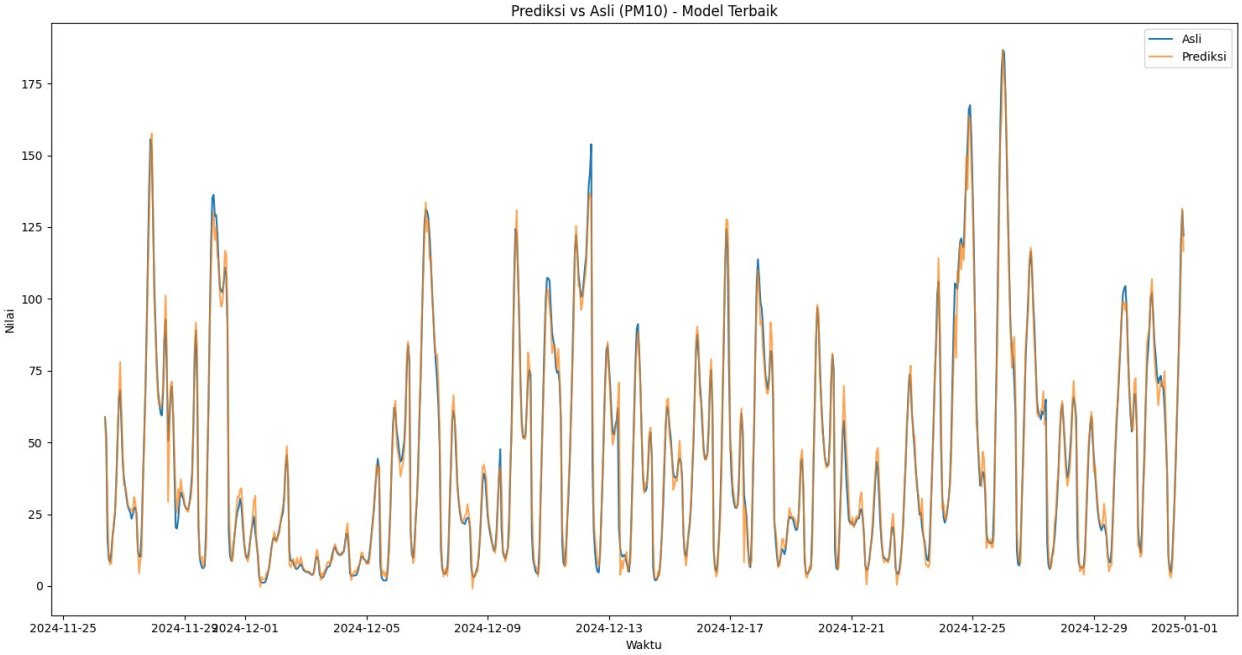
```
--------------------------------------------------
  > Bias (Shape: (2, 384)):
    Sub-matrix (first 2x50):
[[-0.6623, -0.3301, -0.0963, -0.323 , -0.1191, -0.4439, -0.2975, -
0.1799,
  -0.368 , -0.2103, -0.5757, -0.1433,  0.254 , -0.0725,  0.4329, -
0.1189,
  -0.8254, -0.1417, -0.4068, -0.3494, -0.2719, -0.6743, -0.0057,
0.1419,
  -0.2448, -0.1634, -0.2146, -0.1231, -0.5429, -0.4347, -0.6381, -
0.4444,
  -0.2102, -0.0618,  0.0544,  0.238 , -0.5136, -0.5238, -0.4259, -
0.3283,
  -0.2524, -0.99  , -0.5367, -0.5357, -0.2674, -0.6598, -0.5171,
0.0107,
  -0.5569,  0.128 ],
 [-0.6623, -0.3301, -0.0963, -0.323 , -0.1191, -0.4439, -0.2975, -
0.1799,
  -0.368 , -0.2103, -0.5757, -0.1433,  0.254 , -0.0725,  0.4329, -
0.1189,
  -0.8254, -0.1417, -0.4068, -0.3494, -0.2719, -0.6743, -0.0057,
0.1419,
  -0.2448, -0.1634, -0.2146, -0.1231, -0.5429, -0.4347, -0.6381, -
0.4444,
  -0.2102, -0.0618,  0.0544,  0.238 , -0.5136, -0.5238, -0.4259, -
0.3283,
  -0.2524, -0.99  , -0.5367, -0.5357, -0.2674, -0.6598, -0.5171,
0.0107,
  -0.5569,  0.128 ]]
    ... (truncated)
    Min Value: -1.025260, Max Value: 0.432877
--------------------------------------------------

==================================================
  LAYER 2: GRU_25 (Type: GRU)
==================================================
  > Kernel Weights (Input to Gates) (Shape: (128, 192)):
    Sub-matrix (first 50x50):
[[-0.0278,  0.3371,  0.2014, ..., -0.2524,  0.8297,  0.1835],
 [-0.1057, -0.3075,  0.2086, ...,  0.3791,  0.7787,  0.2653],
 [-0.0494,  0.6514,  0.52  , ..., -0.2422,  0.0032,  0.075 ],
 ...,
 [-0.1967, -0.169 ,  0.0248, ..., -0.0508, -0.3298, -0.0847],
 [ 0.2781,  0.2055, -0.5157, ..., -0.1538,  0.1181, -0.3042],
 [-0.2431, -0.2674, -0.0127, ..., -0.5989, -0.3372, -0.5523]]
    ... (truncated)
    Min Value: -1.550081, Max Value: 1.776468
--------------------------------------------------
  > Recurrent Kernel Weights (Hidden to Gates) (Shape: (64, 192)):
```

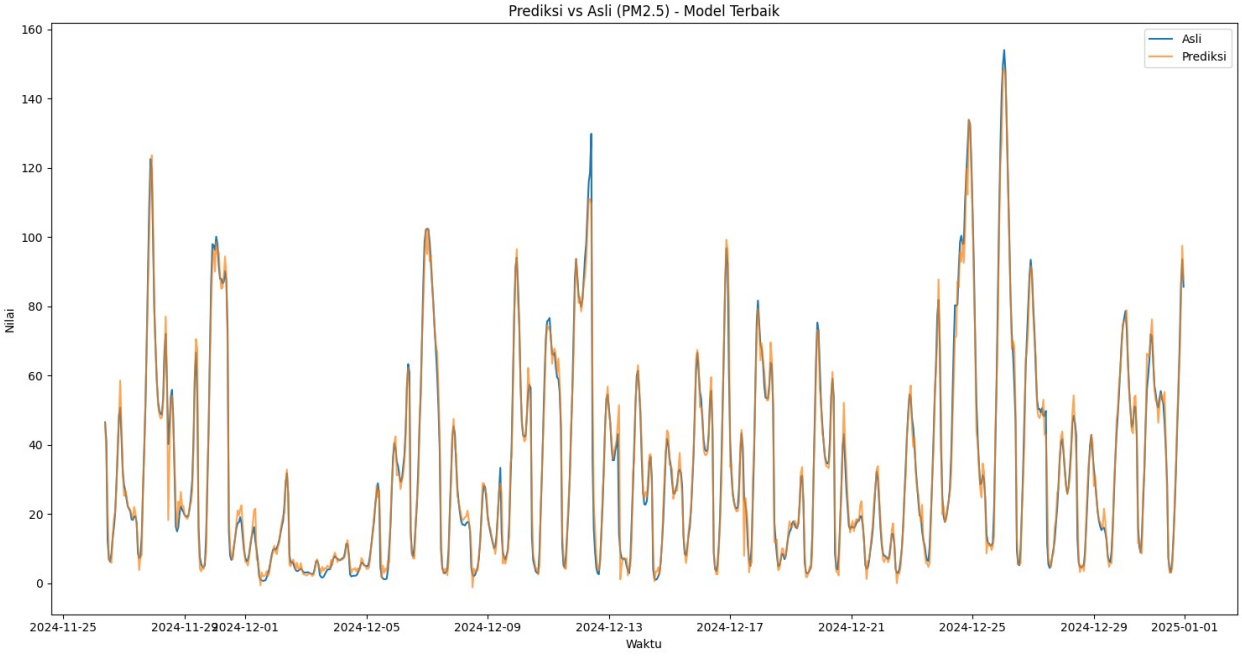
```
    Sub-matrix (first 50x50):
[[ 0.1486, -0.3607,  0.107 , ...,  0.4167, -0.0675,  0.2401],
 [-0.2215, -0.3197,  0.1359, ..., -0.0224,  0.0714, -0.1088],
 [-0.3055, -0.5917, -0.0593, ...,  0.3538,  0.3273, -0.1477],
 ...,
 [-0.4909, -0.235 ,  0.0401, ..., -0.0388, -0.7321, -0.1579],
 [ 0.0666, -0.181 ,  0.0634, ..., -0.2212,  0.4152,  0.2726],
 [ 0.173 ,  0.1429, -0.0724, ..., -0.1274,  0.271 ,  0.1476]]
    ... (truncated)
    Min Value: -1.267365, Max Value: 1.674186
--------------------------------------------------
  > Bias (Shape: (2, 192)):
    Sub-matrix (first 2x50):
[[-0.7301, -0.36  , -0.1797, -0.4211, -0.1197,  0.105 , -0.3871, -
0.4143,
  -0.4959, -0.3581, -0.2525, -0.3356, -0.3675, -0.1958, -0.451 , -
0.4299,
  -0.4954, -0.033 , -0.0077, -0.5389, -0.1216,  0.0257,  0.1489, -
0.1589,
  -0.0187, -0.1772, -0.7968,  0.0073, -0.4651, -0.1543, -0.1527, -
0.1491,
  -0.5423, -0.3076, -0.2645, -0.4968, -0.0336, -0.3651, -0.3048, -
0.3777,
  -0.461 ,  0.0442, -0.0026, -0.4278, -0.1771, -0.2527, -0.4216, -
0.0204,
  -0.3955, -0.487 ],
 [-0.7301, -0.36  , -0.1797, -0.4211, -0.1197,  0.105 , -0.3871, -
0.4143,
  -0.4959, -0.3581, -0.2525, -0.3356, -0.3675, -0.1958, -0.451 , -
0.4299,
  -0.4954, -0.033 , -0.0077, -0.5389, -0.1216,  0.0257,  0.1489, -
0.1589,
  -0.0187, -0.1772, -0.7968,  0.0073, -0.4651, -0.1543, -0.1527, -
0.1491,
  -0.5423, -0.3076, -0.2645, -0.4968, -0.0336, -0.3651, -0.3048, -
0.3777,
  -0.461 ,  0.0442, -0.0026, -0.4278, -0.1771, -0.2527, -0.4216, -
0.0204,
  -0.3955, -0.487 ]]
    ... (truncated)
    Min Value: -0.796782, Max Value: 0.206231
--------------------------------------------------

==================================================
  LAYER 3: GRU_26 (Type: GRU)
==================================================
  > Kernel Weights (Input to Gates) (Shape: (64, 96)):
    Sub-matrix (first 50x50):
[[ 0.2083, -0.2514, -0.6507, ..., -0.1204, -0.1822, -0.3129],
```

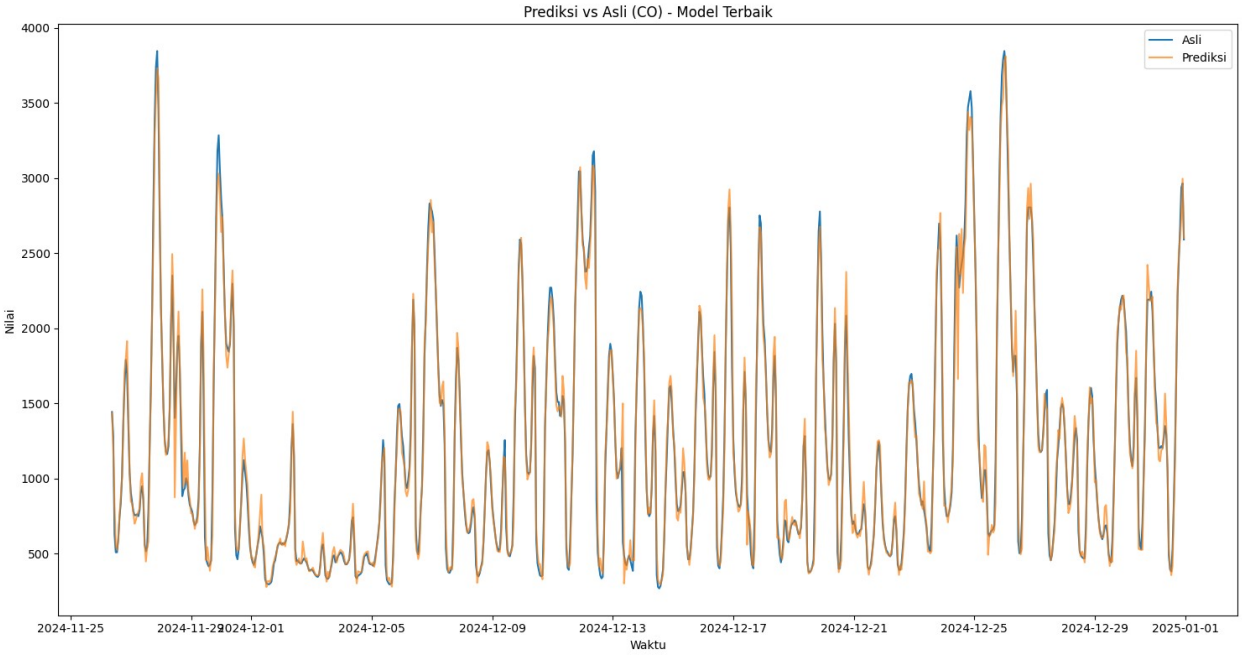
```
 [-0.1626, -0.2796, -0.7537, ..., -0.2505, -0.1682,  0.0104],
 [-0.7712, -0.1035, -0.8629, ..., -0.1831, -0.0895, -0.185 ],
 ...,
 [-0.1567, -0.3659, -0.0786, ...,  0.1016, -0.1493,  0.2111],
 [ 0.0146,  0.241 , -0.0416, ...,  0.1733, -0.3829, -0.1286],
 [ 1.2256,  0.6706,  0.2515, ..., -0.0067, -0.2633, -0.0074]]
    ... (truncated)
    Min Value: -1.549391, Max Value: 1.270864
--------------------------------------------------
  > Recurrent Kernel Weights (Hidden to Gates) (Shape: (32, 96)):
    Sub-matrix (first 32x50):
[[-0.2327, -0.1212, -0.3785, ..., -0.1145, -0.0731,  0.0477],
 [-0.4775, -0.2804, -0.367 , ...,  0.0165,  0.2206,  0.0207],
 [ 0.4522,  0.19  ,  0.5499, ..., -0.2305,  0.1207,  0.0269],
 ...,
 [-0.1663, -0.0698, -0.4121, ..., -0.0306,  0.0445,  0.0004],
 [-0.1526, -0.3139, -0.1182, ..., -0.0791,  0.0527,  0.0201],
 [-0.3312, -0.2304, -0.449 , ..., -0.0359,  0.0344,  0.0511]]
    ... (truncated)
    Min Value: -0.894200, Max Value: 0.890323
--------------------------------------------------
  > Bias (Shape: (2, 96)):
    Sub-matrix (first 2x50):
[[-0.0998, -0.3703, -0.4455, -0.4331, -0.1114, -0.2087, -0.3944, -
0.312 ,
  -0.054 , -0.3354, -0.0268, -0.3734,  0.4526, -0.4741, -0.2138, -
0.4305,
  -0.2926, -0.1281, -0.6684, -0.536 , -0.4899, -0.0567, -0.2178,
0.1846,
  -0.5579, -0.3643, -0.3255, -0.1638, -0.3487, -0.3348, -0.4215, -
0.3592,
   0.0782, -0.0323,  0.0536,  0.0634,  0.0562,  0.0962,  0.1532,
0.1008,
   0.0679,  0.006 ,  0.0812,  0.1195,  0.0487,  0.0979,  0.2531,
0.0414,
   0.0235, -0.0516],
 [-0.0998, -0.3703, -0.4455, -0.4331, -0.1114, -0.2087, -0.3944, -
0.312 ,
  -0.054 , -0.3354, -0.0268, -0.3734,  0.4526, -0.4741, -0.2138, -
0.4305,
  -0.2926, -0.1281, -0.6684, -0.536 , -0.4899, -0.0567, -0.2178,
0.1846,
  -0.5579, -0.3643, -0.3255, -0.1638, -0.3487, -0.3348, -0.4215, -
0.3592,
   0.0782, -0.0323,  0.0536,  0.0634,  0.0562,  0.0962,  0.1532,
0.1008,
   0.0679,  0.006 ,  0.0812,  0.1195,  0.0487,  0.0979,  0.2531,
0.0414,
   0.0235, -0.0516]]
```

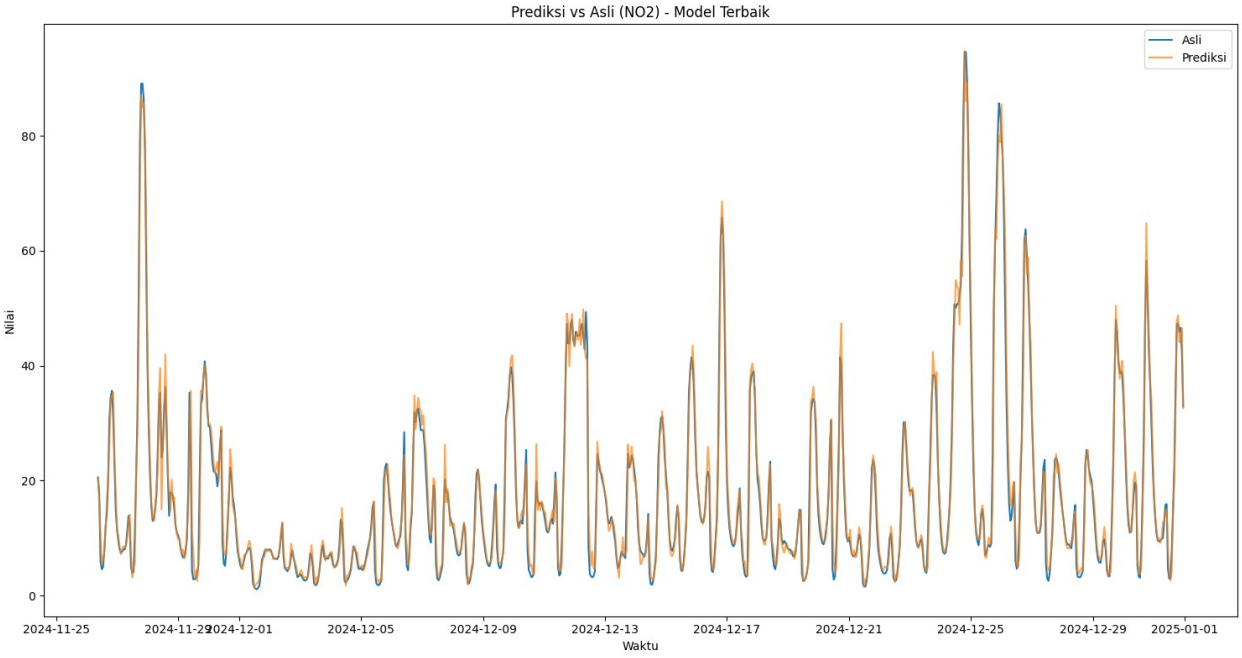
```
    ... (truncated)
    Min Value: -0.668431, Max Value: 0.452589
--------------------------------------------------

==================================================
  LAYER 4: DENSE_8 (Type: Dense)
==================================================
  > Weights (Shape: (32, 7)):
    Sub-matrix (first 32x7):
[[-0.0354,  0.0903,  0.0434,  0.0125, -0.0093, -0.1184,  0.0118],
 [ 0.2812,  0.3177,  0.1035,  0.1047,  0.2216, -0.2423,  0.1498],
 [ 0.1311, -0.1409,  0.1905,  0.025 ,  0.287 ,  0.0442, -0.2657],
 [ 0.0855,  0.0476, -0.2604,  0.2559, -0.0856, -0.2178, -0.0579],
 [ 0.209 , -0.1226,  0.0631,  0.1016,  0.0053, -0.0154, -0.1819],
 [ 0.1526, -0.1119, -0.2649, -0.026 ,  0.146 , -0.0833, -0.0885],
 [ 0.0558,  0.093 , -0.1113,  0.2299, -0.1367,  0.266 , -0.2058],
 [-0.0617, -0.0002, -0.3333, -0.3682, -0.328 , -0.1916, -0.0888],
 [-0.0051, -0.1014,  0.0395,  0.2523, -0.0844,  0.0428,  0.0702],
 [ 0.1098,  0.2386, -0.226 , -0.145 , -0.0383,  0.2092,  0.0447],
 [-0.0519, -0.0285, -0.037 ,  0.1323,  0.181 , -0.1197, -0.1021],
 [-0.236 , -0.1922, -0.0338,  0.1086, -0.1605, -0.0728, -0.3049],
 [-0.0357,  0.0368, -0.2295,  0.0482, -0.1623, -0.0715, -0.2292],
 [ 0.2802,  0.1378,  0.044 ,  0.0092, -0.1828,  0.2921,  0.0199],
 [-0.0929, -0.0019, -0.0359,  0.3111, -0.258 , -0.0509,  0.062 ],
 [-0.129 , -0.0863,  0.0166,  0.2504, -0.2452, -0.3101, -0.1255],
 [-0.0163, -0.2189,  0.2041, -0.1256, -0.1986, -0.2781, -0.1851],
 [ 0.1353, -0.1524,  0.1322,  0.1068,  0.1957,  0.1363,  0.212 ],
 [-0.1708,  0.1397,  0.2156, -0.1922,  0.1268,  0.2738, -0.127 ],
 [ 0.1172,  0.2761,  0.0552,  0.0408, -0.14  ,  0.0241, -0.2889],
 [-0.1428, -0.1414,  0.2085,  0.0081, -0.105 ,  0.1049, -0.2505],
 [ 0.1984,  0.0747,  0.2537,  0.1782,  0.0518, -0.0034, -0.2311],
 [ 0.0734,  0.0754,  0.018 ,  0.1546,  0.2334, -0.1873, -0.0171],
 [-0.0162,  0.1414, -0.1915,  0.2062, -0.0532, -0.0973, -0.0011],
 [ 0.008 , -0.1164,  0.1148,  0.0345, -0.1365,  0.1381,  0.0583],
 [ 0.2497,  0.3177,  0.2034,  0.0147,  0.0396,  0.3239,  0.2706],
 [-0.05  ,  0.075 ,  0.0055,  0.2395, -0.0922, -0.1863,  0.1492],
 [-0.1607,  0.0404, -0.0894,  0.3162, -0.0098,  0.0901,  0.1728],
 [-0.2084, -0.0694, -0.3164,  0.2388,  0.1234, -0.0894, -0.31  ],
 [ 0.0868,  0.2506,  0.0138, -0.2576, -0.187 , -0.2304, -0.0198],
 [-0.1191, -0.2277,  0.329 ,  0.3105, -0.0787,  0.1478,  0.2337],
 [ 0.1803,  0.2008,  0.2904,  0.2726,  0.3023, -0.2828, -0.0729]]
    Min Value: -0.368225, Max Value: 0.329004
--------------------------------------------------
  > Biases (Shape: (7,)):
    [0.05939773 0.06612913 0.05656123 0.05120447 0.04729079 0.04874356
 0.09021664]
    Min Value: 0.047291, Max Value: 0.090217
--------------------------------------------------
```

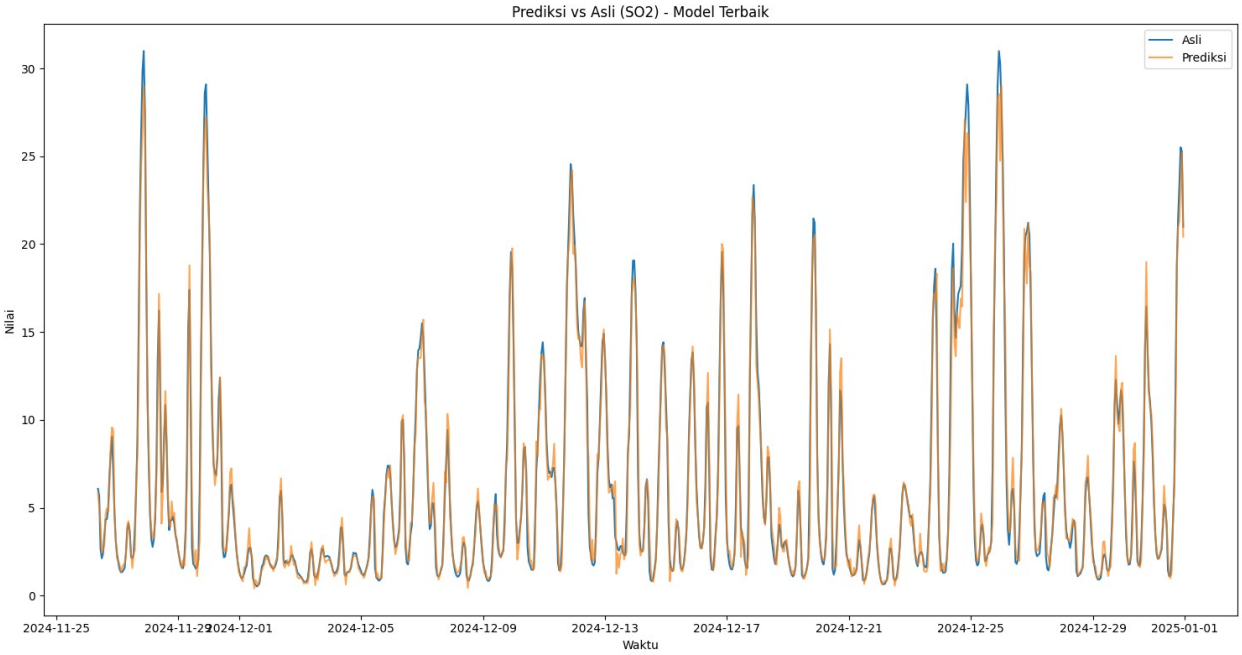
```
================================================================================
  WEIGHTS AND BIASES DISPLAY COMPLETE
================================================================================
```

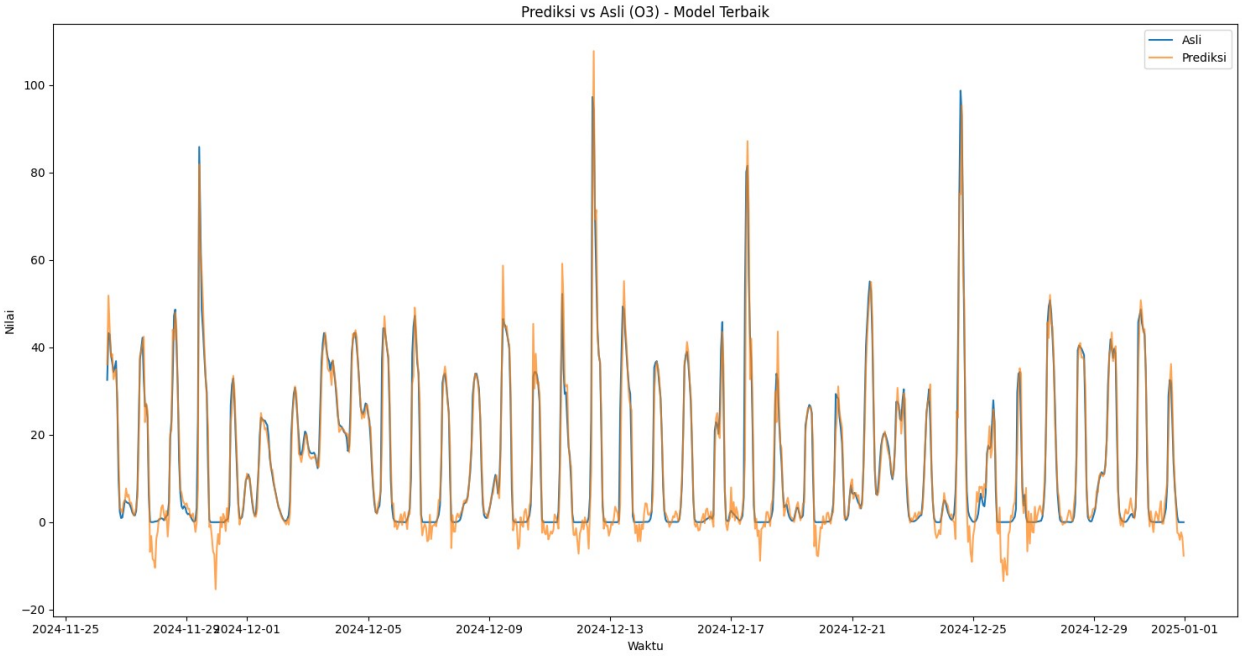


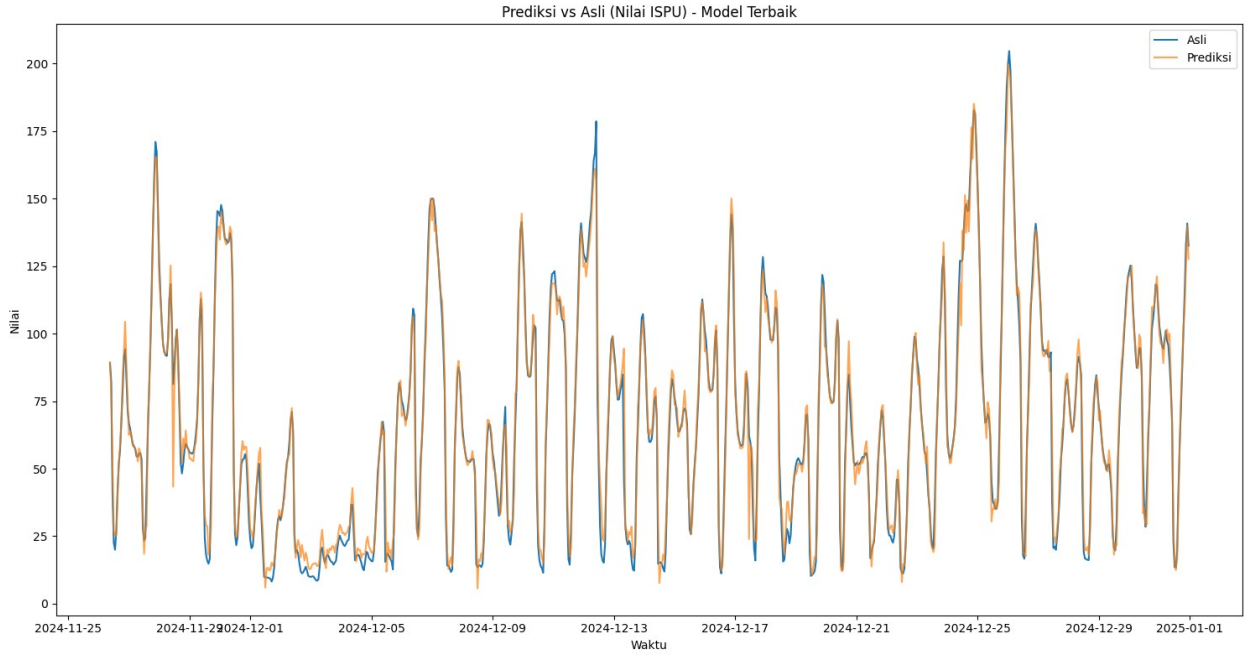



Prediksi vs Asli (CO) - Model Terbaik
Asli
Prediksi
Nilai
Waktu
4000
3500
3000
2500
2000
1500
1000
500
2024-11-25
2024-11-29
2024-12-01
2024-12-05
2024-12-09
2024-12-13
2024-12-17
2024-12-21
2024-12-25
2024-12-29
2025-01-01


Prediksi vs Asli (NO2) - Model Terbaik
Asli
Prediksi
Nilai
Waktu
80
60
40
20
0
2024-11-25
2024-11-29
2024-12-01
2024-12-05
2024-12-09
2024-12-13
2024-12-17
2024-12-21
2024-12-25
2024-12-29
2025-01-01


Prediksi vs Asli (SO2) - Model Terbaik
Asli
Prediksi
Nilai
Waktu
0
5
10
15
20
25
30
2024-11-25
2024-11-29
2024-12-01
2024-12-05
2024-12-09
2024-12-13
2024-12-17
2024-12-21
2024-12-25
2024-12-29
2025-01-01


Prediksi vs Asli (O3) - Model Terbaik
Asli
Prediksi
Nilai
Waktu
−20
0
20
40
60
80
100
2024-11-25
2024-11-29
2024-12-01
2024-12-05
2024-12-09
2024-12-13
2024-12-17
2024-12-21
2024-12-25
2024-12-29
2025-01-01



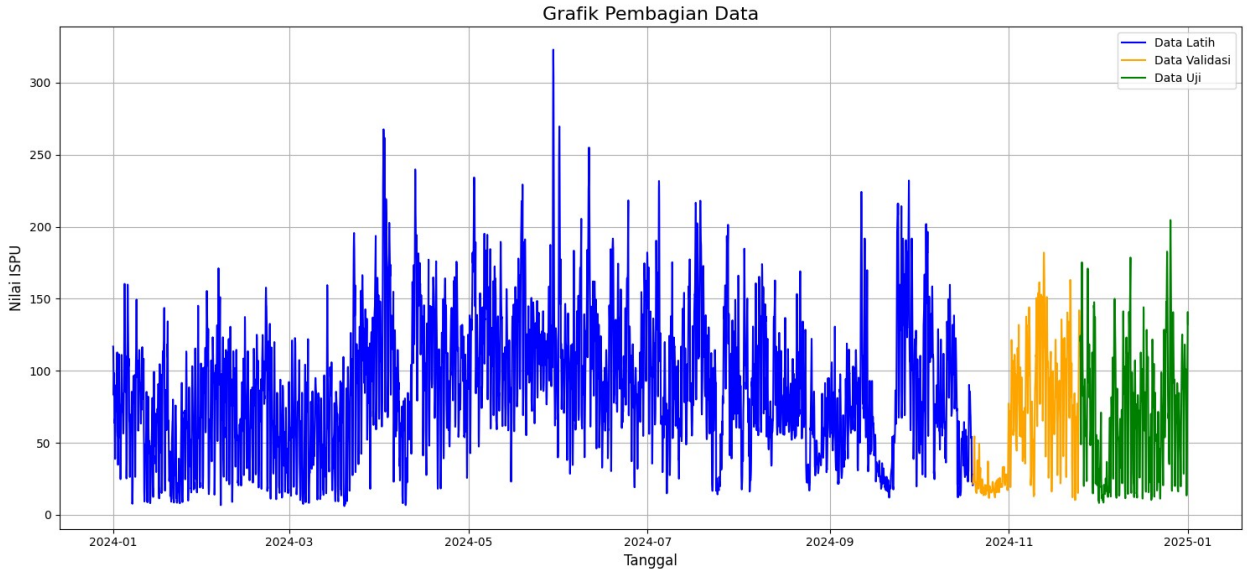
TEST

```
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
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import GRU, Dense, Input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import backend as K
from tensorflow.keras.losses import MeanSquaredError
from math import sqrt
import os

# --- 1. Custom Metric ---
def root_mean_squared_error(y_true, y_pred):
    return K.sqrt(K.mean(K.square(y_pred - y_true)))

# --- 2. Load & Preprocess Data ---
data = pd.read_csv('/content/05-data_ISPU_2024.csv')
data['Waktu'] = pd.to_datetime(data['Waktu'])

# Drop 'Kategori' jika ada dan tidak diinginkan sebagai fitur
if 'Kategori' in data.columns:
    data = data.drop(columns=['Kategori'])

# Define ALL columns as target columns for prediction (6 pollutants +
```

```
'Nilai ISPU')
target_columns = data.drop(columns=['Waktu']).columns.tolist()

# Separate input features and the target features
timestamps = data['Waktu'].values

# Input features will include all columns except 'Waktu'
input_features_df = data.drop(columns=['Waktu'])
target_feature_df = data.drop(columns=['Waktu']) # TARGET JUGA SEMUA 7
KOLOM

input_values = input_features_df.values
target_values = target_feature_df.values

# Assert the number of input features
num_input_features = input_features_df.shape[1]
print(f"Number of input features: {num_input_features}")
assert num_input_features == 7, f"Expected 7 input features (6
pollutants + ISPU), got {num_input_features}"
input_feature_names = input_features_df.columns.tolist()
print(f"Input feature names: {input_feature_names}")
print(f"Output target names: {target_columns}")


# --- 3. Split Data ---
n_total = len(input_values)
n_train = int(n_total * 0.80)
n_val = int(n_total * 0.10)
n_test = n_total - n_train - n_val

# Splitting input features data
train_input_data = input_values[:n_train]
val_input_data = input_values[n_train:n_train + n_val]
test_input_data = input_values[n_train + n_val:]

# Splitting target feature data
train_target_data = target_values[:n_train]
val_target_data = target_values[n_train:n_train + n_val]
test_target_data = target_values[n_train + n_val:]

train_time = timestamps[:n_train]
val_time = timestamps[n_train:n_train + n_val]
test_time = timestamps[n_train + n_val:]

# ==== Plot Data Division ====
print("\n Grafik Pembagian Data ")
plt.figure(figsize=(15, 7))
plt.plot(data['Waktu'].iloc[:n_train], data['Nilai
ISPU'].iloc[:n_train], 'blue', label='Data Latih')
plt.plot(data['Waktu'].iloc[n_train-1 : n_train + n_val], data['Nilai
```

```
ISPU'].iloc[n_train-1 : n_train + n_val], 'orange', label='Data
Validasi')
plt.plot(data['Waktu'].iloc[n_train + n_val - 1 : ], data['Nilai
ISPU'].iloc[n_train + n_val - 1 : ], 'green', label='Data Uji')

plt.title('Grafik Pembagian Data', fontsize=16)
plt.xlabel('Tanggal', fontsize=12)
plt.ylabel('Nilai ISPU', fontsize=12)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()


print("\n Data Setelah Normalisasi")

# --- 4. Normalisasi ---
# Create separate scalers for input features and the target feature
input_scaler = MinMaxScaler()
target_scaler = MinMaxScaler() # target_scaler now scales all 7 output
features

train_input_scaled = input_scaler.fit_transform(train_input_data)
train_target_scaled = target_scaler.fit_transform(train_target_data)

val_input_scaled = input_scaler.transform(val_input_data)
val_target_scaled = target_scaler.transform(val_target_data)

test_input_scaled = input_scaler.transform(test_input_data)
test_target_scaled = target_scaler.transform(test_target_data)


# --- 5. Sequence Maker ---
def create_sequences(input_data, target_data, lookback):
    """
    Creates sequences for time series prediction.
    X contains the input features for 'lookback' steps.
    y contains ALL target features at the next step.
    """
    X, y = [], []
    for i in range(len(input_data) - lookback):
        X.append(input_data[i:i + lookback])
        y.append(target_data[i + lookback]) # y now appends all target
features
    return np.array(X), np.array(y)

# --- 6. Hyperparameters ---
lookback = 24 # Menggunakan lookback yang Anda berikan
grulayers = [128, 64, 32] # Arsitektur GRU yang Anda berikan
batch_size = 32
```

```
lr = 0.001
epochs = 68

# --- 7. Buat Dataset Berdasarkan Lookback ---
X_train, y_train = create_sequences(train_input_scaled,
train_target_scaled, lookback)
X_val, y_val = create_sequences(val_input_scaled, val_target_scaled,
lookback)
X_test, y_test = create_sequences(test_input_scaled,
test_target_scaled, lookback)

Number of input features: 7
Input feature names: ['PM10', 'PM2.5', 'CO', 'NO2', 'SO2', 'O3',
'Nilai ISPU']
Output target names: ['PM10', 'PM2.5', 'CO', 'NO2', 'SO2', 'O3',
'Nilai ISPU']

 Grafik Pembagian Data
```




```
 Data Setelah Normalisasi

# --- 8. Build Model GRU ---
model = Sequential()
# Input shape: (lookback, num_input_features) - num_input_features = 7
model.add(Input(shape=(lookback, X_train.shape[2])))
model.add(GRU(grulayers[0], return_sequences=True))
model.add(GRU(grulayers[1], return_sequences=True))
model.add(GRU(grulayers[2]))
model.add(Dense(len(target_columns))) # OUTPUT LAYER SEKARANG
MEMPREDIKSI SEMUA 7 KOLOM
```

```
model.compile(optimizer=Adam(learning_rate=lr),
              loss='mse',
              metrics=['mse', root_mean_squared_error])

print("\n--- Model Summary ---")
model.summary()

# --- 9. Training Manual Per Epoch & Mengumpulkan Metrik ---
train_mse_list = []
val_mse_list = []
test_mse_list = []
train_rmse_list = []
val_rmse_list = []
test_rmse_list = []

print(f"\nMemulai pelatihan selama {epochs} epoch...")
for epoch in range(epochs):
    history = model.fit(X_train, y_train,
                        validation_data=(X_val, y_val),
                        epochs=1,
                        batch_size=batch_size,
                        verbose=0)

    y_test_pred_norm = model.predict(X_test, verbose=0)

    current_test_mse = mean_squared_error(y_test, y_test_pred_norm)
    current_test_rmse = sqrt(current_test_mse)

    train_mse_list.append(history.history['mse'][0])
    val_mse_list.append(history.history['val_mse'][0])
    train_rmse_list.append(history.history['root_mean_squared_error']
[0])

val_rmse_list.append(history.history['val_root_mean_squared_error']
[0])
    test_mse_list.append(current_test_mse)
    test_rmse_list.append(current_test_rmse)

    print(f"Epoch {epoch+1:3d}: Train MSE={train_mse_list[-1]:.4f},
Val MSE={val_mse_list[-1]:.4f}, Test MSE={current_test_mse:.4f} | "
          f"Train RMSE={train_rmse_list[-1]:.4f}, Val
RMSE={val_rmse_list[-1]:.4f}, Test RMSE={current_test_rmse:.4f}")

# --- 10. Evaluasi Final untuk Semua Set (Normalized & Denormalized)
---
y_train_pred_norm = model.predict(X_train, verbose=0)
y_val_pred_norm = model.predict(X_val, verbose=0)
y_test_pred_norm = model.predict(X_test, verbose=0)

y_train_true_denorm = target_scaler.inverse_transform(y_train)
```

```
y_train_pred_denorm =
target_scaler.inverse_transform(y_train_pred_norm)
#y_train_pred_denorm = np.clip(y_train_pred_denorm, 0, None) #
Pastikan nilai tidak negatif

y_val_true_denorm = target_scaler.inverse_transform(y_val)
y_val_pred_denorm = target_scaler.inverse_transform(y_val_pred_norm)
#y_val_pred_denorm = np.clip(y_val_pred_denorm, 0, None) # Pastikan
nilai tidak negatif

y_test_true_denorm = target_scaler.inverse_transform(y_test)
y_test_pred_denorm = target_scaler.inverse_transform(y_test_pred_norm)
#y_test_pred_denorm = np.clip(y_test_pred_denorm, 0, None) # Pastikan
nilai tidak negatif


# Hitung Metrik Final (Normalized)
final_train_mse_norm = mean_squared_error(y_train, y_train_pred_norm)
final_val_mse_norm = mean_squared_error(y_val, y_val_pred_norm)
final_test_mse_norm = mean_squared_error(y_test, y_test_pred_norm)

final_train_rmse_norm = sqrt(final_train_mse_norm)
final_val_rmse_norm = sqrt(final_val_mse_norm)
final_test_rmse_norm = sqrt(final_test_mse_norm)

# Hitung Metrik Final (Denormalized)
final_train_mse_denorm = mean_squared_error(y_train_true_denorm,
y_train_pred_denorm)
final_val_mse_denorm = mean_squared_error(y_val_true_denorm,
y_val_pred_denorm)
final_test_mse_denorm = mean_squared_error(y_test_true_denorm,
y_test_pred_denorm)

final_train_rmse_denorm = sqrt(final_train_mse_denorm)
final_val_rmse_denorm = sqrt(final_val_mse_denorm)
final_test_rmse_denorm = sqrt(final_test_mse_denorm)


# --- Tampilan Semua Metrik Final ---
print("\n" + "="*50)
print("  FINAL MODEL METRICS SUMMARY (MULTI-TARGET Prediction)")
print("="*50)

print("\n--- Normalized Metrics ---")
print(f"Train MSE:  {final_train_mse_norm:.4f}")
print(f"Val   MSE:  {final_val_mse_norm:.4f}")
print(f"Test  MSE:  {final_test_mse_norm:.4f}")
print(f"Train RMSE: {final_train_rmse_norm:.4f}")
print(f"Val   RMSE: {final_val_rmse_norm:.4f}")
print(f"Test  RMSE: {final_test_rmse_norm:.4f}")
```

```
print("\n--- Denormalized Metrics ---")
print(f"Train MSE:  {final_train_mse_denorm:.4f}")
print(f"Val   MSE:  {final_val_mse_denorm:.4f}")
print(f"Test  MSE:  {final_test_mse_denorm:.4f}")
print(f"Train RMSE: {final_train_rmse_denorm:.4f}")
print(f"Val   RMSE: {final_val_rmse_denorm:.4f}")
print(f"Test  RMSE: {final_test_rmse_denorm:.4f}")
print("="*50)


# --- 11. Simpan Model Terlatih ---
model_save_path = 'final_gru_multi_target_model_7_inputs_7_outputs.h5'
# Nama file baru
model.save(model_save_path)
print(f"\nModel terlatih disimpan sebagai '{model_save_path}'")

# --- 12. Visualisasi Kurva Loss (HANYA MSE) ---
output_loss_folder = 'loss_plots_multi_target_7_inputs_7_outputs' #
Nama folder baru
os.makedirs(output_loss_folder, exist_ok=True)

print(f"\nMenghasilkan visualisasi kurva MSE dan menyimpannya di
folder: {output_loss_folder}")

# --- Plot MSE Curves ---
plt.figure(figsize=(15, 8))
plt.plot(train_mse_list, label='Train MSE')
plt.plot(val_mse_list, label='Val MSE')
plt.title(f'Train vs Val MSE (Multi-Target)') # Judul disesuaikan
plt.xlabel('Epoch')
plt.ylabel('MSE')
plt.ylim(0, 0.02)
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.savefig(os.path.join(output_loss_folder,
'mse_train_val_curve.png'))
plt.close()

plt.figure(figsize=(15, 8))
plt.plot(test_mse_list, label='Test MSE', color='orange')
plt.title(f'Test MSE (Multi-Target)') # Judul disesuaikan
plt.xlabel('Epoch')
plt.ylabel('MSE')
plt.ylim(0, 0.02)
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.savefig(os.path.join(output_loss_folder, 'mse_test_curve.png'))
```

```
plt.close()

plt.figure(figsize=(15, 8))
plt.plot(train_mse_list, label='Train MSE')
plt.plot(val_mse_list, label='Val MSE')
plt.plot(test_mse_list, label='Test MSE')
plt.title(f'Train, Val, and Test MSE (Multi-Target)') # Judul
disesuaikan
plt.xlabel('Epoch')
plt.ylabel('MSE')
plt.ylim(0, 0.02)
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.savefig(os.path.join(output_loss_folder,
'mse_combined_curve.png'))
plt.close()


# --- 13. Visualisasi Prediksi vs Asli untuk Setiap Target ---
output_pred_folder = 'predictions_multi_target_7_inputs_7_outputs' #
Nama folder baru
os.makedirs(output_pred_folder, exist_ok=True)

print(f"\nMenghasilkan visualisasi prediksi vs asli untuk setiap fitur
target di data test dan menyimpannya di folder: {output_pred_folder}")

start_index_test_for_plot = lookback # Menggunakan lookback yang Anda
berikan
test_time_plot = test_time[start_index_test_for_plot :
start_index_test_for_plot + len(y_test_true_denorm)]

for i, feature_name in enumerate(target_columns):
    plt.figure(figsize=(15, 8))
    plt.plot(test_time_plot, y_test_true_denorm[:, i], label='Asli') #
Plot untuk setiap kolom target
    plt.plot(test_time_plot, y_test_pred_denorm[:, i],
label='Prediksi', alpha=0.7) # Plot untuk setiap kolom target
    plt.title(f'Prediksi vs Asli ({feature_name})')
    plt.xlabel('Waktu')
    plt.ylabel('Nilai')
    plt.legend()
    # Menghapus pengaturan ylim dinamis, kembali ke skala otomatis
matplotlib
    plt.tight_layout()
    plt.savefig(os.path.join(output_pred_folder,
f'prediction_vs_actual_{feature_name.replace(" ", "_")}.png'))


# --- 14. Simpan Hasil Prediksi Test ke CSV ---
```

```
print(f"\nMenyimpan hasil prediksi test dan data asli test untuk semua
fitur target ke CSV...")
df_results_multi_target = pd.DataFrame({'Waktu': test_time_plot})
for i, feature_name in enumerate(target_columns):
    df_results_multi_target[f'Asli_{feature_name}'] =
y_test_true_denorm[:, i]
    df_results_multi_target[f'Prediksi_{feature_name}'] =
y_test_pred_denorm[:, i]

df_results_multi_target.to_csv('prediksi_asli_test_multi_target_7_inpu
ts_7_outputs.csv', index=False) # Nama file baru
print("File 'prediksi_asli_test_multi_target_7_inputs_7_outputs.csv'
telah dibuat.")

## Menampilkan Bobot dan Bias Model Terlatih

print("\n" + "="*70)
print("  MENAMPILKAN BOBOT DAN BIAS DARI MODEL GRU YANG TELAH DILATIH
(MULTI-TARGET)") # Judul disesuaikan
print("  (Menampilkan sekitar 50 nilai per dimensi untuk ringkasan)")
print("="*70)

model_path = model_save_path # Menggunakan path model yang sudah
disimpan

if os.path.exists(model_path):
    try:
        loaded_model = load_model(model_path, custom_objects={
            'root_mean_squared_error': root_mean_squared_error,
            'mse': MeanSquaredError()
        })

        print(f"\nBerhasil memuat model dari: {model_path}")
        print("\n--- Ringkasan Model yang Dimuat ---")
        loaded_model.summary()

        print("\n--- Detail Bobot dan Bias untuk Setiap Layer ---")
        for i, layer in enumerate(loaded_model.layers):
            weights_and_biases = layer.get_weights()

            if weights_and_biases:
                print(f"\n" + "="*50)
                print(f"  LAYER {i+1}: {layer.name.upper()} (Tipe:
{layer.__class__.__name__})")
                print("="*50)

                for j, wb_tensor in enumerate(weights_and_biases):
                    param_type = ""
                    if layer.name.startswith('gru'):
                        if j == 0: param_type = "Kernel Weights (Input
```

```
to Gates)"
                            elif j == 1: param_type = "Recurrent Kernel
Weights (Hidden to Gates)"
                            elif j == 2: param_type = "Bias"
                        elif layer.name.startswith('dense'):
                            if j == 0: param_type = "Weights"
                            elif j == 1: param_type = "Biases"
                        else:
                            param_type = f"Parameter {j}"

                        print(f"  > {param_type} (Shape:
{wb_tensor.shape}):")

                        if wb_tensor.ndim == 1:
                            display_len = min(len(wb_tensor), 50)
                            if display_len < len(wb_tensor):
                                print(f"    [ {', '.join(f'{x:.4f}' for x
in wb_tensor[:display_len])} ... (dipotong) ]")
                            else:
                                print(f"    {wb_tensor}")
                            print(f"    Min Value:
{np.min(wb_tensor):.6f}, Max Value: {np.max(wb_tensor):.6f}")

                        elif wb_tensor.ndim == 2:
                            rows_to_display = min(wb_tensor.shape[0], 50)
                            cols_to_display = min(wb_tensor.shape[1], 50)

                            print(f"    Sub-matriks (pertama
{rows_to_display}x{cols_to_display}):")

print(np.array2string(wb_tensor[:rows_to_display, :cols_to_display],
precision=4, separator=', ', suppress_small=True))

                            if rows_to_display < wb_tensor.shape[0] or
cols_to_display < wb_tensor.shape[1]:
                                print("    ... (dipotong)")

                            print(f"    Min Value:
{np.min(wb_tensor):.6f}, Max Value: {np.max(wb_tensor):.6f}")
                        else:
                            print(f"    {np.array2string(wb_tensor,
precision=4, separator=', ', suppress_small=True)}")
                            print(f"    Min Value:
{np.min(wb_tensor):.6f}, Max Value: {np.max(wb_tensor):.6f}")
                        print("-" * 50)

                else:
                    print(f"\nLayer {i+1}: {layer.name.upper()} (Tipe:
{layer.__class__.__name__}) - Tidak memiliki bobot yang bisa
dilatih.")
```

```
    except Exception as e:
        print(f"\nError memuat atau memeriksa model: {e}")
        print("Pastikan:")
        print(f"1. File model '{model_path}' ada di direktori saat
ini.")
        print(f"2. Metrik kustom 'root_mean_squared_error' dan 'mse'
(MeanSquaredError) didefinisikan secara identik seperti saat
disimpan/dikompilasi.")
else:
    print(f"\nError: File model '{model_path}' tidak ditemukan.")
    print(f"Pastikan Anda telah melatih model dan menyimpannya sebagai
'{model_save_path}' di direktori yang sama.")

print("\n" + "="*70)
print("  TAMPILAN BOBOT DAN BIAS SELESAI")
print("="*70)
```

```
--- Model Summary ---
```

```
Model: "sequential_2"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                          ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ gru_6 (GRU)                           │ (None, 24, 128)             │
52,608 │
├───────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ gru_7 (GRU)                           │ (None, 24, 64)              │
37,248 │
├───────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ gru_8 (GRU)                           │ (None, 32)                  │
9,408 │
├───────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_2 (Dense)                       │ (None, 7)                   │
231 │
└───────────────────────────────────────┴─────────────────────────────┴─────────────────┘
```

```
 Total params: 99,495 (388.65 KB)
```

```
 Trainable params: 99,495 (388.65 KB)
```

```
 Non-trainable params: 0 (0.00 B)
```

```
Memulai pelatihan selama 68 epoch...
Epoch   1: Train MSE=0.0078, Val MSE=0.0021, Test MSE=0.0028 | Train
RMSE=0.0821, Val RMSE=0.0418, Test RMSE=0.0532
Epoch   2: Train MSE=0.0023, Val MSE=0.0012, Test MSE=0.0017 | Train
RMSE=0.0474, Val RMSE=0.0321, Test RMSE=0.0416
Epoch   3: Train MSE=0.0018, Val MSE=0.0010, Test MSE=0.0014 | Train
RMSE=0.0418, Val RMSE=0.0283, Test RMSE=0.0377
Epoch   4: Train MSE=0.0015, Val MSE=0.0010, Test MSE=0.0014 | Train
RMSE=0.0384, Val RMSE=0.0284, Test RMSE=0.0368
Epoch   5: Train MSE=0.0014, Val MSE=0.0009, Test MSE=0.0012 | Train
RMSE=0.0363, Val RMSE=0.0271, Test RMSE=0.0348
Epoch   6: Train MSE=0.0013, Val MSE=0.0008, Test MSE=0.0011 | Train
RMSE=0.0352, Val RMSE=0.0254, Test RMSE=0.0332
Epoch   7: Train MSE=0.0013, Val MSE=0.0008, Test MSE=0.0011 | Train
RMSE=0.0350, Val RMSE=0.0271, Test RMSE=0.0335
Epoch   8: Train MSE=0.0012, Val MSE=0.0007, Test MSE=0.0011 | Train
RMSE=0.0339, Val RMSE=0.0256, Test RMSE=0.0324
Epoch   9: Train MSE=0.0011, Val MSE=0.0009, Test MSE=0.0012 | Train
RMSE=0.0328, Val RMSE=0.0284, Test RMSE=0.0340
Epoch  10: Train MSE=0.0011, Val MSE=0.0007, Test MSE=0.0010 | Train
RMSE=0.0320, Val RMSE=0.0252, Test RMSE=0.0311
Epoch  11: Train MSE=0.0010, Val MSE=0.0007, Test MSE=0.0009 | Train
RMSE=0.0309, Val RMSE=0.0252, Test RMSE=0.0303
Epoch  12: Train MSE=0.0009, Val MSE=0.0007, Test MSE=0.0010 | Train
RMSE=0.0301, Val RMSE=0.0249, Test RMSE=0.0311
Epoch  13: Train MSE=0.0009, Val MSE=0.0006, Test MSE=0.0009 | Train
RMSE=0.0295, Val RMSE=0.0231, Test RMSE=0.0304
Epoch  14: Train MSE=0.0009, Val MSE=0.0007, Test MSE=0.0009 | Train
RMSE=0.0288, Val RMSE=0.0233, Test RMSE=0.0296
Epoch  15: Train MSE=0.0008, Val MSE=0.0007, Test MSE=0.0009 | Train
RMSE=0.0281, Val RMSE=0.0248, Test RMSE=0.0305
Epoch  16: Train MSE=0.0008, Val MSE=0.0006, Test MSE=0.0009 | Train
RMSE=0.0278, Val RMSE=0.0233, Test RMSE=0.0295
Epoch  17: Train MSE=0.0008, Val MSE=0.0006, Test MSE=0.0008 | Train
RMSE=0.0273, Val RMSE=0.0227, Test RMSE=0.0282
Epoch  18: Train MSE=0.0008, Val MSE=0.0007, Test MSE=0.0009 | Train
RMSE=0.0272, Val RMSE=0.0238, Test RMSE=0.0292
Epoch  19: Train MSE=0.0008, Val MSE=0.0006, Test MSE=0.0008 | Train
RMSE=0.0271, Val RMSE=0.0245, Test RMSE=0.0284
Epoch  20: Train MSE=0.0007, Val MSE=0.0007, Test MSE=0.0009 | Train
RMSE=0.0265, Val RMSE=0.0266, Test RMSE=0.0304
Epoch  21: Train MSE=0.0007, Val MSE=0.0007, Test MSE=0.0009 | Train
RMSE=0.0261, Val RMSE=0.0236, Test RMSE=0.0293
Epoch  22: Train MSE=0.0007, Val MSE=0.0006, Test MSE=0.0008 | Train
RMSE=0.0259, Val RMSE=0.0229, Test RMSE=0.0284
Epoch  23: Train MSE=0.0007, Val MSE=0.0006, Test MSE=0.0007 | Train
RMSE=0.0262, Val RMSE=0.0223, Test RMSE=0.0271
Epoch  24: Train MSE=0.0007, Val MSE=0.0006, Test MSE=0.0007 | Train
RMSE=0.0256, Val RMSE=0.0223, Test RMSE=0.0263
```

```
Epoch  25: Train MSE=0.0007, Val MSE=0.0006, Test MSE=0.0007 | Train
RMSE=0.0256, Val RMSE=0.0216, Test RMSE=0.0260
Epoch  26: Train MSE=0.0007, Val MSE=0.0006, Test MSE=0.0007 | Train
RMSE=0.0254, Val RMSE=0.0218, Test RMSE=0.0262
Epoch  27: Train MSE=0.0006, Val MSE=0.0006, Test MSE=0.0007 | Train
RMSE=0.0249, Val RMSE=0.0218, Test RMSE=0.0274
Epoch  28: Train MSE=0.0006, Val MSE=0.0005, Test MSE=0.0006 | Train
RMSE=0.0245, Val RMSE=0.0213, Test RMSE=0.0254
Epoch  29: Train MSE=0.0006, Val MSE=0.0005, Test MSE=0.0006 | Train
RMSE=0.0244, Val RMSE=0.0210, Test RMSE=0.0249
Epoch  30: Train MSE=0.0006, Val MSE=0.0005, Test MSE=0.0007 | Train
RMSE=0.0242, Val RMSE=0.0210, Test RMSE=0.0261
Epoch  31: Train MSE=0.0006, Val MSE=0.0005, Test MSE=0.0007 | Train
RMSE=0.0243, Val RMSE=0.0215, Test RMSE=0.0261
Epoch  32: Train MSE=0.0006, Val MSE=0.0005, Test MSE=0.0008 | Train
RMSE=0.0238, Val RMSE=0.0212, Test RMSE=0.0276
Epoch  33: Train MSE=0.0006, Val MSE=0.0006, Test MSE=0.0007 | Train
RMSE=0.0241, Val RMSE=0.0225, Test RMSE=0.0267
Epoch  34: Train MSE=0.0006, Val MSE=0.0005, Test MSE=0.0006 | Train
RMSE=0.0236, Val RMSE=0.0207, Test RMSE=0.0255
Epoch  35: Train MSE=0.0006, Val MSE=0.0005, Test MSE=0.0006 | Train
RMSE=0.0233, Val RMSE=0.0208, Test RMSE=0.0253
Epoch  36: Train MSE=0.0006, Val MSE=0.0006, Test MSE=0.0008 | Train
RMSE=0.0230, Val RMSE=0.0224, Test RMSE=0.0275
Epoch  37: Train MSE=0.0006, Val MSE=0.0006, Test MSE=0.0007 | Train
RMSE=0.0231, Val RMSE=0.0220, Test RMSE=0.0256
Epoch  38: Train MSE=0.0005, Val MSE=0.0005, Test MSE=0.0007 | Train
RMSE=0.0228, Val RMSE=0.0208, Test RMSE=0.0256
Epoch  39: Train MSE=0.0005, Val MSE=0.0005, Test MSE=0.0006 | Train
RMSE=0.0227, Val RMSE=0.0199, Test RMSE=0.0244
Epoch  40: Train MSE=0.0005, Val MSE=0.0005, Test MSE=0.0006 | Train
RMSE=0.0227, Val RMSE=0.0202, Test RMSE=0.0245
Epoch  41: Train MSE=0.0005, Val MSE=0.0005, Test MSE=0.0006 | Train
RMSE=0.0221, Val RMSE=0.0197, Test RMSE=0.0244
Epoch  42: Train MSE=0.0005, Val MSE=0.0005, Test MSE=0.0006 | Train
RMSE=0.0223, Val RMSE=0.0196, Test RMSE=0.0241
Epoch  43: Train MSE=0.0005, Val MSE=0.0005, Test MSE=0.0006 | Train
RMSE=0.0220, Val RMSE=0.0200, Test RMSE=0.0246
Epoch  44: Train MSE=0.0005, Val MSE=0.0005, Test MSE=0.0006 | Train
RMSE=0.0216, Val RMSE=0.0200, Test RMSE=0.0243
Epoch  45: Train MSE=0.0005, Val MSE=0.0004, Test MSE=0.0006 | Train
RMSE=0.0220, Val RMSE=0.0194, Test RMSE=0.0241
Epoch  46: Train MSE=0.0005, Val MSE=0.0005, Test MSE=0.0006 | Train
RMSE=0.0215, Val RMSE=0.0206, Test RMSE=0.0251
Epoch  47: Train MSE=0.0005, Val MSE=0.0005, Test MSE=0.0006 | Train
RMSE=0.0214, Val RMSE=0.0199, Test RMSE=0.0250
Epoch  48: Train MSE=0.0005, Val MSE=0.0005, Test MSE=0.0006 | Train
RMSE=0.0213, Val RMSE=0.0193, Test RMSE=0.0235
Epoch  49: Train MSE=0.0005, Val MSE=0.0005, Test MSE=0.0006 | Train
RMSE=0.0212, Val RMSE=0.0209, Test RMSE=0.0248
```

```
Epoch  50: Train MSE=0.0005, Val MSE=0.0005, Test MSE=0.0006 | Train
RMSE=0.0210, Val RMSE=0.0208, Test RMSE=0.0253
Epoch  51: Train MSE=0.0005, Val MSE=0.0005, Test MSE=0.0006 | Train
RMSE=0.0209, Val RMSE=0.0198, Test RMSE=0.0248
Epoch  52: Train MSE=0.0004, Val MSE=0.0004, Test MSE=0.0006 | Train
RMSE=0.0205, Val RMSE=0.0194, Test RMSE=0.0241
Epoch  53: Train MSE=0.0005, Val MSE=0.0005, Test MSE=0.0006 | Train
RMSE=0.0209, Val RMSE=0.0194, Test RMSE=0.0245
Epoch  54: Train MSE=0.0004, Val MSE=0.0004, Test MSE=0.0005 | Train
RMSE=0.0206, Val RMSE=0.0187, Test RMSE=0.0232
Epoch  55: Train MSE=0.0005, Val MSE=0.0005, Test MSE=0.0006 | Train
RMSE=0.0207, Val RMSE=0.0201, Test RMSE=0.0250
Epoch  56: Train MSE=0.0004, Val MSE=0.0004, Test MSE=0.0006 | Train
RMSE=0.0204, Val RMSE=0.0185, Test RMSE=0.0242
Epoch  57: Train MSE=0.0004, Val MSE=0.0005, Test MSE=0.0007 | Train
RMSE=0.0201, Val RMSE=0.0200, Test RMSE=0.0264
Epoch  58: Train MSE=0.0004, Val MSE=0.0005, Test MSE=0.0005 | Train
RMSE=0.0200, Val RMSE=0.0194, Test RMSE=0.0233
Epoch  59: Train MSE=0.0004, Val MSE=0.0004, Test MSE=0.0006 | Train
RMSE=0.0197, Val RMSE=0.0191, Test RMSE=0.0243
Epoch  60: Train MSE=0.0004, Val MSE=0.0004, Test MSE=0.0006 | Train
RMSE=0.0197, Val RMSE=0.0185, Test RMSE=0.0235
Epoch  61: Train MSE=0.0004, Val MSE=0.0005, Test MSE=0.0007 | Train
RMSE=0.0196, Val RMSE=0.0223, Test RMSE=0.0260
Epoch  62: Train MSE=0.0004, Val MSE=0.0005, Test MSE=0.0006 | Train
RMSE=0.0197, Val RMSE=0.0190, Test RMSE=0.0249
Epoch  63: Train MSE=0.0004, Val MSE=0.0004, Test MSE=0.0005 | Train
RMSE=0.0194, Val RMSE=0.0185, Test RMSE=0.0226
Epoch  64: Train MSE=0.0004, Val MSE=0.0005, Test MSE=0.0007 | Train
RMSE=0.0193, Val RMSE=0.0207, Test RMSE=0.0259
Epoch  65: Train MSE=0.0004, Val MSE=0.0004, Test MSE=0.0006 | Train
RMSE=0.0190, Val RMSE=0.0187, Test RMSE=0.0243
Epoch  66: Train MSE=0.0004, Val MSE=0.0005, Test MSE=0.0007 | Train
RMSE=0.0189, Val RMSE=0.0196, Test RMSE=0.0256
Epoch  67: Train MSE=0.0004, Val MSE=0.0005, Test MSE=0.0006 | Train
RMSE=0.0193, Val RMSE=0.0203, Test RMSE=0.0251
Epoch  68: Train MSE=0.0004, Val MSE=0.0004, Test MSE=0.0005 | Train
RMSE=0.0191, Val RMSE=0.0178, Test RMSE=0.0230

WARNING:absl:You are saving your model as an HDF5 file via
`model.save()` or `keras.saving.save_model(model)`. This file format
is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.


================================================
  FINAL MODEL METRICS SUMMARY (MULTI-TARGET Prediction)
================================================
```

```
--- Normalized Metrics ---
Train MSE:  0.0003
Val   MSE:  0.0004
Test  MSE:  0.0005
Train RMSE: 0.0185
Val   RMSE: 0.0196
Test  RMSE: 0.0230

--- Denormalized Metrics ---
Train MSE:  710.7772
Val   MSE:  951.1056
Test  MSE:  1383.3507
Train RMSE: 26.6604
Val   RMSE: 30.8400
Test  RMSE: 37.1934
==================================================

Model terlatih disimpan sebagai
'final_gru_multi_target_model_7_inputs_7_outputs.h5'

Menghasilkan visualisasi kurva MSE dan menyimpannya di folder:
loss_plots_multi_target_7_inputs_7_outputs

Menghasilkan visualisasi prediksi vs asli untuk setiap fitur target di
data test dan menyimpannya di folder:
predictions_multi_target_7_inputs_7_outputs

WARNING:absl:Compiled the loaded model, but the compiled metrics have
yet to be built. `model.compile_metrics` will be empty until you train
or evaluate the model.


Menyimpan hasil prediksi test dan data asli test untuk semua fitur
target ke CSV...
File 'prediksi_asli_test_multi_target_7_inputs_7_outputs.csv' telah
dibuat.

======================================================================
  MENAMPILKAN BOBOT DAN BIAS DARI MODEL GRU YANG TELAH DILATIH (MULTI-
TARGET)
  (Menampilkan sekitar 50 nilai per dimensi untuk ringkasan)
======================================================================

Berhasil memuat model dari:
final_gru_multi_target_model_7_inputs_7_outputs.h5

--- Ringkasan Model yang Dimuat ---

Model: "sequential_2"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ gru_6 (GRU)                          │ (None, 24, 128)             │          52,608 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ gru_7 (GRU)                          │ (None, 24, 64)              │          37,248 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ gru_8 (GRU)                          │ (None, 32)                  │           9,408 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_2 (Dense)                      │ (None, 7)                   │             231 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘

 Total params: 99,497 (388.66 KB)

 Trainable params: 99,495 (388.65 KB)

 Non-trainable params: 0 (0.00 B)

 Optimizer params: 2 (12.00 B)


--- Detail Bobot dan Bias untuk Setiap Layer ---

==================================================
  LAYER 1: GRU_6 (Tipe: GRU)
==================================================
  > Kernel Weights (Input to Gates) (Shape: (7, 384)):
    Sub-matriks (pertama 7x50):
[[-0.3547, -0.082 , -0.1166, -0.4704, -0.6003, -0.111 , -0.1873, -
0.5684,
   0.1393, -0.1469, -0.1242, -0.2509, -0.0606, -0.1391, -0.0878, -
0.3756,
  -0.0998, -0.1645, -0.4281, -0.1511, -0.1616, -0.6422, -0.0874, -
0.1998,
  -0.3588, -0.2846, -0.0963, -0.0891, -0.2734, -0.4632, -0.3536, -
0.1221,
  -0.0088, -0.399 , -0.3647, -0.2138, -0.3839, -0.3197, -0.2398, -
0.2806,
  -0.0891, -0.1587,  0.4013, -0.1058, -0.2323, -0.2794,  0.1705,
0.0046,
```

```
   0.4851,  0.4635],
 [-0.3179, -0.1407, -0.1957, -0.5112, -0.6116, -0.0084, -0.1333, -
0.3749,
   0.0994, -0.2323, -0.0444, -0.3675, -0.0976, -0.161 , -0.2253, -
0.4049,
  -0.2361, -0.1875, -0.4391, -0.2765, -0.2432, -0.6318, -0.2533, -
0.2542,
  -0.3244, -0.3395, -0.1539,  0.0448, -0.3332, -0.3729, -0.1482, -
0.3119,
   0.1106, -0.2991, -0.3036, -0.0524, -0.3047, -0.2448, -0.4273, -
0.4409,
  -0.1185, -0.1382,  0.3085, -0.1773, -0.1275, -0.3946,  0.1389, -
0.2066,
   0.6524,  0.4428],
 [-0.3539, -0.1923, -0.3266, -0.3633, -0.6146, -0.3148,  0.0706, -
0.6086,
   0.1365, -0.1729,  0.0632, -0.2692, -0.0299,  0.2748,  0.0795, -
0.1348,
  -0.2212, -0.0355, -0.314 , -0.1685, -0.3477, -0.6822,  0.0329, -
0.0703,
  -0.3946, -0.1163,  0.2823, -0.0365, -0.3485, -0.7778, -0.2697,
0.0119,
   0.1481, -0.0737, -0.2241, -0.315 , -0.6673, -0.4408,  0.0362, -
0.2416,
  -0.051 , -0.3549,  0.261 , -0.0901, -0.3551, -0.5297,  0.2711, -
0.1398,
   0.3665,  0.617 ],
 [-0.4537,  0.1632, -0.1821, -0.3756, -0.2962, -0.4306, -0.1573, -
0.9869,
  -0.1111, -0.2326, -0.4802, -0.4283,  0.1101, -0.1911, -0.08  , -
0.1155,
  -0.1669,  0.0953, -0.1882, -0.0908, -0.5051, -0.5656, -0.0076, -
0.2576,
  -0.1173, -0.0205,  0.1956, -0.2985, -0.6699, -0.5154,  0.0865, -
0.1199,
   0.1792, -0.1065, -0.2084, -0.2841, -0.1006, -0.444 , -0.0303, -
0.4397,
   0.0278, -0.0512,  0.1226, -0.2925, -0.1579, -0.6011,  0.1478,
0.1251,
   0.2999,  0.3608],
 [ 0.0187, -0.0102, -0.0353, -0.2913, -0.4898, -0.4652, -0.1432, -
0.3387,
  -0.1258,  0.0019, -0.4897,  0.113 , -0.187 ,  0.1136, -0.018 ,
0.1516,
  -0.2312, -0.0697, -0.2177, -0.1736, -0.204 , -0.1779, -0.2027,
0.0919,
  -0.2114,  0.0336,  0.3074, -0.2442, -0.4622, -0.2576, -0.1066, -
0.3336,
   0.0711, -0.1667, -0.0177, -0.5697,  0.1679, -0.0985, -0.1306, -
```

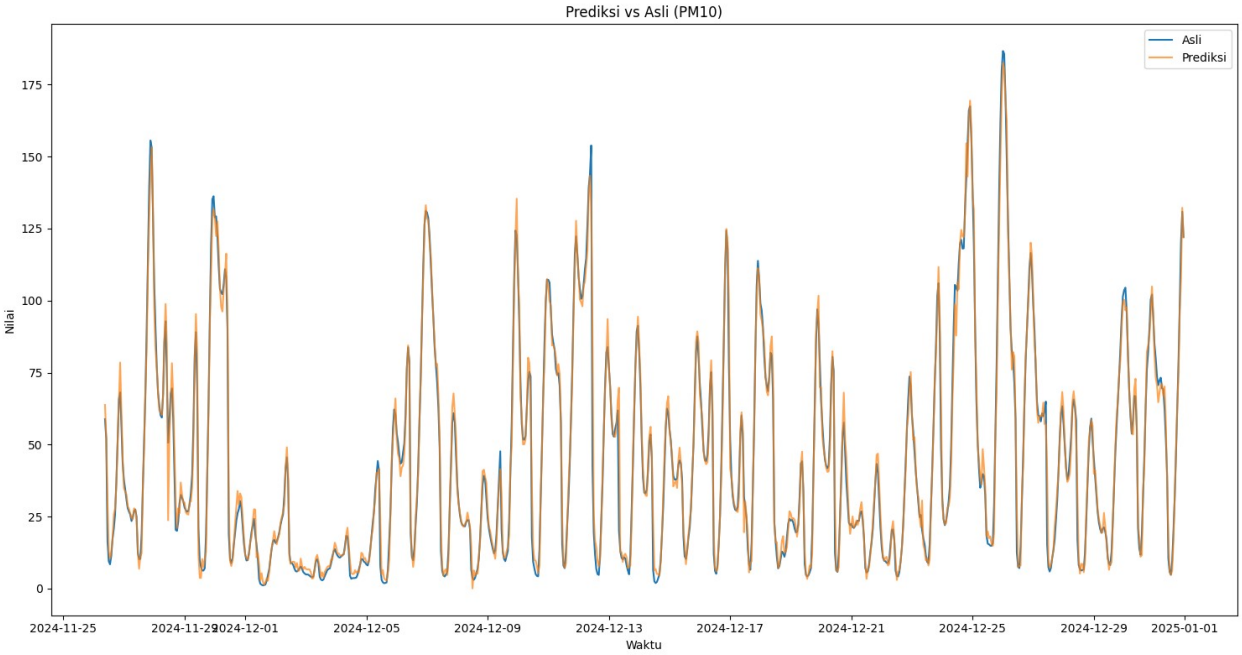
```
0.3632,
  -0.2279, -0.0587,  0.0616, -0.3414, -0.4761, -0.0584,  0.0361, -
0.3165,
   0.0146,  0.1908],
 [ 0.3642, -0.0006, -0.2163, -0.1631, -0.4297,  0.0108,  0.2441,
0.0888,
  -0.2162, -0.0276,  0.1812, -0.0271, -0.1617, -0.5911,  0.081 , -
0.752 ,
  -0.24  , -0.2997, -0.6598, -0.1937, -0.0962, -0.1598, -0.1889, -
0.1559,
  -0.3748, -0.8846, -0.3521, -0.0142, -0.1393,  0.4007, -0.2747, -
0.6938,
  -0.1963, -0.4344,  0.0118, -0.0937,  0.2225, -0.1172, -0.3805, -
0.1329,
  -0.0194, -0.2051, -0.0051, -0.18  , -0.27  ,  0.1268, -0.1164, -
0.3027,
   0.0091,  0.3077],
 [-0.1956, -0.0257,  0.0258, -0.422 , -0.4843, -0.1132, -0.0015, -
0.2867,
   0.1429, -0.2259, -0.2177, -0.1966, -0.1696, -0.1998, -0.2493, -
0.1851,
  -0.2373, -0.0828, -0.6477, -0.2696, -0.1989, -0.5651, -0.1159, -
0.1009,
  -0.5846, -0.4139, -0.0541, -0.1812, -0.3476, -0.2568, -0.4077, -
0.2192,
  -0.1236, -0.3936, -0.5062, -0.1253, -0.3367, -0.2623, -0.3505, -
0.2508,
  -0.1451, -0.1619,  0.3244, -0.158 , -0.4952, -0.2845,  0.1563, -
0.1785,
   0.5266,  0.2963]]
    ... (dipotong)
    Min Value: -1.547076, Max Value: 0.652410
--------------------------------------------------
  > Recurrent Kernel Weights (Hidden to Gates) (Shape: (128, 384)):
    Sub-matriks (pertama 50x50):
[[-0.174 ,  0.4688,  0.0642, ..., -0.0809, -0.148 ,  0.1273],
 [ 0.3663,  0.6124, -0.3431, ..., -0.0704,  0.0429,  0.0232],
 [ 0.1034,  0.4908,  0.1186, ...,  0.0616,  0.0278, -0.0729],
 ...,
 [-0.2125,  0.2927, -0.2842, ..., -0.1216,  0.1753,  0.284 ],
 [-0.0515, -0.2544, -0.094 , ...,  0.0256,  0.2293, -0.074 ],
 [ 0.0697, -0.108 , -0.194 , ...,  0.1459, -0.1087, -0.0544]]
    ... (dipotong)
    Min Value: -1.110686, Max Value: 1.195120
--------------------------------------------------
  > Bias (Shape: (2, 384)):
    Sub-matriks (pertama 2x50):
[[-0.4698,  0.1574, -0.2474, -0.6394, -0.9179, -0.7172, -0.2045, -
1.0938,
```

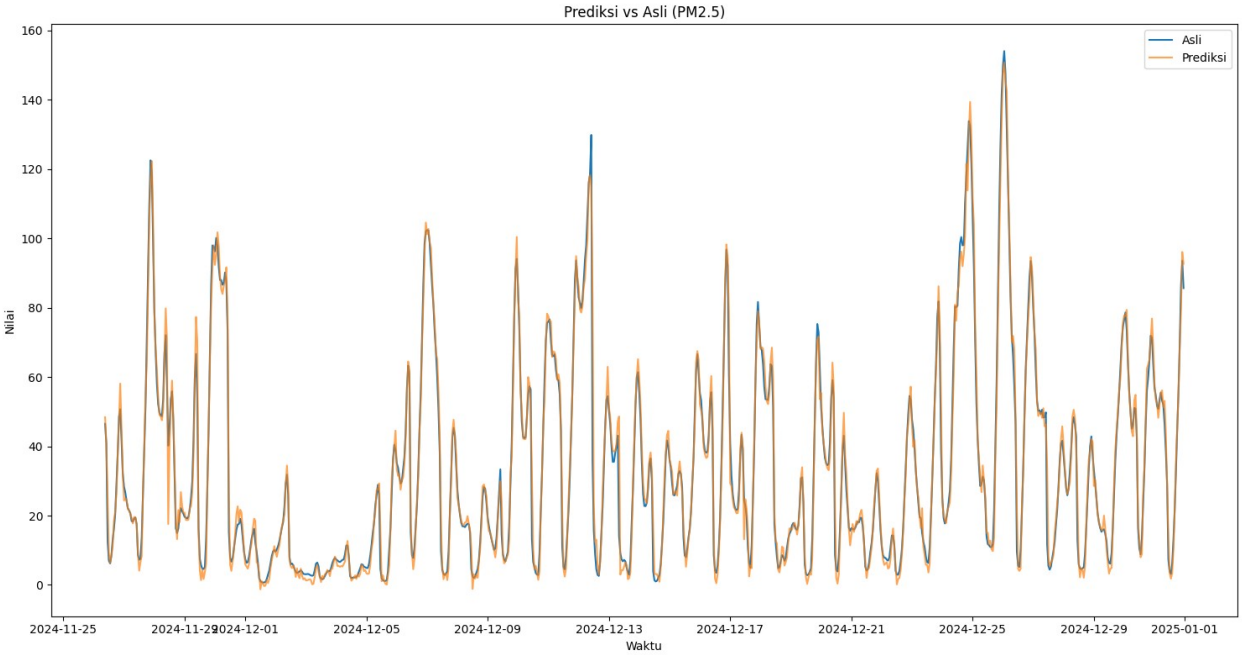
```
  -0.116 , -0.4858, -0.3605, -0.315 , -0.0493, -0.4648, -0.2926, -
0.1948,
  -0.4247, -0.1968, -0.7239, -0.5195, -0.5393, -0.818 , -0.1106, -
0.3249,
  -0.9236, -0.359 , -0.327 , -0.371 , -0.5792, -0.7048, -0.365 , -
0.2155,
  -0.2957, -0.3584, -0.8099, -0.4916, -0.4058, -0.4506, -0.2581, -
0.3964,
  -0.2969, -0.4976,  0.2142, -0.2754, -0.7261, -0.6799, -0.1653, -
0.5258,
   0.239 ,  0.0441],
 [-0.4698,  0.1574, -0.2474, -0.6394, -0.9179, -0.7172, -0.2045, -
1.0938,
  -0.116 , -0.4858, -0.3605, -0.315 , -0.0493, -0.4648, -0.2926, -
0.1948,
  -0.4247, -0.1968, -0.7239, -0.5195, -0.5393, -0.818 , -0.1106, -
0.3249,
  -0.9236, -0.359 , -0.327 , -0.371 , -0.5792, -0.7048, -0.365 , -
0.2155,
  -0.2957, -0.3584, -0.8099, -0.4916, -0.4058, -0.4506, -0.2581, -
0.3964,
  -0.2969, -0.4976,  0.2142, -0.2754, -0.7261, -0.6799, -0.1653, -
0.5258,
   0.239 ,  0.0441]]
    ... (dipotong)
    Min Value: -1.093820, Max Value: 0.565435
--------------------------------------------------

==================================================
  LAYER 2: GRU_7 (Tipe: GRU)
==================================================
  > Kernel Weights (Input to Gates) (Shape: (128, 192)):
    Sub-matriks (pertama 50x50):
[[ 0.022 , -0.2597, -0.3676, ..., -0.0263, -0.3353, -0.0048],
 [ 0.3028,  0.0174,  0.0542, ...,  0.2592,  0.5108, -0.0445],
 [-0.6316,  0.2423, -0.249 , ...,  0.0072, -0.0013, -0.0627],
 ...,
 [-0.0634, -0.1894,  0.1049, ..., -0.3565, -0.4661, -0.1523],
 [ 0.3581,  0.4437,  0.2786, ...,  0.2389,  0.1605,  0.0088],
 [ 0.0759,  0.1214,  0.3323, ...,  0.2306,  0.0969,  0.109 ]]
    ... (dipotong)
    Min Value: -1.341249, Max Value: 1.435881
--------------------------------------------------
  > Recurrent Kernel Weights (Hidden to Gates) (Shape: (64, 192)):
    Sub-matriks (pertama 50x50):
[[ 0.1161, -0.3182,  0.0099, ...,  0.067 ,  0.2585, -0.0846],
 [-0.1566,  0.3006,  0.0152, ...,  0.2662,  0.5176,  0.1843],
 [ 0.2129,  0.1431, -0.2264, ...,  0.1853, -0.0246,  0.0237],
 ...,
```

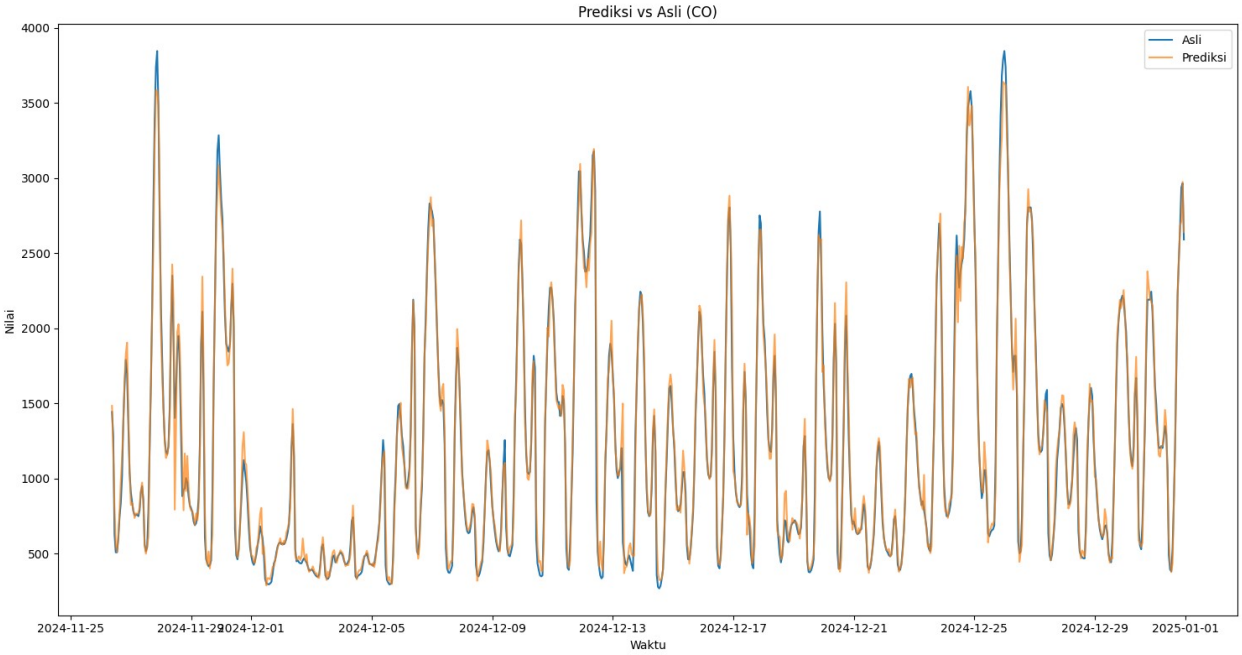
```
 [ 0.1383, -0.5764,  0.3592, ..., -0.1872, -0.2133, -0.245 ],
 [ 0.3315, -0.1049,  0.2257, ...,  0.1986,  0.08  , -0.0613],
 [ 0.0123,  0.2276,  0.5494, ..., -0.1103, -0.1532,  0.3665]]
    ... (dipotong)
    Min Value: -1.283229, Max Value: 1.208167
--------------------------------------------------
  > Bias (Shape: (2, 192)):
    Sub-matriks (pertama 2x50):
[[-0.4203, -0.3721, -0.4998, -0.6094, -0.3097, -0.6458, -0.1578, -
0.1673,
  -0.3542, -0.1574, -0.1373, -0.4696, -0.0627, -0.4325, -0.6032, -
0.7939,
  -0.0484, -0.2954, -0.0619, -0.4515, -0.1487, -0.2278, -0.4711,
0.0241,
  -0.3013, -0.3992, -0.1328, -0.5181, -0.2575, -0.4713,  0.0844, -
0.3776,
  -0.1302, -0.4536, -0.34  , -0.2982, -0.3686, -0.3082, -0.3369, -
0.8667,
  -0.9527,  0.0728, -0.7951, -0.0921, -0.8028, -0.4177, -0.024 , -
0.1671,
  -0.3593, -0.1897],
 [-0.4203, -0.3721, -0.4998, -0.6094, -0.3097, -0.6458, -0.1578, -
0.1673,
  -0.3542, -0.1574, -0.1373, -0.4696, -0.0627, -0.4325, -0.6032, -
0.7939,
  -0.0484, -0.2954, -0.0619, -0.4515, -0.1487, -0.2278, -0.4711,
0.0241,
  -0.3013, -0.3992, -0.1328, -0.5181, -0.2575, -0.4713,  0.0844, -
0.3776,
  -0.1302, -0.4536, -0.34  , -0.2982, -0.3686, -0.3082, -0.3369, -
0.8667,
  -0.9527,  0.0728, -0.7951, -0.0921, -0.8028, -0.4177, -0.024 , -
0.1671,
  -0.3593, -0.1897]]
    ... (dipotong)
    Min Value: -0.952653, Max Value: 0.184190
--------------------------------------------------

==================================================
  LAYER 3: GRU_8 (Tipe: GRU)
==================================================
  > Kernel Weights (Input to Gates) (Shape: (64, 96)):
    Sub-matriks (pertama 50x50):
[[ 0.2652, -0.6561, -0.4551, ...,  0.2869,  0.3246, -0.0404],
 [ 0.5129, -0.2057,  0.7098, ..., -0.0199, -0.1474, -0.128 ],
 [ 0.3107, -0.3133, -0.0915, ..., -0.0781,  0.1775,  0.1293],
 ...,
 [ 0.2248, -0.0303, -0.147 , ..., -0.1388,  0.125 ,  0.4813],
 [-0.5691,  0.4376, -0.2104, ...,  0.0607,  0.1875,  0.6947],
```

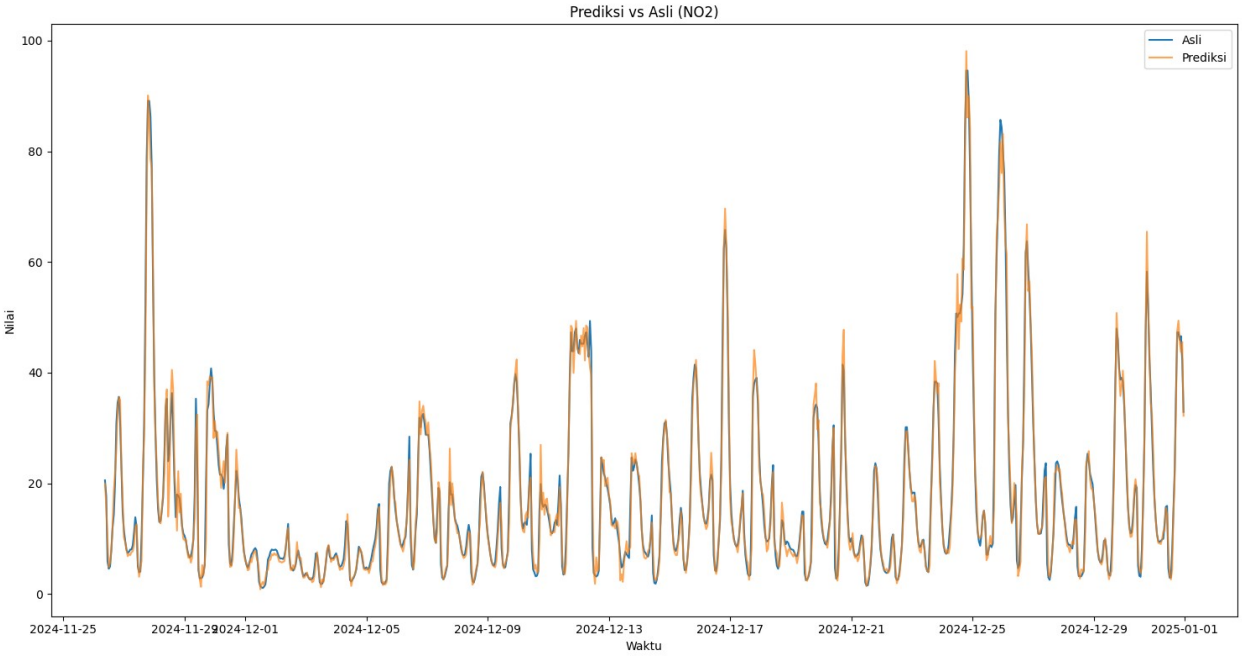
```
 [-0.1575, -0.1999,  0.2561, ..., -0.0201,  0.0515, -0.0523]]
    ... (dipotong)
    Min Value: -2.026096, Max Value: 1.371543
--------------------------------------------------
  > Recurrent Kernel Weights (Hidden to Gates) (Shape: (32, 96)):
    Sub-matriks (pertama 32x50):
[[ 0.7574,  0.1522, -0.4349, ..., -0.0905,  0.3506,  0.1217],
 [ 0.2816, -0.1765,  0.0551, ...,  0.3419,  0.0975,  0.1272],
 [-0.2504,  0.1545,  0.6904, ...,  0.1714,  0.1165, -0.1131],
 ...,
 [ 0.2268, -0.3755, -0.0871, ..., -0.0821,  0.2181, -0.0407],
 [ 0.2086,  0.0662, -0.0412, ..., -0.0011,  0.0719,  0.2613],
 [-0.1132,  0.1735, -0.2173, ..., -0.2731, -0.2129,  0.0196]]
    ... (dipotong)
    Min Value: -0.880799, Max Value: 1.006161
--------------------------------------------------
  > Bias (Shape: (2, 96)):
    Sub-matriks (pertama 2x50):
[[ 0.1889, -0.0912, -0.3478, -0.1015, -0.2235, -0.3528, -0.1844, -
0.2432,
  -0.4383, -0.3004, -0.2592, -0.1524, -0.2647, -0.0473, -0.1821, -
0.2513,
  -0.3083,  0.0811, -0.2426, -0.2665, -0.5557, -0.1852, -0.4098, -
0.3286,
  -0.5838, -0.3504, -0.3559, -0.5404, -0.5096, -0.0389, -0.3232, -
0.132 ,
  -0.1882,  0.0322,  0.0623,  0.005 , -0.0578, -0.0146, -0.0225, -
0.0803,
   0.0086,  0.2207,  0.1212,  0.0088,  0.0268,  0.0793,  0.0215,
0.0022,
   0.0578, -0.1484],
 [ 0.1889, -0.0912, -0.3478, -0.1015, -0.2235, -0.3528, -0.1844, -
0.2432,
  -0.4383, -0.3004, -0.2592, -0.1524, -0.2647, -0.0473, -0.1821, -
0.2513,
  -0.3083,  0.0811, -0.2426, -0.2665, -0.5557, -0.1852, -0.4098, -
0.3286,
  -0.5838, -0.3504, -0.3559, -0.5404, -0.5096, -0.0389, -0.3232, -
0.132 ,
  -0.1882,  0.0322,  0.0623,  0.005 , -0.0578, -0.0146, -0.0225, -
0.0803,
   0.0086,  0.2207,  0.1212,  0.0088,  0.0268,  0.0793,  0.0215,
0.0022,
   0.0578, -0.1484]]
    ... (dipotong)
    Min Value: -0.583800, Max Value: 0.279012
--------------------------------------------------

==================================================
```

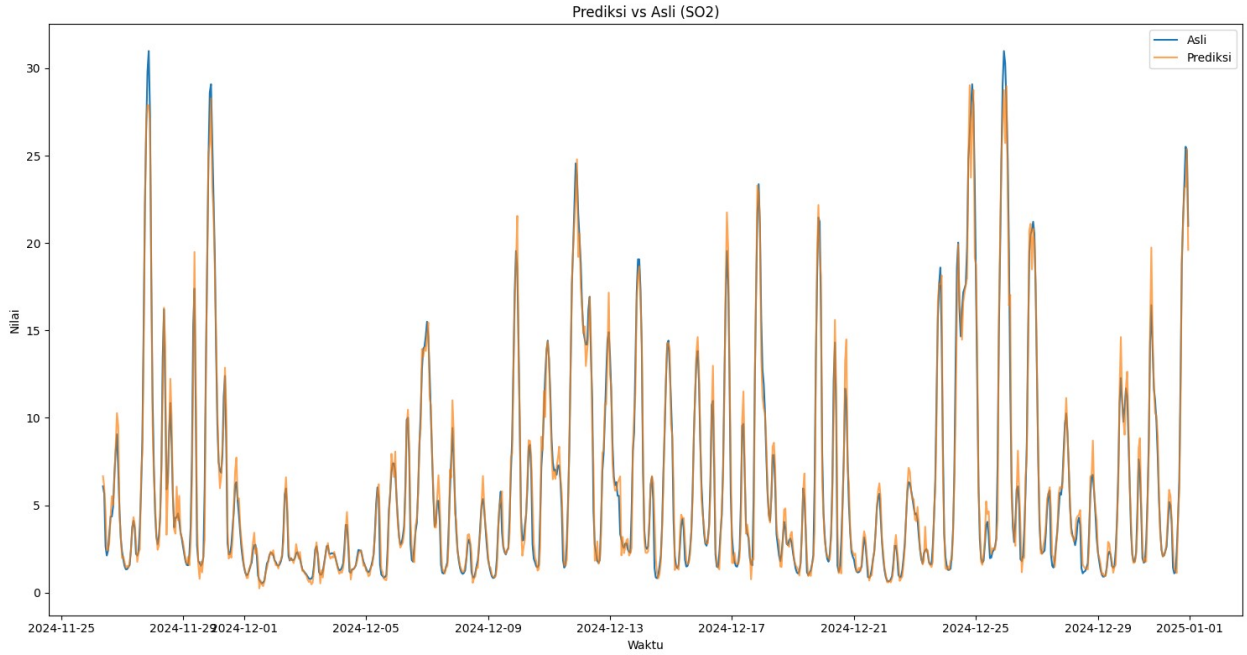
```
  LAYER 4: DENSE_2 (Tipe: Dense)
================================================
  > Weights (Shape: (32, 7)):
    Sub-matriks (pertama 32x7):
[[-0.0321, -0.1178,  0.0675,  0.0634,  0.1216,  0.0462,  0.0622],
 [ 0.1363, -0.0346,  0.1331,  0.0814, -0.1274,  0.0411,  0.1261],
 [-0.1646,  0.0082,  0.3597, -0.1848,  0.2864,  0.0123, -0.0849],
 [-0.218 , -0.3398,  0.1677, -0.3013,  0.007 , -0.1648, -0.18  ],
 [-0.148 , -0.2175,  0.0944,  0.1552, -0.1118, -0.2031,  0.1344],
 [-0.1682, -0.2546,  0.0772,  0.1779, -0.0247, -0.0363, -0.2725],
 [ 0.1331, -0.0015,  0.2936,  0.0852,  0.1118,  0.1629, -0.0684],
 [ 0.0987,  0.0943, -0.2446, -0.2512, -0.1237,  0.2512, -0.1566],
 [ 0.1948,  0.1636,  0.3138,  0.1396,  0.1121, -0.0786,  0.1715],
 [-0.0364,  0.1827,  0.1026,  0.041 , -0.2313, -0.1541,  0.0613],
 [ 0.1395,  0.0868, -0.2873, -0.2525, -0.1192,  0.2681,  0.1903],
 [ 0.0184, -0.2161,  0.0314,  0.215 ,  0.0987,  0.1409,  0.2456],
 [-0.0635, -0.0749,  0.1166,  0.1097, -0.2499, -0.2755, -0.04  ],
 [ 0.078 , -0.1409, -0.1022, -0.0292, -0.064 ,  0.0867, -0.2571],
 [-0.0836, -0.2047,  0.1488,  0.0271,  0.1126, -0.0111, -0.0554],
 [ 0.0487, -0.1602, -0.023 , -0.215 , -0.3085, -0.3066, -0.3122],
 [-0.1537,  0.0077,  0.0701,  0.1817,  0.2481, -0.1239, -0.2331],
 [-0.066 , -0.0964, -0.0361,  0.1493, -0.0883, -0.1417, -0.0065],
 [-0.0333,  0.1408,  0.0889,  0.1006, -0.1286,  0.0863, -0.0718],
 [-0.0104, -0.2246, -0.1731,  0.0093,  0.1859, -0.0881,  0.0481],
 [ 0.1218, -0.1137, -0.1998,  0.093 , -0.079 , -0.2594,  0.0808],
 [-0.233 , -0.0931,  0.1868, -0.2436, -0.3086, -0.1279, -0.1163],
 [ 0.0329, -0.0941, -0.2085, -0.058 , -0.0592, -0.3029,  0.045 ],
 [ 0.2681,  0.0893,  0.1043,  0.0567,  0.1347, -0.2345,  0.1826],
 [ 0.2184, -0.0309,  0.2268, -0.2428, -0.0762,  0.1933, -0.0213],
 [-0.2683,  0.0808,  0.1255, -0.0417,  0.1172, -0.1372, -0.0968],
 [ 0.1265,  0.0971, -0.2917,  0.3099, -0.2589, -0.2704, -0.2819],
 [ 0.0158, -0.1392, -0.1208,  0.2083, -0.216 ,  0.2233,  0.2219],
 [-0.0568, -0.0905,  0.1701,  0.1135,  0.0876,  0.3289,  0.139 ],
 [-0.0635,  0.0988,  0.0261, -0.2294, -0.2686,  0.1616, -0.0792],
 [-0.1283, -0.1293, -0.3147, -0.3114,  0.129 ,  0.0211,  0.0847],
 [-0.0684,  0.1894, -0.0332,  0.1192, -0.1233,  0.0001, -0.0102]]
    Min Value: -0.339832, Max Value: 0.359689
------------------------------------------------
  > Biases (Shape: (7,)):
    [0.06580596 0.06673248 0.06901591 0.03951578 0.04520005 0.05276629
 0.07644652]
    Min Value: 0.039516, Max Value: 0.076447
------------------------------------------------

================================================================
  TAMPILAN BOBOT DAN BIAS SELESAI
================================================================
```

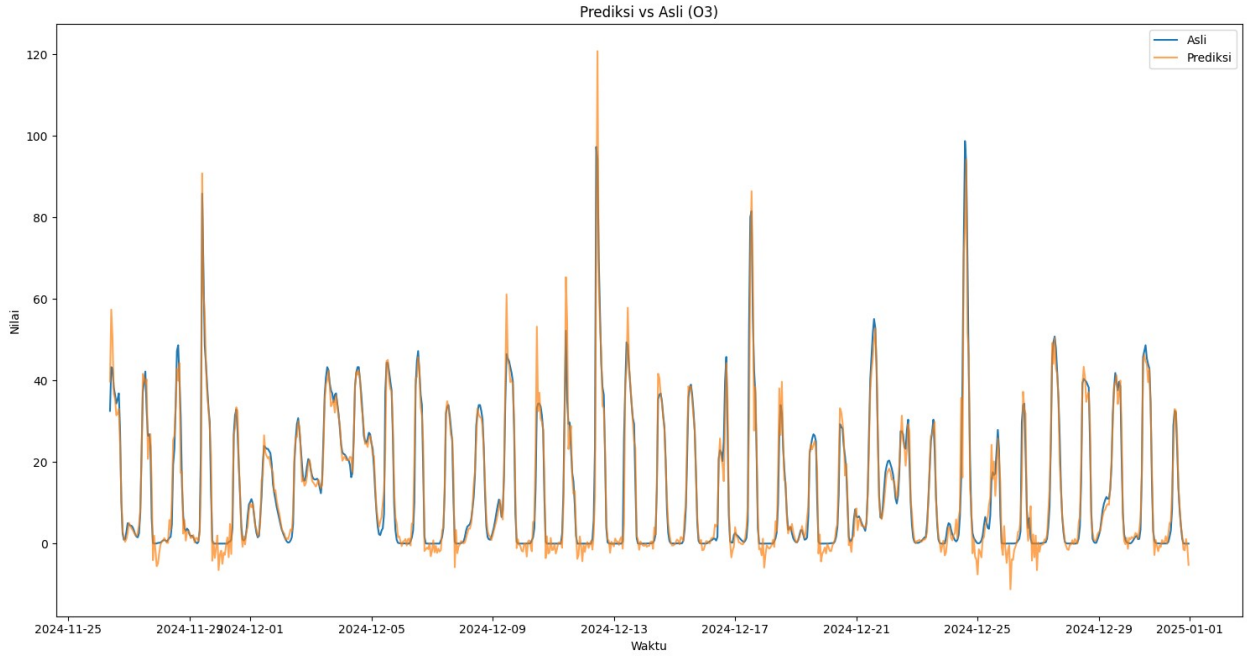

Prediksi vs Asli (PM10)
Asli
Prediksi
Nilai
Waktu
0
25
50
75
100
125
150
175
2024-11-25
2024-11-29
2024-12-01
2024-12-05
2024-12-09
2024-12-13
2024-12-17
2024-12-21
2024-12-25
2024-12-29
2025-01-01

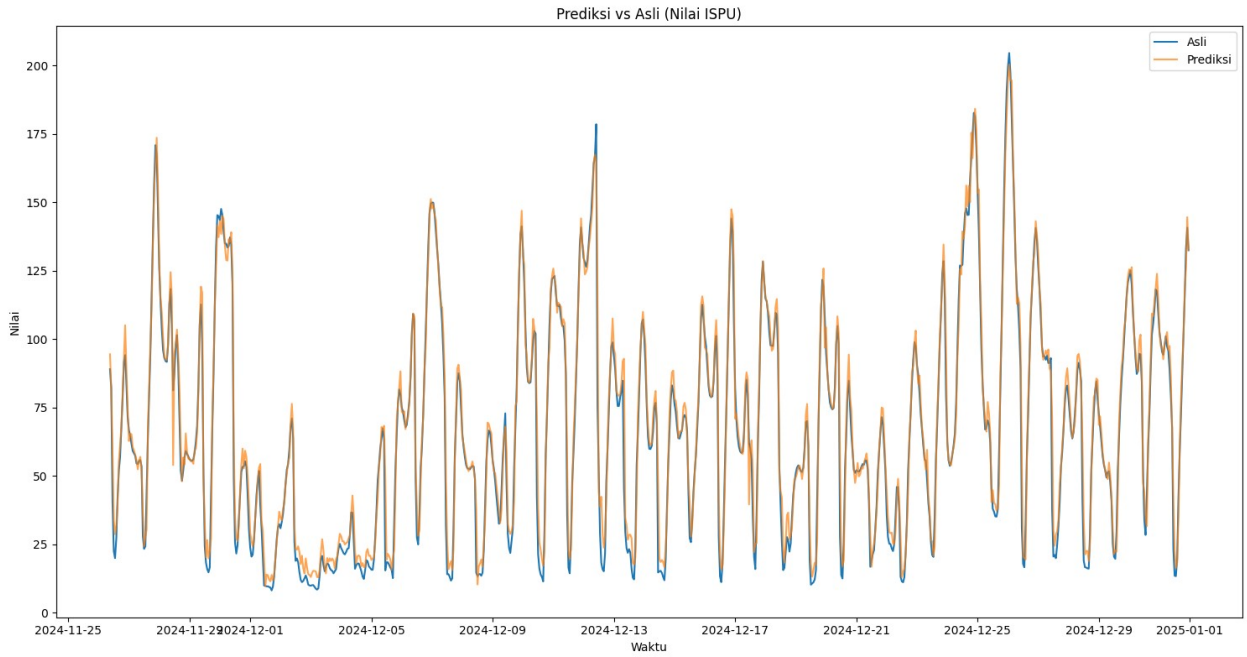

Prediksi vs Asli (PM2.5)
Asli
Prediksi
Nilai
Waktu
0
20
40
60
80
100
120
140
160
2024-11-25
2024-11-29
2024-12-01
2024-12-05
2024-12-09
2024-12-13
2024-12-17
2024-12-21
2024-12-25
2024-12-29
2025-01-01


Prediksi vs Asli (CO)
Asli
Prediksi
Nilai
Waktu
4000
3500
3000
2500
2000
1500
1000
500
2024-11-25
2024-11-29
2024-12-01
2024-12-05
2024-12-09
2024-12-13
2024-12-17
2024-12-21
2024-12-25
2024-12-29
2025-01-01


Prediksi vs Asli (NO2)
Asli
Prediksi
Nilai
Waktu
100
80
60
40
20
0
2024-11-25
2024-11-29
2024-12-01
2024-12-05
2024-12-09
2024-12-13
2024-12-17
2024-12-21
2024-12-25
2024-12-29
2025-01-01


Prediksi vs Asli (SO2)
Asli
Prediksi
Nilai
Waktu
0
5
10
15
20
25
30
2024-11-25
2024-11-29
2024-12-01
2024-12-05
2024-12-09
2024-12-13
2024-12-17
2024-12-21
2024-12-25
2024-12-29
2025-01-01


Prediksi vs Asli (O3)
Asli
Prediksi
Nilai
Waktu
0
20
40
60
80
100
120
2024-11-25
2024-11-29
2024-12-01
2024-12-05
2024-12-09
2024-12-13
2024-12-17
2024-12-21
2024-12-25
2024-12-29
2025-01-01




```
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
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import GRU, Dense, Input
from tensorflow.keras import backend as K
from tensorflow.keras.losses import MeanSquaredError # Penting untuk
memuat MSE jika digunakan
from math import sqrt
import os

# --- 1. Definisi Fungsi Kustom ---

# Metric kustom root_mean_squared_error
# Sangat penting untuk memuat model yang dilatih dengan metrik ini
def root_mean_squared_error(y_true, y_pred):
    return K.sqrt(K.mean(K.square(y_pred - y_true)))

# Fungsi untuk membuat sekuens data
def create_sequences(input_data, target_data, lookback):
    """
    Membuat sekuens untuk prediksi deret waktu.
    X berisi fitur input untuk 'lookback' langkah.
    y berisi SEMUA fitur target pada langkah berikutnya.
    """
    X, y = [], []
    for i in range(len(input_data) - lookback):
```

```
        X.append(input_data[i:i + lookback])
        y.append(target_data[i + lookback])
    return np.array(X), np.array(y)

# --- 2. Load dan Pra-pemrosesan Data ---
print("1. Memuat dan melakukan pra-pemrosesan data...")
# Pastikan path file ini benar sesuai lokasi di komputer Anda
data_file_path = '/content/05-data_ISPU_2024.csv'
if not os.path.exists(data_file_path):
    print(f"ERROR: File data '{data_file_path}' tidak ditemukan.")
    print("Pastikan file berada di direktori yang sama atau ubah
'data_file_path' ke lokasi yang benar.")
    exit() # Keluar jika file tidak ditemukan

data = pd.read_csv(data_file_path)
data['Waktu'] = pd.to_datetime(data['Waktu'])
data = data.drop(columns=['Kategori'])

# Semua kolom selain 'Waktu' dianggap sebagai fitur input DAN fitur
target
target_columns = data.drop(columns=['Waktu']).columns.tolist()

input_features_df = data.drop(columns=['Waktu'])
target_feature_df = data.drop(columns=['Waktu'])

input_values = input_features_df.values
target_values = target_feature_df.values

# --- 3. Pembagian Data (Mengikuti Rasio Asli) ---
print("2. Membagi data menjadi train, validasi, dan test...")
n_total = len(data)
n_train = int(n_total * 0.80)
n_val = int(n_total * 0.10)
n_test = n_total - n_train - n_val

train_input_data = input_values[:n_train]
val_input_data = input_values[n_train : n_train + n_val]
test_input_data = input_values[n_train + n_val : ]

train_target_data = target_values[:n_train]
val_target_data = target_values[n_train : n_train + n_val]
test_target_data = target_values[n_train + n_val : ]

# --- 4. Normalisasi Data ---
print("3. Melakukan normalisasi data...")
scaler_features = MinMaxScaler()
scaler_target = MinMaxScaler() # Skaler terpisah untuk target

# Fit dan transform data training
X_train_scaled = scaler_features.fit_transform(train_input_data)
```

```
y_train_scaled = scaler_target.fit_transform(train_target_data)

# Transform data test (menggunakan skaler yang sudah fit dari data
train)
X_test_scaled = scaler_features.transform(test_input_data)
y_test_scaled = scaler_target.transform(test_target_data)

# --- 5. Ambil Lookback dari Hasil Model Terbaik yang Tersimpan ---
print("4. Mencari nilai lookback dari hasil pelatihan terbaik...")
best_model_lookback = None
results_file_path =
'/content/prediksi_asli_test_multi_target_7_inputs_7_outputs.csv'
if os.path.exists(results_file_path):
    try:
        results_df = pd.read_csv(results_file_path)
        # Ambil lookback dari baris dengan Val RMSE terendah
        best_model_row = results_df.loc[results_df['Val
RMSE'].idxmin()]
        best_model_lookback = int(best_model_row['Lookback'])
        print(f"  Lookback terbaik yang ditemukan dari
'{results_file_path}': {best_model_lookback}")
    except Exception as e:
        print(f"  Gagal membaca lookback dari '{results_file_path}':
{e}")
else:
    print(f"  Peringatan: File hasil '{results_file_path}' tidak
ditemukan.")

if best_model_lookback is None:
    # Fallback jika file hasil tidak ada atau gagal dibaca
    # Anda bisa menggantinya dengan nilai lookback yang Anda ingat
    best_model_lookback = 24
    print(f"  Menggunakan nilai lookback default:
{best_model_lookback}")


# --- 6. Buat Urutan Data Test untuk Prediksi ---
print(f"5. Membuat urutan data test dengan lookback =
{best_model_lookback}...")
X_test_seq, y_test_seq = create_sequences(X_test_scaled,
y_test_scaled, best_model_lookback)

# --- 7. Muat Model Terbaik ---
model_path =
'/content/final_gru_multi_target_model_7_inputs_7_outputs.h5'
print(f"6. Memuat model terbaik dari '{model_path}'...")
loaded_model = None
if os.path.exists(model_path):
    try:
        loaded_model = load_model(model_path, custom_objects={
```

```
            'root_mean_squared_error': root_mean_squared_error,
            'mse': MeanSquaredError() # Sertakan MeanSquaredError juga
        })
        print("  Model berhasil dimuat.")
    except Exception as e:
        print(f"ERROR: Gagal memuat model. Pastikan metrik kustom dan
kelas loss didefinisikan dengan benar saat memuat:\n {e}")
else:
    print(f"ERROR: File model '{model_path}' tidak ditemukan. Pastikan
sudah ada di direktori yang sama.")

# --- 8. Lakukan Prediksi dan Inverse Transform ---
if loaded_model:
    print("7. Melakukan prediksi pada data test dan mengembalikan ke
skala asli...")
    y_test_pred_scaled = loaded_model.predict(X_test_seq)

    # Inverse transform nilai target asli dan prediksi kembali ke
skala aslinya
    y_test_true_orig = scaler_target.inverse_transform(y_test_seq)
    y_test_pred_orig =
scaler_target.inverse_transform(y_test_pred_scaled)

    # --- 9. Hitung dan Tampilkan RMSE Per Fitur ---
    print("\n" + "="*70)
    print("  HASIL RMSE PER FITUR PADA DATA UJI (SKALA ASLI)")
    print("="*70)

    for i, feature_name in enumerate(target_columns):
        true_values_feature = y_test_true_orig[:, i]
        pred_values_feature = y_test_pred_orig[:, i]

        rmse_feature = sqrt(mean_squared_error(true_values_feature,
pred_values_feature))
        print(f"  RMSE untuk '{feature_name}': {rmse_feature:.4f}")

    print("="*70)

    # Opsional: Hitung dan tampilkan RMSE rata-rata keseluruhan (untuk
semua fitur)
    overall_rmse = sqrt(mean_squared_error(y_test_true_orig,
y_test_pred_orig))
    print(f"\nRMSE Rata-rata Keseluruhan (untuk semua fitur):
{overall_rmse:.4f}")

print("----------------------------------------------------------------
-------")

else:
    print("\nTidak dapat melanjutkan karena model tidak berhasil
```

```
dimuat. Mohon periksa pesan error di atas.")

print("\n--- Proses Perhitungan RMSE Per Fitur Selesai ---")
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have
yet to be built. `model.compile_metrics` will be empty until you train
or evaluate the model.

1. Memuat dan melakukan pra-pemrosesan data...
2. Membagi data menjadi train, validasi, dan test...
3. Melakukan normalisasi data...
4. Mencari nilai lookback dari hasil pelatihan terbaik...
  Gagal membaca lookback dari
'/content/prediksi_asli_test_multi_target_7_inputs_7_outputs.csv':
'Val RMSE'
  Menggunakan nilai lookback default: 24
5. Membuat urutan data test dengan lookback = 24...
6. Memuat model terbaik dari
'/content/final_gru_multi_target_model_7_inputs_7_outputs.h5'...
  Model berhasil dimuat.
7. Melakukan prediksi pada data test dan mengembalikan ke skala
asli...
27/27 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step

======================================================================
  HASIL RMSE PER FITUR PADA DATA UJI (SKALA ASLI)
======================================================================
  RMSE untuk 'PM10': 4.8719
  RMSE untuk 'PM2.5': 4.0615
  RMSE untuk 'CO': 97.8827
  RMSE untuk 'NO2': 2.3267
  RMSE untuk 'SO2': 0.9245
  RMSE untuk 'O3': 3.8858
  RMSE untuk 'Nilai ISPU': 6.3903
======================================================================

RMSE Rata-rata Keseluruhan (untuk semua fitur): 37.1934
----------------------------------------------------------------------

--- Proses Perhitungan RMSE Per Fitur Selesai ---

Input: shape = (1, 4, 3)
Timesteps:       t0       t1       t2       t3
               [1,2,3]  [4,5,6]  [7,8,9]  [10,11,12]
                          |        |        |
GRU w/ return_sequences=True:
Output:          [..]     [..]     [..]     [..]   → shape (1, 4,
units)

GRU w/ return_sequences=False:
```

```
Output:                                [..]         → shape (1, units)
only from t3
```