

▼ Introduction

Jakarta Selatan - Stasiun Jagakarsa

Koordinat Geospasial: 106.80367, -6.35693

Jakarta merupakan salah satu kota metropolitan dengan tingkat aktivitas ekonomi, transportasi, dan urbanisasi yang sangat tinggi. Salah satu titik representatif di pusat kota adalah **Jagakarsa**, yang berfungsi sebagai area observasi penting untuk memantau dinamika kualitas udara di kawasan perkotaan padat penduduk.

Dalam penelitian ini, digunakan **model Long Short-Term Memory (LSTM)** — sebuah arsitektur jaringan saraf tiruan (neural network) berbasis *Recurrent Neural Network (RNN)* — untuk memprediksi variasi konsentrasi **PM2.5** dan parameter atmosfer seperti **Aerosol Optical Depth (AOD)**. Model LSTM dipilih karena kemampuannya dalam menangkap **pola temporal dan dependensi jangka panjang** dari data deret waktu (*time series*), yang umum ditemukan dalam dinamika atmosfer.

Secara konseptual, LSTM memiliki tiga gerbang utama:

1. **Forget Gate** – menentukan informasi mana yang perlu dihapus dari memori sel.
2. **Input Gate** – mengontrol seberapa banyak informasi baru akan disimpan.
3. **Output Gate** – mengatur keluaran dari memori sel ke langkah waktu berikutnya.

Dengan memanfaatkan data historis dari **satelit MODIS (MAIAC AOD)** dan **data pengamatan stasiun ISPU**, model LSTM diharapkan mampu memberikan prediksi yang lebih akurat terhadap variasi harian PM2.5 di wilayah Jagakarsa, serta membantu dalam analisis sumber polutan dan pola sebarannya.

Tujuan Penelitian

- Menerapkan model LSTM untuk memprediksi konsentrasi PM2.5 berbasis data AOD dan variabel meteorologi.
 - Mengevaluasi performa model menggunakan metrik statistik seperti RMSE, MAE, MAPE, R^2 , dan NSE.
 - Menganalisis keterkaitan spasial antara variabilitas AOD dengan dinamika polusi udara di Jakarta Pusat.
-

Ringkasan Teknis

Komponen	Deskripsi
Model	Long Short-Term Memory (LSTM)
Input Data	AOD (MODIS MCD19A2), Temperatur, Kelembapan, Kecepatan Angin, dll
Output Target	Konsentrasi PM2.5 Harian
Periode Data	2022 – 2024
Lokasi Studi	Jagakarsa (106.80367, -6.35693)
Evaluasi Kinerja	RMSE, MAE, MAPE, R ² , NSE

Gagasan Inti

Melalui pendekatan *deep learning* berbasis LSTM, penelitian ini berupaya membangun model prediktif yang tidak hanya mampu mengikuti pola temporal jangka pendek, tetapi juga menangkap *lag effect* antarvariabel atmosferik terhadap kualitas udara. Pendekatan ini diharapkan dapat memberikan kontribusi ilmiah terhadap **analisis sumber polutan PM2.5 dan pemantauan kualitas udara perkotaan secara berkelanjutan.**

1. Importing Required Libraries

- Mempersiapkan library yang diperlukan dalam pemrosesan data deret waktu dan pembangunan model LSTM.
- Mempersiapkan library untuk visualisasi utk mempermudah interpretasi data

```
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.preprocessing.sequence import TimeseriesGenerator
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import math
```

2. Data Cleaning and Interpolation

- Pada tahap ini dilakukan pembersihan data dengan mengganti nilai nol (0) dan simbol kosong (-) menjadi nilai hilang (NaN).
- Proses interpolasi linier untuk memperkirakan nilai yang hilang pada variabel seperti *PM2.5*, suhu, kelembapan, kecepatan angin, curah hujan, dan AOD, sehingga menghasilkan dataset yang lebih lengkap dan siap digunakan untuk pemodelan LSTM.

```
# path = "C:\\\\Users\\\\user\\\\OneDrive\\\\IPB\\\\Thesis\\\\02. Development\\\\03. LSTM\\\\da
# df_cleaned = pd.read_csv(path)
# columns = ['ISPU PM2.5', 'temp', 'dew', 'humidity', 'windspeed', 'AOD'] # unt
# for col in columns:
```

```

#     df_cleaned.loc[df_cleaned[col] == 0, col] = pd.NA

# df_cleaned.replace("-", pd.NA, inplace=True)

# columns = ['ISPU PM2.5', 'temp', 'dew', 'humidity', 'windspeed', 'precip', 'A
# for col in columns:
#     df_cleaned[col] = df_cleaned[col].interpolate(method='linear')

# df_cleaned = df_cleaned.dropna()

# df_cleaned

```

```

import pandas as pd
import numpy as np # Sebaiknya gunakan np.nan untuk 'pd.NA' numeri

path = r"C:\Users\user\OneDrive\IPB\Thesis\02. Development\03. LS
df_cleaned = pd.read_csv(path)

# --- BAGIAN 1: Membersihkan nilai 0 dan '-' ---

columns = ['ISPU PM2.5', 'temp', 'dew', 'humidity', 'windspeed',
for col in columns:

    # --- PERUBAHAN 1: TAMBAHKAN PENGECEKAN 'if col in ...' ---
    # Ini untuk memastikan skrip tidak error jika salah satu kolo
    # tidak ada di file Anda.
    if col in df_cleaned.columns:

        # --- PERUBAHAN 2: TAMBAHKAN 'pd.to_numeric' ---
        # Ini penting. Kita ubah dulu jadi angka (jika ada string
        # agar perbandingan '== 0' di baris berikutnya bekerja de
        df_cleaned[col] = pd.to_numeric(df_cleaned[col], errors='i

        # Ganti nilai 0 dengan NaN (pd.NA atau np.nan)
        df_cleaned.loc[df_cleaned[col] == 0, col] = np.nan # np.n
    else:
        print(f"Info: Kolom '{col}' untuk pembersihan 0 tidak dit
    # --- Akhir Perubahan 1 & 2 ---

# Baris ini sekarang akan mengganti sisa '-' (jika ada) yang gaga
# oleh 'pd.to_numeric' (meskipun seharusnya sudah ter-handle)
df_cleaned.replace("-", np.nan, inplace=True)

# --- BAGIAN 2: Interpolasi ---

columns = ['ISPU PM2.5', 'temp', 'dew', 'humidity', 'windspeed',
for col in columns:

    # --- PERUBAHAN 3: TAMBAHKAN PENGECEKAN 'if col in ...' ---
    # Ini adalah perbaikan utama Anda untuk 'KeyError: 'precip''
    # Skrip akan menginterpolasi 'precip' HANYA JIKA kolom itu ad

```

```

if col in df_cleaned.columns:
    # Menjalankan interpolasi.
    # Jika data sudah diinterpolasi (tidak ada NA), baris ini
    # tidak akan mengubah apapun, jadi aman untuk dijalankan.
    df_cleaned[col] = df_cleaned[col].interpolate(method='linear')
else:
    # Ini akan memberi tahu Anda bahwa 'precip' dilewati, bukan
    print(f"Info: Kolom '{col}' untuk interpolasi tidak ditemui")
# --- Akhir Perubahan 3 ---

# --- BAGIAN 3: Membersihkan sisa NA ---

# Interpolasi linear tidak bisa mengisi NA di awal atau akhir data.
# Tambahkan 'bfill' (back-fill) dan 'ffill' (forward-fill) untuk ini.
df_cleaned = df_cleaned.bfill().ffill()

# 'dropna()' Anda sekarang akan menghapus baris HANYA jika
# seluruh kolomnya gagal diisi (misal, jika file Anda kosong)
df_cleaned = df_cleaned.dropna()
# df = df_cleaned

print("Skrip selesai dijalankan.")
df_cleaned.head()

```

Skrip selesai dijalankan.

	datetime	ISPU	PM2.5	temp	dew	humidity	precip	windspeed	AOD	
0	1/1/2022		52.75	26.35	22.13		77.6	0.50	2.06	0.112
1	1/2/2022		52.75	25.62	21.82		79.6	0.01	1.58	0.112
2	1/3/2022		52.75	26.51	21.08		72.1	0.00	0.98	0.112
3	1/4/2022		52.75	26.30	21.76		76.1	0.00	1.23	0.112
4	1/5/2022		52.75	26.68	22.51		77.9	16.89	1.00	0.112

```

# Menghitung nilai minimum dan maksimum
min_vals = df_cleaned.min(numeric_only=True)
max_vals = df_cleaned.max(numeric_only=True)

print("== Nilai Minimum ==")
print(min_vals)
print("\n== Nilai Maksimum ==")
print(max_vals)

# Opsi alternatif satu baris untuk melihat keduanya sekaligus:
# print(df_cleaned.describe().loc[['min', 'max']])

```

```

== Nilai Minimum ==
ISPU PM2.5      2.550

```

```
temp          23.290
dew           17.860
humidity      56.000
precip         0.000
windspeed     0.690
AOD            0.007
dtype: float64
```

```
==== Nilai Maksimum ====
ISPU PM2.5    137.170
temp          28.510
dew           24.500
humidity      93.000
precip        110.390
windspeed     2.870
AOD            1.834
dtype: float64
```

```
print("\nData Information")
print("=*30)
df_cleaned.info()

print("\nData Summary")
print("=*30)
df_cleaned.isna().sum()
# df.duplicated().sum()
```

Data Information

```
=====
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1096 entries, 0 to 1095
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype  
---  --  
 0   datetime     1096 non-null   object 
 1   ISPU PM2.5  1096 non-null   float64
 2   temp         1096 non-null   float64
 3   dew          1096 non-null   float64
 4   humidity     1096 non-null   float64
 5   precip       1096 non-null   float64
 6   windspeed    1096 non-null   float64
 7   AOD          1096 non-null   float64
dtypes: float64(7), object(1)
memory usage: 68.6+ KB
```

Data Summary

```
=====
datetime      0
ISPU PM2.5   0
temp          0
dew           0
humidity      0
precip        0
windspeed     0
```

```
AOD          0  
dtype: int64
```

```
import matplotlib.pyplot as plt  
import seaborn as sns  
  
# hanya ambil kolom numerik  
corr = df_cleaned.select_dtypes(include=['number']).corr()  
  
plt.figure(figsize=(8,6))  
sns.heatmap(corr, annot=True, fmt=".2f", cmap="coolwarm", linewidths=0.5)  
plt.title("Correlation Matrix for each variable in Bundaran HI Dataset\n")  
plt.show()
```

Correlation Matrix for each variable in Bundaran HI Dataset



```
# daftar variabel yang akan diplot  
features = ['ISPU PM2.5', 'temp', 'dew', 'humidity', 'precip', 'windspeed']  
  
# atur tema  
sns.set(style="whitegrid", palette="viridis", context="talk")  
  
# hitung jumlah baris dan kolom otomatis  
n_cols = 3  
n_rows = math.ceil(len(features) / n_cols)
```

```

# buat grid
fig, axes = plt.subplots(n_rows, n_cols, figsize=(18, 6 * n_rows))
axes = axes.flatten()

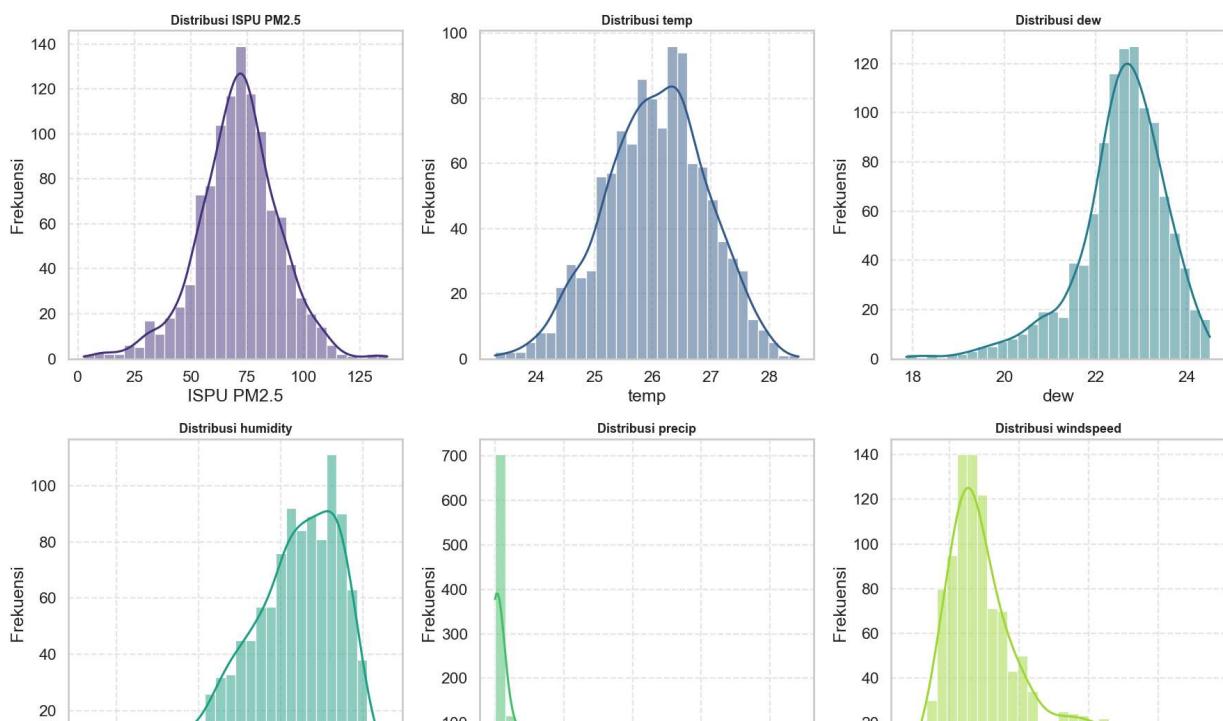
# looping untuk setiap fitur
for i, feature in enumerate(features):
    sns.histplot(df_cleaned[feature], kde=True, bins=30,
                 color=sns.color_palette("viridis", len(features))[i],
                 ax=axes[i])
    axes[i].set_title(f"Distribusi {feature}", fontsize=13, fontweight='bold')
    axes[i].set_xlabel(feature)
    axes[i].set_ylabel("Frekuensi")
    axes[i].grid(True, linestyle="--", alpha=0.4)

# hapus subplot kosong kalau ada
for j in range(len(features), len(axes)):
    fig.delaxes(axes[j])

# # judul besar
# plt.suptitle("Distribusi Histogram dari Variabel Dataset PM2.5 Jakarta",
#             fontsize=18, fontweight='bold', y=1.02)

plt.tight_layout()
plt.show()

```



```

import pandas as pd
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# 1 Load data
df = pd.read_csv(path, parse_dates=['datetime'])

```

```

df.columns = df.columns.str.strip()

# ❷ Pilih kolom numerik dan drop target untuk PCA
target_col = 'ISPU PM2.5'
df_num = df.select_dtypes(include=['number'])
features_only = df_num.drop(columns=[target_col])

# ❸ Pearson Correlation dengan Target
corr_series = df_num.corr()[target_col].drop(target_col)
top_corr_features = corr_series.abs().sort_values(ascending=False).head(5).index

print(f"Top features by Pearson correlation: {top_corr_features}")

# ❹ PCA (sebagai PPCA simplifikasi)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(features_only.dropna()) # Hapus NaN dulu

pca = PCA(n_components=2)
pca.fit(X_scaled)

# Loadings
loadings = pd.DataFrame(pca.components_.T,
                         index=features_only.columns,
                         columns=[f'PC{i+1}' for i in range(pca.n_components_)])

# Ambil fitur dengan loading tertinggi (komponen 1)
top_pca_features = loadings['PC1'].abs().sort_values(ascending=False).head(5).index

print(f"Top features by PCA loadings: {top_pca_features}")

# ❺ Gabungkan hasil Pearson + PCA
final_features = list(set(top_corr_features + top_pca_features))
print(f"\n✓ Fitur terpilih untuk model: {final_features}")

import seaborn as sns
import matplotlib.pyplot as plt

# ❶ Pearson correlation dengan target
plt.figure(figsize=(8,5))
sns.barplot(x=corr_series.abs().sort_values(ascending=False).values,
            y=corr_series.abs().sort_values(ascending=False).index, palette='viridis')
plt.title('Pearson Correlation dengan ISPU PM2.5')
plt.xlabel('Nilai Korelasi Absolut')
plt.ylabel('Fitur')
plt.show()

# ❷ PCA Loadings
plt.figure(figsize=(8,6))
plt.scatter(loadings['PC1'], loadings['PC2'], c='blue')

for i, feature in enumerate(loadings.index):
    plt.text(loadings['PC1'][i], loadings['PC2'][i], feature)

```

```
plt.text(loadings['PC1'][i], loadings['PC2'][i], feature, fontsize=10)

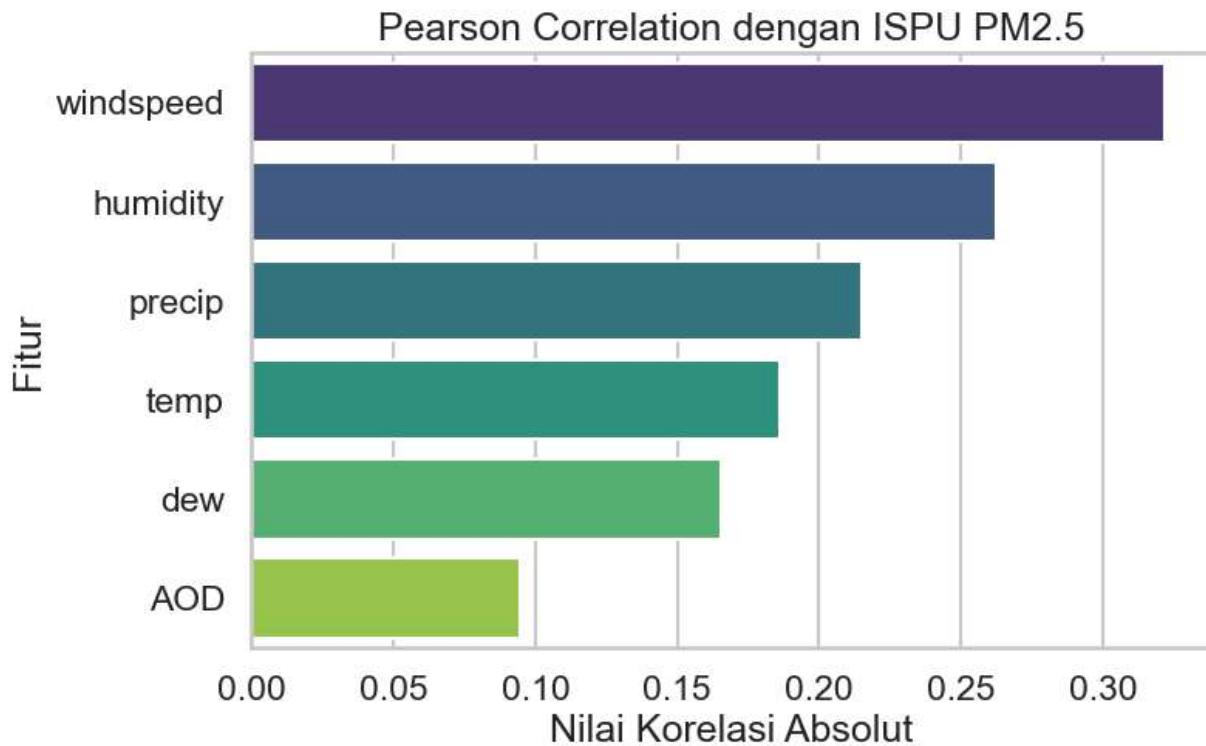
plt.axhline(0, color='grey', linestyle='--')
plt.axvline(0, color='grey', linestyle='--')
plt.title('PCA Loadings Plot (PC1 vs PC2)')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.grid()
plt.show()
```

Top features by Pearson correlation: ['windspeed', 'humidity', 'precip', 'temp',
Top features by PCA loadings: ['humidity', 'temp', 'precip', 'dew', 'windspeed']

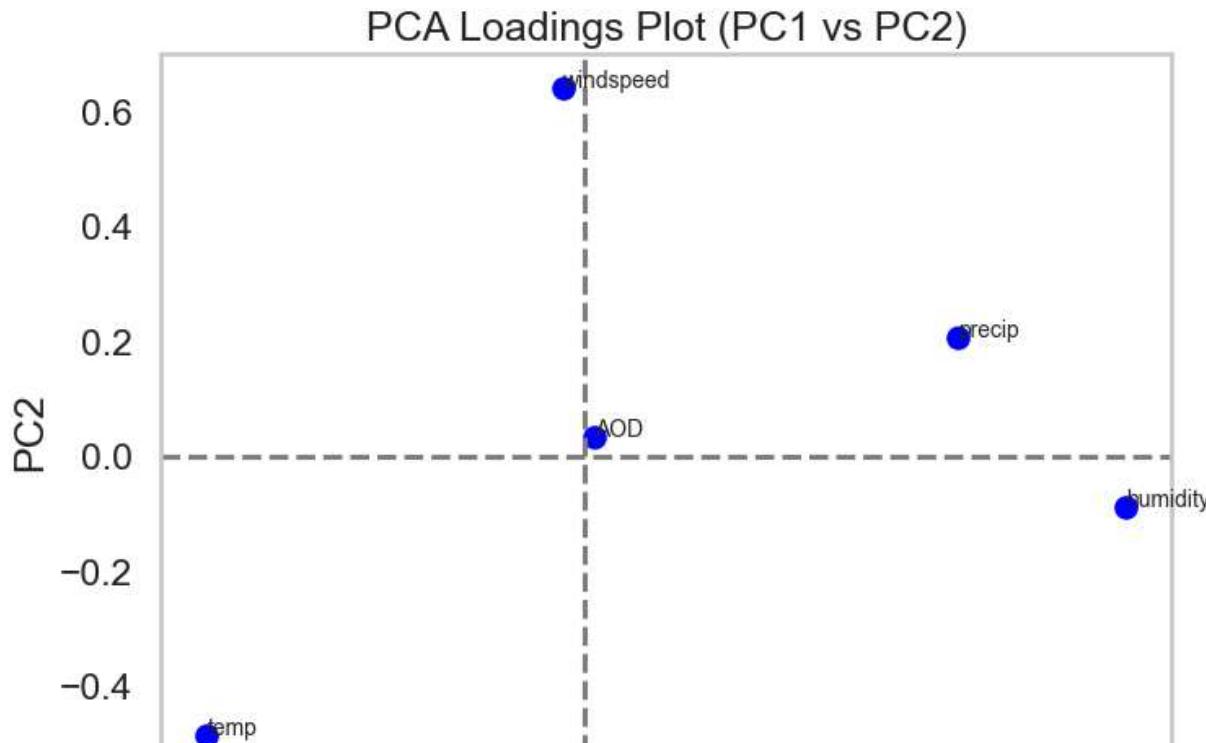
✓ Fitur terpilih untuk model: ['precip', 'windspeed', 'humidity', 'dew', 'temp']
C:\Users\user\AppData\Local\Temp\ipykernel_54484\4150998722.py:46: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v

```
sns.barplot(x=corr_series.abs().sort_values(ascending=False).values,
```



C:\Users\user\AppData\Local\Temp\ipykernel_54484\4150998722.py:58: FutureWarning
plt.text(loadings['PC1'][i], loadings['PC2'][i], feature, fontsize=10)



df_cleaned

	datetime	ISPU	PM2.5	temp	dew	humidity	precip	windspeed	AOD
0	1/1/2022	52.75	26.35	22.13	77.6	0.50	2.06	0.112	
1	1/2/2022	52.75	25.62	21.82	79.6	0.01	1.58	0.112	
2	1/3/2022	52.75	26.51	21.08	72.1	0.00	0.98	0.112	
3	1/4/2022	52.75	26.30	21.76	76.1	0.00	1.23	0.112	
4	1/5/2022	52.75	26.68	22.51	77.9	16.89	1.00	0.112	
...
1091	12/27/2024	82.89	27.38	22.78	76.0	0.00	1.49	0.474	
1092	12/28/2024	74.83	27.52	21.67	70.5	0.04	1.80	0.474	
1093	12/29/2024	66.78	27.22	23.24	78.9	0.05	1.78	0.474	
1094	12/30/2024	58.72	25.41	22.39	83.4	3.34	2.64	0.474	
1095	12/31/2024	50.67	26.41	22.85	80.8	4.01	2.35	0.474	

1096 rows × 8 columns

```

df = df_cleaned

features = ["temp", "dew", "humidity", "precip", "windspeed", "AOD", "ISPU PM2.5"]
target = "ISPU PM2.5"

# Normalisasi data
scaler = MinMaxScaler()
df_scaled = scaler.fit_transform(df[features])

# Tentukan lookback (jumlah timestep sebelumnya digunakan untuk prediksi)
lookback = 30

# Bagi data menjadi training (70%), validasi (15%), dan testing (15%)
train_size = int(len(df_scaled) * 0.7)
val_size = int(len(df_scaled) * 0.15)

train_data = df_scaled[:train_size]
val_data = df_scaled[train_size:train_size+val_size]
test_data = df_scaled[train_size+val_size:]

# Gunakan TimeseriesGenerator untuk membentuk data dalam format sekuensial
train_gen = TimeseriesGenerator(train_data, train_data[:, 6], length=lookback,
val_gen = TimeseriesGenerator(val_data, val_data[:, 6], length=lookback, batch_
test_gen = TimeseriesGenerator(test_data, test_data[:, 6], length=lookback, bat

```

```
# Cek bentuk data generator
len(train_gen), len(val_gen), len(test_gen)

(185, 34, 34)
```

3. LSTM Model Building and Evaluation

- Bagian ini berisi proses pembangunan model **LSTM (Long Short-Term Memory)** untuk prediksi konsentrasi PM2.5 dengan optimasi parameter seperti *learning rate*, *optimizer*, dan jumlah *LSTM units*.
- Fungsi `build_lstm_model()` mendefinisikan arsitektur jaringan LSTM dengan dua lapisan tersembunyi dan mekanisme *dropout* untuk mencegah overfitting. Selain itu, ditambahkan fungsi evaluasi `evaluate_model()` yang menghitung berbagai metrik performa model, termasuk **R²**, **RMSE**, **MBE**, **NSE**, dan **MAPE** untuk menilai akurasi hasil prediksi.
- Proses *grid search* digunakan dalam `grid_search_lstm()` untuk menguji berbagai kombinasi hiperparameter, dengan penerapan **EarlyStopping** agar pelatihan berhenti otomatis ketika model tidak lagi membaik, serta callback `R2History` untuk memantau peningkatan kinerja model pada data validasi.

```
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam, Nadam, RMSprop, SGD
from tensorflow.keras import Input
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_percent
from tensorflow.keras.callbacks import Callback, EarlyStopping
import numpy as np

# --- 1. FUNGSI BARU: MEAN ABSOLUTE PERCENTAGE ERROR (MAPE) ---
# def mape_score(y_true, y_pred):
#     """
#     Menghitung Mean Absolute Percentage Error (MAPE).
#     Mencegah pembagian dengan nol dengan menambahkan epsilon (np.finfo(float))
#     """
#     # Menghindari pembagian dengan nol
#     y_true, y_pred = np.array(y_true), np.array(y_pred)
#     return np.mean(np.abs((y_true - y_pred) / (y_true + np.finfo(float).eps)))

# -----


early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=5,
```

```

        restore_best_weights=True
    )

def build_lstm_model(input_shape, lstm_units, optimizer, learning_rate):
    model = Sequential([
        Input(shape=input_shape),
        LSTM(lstm_units, activation='tanh', return_sequences=True),
        Dropout(0.1),
        LSTM(lstm_units, activation='tanh', return_sequences=False),
        Dropout(0.1),
        Dense(25, activation='tanh'),
        Dense(1)
    ])

    opt = Adam(learning_rate=learning_rate) if optimizer == 'adam' else RMSprop
    model.compile(optimizer=opt, loss='mse')
    return model

def evaluate_model(model, test_gen):
    test_pred = model.predict(test_gen)

    test_true_list = []
    for i in range(len(test_gen)):
        _, y = test_gen[i]
        test_true_list.append(y)

    test_true = np.hstack(test_true_list)

    test_r2 = r2_score(test_true, test_pred)
    test_rmse = np.sqrt(mean_squared_error(test_true, test_pred))
    test_mbe = np.mean(test_true - test_pred)
    test_nse = 1 - (np.sum((test_true - test_pred) ** 2) / np.sum((test_true -
    test_mape = mean_absolute_percentage_error(test_true, test_pred)
    # --- PENAMBAHAN MAPE ---
    # test_mape = mape_score(test_true, test_pred)

    return {'R2': test_r2, 'RMSE': test_rmse, 'MBE': test_mbe, 'NSE': test_nse}

def grid_search_lstm(train_gen, val_gen, test_gen, input_shape, param_grid):
    results = []
    models = {}
    all_history = {}
    all_r2_callback = {}

    # --- Callback kustom untuk R2, diperluas untuk MAPE (jika perlu, tapi kita
    class R2History(Callback):
        def __init__(self, val_gen):
            self.val_gen = val_gen
            self.r2_scores = []

        def on_epoch_end(self, epoch, logs=None):

```

```

        val_pred = self.model.predict(self.val_gen, verbose=0)

        val_true_list = []
        for i in range(len(val_gen)):
            _, y = val_gen[i]
            val_true_list.append(y)

        val_true = np.hstack(val_true_list)

        r2 = r2_score(val_true, val_pred)
        self.r2_scores.append(r2)
    # -----.

    for epochs in param_grid['epochs']:
        for learning_rate in param_grid['learning_rate']:
            for optimizer in param_grid['optimizer']:
                for lstm_units in param_grid['lstm_units']:

                    r2_callback = R2History(val_gen)
                    config_name = f"epochs_{epochs}_lr_{learning_rate}_opt_{optimizer}"
                    print(f"\nTraining model dengan {config_name}")

                    model = build_lstm_model(input_shape, lstm_units, optimizer)

                    # Run 1 epoch warmup (optional but helpful before main loop
                    model.fit(train_gen, validation_data=val_gen, epochs=1, verbose=0)

                    # Main training loop
                    history = model.fit(train_gen, validation_data=val_gen, epochs=10, verbose=0)

                    metrics = evaluate_model(model, test_gen)
                    print(metrics)

                    results.append({
                        'epochs': epochs,
                        'learning_rate': learning_rate,
                        'optimizer': optimizer,
                        'lstm_units': lstm_units,
                        'R2': metrics['R2'],
                        'RMSE': metrics['RMSE'],
                        'MBE': metrics['MBE'],
                        'NSE': metrics['NSE'],
                        'MAPE': metrics['MAPE'] # --- PENAMBAHAN MAPE KE HASIL
                    })
                    models[config_name] = model
                    all_history[config_name] = history
                    all_r2_callback[config_name] = r2_callback

    return results, models, all_history, all_r2_callback

```

4. Hyperparameter Tuning

Pada tahap ini dilakukan **eksperimen pencarian hiperparameter (Grid Search)** untuk menentukan kombinasi terbaik dalam pelatihan model LSTM.

Parameter yang diuji mencakup:

- **epochs** → jumlah iterasi pelatihan model.
- **learning_rate** → laju pembelajaran untuk mengontrol penyesuaian bobot.
- **optimizer** → algoritma optimasi (*Adam, RMSprop, SGD, Nadam*).
- **lstm_units** → jumlah neuron pada lapisan LSTM.

Hasil setiap kombinasi disimpan dalam `df_results` untuk dianalisis lebih lanjut, mencakup metrik performa seperti **R²**, **RMSE**, **MBE**, **NSE**, dan **MAPE** guna menentukan konfigurasi paling optimal.

```
param_grid = {
    'epochs': [1000],
    'learning_rate': [0.0001, 0.001, 0.01, 0.1],
    'optimizer': ['adam', 'rmsprop', 'sgd', 'nadam'],
    'lstm_units': [16, 32, 64]
}

grid_results, models, all_history, all_r2_callback = grid_search_lstm(train_ger
df_results = pd.DataFrame(grid_results)
```

```
Training model dengan epochs_1000_lr_0.0001_opt_adam_units_16
c:\Users\user\AppData\Local\Programs\Python\Python313\Lib\site-packages\keras\
    self._warn_if_super_not_called()
27/34 _____ 0s 4ms/step: c:\Users\user\AppData\Local\Programs\Pyt
    self._warn_if_super_not_called()
34/34 _____ 0s 4ms/step
{'R2': 0.4352583078644886, 'RMSE': np.float64(0.09207442993169296), 'MBE': np.

Training model dengan epochs_1000_lr_0.0001_opt_adam_units_32
34/34 _____ 0s 9ms/step
{'R2': -0.10814345944866233, 'RMSE': np.float64(0.12897695128299208), 'MBE': n

Training model dengan epochs_1000_lr_0.0001_opt_adam_units_64
34/34 _____ 1s 13ms/step
{'R2': 0.18831992785285756, 'RMSE': np.float64(0.1103841086444277), 'MBE': np.

Training model dengan epochs_1000_lr_0.0001_opt_rmsprop_units_16
34/34 _____ 0s 8ms/step
{'R2': -0.19875088491384796, 'RMSE': np.float64(0.1341462645134358), 'MBE': np.

Training model dengan epochs_1000_lr_0.0001_opt_rmsprop_units_32
34/34 _____ 0s 8ms/step
{'R2': 0.3507936144045518, 'RMSE': np.float64(0.09872008138284118), 'MBE': np.
```

```
Training model dengan epochs_1000_lr_0.0001_opt_rmsprop_units_64
34/34 ━━━━━━━━ 0s 7ms/step
{'R2': 0.14736667812250204, 'RMSE': np.float64(0.11313455265412535), 'MBE': np.

Training model dengan epochs_1000_lr_0.0001_opt_sgd_units_16
34/34 ━━━━━━━━ 0s 6ms/step
{'R2': -0.006270542738298079, 'RMSE': np.float64(0.12290554954620515), 'MBE': np.

Training model dengan epochs_1000_lr_0.0001_opt_sgd_units_32
34/34 ━━━━━━━━ 0s 8ms/step
{'R2': 0.2631005129854864, 'RMSE': np.float64(0.10517638516718764), 'MBE': np.

Training model dengan epochs_1000_lr_0.0001_opt_sgd_units_64
34/34 ━━━━━━━━ 0s 11ms/step
{'R2': -0.4166284549258288, 'RMSE': np.float64(0.14582839426198735), 'MBE': np.

Training model dengan epochs_1000_lr_0.0001_opt_nadam_units_16
34/34 ━━━━━━━━ 0s 10ms/step
{'R2': -0.2254265130428521, 'RMSE': np.float64(0.13563062076142074), 'MBE': np.

Training model dengan epochs_1000_lr_0.0001_opt_nadam_units_32
34/34 ━━━━━━━━ 0s 7ms/step
{'R2': -0.023654292811271294, 'RMSE': np.float64(0.12396262645920629), 'MBE': np.

Training model dengan epochs_1000_lr_0.0001_opt_nadam_units_64
34/34 ━━━━━━━━ 0s 6ms/step
{'R2': 0.24443071775264757, 'RMSE': np.float64(0.10650040514954677), 'MBE': np.

Training model dengan epochs_1000_lr_0.001_opt_adam_units_16
34/34 ━━━━━━━━ 0s 7ms/step
{'R2': 0.27800300122807375, 'RMSE': np.float64(0.10410745008939493), 'MBE': np
```

```
df_results
```


	epochs	learning_rate	optimizer	lstm_units	R2	RMSE	ME
0	1000	0.0001	adam	16	0.435258	0.092074	-0.02042
1	1000	0.0001	adam	32	-0.108143	0.128977	-0.06180
2	1000	0.0001	adam	64	0.188320	0.110384	-0.03424
3	1000	0.0001	rmsprop	16	-0.198751	0.134146	-0.03207
4	1000	0.0001	rmsprop	32	0.350794	0.098720	-0.02320
5	1000	0.0001	rmsprop	64	0.147367	0.113135	-0.03259
6	1000	0.0001	sgd	16	-0.006271	0.122906	-0.03280
7	1000	0.0001	sgd	32	0.263101	0.105176	-0.00426
8	1000	0.0001	sgd	64	-0.416628	0.145828	-0.09474
9	1000	0.0001	nadam	16	-0.225427	0.135631	-0.02000
10	1000	0.0001	nadam	32	-0.023654	0.123963	-0.04532
11	1000	0.0001	nadam	64	0.244431	0.106500	-0.03121
12	1000	0.0010	adam	16	0.278003	0.104107	-0.02260
13	1000	0.0010	adam	32	0.575182	0.079857	-0.01937
14	1000	0.0010	adam	64	0.442808	0.091457	-0.00941
15	1000	0.0010	rmsprop	16	0.266541	0.104931	0.00557
16	1000	0.0010	rmsprop	32	0.394183	0.095364	0.02780
17	1000	0.0010	rmsprop	64	0.353126	0.098543	0.03244
18	1000	0.0010	sgd	16	0.181564	0.110843	-0.03139
19	1000	0.0010	sgd	32	0.245230	0.106400	-0.02423
20	1000	0.0010	sgd	64	0.154304	0.112673	0.06922
21	1000	0.0010	nadam	16	0.193091	0.110059	-0.04562
22	1000	0.0010	nadam	32	0.466711	0.119829	0.06210
23	1000	0.0010	nadam	64	0.407187	0.094335	-0.02531
24	1000	0.0100	adam	16	0.591820	0.078278	-0.02113
25	1000	0.0100	adam	32	0.428735	0.092605	0.00887
26	1000	0.0100	adam	64	0.655512	0.071912	-0.01192
27	1000	0.0100	rmsprop	16	-0.535235	0.151810	-0.11515

5. Pemilihan Model Terbaik

Setelah seluruh kombinasi hiperparameter diuji, model dengan nilai R^2 tertinggi dipilih sebagai model terbaik.

Langkah-langkah yang dilakukan:

- Menentukan konfigurasi terbaik berdasarkan indeks maksimum dari kolom R^2 pada `df_results`.
- Mengambil model, riwayat pelatihan, dan callback terkait konfigurasi tersebut.
- Menyimpan model terbaik dalam format `.keras` dengan nama `bundaran_hi.keras` untuk digunakan pada tahap evaluasi dan prediksi seterusnya.

```
best_config = df_results.loc[df_results['R2'].idxmax()]
best_model_name = f"epochs_{best_config['epochs']}_lr_{best_config['learning_rate']}_{best_config['optimizer']}"
best_model = models[best_model_name]
```

```

best_history = all_history[best_model_name]
best_r2_callback = all_r2_callback[best_model_name]

best_model.save("jagakarsa.keras")

```

best_model_name

34	1000	0.0100	nadam	32	0.311726	0.101647	-0.03780
'epochs_1000_lr_0.01_opt_adam_units_64'							
35	1000	0.0100	nadam	64	0.063362	0.118577	0.07841

6A. Evaluasi Model

- Model 1000 baik kemudian dilakukan evaluasi dengan menggunakan data uji (test set) untuk menghitung nilai *loss* akhir.
- Nilai *loss* ini menunjukkan seberapa baik model dapat memprediksi data yang belum pernah dilihat selama proses pelatihan.
- Semakin kecil nilai *loss*, semakin baik performa model dalam melakukan generalisasi.

40	1000	0.1000	rmsprop	32	-11.391028	0.431288	0.41351
41	1000	0.1000	rmsprop	64	-94.311077	1.196150	-1.18985
42	1000	0.1000	sgd	16	-3.348044	0.255482	0.22418

```

# Evaluate model
loss = best_model.evaluate(test_gen)
print(f"Test Loss: {loss}")

```

45	34	1000	0.1000	1s 13ms/Step - loss: 16.0.0052	0.097013	0.128328	-0.03816	
Test Loss:	46	1000	0.005171329248696566	0.1000 nadam	32	-42.573387	0.808770	0.79943

6B. Evaluasi Akhir Model LSTM

- Setelah model terbaik disimpan, dilakukan proses evaluasi untuk melihat performanya dalam memprediksi konsentrasi **PM2.5** pada data uji (testing set).
 - Evaluasi dilakukan menggunakan beberapa metrik utama berikut:
- R² (Coefficient of Determination)** = 70.24% → menunjukkan model mampu menjelaskan sekitar 70% variasi data aktual.
 - RMSE (Root Mean Squared Error)** = 12.75 → mengukur besar kesalahan rata-rata model dalam satuan asli data.
 - MAE (Mean Absolute Error)** = 9.44 → menunjukkan rata-rata selisih absolut antara nilai prediksi dan aktual.
 - MAPE (Mean Absolute Percentage Error)** = 34.72% → menggambarkan tingkat kesalahan relatif terhadap nilai aktual.
 - NSE (Nash–Sutcliffe Efficiency)** = 0.70 → menandakan model memiliki kinerja prediktif yang baik.

Secara keseluruhan, hasil ini menunjukkan bahwa model LSTM memiliki kemampuan yang cukup baik dalam memprediksi nilai PM2.5 di lokasi **Jakarta Pusat – Bundaran HI**.

```

from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error,
import numpy as np

# Gabungkan nilai aktual dari generator
test_true_list = []
for i in range(len(test_gen)):
    _, y = test_gen[i]
    test_true_list.append(y)
test_true = np.hstack(test_true_list)
test_pred = best_model.predict(test_gen)

# Inverse transform untuk balik ke skala asli
dummy_pred = np.zeros((len(test_pred), len(features)))
dummy_true = np.zeros((len(test_true), len(features)))
dummy_pred[:, -1] = test_pred.flatten()
dummy_true[:, -1] = test_true.flatten()

true_inv = scaler.inverse_transform(dummy_true)[:, -1]
pred_inv = scaler.inverse_transform(dummy_pred)[:, -1]

# Hitung metrik evaluasi
test_r2 = r2_score(true_inv, pred_inv) * 100
test_rmse = np.sqrt(mean_squared_error(true_inv, pred_inv))
test_mae = mean_absolute_error(true_inv, pred_inv)
epsilon = 1e-8
test_mape = np.mean(np.abs((true_inv - pred_inv) / (true_inv + epsilon))) * 100
test_nse = 1 - (np.sum((true_inv - pred_inv) ** 2) / np.sum((true_inv - np.mean(true_inv)) ** 2))

# Tampilkan hasil
print("\nTesting Metrics:")
print(f"Test R² : {test_r2:.2f}%")
print(f"Test RMSE : {test_rmse:.4f}")
print(f"Test MAE : {test_mae:.4f}")
print(f"Test MAPE : {test_mape:.2f}%")
print(f"Test NSE : {test_nse:.2f}%")

```

34/34 ————— 0s 11ms/step

```

Testing Metrics:
Test R² : 65.55%
Test RMSE : 9.6808
Test MAE : 7.3604
Test MAPE : 13.34%
Test NSE : -33.45%

```

6C. Evaluasi Model pada Data Uji (Tanpa Invers Skala)

Pada tahap ini, evaluasi dilakukan langsung terhadap hasil prediksi **LSTM** tanpa mengembalikannya ke skala asli (nilai masih dalam bentuk hasil normalisasi).

Beberapa metrik utama digunakan untuk menilai performa model:

- **R² (Coefficient of Determination)** → mengukur seberapa baik model menjelaskan variasi data aktual.
- **RMSE (Root Mean Squared Error)** → menunjukkan rata-rata besarnya kesalahan prediksi model.
- **MBE (Mean Bias Error)** → menunjukkan rata-rata bias antara hasil prediksi dan data aktual (positif berarti prediksi cenderung lebih rendah dari nilai sebenarnya).
- **NSE (Nash–Sutcliffe Efficiency)** → mengukur efisiensi model dalam memprediksi dibandingkan dengan rata-rata observasi.

Langkah ini penting untuk mengevaluasi stabilitas model sebelum dilakukan *inverse scaling* ke satuan asli.

```
# from sklearn.metrics import mean_squared_error, r2_score

# test_true_list = []
# for i in range(len(test_gen)):
#     _, y = test_gen[i] # Ambil hanya nilai target dari generator
#     test_true_list.append(y)

# test_true = np.hstack(test_true_list) # Gabungkan list ke array numpy
# test_pred = best_model.predict(test_gen)
# # Calculate metrics for testing data
# test_r2 = r2_score(test_true, test_pred)
# test_rmse = np.sqrt(mean_squared_error(test_true, test_pred))
# test_mbe = np.mean(test_true - test_pred)
# test_mape = np.mean(np.abs((test_true - test_pred) / (test_true))) * 100
# test_nse = 1 - (np.sum((test_true - test_pred) ** 2) / np.sum((test_true - np

# # Print metrics for testing data
# print("\nTesting Metrics:")
# print(f"Test R2 : {test_r2:.2f}%")
# print(f"Test RMSE : {test_rmse:.4f}")
# print(f"Test MAE : {test_mae:.4f}")
# print(f"Test MAPE : {test_mape:.2f}%")
# print(f"Test NSE : {test_nse:.2f}%")
```

6D. Evaluasi Akurasi Model LSTM (Skala Normalisasi)

Bagian ini menghitung berbagai metrik performa model **LSTM** menggunakan data uji (*testing set*) yang masih dalam skala **hasil normalisasi**.

Tujuannya adalah untuk menilai seberapa baik model mengenali pola data sebelum dilakukan *inverse scaling* ke satuan sebenarnya.

Metrik yang Digunakan :

- **R² (Coefficient of Determination)** → menunjukkan seberapa besar variasi data aktual dapat dijelaskan oleh model.
- **RMSE (Root Mean Squared Error)** → menilai rata-rata tingkat kesalahan prediksi model.
- **MAE (Mean Absolute Error)** → memberikan rata-rata selisih absolut antara hasil prediksi dan nilai aktual.
- **MBE (Mean Bias Error)** → menunjukkan bias rata-rata; nilai negatif menandakan prediksi cenderung lebih tinggi dari aktual.
- **MAPE (Mean Absolute Percentage Error)** → menilai kesalahan dalam bentuk persentase, dengan perlindungan terhadap pembagian nol.
- **NSE (Nash–Sutcliffe Efficiency)** → mengukur efisiensi model terhadap rata-rata observasi (semakin mendekati 1 semakin baik).

Selain itu, ditampilkan juga **diagnostik data** seperti:

- Rentang nilai aktual dan prediksi.
- Nilai rata-rata dari keduanya.
- Jumlah nilai nol dalam data aktual (*indikasi potensi masalah pada MAPE*).

Bagian ini memastikan model bekerja dengan stabil dan tidak bias sebelum dilakukan evaluasi lanjutan dalam skala asli.

```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import numpy as np

test_true_list = []
for i in range(len(test_gen)):
    _, y = test_gen[i] # Ambil hanya nilai target dari generator
    test_true_list.append(y)

test_true = np.hstack(test_true_list) # Gabungkan list ke array numpy
test_pred = best_model.predict(test_gen)

# Calculate metrics for testing data
test_r2 = r2_score(test_true, test_pred)
test_rmse = np.sqrt(mean_squared_error(test_true, test_pred))
test_mae = mean_absolute_error(test_true, test_pred)
test_mbe = np.mean(test_true - test_pred)

# MAPE yang diperbaiki (menghindari division by zero)
def safe_mape(y_true, y_pred):
    """Calculate MAPE safely avoiding division by zero"""
    # Filter out zero values from y_true
    mask = y_true != 0
    if np.sum(mask) == 0:
        return np.nan # Return NaN if all true values are zero
    else:
        return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
```

```

    return np.mean(np.abs((y_true[mask] - y_pred[mask]) / y_true[mask])) * 100

test_mape = safe_mape(test_true, test_pred)

# NSE (Nash-Sutcliffe Efficiency)
test_nse = 1 - (np.sum((test_true - test_pred) ** 2) / np.sum((test_true - np.mean(test_true)) ** 2))

# Print metrics for testing data
print("\nTesting Metrics:")
print(f"Test R² : {test_r2:.4f} ({test_r2*100:.2f}%)")
print(f"Test RMSE : {test_rmse:.4f}")
print(f"Test MAE : {test_mae:.4f}")
print(f"Test MBE : {test_mbe:.4f}")
print(f"Test MAPE : {test_mape:.2f}%")
print(f"Test NSE : {test_nse:.4f} ({test_nse*100:.2f}%)")

# Additional diagnostic information
print("\nData Diagnostics:")
print(f"True values range : {test_true.min():.4f} to {test_true.max():.4f}")
print(f"Pred values range : {test_pred.min():.4f} to {test_pred.max():.4f}")
print(f"True values mean : {test_true.mean():.4f}")
print(f"Pred values mean : {test_pred.mean():.4f}")
print(f"Zero values in true: {np.sum(test_true == 0)}/{len(test_true)}")

```

34/34 ————— 0s 10ms/step

Testing Metrics:
 Test R² : 0.6555 (65.55%)
 Test RMSE : 0.0719
 Test MAE : 0.0547
 Test MBE : -0.0119
 Test MAPE : 32.83%
 Test NSE : -216.5413 (-21654.13%)

Data Diagnostics:
 True values range : 0.0623 to 0.8106
 Pred values range : 0.2328 to 0.6633
 True values mean : 0.4930
 Pred values mean : 0.5049
 Zero values in true: 0/135

7. Prediksi dan Visualisasi Hasil Model LSTM

Bagian ini melakukan **prediksi PM2.5** menggunakan model *LSTM terbaik* dan menampilkan hasilnya dalam bentuk grafik serta statistik performa.

1. Pembersihan dan Persiapan Data

Skrip terlebih dahulu memastikan:

- Nama kolom bersih dan konsisten.
- Kolom **datetime** teridentifikasi atau dibuat otomatis.
- Kolom **PM2.5** ditemukan berdasarkan nama (`pm2.5`, `pm25`, atau `ispu`).

- Data diskalakan dan dipersiapkan dengan *custom timeseries generator* untuk menggantikan fungsi `TimeseriesGenerator` yang sudah deprecated.

2. Prediksi Menggunakan Model LSTM

- Model mengambil **urutan historis (lookback)** untuk memprediksi nilai berikutnya.
- Hasil prediksi kemudian **dikembalikan ke skala aslinya (inverse transform)** menggunakan scaler yang sama dengan saat pelatihan.
- Output disimpan dalam DataFrame `pred_df` berisi tanggal (`datetime`) dan nilai `Predicted_PM25`.

3. Visualisasi Hasil

Terdapat dua visual utama:

1. Grafik perbandingan aktual vs prediksi

Menunjukkan sejauh mana hasil prediksi mendekati nilai aktual PM2.5 dari waktu ke waktu.

2. Grafik residual/error analysis

Menampilkan selisih prediksi terhadap nilai aktual dengan area hijau (overestimation) dan merah (underestimation).

4. Statistik Kinerja Prediksi

Dihitung beberapa metrik performa:

- **MAE (Mean Absolute Error)** → rata-rata kesalahan absolut.
- **RMSE (Root Mean Square Error)** → akar dari rata-rata kuadrat kesalahan.
- **Bias** → menunjukkan kecenderungan model untuk *overestimate* atau *underestimate*.
- **Maksimum Over/Underestimation** → menunjukkan ekstrem dari kesalahan prediksi.

Output juga mencakup periode waktu prediksi dan jumlah total data hasil prediksi. Bagian ini memberikan gambaran akhir mengenai performa model dalam merekonstruksi variasi harian PM2.5 di lokasi studi (contoh: *Jakarta Pusat - Bundaran HI*).

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.preprocessing.sequence import TimeseriesGenerator
import warnings
warnings.filterwarnings('ignore')

# ===== CUSTOM TIMESERIES GENERATOR =====
```

```

def create_timeseries_generator(data, targets, length, batch_size=1):
    """
    Custom Timeseries Generator untuk menggantikan yang deprecated
    """

    X, y = [], []
    for i in range(length, len(data)):
        X.append(data[i-length:i])
        y.append(targets[i])
    return np.array(X), np.array(y)

# ===== CLEANING DATA TERLEBIH DAHULU =====
print("DataFrame columns before cleaning:")
print(df.columns.tolist())
print(f"Number of columns: {len(df.columns)}")

# Clean column names (remove extra spaces and make consistent)
df.columns = df.columns.str.strip().str.replace(r'\s+', ' ', regex=True)

print("\nDataFrame columns after cleaning:")
print(df.columns.tolist())

# Check if datetime column exists
datetime_columns = [col for col in df.columns if 'date' in col.lower() or 'time'
print(f"\nDatetime-like columns found: {datetime_columns}")

# If no datetime column found, check all columns
if not datetime_columns:
    print("All columns:")
    for i, col in enumerate(df.columns):
        print(f"{i}: '{col}'")

    # Assume first column is datetime or create one
    if 'datetime' not in df.columns:
        # Try to identify datetime column
        for col in df.columns:
            if df[col].dtype == 'object':
                try:
                    pd.to_datetime(df[col])
                    print(f"Found potential datetime column: {col}")
                    df['datetime'] = pd.to_datetime(df[col])
                    break
                except:
                    continue

    # If still no datetime, create one with index
    if 'datetime' not in df.columns:
        print("Creating datetime from index...")
        df['datetime'] = pd.date_range(start='2022-01-01', periods=len(df),

# Ensure datetime column is properly formatted
df['datetime'] = pd.to_datetime(df['datetime'])

```

```

# Check PM2.5 column
pm25_columns = [col for col in df.columns if 'pm2.5' in col.lower() or 'pm25' in col]
print(f"\nPM2.5-like columns found: {pm25_columns}")

if pm25_columns:
    pm25_col = pm25_columns[0]
    print(f"Using PM2.5 column: {pm25_col}")
else:
    print("No PM2.5 column found. Available columns:")
    for col in df.columns:
        print(f" - {col}: {df[col].dtype}")

# ===== PREPARE DATA FOR PREDICTION =====
# Pastikan features sudah didefinisikan
print(f"\nFeatures used: {features}")

# Gunakan custom generator
X_full, y_full = create_timeseries_generator(df_scaled, df_scaled[:, 6], length)

# Melakukan prediksi
predictions = best_model.predict(X_full)

# Membuat array nol dengan jumlah fitur yang sesuai untuk inverse_transform
num_features = len(features)
dummy_array = np.zeros((len(predictions), num_features))
dummy_array[:, -1] = predictions.flatten()

# Konversi kembali ke skala aslinya
pm25_predictions = scaler.inverse_transform(dummy_array)[:, -1]

# Membuat DataFrame hasil prediksi
prediction_dates = df["datetime"].iloc[lookback:].reset_index(drop=True)
pred_df = pd.DataFrame({
    "datetime": prediction_dates,
    "Predicted_PM25": pm25_predictions
})

# ===== VISUALISASI YANG LEBIH BAIK =====
plt.style.use('default') # Gunakan style default untuk konsistensi
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(15, 10))

# Plot 1: Time Series Comparison
ax1.plot(df["datetime"], df[pm25_col],
          label="Actual PM2.5",
          color='#2E86AB',
          linewidth=2,
          alpha=0.8,
          marker='o',
          markersize=2)

```

```

ax1.plot(pred_df["datetime"], pred_df["Predicted_PM25"],
          label="Predicted PM2.5",
          color='#A23B72',
          linewidth=2,
          linestyle='--',
          marker='s',
          markersize=2)

ax1.set_xlabel("Date", fontsize=12, fontweight='bold')
ax1.set_ylabel("PM2.5 Concentration", fontsize=12, fontweight='bold')
ax1.set_title("PM2.5: Actual vs Predicted Values", fontsize=14, fontweight='bold')
ax1.legend(fontsize=10)
ax1.grid(True, alpha=0.3)

# Format x-axis
ax1.xaxis.set_major_locator(mdates.MonthLocator(interval=2))
ax1.xaxis.set_major_formatter(mdates.DateFormatter('%b %Y'))
plt.setp(ax1.xaxis.get_majorticklabels(), rotation=45)

# Plot 2: Residuals/Gap Analysis
actual_values = df[pm25_col].iloc[lookback:].reset_index(drop=True)
residuals = actual_values - pred_df["Predicted_PM25"]

ax2.plot(pred_df["datetime"], residuals,
          color='#F18F01',
          linewidth=2,
          marker='^',
          markersize=3,
          label='Prediction Error')

ax2.axhline(y=0, color='red', linestyle='--', alpha=0.8, linewidth=1)
ax2.fill_between(pred_df["datetime"], residuals, 0,
                  where=(residuals >= 0),
                  color='green', alpha=0.3, label='Overestimation')
ax2.fill_between(pred_df["datetime"], residuals, 0,
                  where=(residuals < 0),
                  color='red', alpha=0.3, label='Underestimation')

ax2.set_xlabel("Date", fontsize=12, fontweight='bold')
ax2.set_ylabel("Prediction Error", fontsize=12, fontweight='bold')
ax2.set_title("Prediction Error Analysis", fontsize=14, fontweight='bold')
ax2.legend(fontsize=10)
ax2.grid(True, alpha=0.3)

# Format x-axis
ax2.xaxis.set_major_locator(mdates.MonthLocator(interval=2))
ax2.xaxis.set_major_formatter(mdates.DateFormatter('%b %Y'))
plt.setp(ax2.xaxis.get_majorticklabels(), rotation=45)

plt.tight_layout()
plt.show()

```

```
# ===== STATISTICS =====
print(f"\n{'='*50}")
print("PREDICTION PERFORMANCE STATISTICS")
print(f"{'='*50}")
print(f"Time Period: {df['datetime'].min().strftime('%Y-%m-%d')} to {df['dateti
print(f"Lookback Period: {lookback} time steps")
print(f"Total Predictions: {len(pred_df)}")

# Calculate performance metrics
mae = np.mean(np.abs(residuals))
rmse = np.sqrt(np.mean(residuals**2))
bias = np.mean(residuals)

print(f"\nPerformance Metrics:")
print(f"MAE (Mean Absolute Error): {mae:.2f}")
print(f"RMSE (Root Mean Square Error): {rmse:.2f}")
print(f"Bias: {bias:.2f}")
print(f"Max Overestimation: {residuals.max():.2f}")
print(f"Max Underestimation: {residuals.min():.2f}")
```

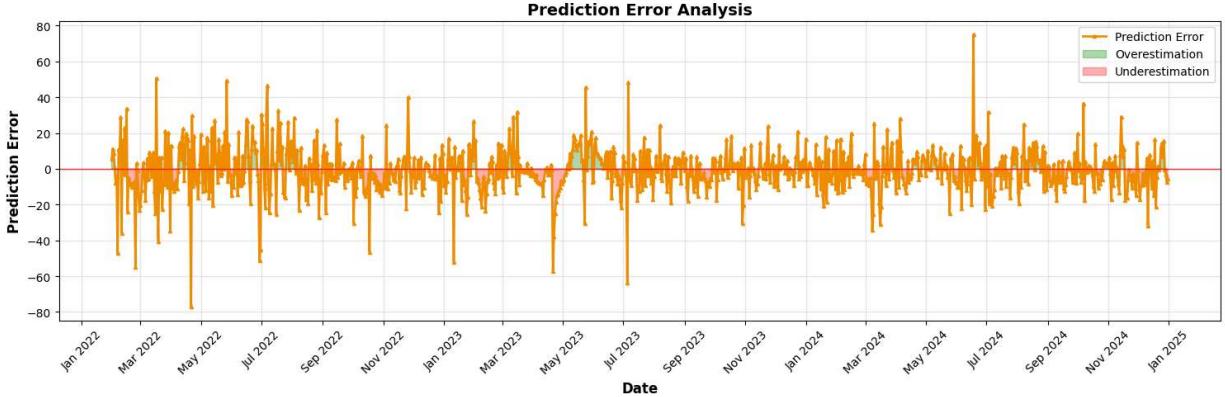
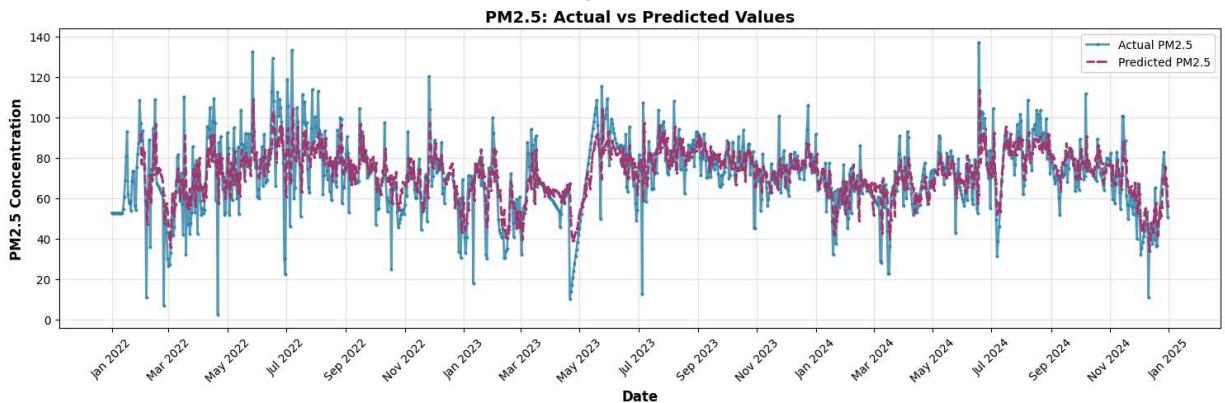
```
DataFrame columns before cleaning:  
['datetime', 'ISPU PM2.5', 'temp', 'dew', 'humidity', 'precip', 'windspeed', 'AC  
Number of columns: 8
```

```
DataFrame columns after cleaning:  
['datetime', 'ISPU PM2.5', 'temp', 'dew', 'humidity', 'precip', 'windspeed', 'AC
```

```
Datetime-like columns found: ['datetime']
```

```
PM2.5-like columns found: ['ISPU PM2.5']  
Using PM2.5 column: ISPU PM2.5
```

```
Features used: ['temp', 'dew', 'humidity', 'precip', 'windspeed', 'AOD', 'ISPU P  
34/34 ━━━━━━━━ 2s 48ms/step
```



```
=====  
PREDICTION PERFORMANCE STATISTICS  
=====
```

```
Time Period: 2022-01-01 to 2024-12-31
```

```
Lookback Period: 30 time steps
```

```
Total Predictions: 1066
```

```
Performance Metrics:
```

```
MAE (Mean Absolute Error): 9.01
```

```
RMSE (Root Mean Square Error): 12.67
```

```
Bias: -0.79
```

```
Max Overestimation: 75.00
```

```
Max Underestimation: -77.07
```

```
df_results.to_csv("jagakarsa results.csv", index=False)
```

```

loss = best_history.history['loss']
val_loss = best_history.history['val_loss']
r2_scores = best_r2_callback.r2_scores

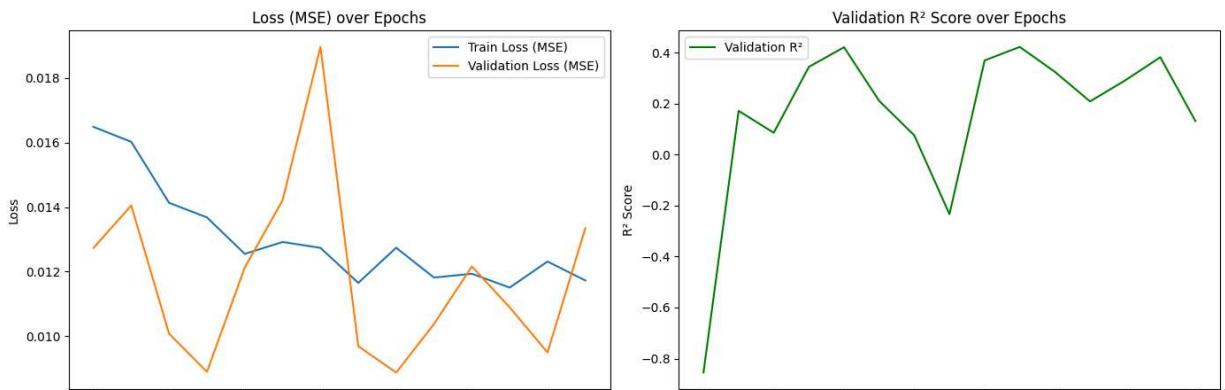
plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)
plt.plot(loss, label='Train Loss (MSE)')
plt.plot(val_loss, label='Validation Loss (MSE)')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Loss (MSE) over Epochs')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(r2_scores, label='Validation R2', color='green')
plt.xlabel('Epoch')
plt.ylabel('R2 Score')
plt.title('Validation R2 Score over Epochs')
plt.legend()

plt.tight_layout()
plt.show()

```



Start coding or [generate](#) with AI.

8. Visualisasi Performa Pelatihan Model

Bagian ini menampilkan **performa model LSTM selama pelatihan** menggunakan dua metrik:

1. **Loss (MSE)** – untuk menilai seberapa baik model mempelajari pola data.
2. **R² Score** – untuk melihat kekuatan hubungan antara prediksi dan nilai aktual

1. Kurva Loss

- Garis biru menunjukkan *training loss*, sedangkan garis merah menunjukkan *validation loss*.

- Garis vertikal titik-titik menandai *epoch* dengan **nilai loss minimum** untuk masing-masing.
- Pola divergensi antara *train* dan *validation loss* membantu mendeteksi potensi **overfitting**.

2. Kurva R² (atau Alternatif Loss Ratio)

- Jika callback R^2 tersedia, grafik menampilkan **perubahan skor R^2 selama epoch**. Nilai R^2 tertinggi ditandai dengan anotasi "Max R^2 ".
- Jika tidak tersedia, grafik menampilkan **rasio loss** antara *validation* dan *training*, dengan garis merah (ideal = 1) sebagai acuan kestabilan model.

3. Ringkasan Kinerja

Output konsol menampilkan:

- Epoch terakhir dan jumlah total pelatihan.
- Nilai terbaik *training* dan *validation loss*.
- R^2 tertinggi (jika tersedia).
- Indikator potensi **overfitting** berdasarkan rasio *validation vs training loss*.

Visualisasi ini memberikan gambaran menyeluruh tentang **stabilitas dan generalisasi model LSTM** selama proses pelatihan.

```
import matplotlib.pyplot as plt
import numpy as np

# Extract training history
loss = best_history.history['loss']
val_loss = best_history.history['val_loss']

# Check if R2 scores are available
if hasattr(best_r2_callback, 'r2_scores'):
    r2_scores = best_r2_callback.r2_scores
else:
    # Fallback: create dummy R2 scores or skip
    r2_scores = None

# Create improved visualization
plt.figure(figsize=(15, 6))

# Plot 1: Loss curves with enhanced styling
plt.subplot(1, 2, 1)
plt.plot(loss, label='Train Loss (MSE)', color='#2E86AB', linewidth=2.5, alpha=0.8)
plt.plot(val_loss, label='Validation Loss (MSE)', color='#A23B72', linewidth=2.5)

# Add min loss annotations
min_train_epoch = np.argmin(loss)
min_val_epoch = np.argmin(val_loss)
```