Air Temperature Forecasting with LSTM

Oleh: Crista Livia Budiman

```
# !pip install tensorflow

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import RobustScaler, StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout,
BatchNormalization
from tensorflow.keras.optimizers import Adam
import warnings
import random
warnings.filterwarnings('ignore')

SEED_VALUE = 42
np.random.seed(SEED_VALUE)
random.seed(SEED_VALUE)
tf.random.set_seed(SEED_VALUE)

df = pd.read_csv("AP004.csv")
```

Data Preparation

EDA yang perlu dilakukan:

1. Memeriksa informasi dasar dalam dataset
2. Cek missing values dan outliers
3. Visualisasi time series
4. Cek korelasi antar fitur

```
df.shape
```

```
(48802, 25)
```

Dataset awal terdiri atas 48,802 baris dan 25 kolom.

```
df
```

{"type":"dataframe","variable_name":"df"}

Berdasarkan data di atas, dapat diketahui bahwa dataset ini merupakan hasil observasi per jam, karena selisih waktu antara kolom 'From Date' dan 'To Date' hanya 1 jam, di mana hal tersebut tidak akan memberikan kontribusi yang signifikan terhadap proses prediksi. Oleh karena itu, diputuskan untuk drop kolom 'To Date' dari dataset.

```
df.drop(columns='To Date', inplace=True)

df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48802 entries, 0 to 48801
Data columns (total 24 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   From Date           48802 non-null  object 
 1   PM2.5 (ug/m3)       46344 non-null  float64
 2   PM10 (ug/m3)        46917 non-null  float64
 3   NO (ug/m3)          47244 non-null  float64
 4   NO2 (ug/m3)         47224 non-null  float64
 5   NOx (ppb)           46628 non-null  float64
 6   NH3 (ug/m3)         47140 non-null  float64
 7   SO2 (ug/m3)         46649 non-null  float64
 8   CO (mg/m3)          46387 non-null  float64
 9   Ozone (ug/m3)       47156 non-null  float64
 10  Benzene (ug/m3)     46914 non-null  float64
 11  Toluene (ug/m3)     46908 non-null  float64
 12  Eth-Benzene (ug/m3)  23988 non-null  float64
 13  MP-Xylene (ug/m3)   39256 non-null  float64
 14  Temp (degree C)     21599 non-null  float64
 15  RH (%)              47364 non-null  float64
 16  WS (m/s)            47375 non-null  float64
 17  WD (degree)         47373 non-null  float64
 18  SR (W/mt2)          47146 non-null  float64
 19  BP (mmHg)           47373 non-null  float64
 20  VWS (m/s)           47176 non-null  float64
 21  AT (degree C)       47286 non-null  float64
 22  RF (mm)             47510 non-null  float64
 23  Xylene (ug/m3)      47075 non-null  float64
dtypes: float64(23), object(1)
memory usage: 8.9+ MB

df['From Date'] = pd.to_datetime(df['From Date'])

df['hour'] = df['From Date'].dt.hour
df['month'] = df['From Date'].dt.month

df['hour_sin'] = np.sin(2 * np.pi * df['hour'] / 24)
df['hour_cos'] = np.cos(2 * np.pi * df['hour'] / 24)
```

```
df['month_sin'] = np.sin(2 * np.pi * df['month'] / 12)
df['month_cos'] = np.cos(2 * np.pi * df['month'] / 12)

df.drop(['hour', 'month'], axis=1, inplace=True)

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48802 entries, 0 to 48801
Data columns (total 28 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   From Date            48802 non-null  datetime64[ns]
 1   PM2.5 (ug/m3)        46344 non-null  float64
 2   PM10 (ug/m3)         46917 non-null  float64
 3   NO (ug/m3)           47244 non-null  float64
 4   NO2 (ug/m3)          47224 non-null  float64
 5   NOx (ppb)            46628 non-null  float64
 6   NH3 (ug/m3)          47140 non-null  float64
 7   SO2 (ug/m3)          46649 non-null  float64
 8   CO (mg/m3)           46387 non-null  float64
 9   Ozone (ug/m3)        47156 non-null  float64
 10  Benzene (ug/m3)      46914 non-null  float64
 11  Toluene (ug/m3)      46908 non-null  float64
 12  Eth-Benzene (ug/m3)  23988 non-null  float64
 13  MP-Xylene (ug/m3)    39256 non-null  float64
 14  Temp (degree C)      21599 non-null  float64
 15  RH (%)               47364 non-null  float64
 16  WS (m/s)             47375 non-null  float64
 17  WD (degree)          47373 non-null  float64
 18  SR (W/mt2)           47146 non-null  float64
 19  BP (mmHg)            47373 non-null  float64
 20  VWS (m/s)            47176 non-null  float64
 21  AT (degree C)        47286 non-null  float64
 22  RF (mm)              47510 non-null  float64
 23  Xylene (ug/m3)       47075 non-null  float64
 24  hour_sin             48802 non-null  float64
 25  hour_cos             48802 non-null  float64
 26  month_sin            48802 non-null  float64
 27  month_cos            48802 non-null  float64
dtypes: datetime64[ns](1), float64(27)
memory usage: 10.4 MB
```

```
df.set_index('From Date', inplace=True)

df = df.dropna(subset=['AT (degree C)'])

df.duplicated().sum()
```

```
np.int64(0)
```

Check Missing Values

```
df.isna().sum()
```

```
PM2.5 (ug/m3)           1042
PM10 (ug/m3)             470
NO (ug/m3)               143
NO2 (ug/m3)              164
NOx (ppb)                760
NH3 (ug/m3)              240
SO2 (ug/m3)              740
CO (mg/m3)               999
Ozone (ug/m3)            232
Benzene (ug/m3)          468
Toluene (ug/m3)          473
Eth-Benzene (ug/m3)    23387
MP-Xylene (ug/m3)       8239
Temp (degree C)        25829
RH (%)                    11
WS (m/s)                   1
WD (degree)                3
SR (W/mt2)               229
BP (mmHg)                  2
VWS (m/s)                352
AT (degree C)              0
RF (mm)                   25
Xylene (ug/m3)           461
hour_sin                   0
hour_cos                   0
month_sin                  0
month_cos                  0
dtype: int64
```

Kolom 'Temp (degree C)' memiliki lebih dari 50% missing values (27.203), sehingga dianggap kurang ideal untuk dipertahankan dalam dataset. Selain itu, karena tujuan analisis adalah memprediksi kualitas udara di suatu wilayah (AT), kolom ini dinilai tidak memberikan pengaruh yang signifikan terhadap proses prediksi sehingga diputuskan untuk drop kolom 'Temp' dari dataset.

Demikian pula, kolom 'Eth-Benzene (ug/m3)' juga memiliki lebih dari 50% missing value (24.814). Terlebih lagi, sudah ada kolom 'Benzene (ug/m3)' yang dianggap lebih relevan untuk prediksi dengan jumlah missing value yang jauh lebih sedikit (1.888), sehingga masih dapat diatasi dengan imputasi. Maka dari itu, diputuskan untuk juga drop kolom 'Eth-Benzene (ug/m3)' dari dataset.

Kolom ketiga dengan jumlah missing values terbanyak adalah 'MP-Xylene (ug/m3)', yaitu sebanyak 9546 missing values. Meskipun tidak sebanyak 2 kolom sebelumnya, jumlah ini hampir mencakup 20% dari jumlah baris sehingga perlu dipertambangkan apakah sebaiknya di drop atau imputasi. Namun setelah mengetahui bahwa terdapat kolom 'Xylene (ug/m3)' yang mencakup Xylene secara keseluruhan dan jumlah missing values yang jauh lebih sedikit (1.727) maka diputuskan untuk juga drop kolom 'MP-Xylene (ug/m3)' untuk mencegah terjadinya overfitting karena model mempelajari pola berulang dari dua fitur yang mirip.

Untuk kolom lain, tahap imputasi akan dilakukan setelah splitting

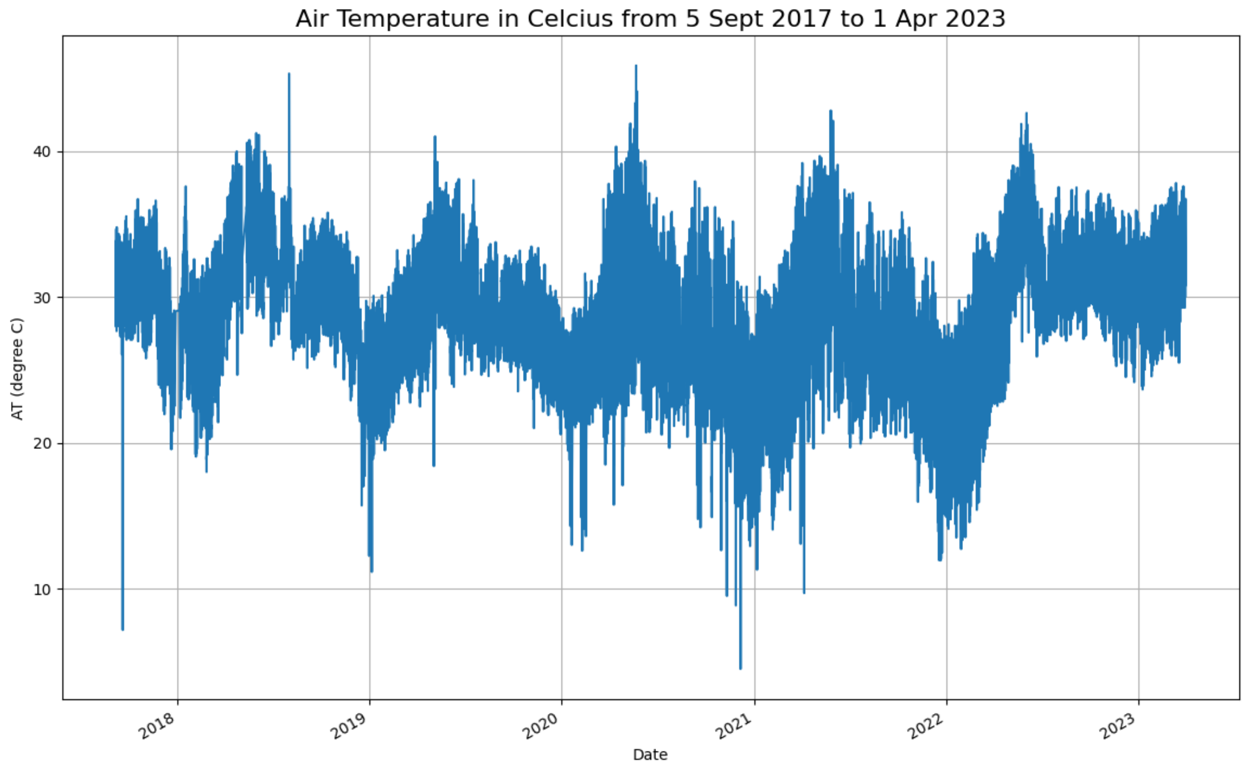
```
df = df.drop(columns=['Temp (degree C)', 'Eth-Benzene (ug/m3)', 'MP-
Xylene (ug/m3)'])

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 47286 entries, 2017-09-05 14:00:00 to 2023-03-31
23:00:00
Data columns (total 24 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   PM2.5 (ug/m3)      46244 non-null  float64
 1   PM10 (ug/m3)       46816 non-null  float64
 2   NO (ug/m3)         47143 non-null  float64
 3   NO2 (ug/m3)        47122 non-null  float64
 4   NOx (ppb)          46526 non-null  float64
 5   NH3 (ug/m3)        47046 non-null  float64
 6   SO2 (ug/m3)        46546 non-null  float64
 7   CO (mg/m3)         46287 non-null  float64
 8   Ozone (ug/m3)      47054 non-null  float64
 9   Benzene (ug/m3)    46818 non-null  float64
 10  Toluene (ug/m3)    46813 non-null  float64
 11  RH (%)             47275 non-null  float64
 12  WS (m/s)           47285 non-null  float64
 13  WD (degree)        47283 non-null  float64
 14  SR (W/mt2)         47057 non-null  float64
 15  BP (mmHg)          47284 non-null  float64
 16  VWS (m/s)          46934 non-null  float64
 17  AT (degree C)      47286 non-null  float64
 18  RF (mm)            47261 non-null  float64
 19  Xylene (ug/m3)     46825 non-null  float64
 20  hour_sin           47286 non-null  float64
 21  hour_cos           47286 non-null  float64
 22  month_sin          47286 non-null  float64
 23  month_cos          47286 non-null  float64
dtypes: float64(24)
memory usage: 9.0 MB
```

Visualisasi Time Series

```
df['AT (degree C)'].plot(figsize=(14, 9))
plt.title('Air Temperature in Celcius from 5 Sept 2017 to 1 Apr
2023',fontsize=16)
plt.xlabel('Date')
plt.ylabel('AT (degree C)')
```

```
plt.grid(True)
plt.show()
```



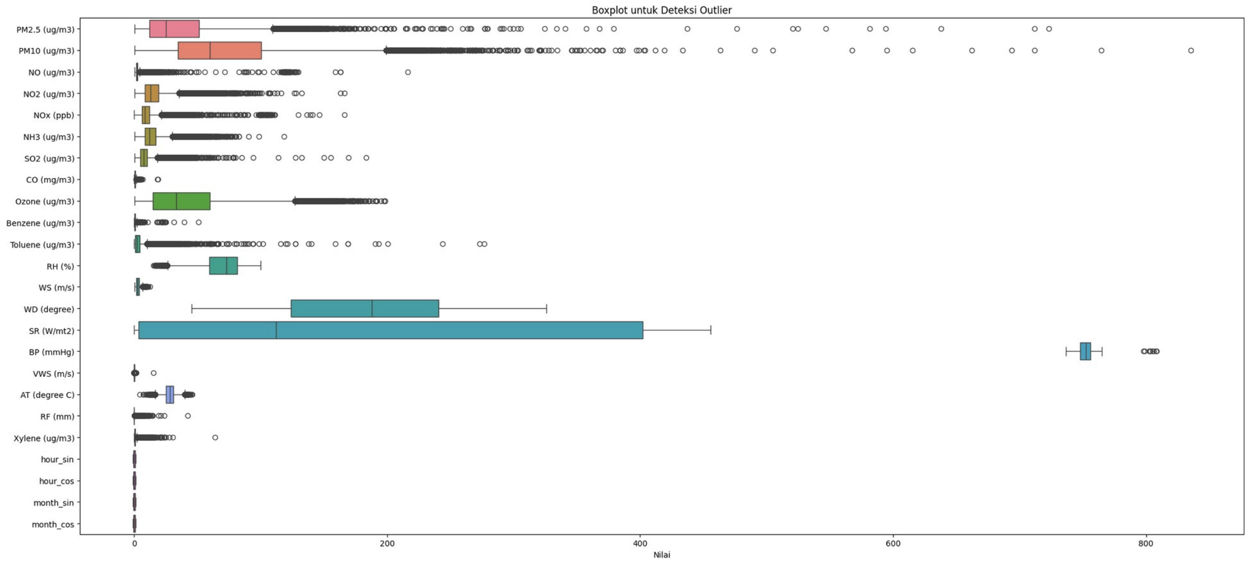
Dari plot tersebut, dapat diperoleh beberapa informasi yaitu:

- Grafik terlihat naik turun secara berkala.
- Terlihat ada fluktuasi ekstrim pada awal yang kemungkinan besar disebabkan oleh anomali cuaca atau kesalahan sensor.
- Tidak terlihat peningkatan atau penurunan suhu yang signfikan

Informasi ini penting untuk diketahui karena dapat membantu dalam memahami karakteristik data secara menyeluruh sebelum digunakan untuk pemodelan. Dalam membuat model LSTM, pemahaman terhadap pola musiman dan urutan waktu sangat diperlukan agar model dapat belajar secara optimal. Selain itu, deteksi awal terhadap outlier atau anomali juga sangat penting agar data yang digunakan bersih dan tidak mengganggu performa model.

Check Outliers

```
plt.figure(figsize=(22, 10))
sns.boxplot(data=df, orient="h")
plt.title("Boxplot untuk Deteksi Outlier")
plt.xlabel("Nilai")
plt.tight_layout()
plt.show()
```

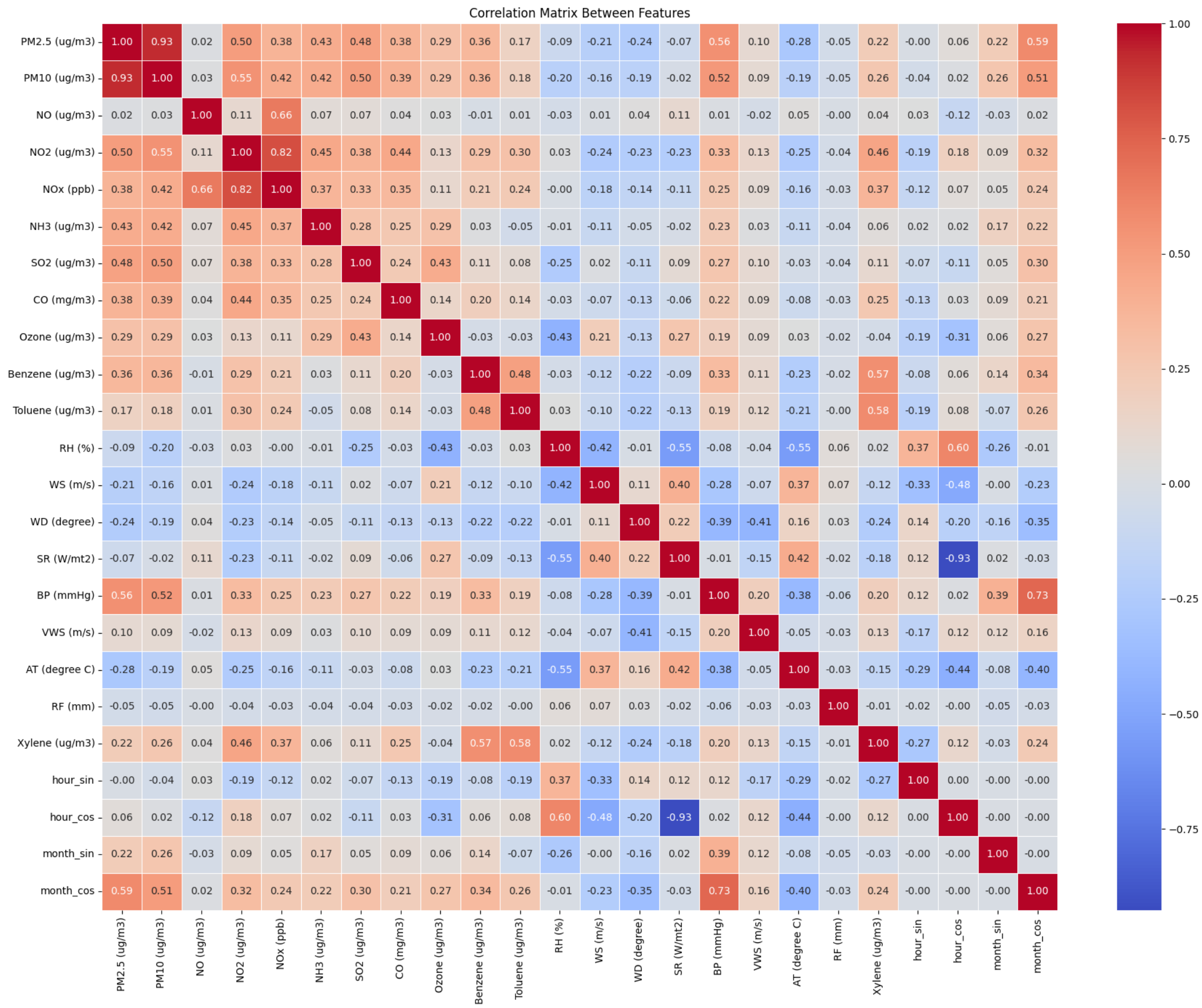



```
df.describe()

{"type":"dataframe"}
```

Melalui EDA ini, dapat diketahui bahwa terdapat beberapa kolom yang memiliki outliers ekstrim. Informasi ini penting untuk mengetahui metode scaling yang sesuai untuk dataset ini, yaitu Robust Scaler.

Correlation Matrix Between Features

```
corr_matrix = df.corr()

plt.figure(figsize=(18, 14))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm',
linewidths=0.5)
plt.title('Correlation Matrix Between Features')
plt.tight_layout()
plt.show()
```



Melalui correlation matrix, dapat diperoleh beberapa informasi yaitu:

- 'RH (%)' memiliki korelasi negatif terkuat terhadap AT (-0.55)
- 'SR (W/mt2)' memiliki korelasi positif terkuat terhadap AT (+0.42)
- 'SO2 (ug/m3)' dan 'RF (mm)' dengan korelasi terendah terhadap AT (-0.03)
- 'RF (mm)' tidak memiliki hubungan signifikan dengan fitur manapun karena nilai korelasinya sangat kecil, yaitu kurang dari -0.05 s/d +0.07

Informasi ini perlu diketahui agar dapat mengetahui korelasi antar fitur dengan target prediksi (AT), sehingga dapat membantu menghindari penggunaan fitur yang tidak memberikan kontribusi signifikan dalam modelling. Melalui informasi tersebut diputuskan untuk menghapus tiga kolom, yaitu 'SO2 (ug/m3)' dan 'RF (mm)' karena memiliki korelasi terendah terhadap target prediksi (AT).

```
df = df.drop(columns=['SO2 (ug/m3)','RF (mm)'])

df.columns
```

```
Index(['PM2.5 (ug/m3)', 'PM10 (ug/m3)', 'NO (ug/m3)', 'NO2 (ug/m3)',
       'NOx (ppb)', 'NH3 (ug/m3)', 'CO (mg/m3)', 'Ozone (ug/m3)',
       'Benzene (ug/m3)', 'Toluene (ug/m3)', 'RH (%)', 'WS (m/s)',
       'WD (degree)', 'SR (W/mt2)', 'BP (mmHg)', 'VWS (m/s)', 'AT
(degree C)',
       'Xylene (ug/m3)', 'hour_sin', 'hour_cos', 'month_sin',
'month_cos'],
      dtype='object')
```

Split Train Test Val

80% training, 10% validasi, dan 10% testing

```
train, val, test = np.split(
    df,
     [int(0.8 * len(df)),
      int(0.9 * len(df))])

print(f"Train: {len(train)}")
print(f"Val: {len(val)}")
print(f"Test: {len(test)}")
```

```
Train: 37828
Val: 4729
Test: 4729
```

Handle Missing Values and Outliers

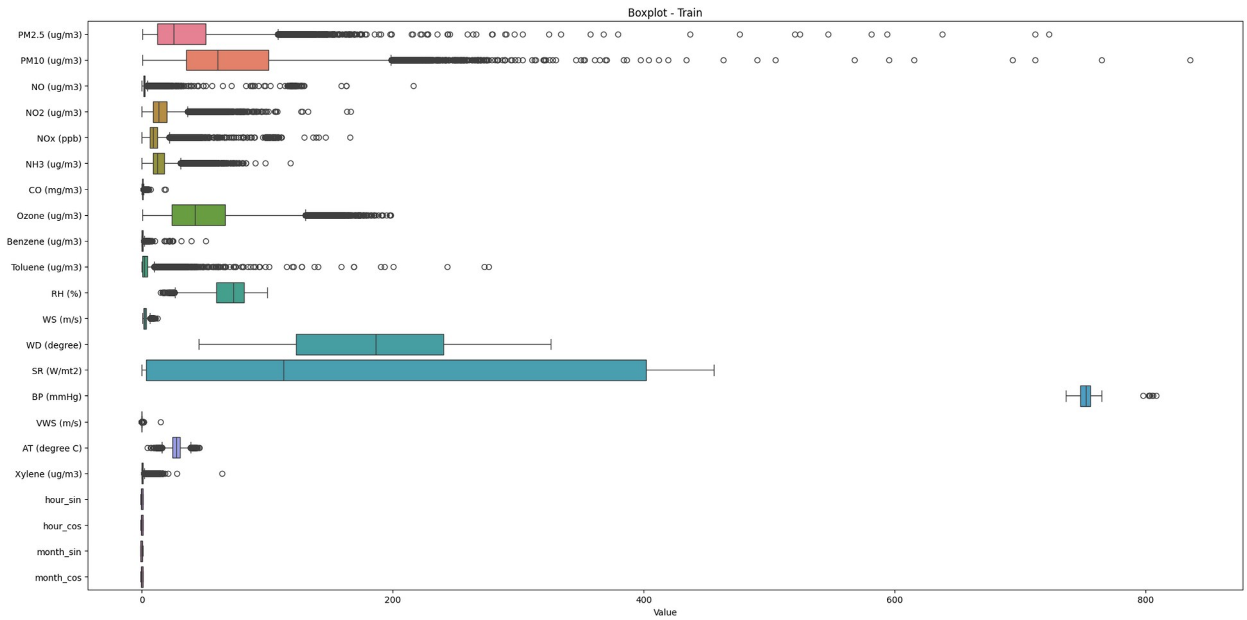
```
print("NULL in train:", np.isnan(train).sum().sum())
print("NULL in val:", np.isnan(val).sum().sum())
print("NULL in test:", np.isnan(test).sum().sum())
```

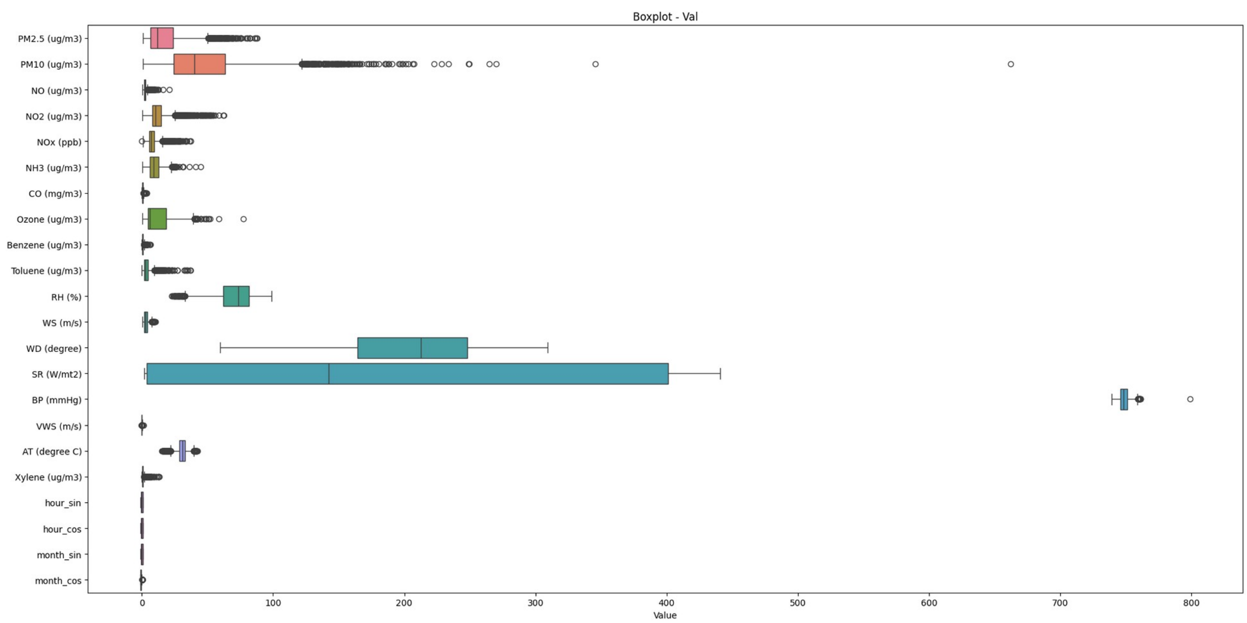
```
NULL in train: 5201
NULL in val: 483
NULL in test: 366
```

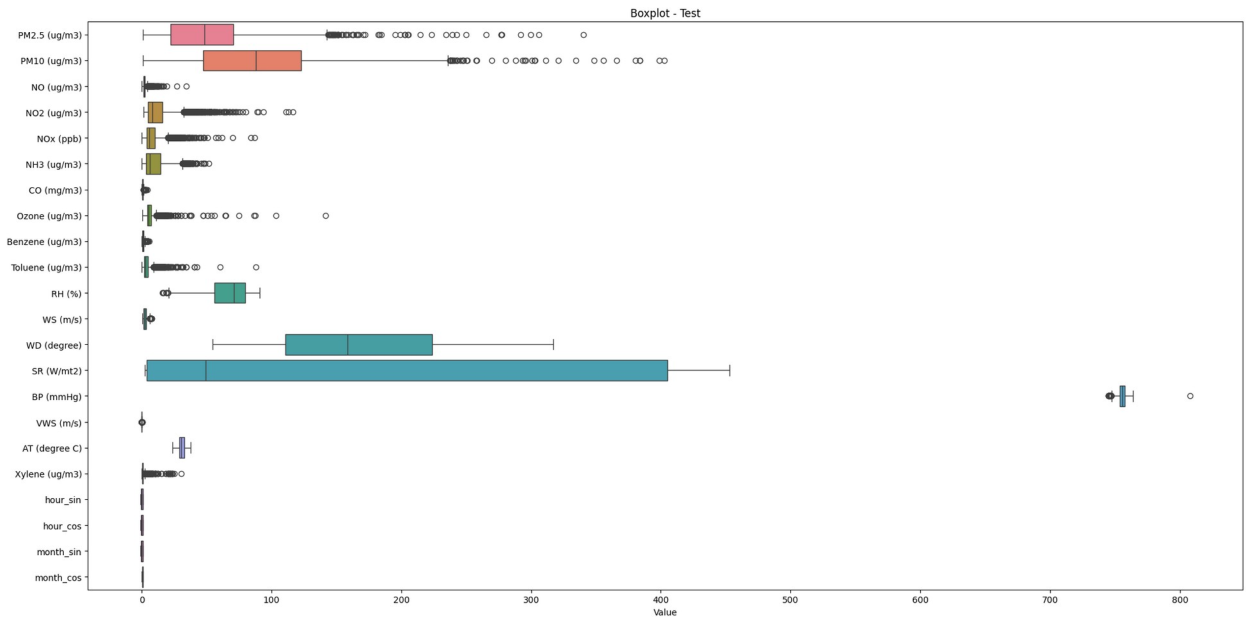
Metode imputasi yang digunakan adalah gabungan dari interpolasi liear dua arah dan metode fillna. Interpolasi linear memperkirakan nilai berdasarkan tren data di sekitar titik yang hilang, sedangkan fillna dengan metode forward-fill dan backward-fill digunakan sebagai pelengkap untuk mengisi nilai yang mungkin tidak terisi oleh interpolasi, terutama pada bagian awal atau akhir data. Oleh karena itu, metode ini sangat cocok karena dataset merupakan data time series yang sensitif terhadap urutan waktu.

```
def plot_boxplots(df, title):
    plt.figure(figsize=(20, 10))
    sns.boxplot(data=df, orient="h")
    plt.title(title)
    plt.xlabel("Value")
```

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

plot_boxplots(train, "Boxplot - Train")
plot_boxplots(val, "Boxplot - Val")
plot_boxplots(test, "Boxplot - Test")
```

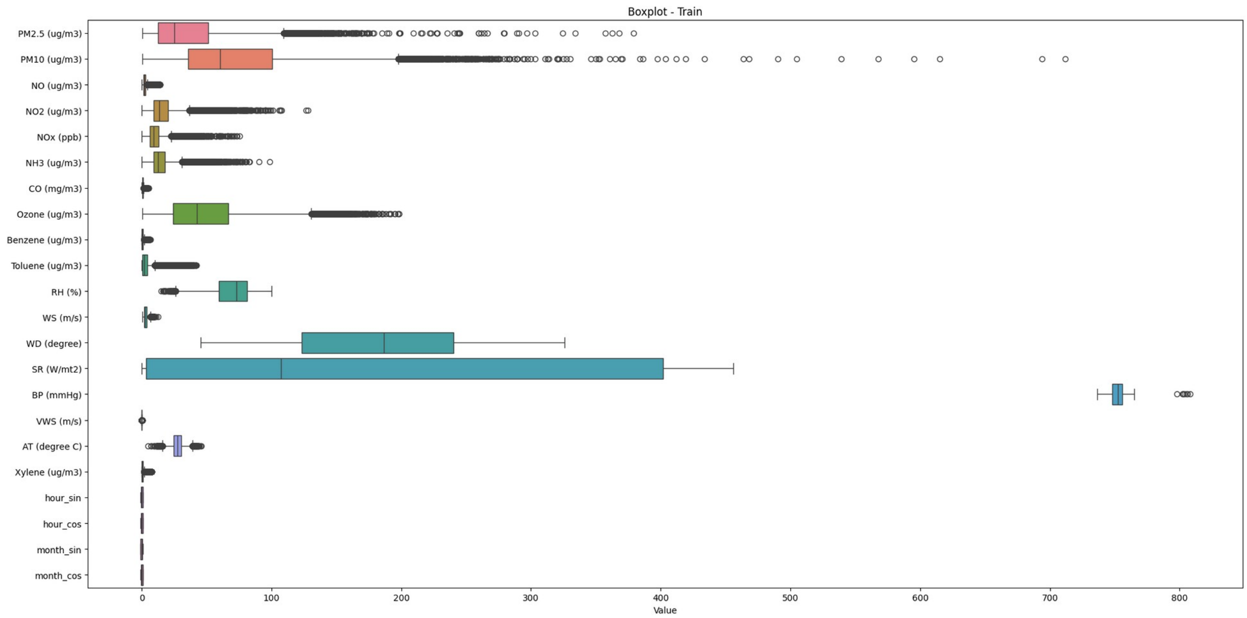


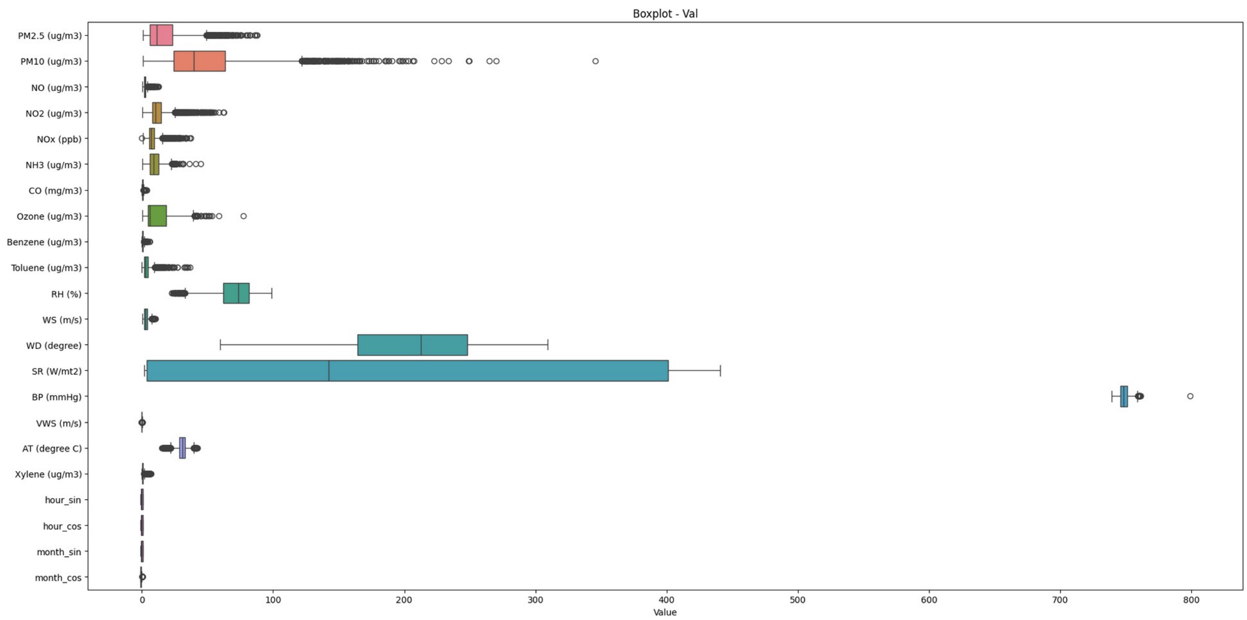


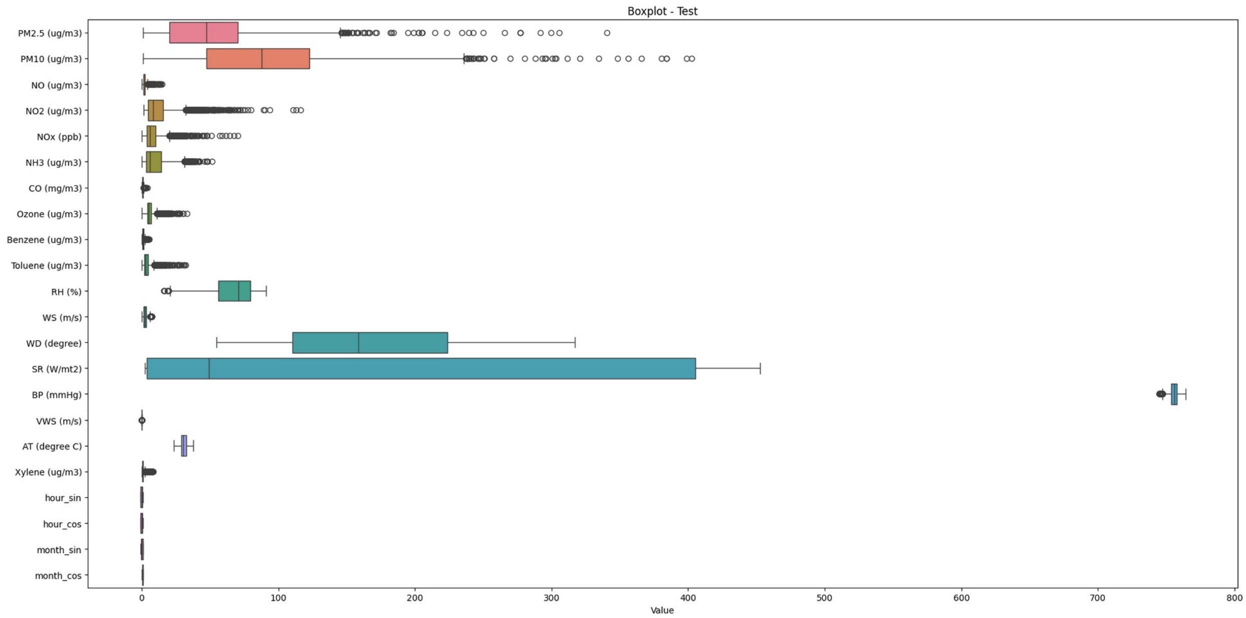



```
def handle_outliers(df_subset):
    for col in df_subset.columns:
        if col == 'AT (degree C)':
            continue

        col_data = df_subset[col].dropna()
        if not col_data.empty:
            Q1 = col_data.quantile(0.25)
            Q3 = col_data.quantile(0.75)
            IQR = Q3 - Q1
            extreme_upper = Q3 + 10 * IQR

            df_subset[col] = df_subset[col].mask(df_subset[col] >
extreme_upper)
            df_subset[col] = df_subset[col].interpolate(method='time')
            df_subset[col] =
df_subset[col].fillna(method='ffill').fillna(method='bfill')
    return df_subset

train = handle_outliers(train)
val = handle_outliers(val)
test = handle_outliers(test)

print("NULL in train:", np.isnan(train).sum().sum())
print("NULL in val:", np.isnan(val).sum().sum())
print("NULL in test:", np.isnan(test).sum().sum())

NULL in train: 0
NULL in val: 0
NULL in test: 0
```

```python
def plot_boxplots(df, title):
    plt.figure(figsize=(20, 10))
    sns.boxplot(data=df, orient="h")
    plt.title(title)
    plt.xlabel("Value")
    plt.tight_layout()
    plt.show()

plot_boxplots(train, "Boxplot - Train")
plot_boxplots(val, "Boxplot - Val")
plot_boxplots(test, "Boxplot - Test")
```







outliers yang diimpute hanya extreme outliers saja

Scaling

```
no_scale = train.columns[19:]
features_scaled = [col for col in train.columns if col not in
no_scale]
target = 'AT (degree C)'

train_scale = train.copy()
val_scale = val.copy()
test_scale = test.copy()

rs_x = RobustScaler()
train_scale[features_scaled] =
rs_x.fit_transform(train_scale[features_scaled])
val_scale[features_scaled] =
rs_x.transform(val_scale[features_scaled])
test_scale[features_scaled] =
rs_x.transform(test_scale[features_scaled])

rs_y = RobustScaler()
train_scale[[target]] = rs_y.fit_transform(train_scale[[target]])
val_scale[[target]] = rs_y.transform(val_scale[[target]])
test_scale[[target]] = rs_y.transform(test_scale[[target]])

train_df = pd.DataFrame(train_scale, columns=train.columns,
index=train.index)
val_df = pd.DataFrame(val_scale, columns=val.columns, index=val.index)
test_df = pd.DataFrame(test_scale, columns=test.columns,
index=test.index)
```

Windowing

Prediksi AT 1 jam ke depan menggunakan data 5 jam sebelumnya

```
def create_sequences(data, target_col, windowing=5, pred_step=1):
    x, y = [], []
    target_idx = data.columns.get_loc(target_col)
    for i in range(len(data) - windowing - pred_step + 1):
        x.append(data.iloc[i:i+windowing].values)
        y.append(data.iloc[i+windowing+pred_step-1, target_idx])
    return np.array(x), np.array(y)

x_train, y_train = create_sequences(train_df, target, windowing=5,
pred_step=1)
x_val, y_val = create_sequences(val_df, target, windowing=5,
pred_step=1)
x_test, y_test = create_sequences(test_df, target, windowing=5,
pred_step=1)
```

Modeling

```
epochs = 100
batch_size = 32
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                                  patience=10,

restore_best_weights=True)
```

LSTM Baseline

```
input_shape = (x_train.shape[1], x_train.shape[2])
hidden_size = 10

base_model = Sequential([
    LSTM(hidden_size, input_shape=input_shape,
return_sequences=False),
    Dense(1, activation='linear')
])

base_model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='mse',
    metrics=['mae']
)

base_model.summary()

Model: "sequential_1"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━
━━━━━┓
┃ Layer (type)                     ┃ Output Shape            ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
━━━━━┩
│ lstm_2 (LSTM)                    │ (None, 10)              │
1,320 │
├──────────────────────────────────┼─────────────────────────┼──────────
─────┤
│ dense_2 (Dense)                  │ (None, 1)               │
11 │
└──────────────────────────────────┴─────────────────────────┴──────────
─────┘

 Total params: 1,331 (5.20 KB)

 Trainable params: 1,331 (5.20 KB)

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

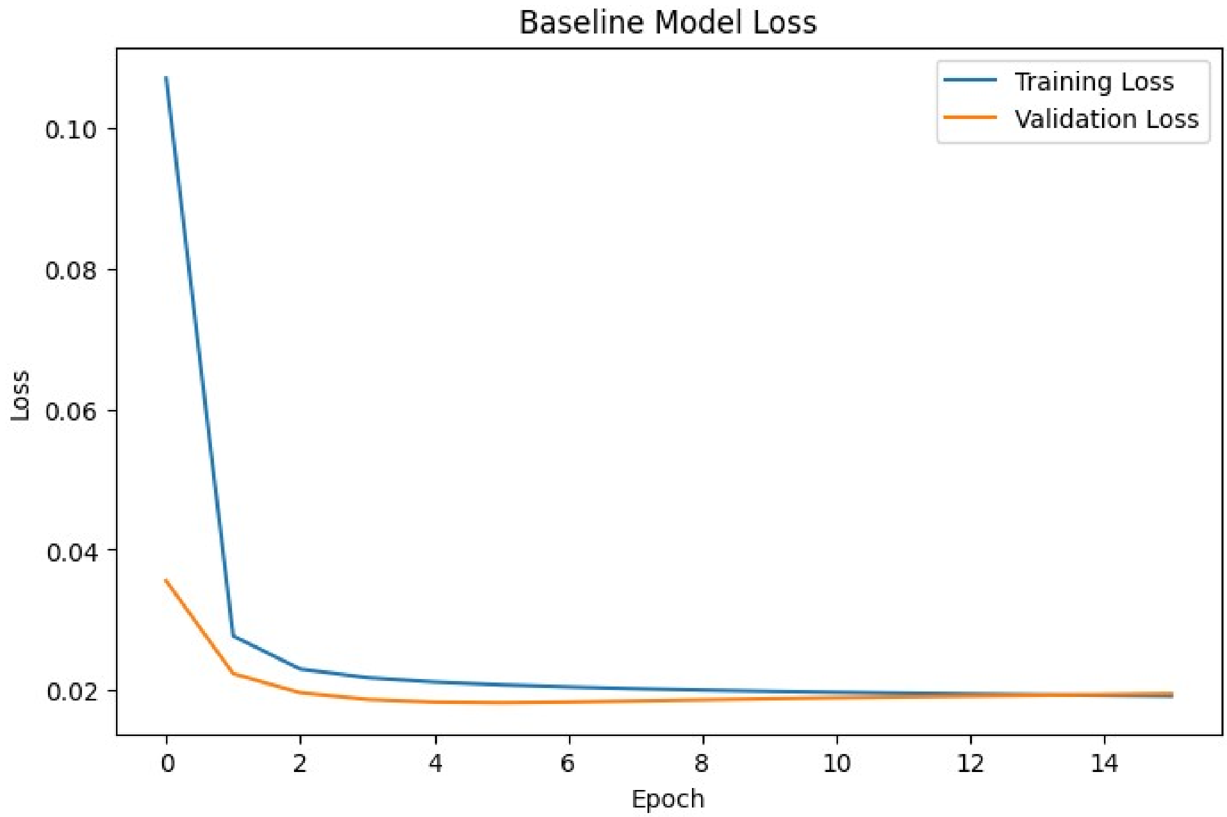
```
history = base_model.fit(
    x_train, y_train,
    validation_data=(x_val, y_val),
    epochs=epochs,
    batch_size=batch_size,
    callbacks=[early_stopping],
    verbose=1
)
```

```
Epoch 1/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 9s 6ms/step - loss: 0.2213 - mae:
0.3360 - val_loss: 0.0356 - val_mae: 0.1474
Epoch 2/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - loss: 0.0292 - mae:
0.1166 - val_loss: 0.0223 - val_mae: 0.1143
Epoch 3/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - loss: 0.0226 - mae:
0.0979 - val_loss: 0.0196 - val_mae: 0.1058
Epoch 4/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - loss: 0.0212 - mae:
0.0928 - val_loss: 0.0187 - val_mae: 0.1025
Epoch 5/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 0.0206 - mae:
0.0904 - val_loss: 0.0183 - val_mae: 0.1011
Epoch 6/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - loss: 0.0202 - mae:
0.0890 - val_loss: 0.0182 - val_mae: 0.1008
Epoch 7/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 0.0198 - mae:
```

```
0.0880 - val_loss: 0.0183 - val_mae: 0.1010
Epoch 8/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - loss: 0.0196 - mae:
0.0872 - val_loss: 0.0184 - val_mae: 0.1014
Epoch 9/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - loss: 0.0194 - mae:
0.0866 - val_loss: 0.0186 - val_mae: 0.1019
Epoch 10/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 0.0192 - mae:
0.0861 - val_loss: 0.0188 - val_mae: 0.1023
Epoch 11/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 0.0190 - mae:
0.0857 - val_loss: 0.0189 - val_mae: 0.1026
Epoch 12/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 0.0189 - mae:
0.0854 - val_loss: 0.0190 - val_mae: 0.1030
Epoch 13/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - loss: 0.0187 - mae:
0.0852 - val_loss: 0.0191 - val_mae: 0.1033
Epoch 14/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 0.0186 - mae:
0.0850 - val_loss: 0.0193 - val_mae: 0.1036
Epoch 15/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - loss: 0.0185 - mae:
0.0848 - val_loss: 0.0194 - val_mae: 0.1039
Epoch 16/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - loss: 0.0184 - mae:
0.0846 - val_loss: 0.0195 - val_mae: 0.1041
```

```python
plt.figure(figsize=(8, 5))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Baseline Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
```

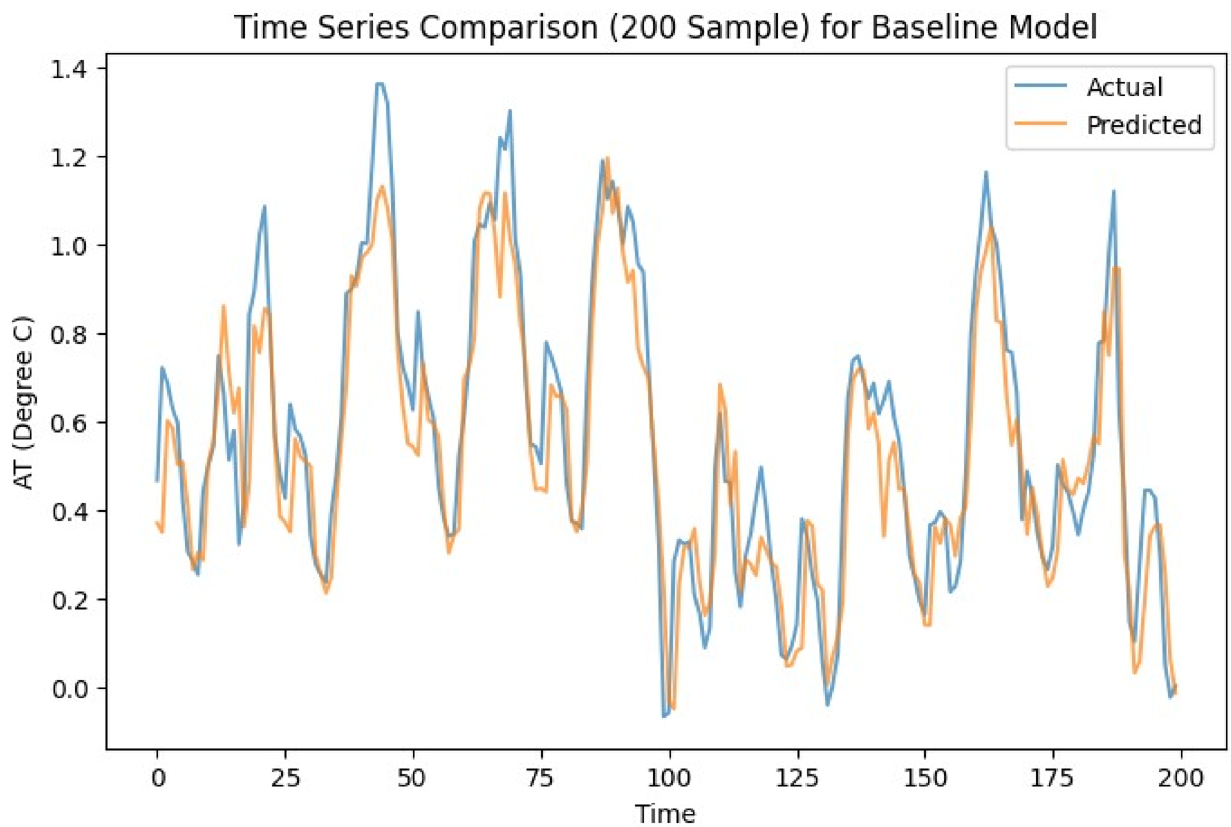
```
<matplotlib.legend.Legend at 0x7bab06987320>
```




```
y_pred_base_unscaled =
rs_y.inverse_transform(base_model.predict(x_test))

plt.figure(figsize=(8, 5))
sample_size = min(200, len(y_test))
plt.plot(y_test[:sample_size], label='Actual', alpha=0.7)
plt.plot(y_pred_base_unscaled[:sample_size], label='Predicted',
alpha=0.7)
plt.title('Time Series Comparison (200 Sample) for Baseline Model')
plt.xlabel('Time')
plt.ylabel('AT (Degree C)')
plt.legend()
```

```
148/148 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
```

```
<matplotlib.legend.Legend at 0x7baae7163980>
```



Modified Model

```
modified_model = Sequential([
    LSTM(128, return_sequences=True, input_shape=input_shape),
    Dropout(0.2),
    LSTM(64),
    Dense(32, activation='relu'),
    Dense(1, activation='linear')
])

modified_model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='mse',
    metrics=['mae'])

modified_model.summary()

Model: "sequential_2"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━
━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━
━━━━━┩
```

```
│ lstm_3 (LSTM)                   │ (None, 5, 128)         │
77,312 │
├─────────────────────────────────┼────────────────────────┼───────
───────┤
│ dropout_1 (Dropout)             │ (None, 5, 128)         │
0 │
├─────────────────────────────────┼────────────────────────┼───────
───────┤
│ lstm_4 (LSTM)                   │ (None, 64)             │
49,408 │
├─────────────────────────────────┼────────────────────────┼───────
───────┤
│ dense_3 (Dense)                 │ (None, 32)             │
2,080 │
├─────────────────────────────────┼────────────────────────┼───────
───────┤
│ dense_4 (Dense)                 │ (None, 1)              │
33 │
└─────────────────────────────────┴────────────────────────┴───────
───────┘

 Total params: 128,833 (503.25 KB)

 Trainable params: 128,833 (503.25 KB)

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

Hyperparameter Tuning for Learning Rate

```
learning_rates = [0.001, 0.0005, 0.0001]
results = {}
best_model = None
min_test_mse = float('inf')

for lr in learning_rates:
    tf.keras.backend.clear_session()
    modified_model =  Sequential([
        LSTM(128, return_sequences=True, input_shape=input_shape),
        Dropout(0.2),
        LSTM(64),
        Dense(32, activation='relu'),
        Dense(1, activation='linear')
        ])

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

    history_modified = modified_model.fit(
```

```
        x_train, y_train,
        validation_data=(x_val, y_val),
        epochs=epochs,
        batch_size=batch_size,
        callbacks=[early_stopping],
        verbose=1)

    y_pred_modified = modified_model.predict(x_test)
    loss = mean_squared_error(y_test, y_pred_modified)
    mae = mean_absolute_error(y_test, y_pred_modified)
    r2 = r2_score(y_test, y_pred_modified)

    results[lr] = {
        'MSE': loss,
        'MAE': mae,
        'R2' : r2,
        'history': history_modified
    }
    print(f"Learning rate: {lr} | MSE: {loss:.5f} | MAE: {mae:.5f} |
R2: {r2:.5f}")

    if loss < min_test_mse:
        min_test_mse = loss
        best_model = modified_model
        best_lr = lr

if best_model:
    print(f"Best learning rate untuk model LSTM yang sudah
dimodifikasi: {best_lr}")

Epoch 1/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 24s 17ms/step - loss: 0.0864 - mae:
0.2006 - val_loss: 0.0208 - val_mae: 0.1098
Epoch 2/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 19s 16ms/step - loss: 0.0273 - mae:
0.1124 - val_loss: 0.0220 - val_mae: 0.1141
Epoch 3/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 20s 17ms/step - loss: 0.0246 - mae:
0.1039 - val_loss: 0.0191 - val_mae: 0.1044
Epoch 4/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 19s 16ms/step - loss: 0.0230 - mae:
0.1000 - val_loss: 0.0169 - val_mae: 0.0961
Epoch 5/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 19s 16ms/step - loss: 0.0223 - mae:
0.0977 - val_loss: 0.0174 - val_mae: 0.0986
Epoch 6/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 19s 16ms/step - loss: 0.0216 - mae:
0.0962 - val_loss: 0.0173 - val_mae: 0.0982
Epoch 7/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 20s 16ms/step - loss: 0.0205 - mae:
```

```
0.0939 - val_loss: 0.0181 - val_mae: 0.1008
Epoch 8/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 19s 16ms/step - loss: 0.0202 - mae:
0.0932 - val_loss: 0.0173 - val_mae: 0.0982
Epoch 9/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 18s 15ms/step - loss: 0.0195 - mae:
0.0920 - val_loss: 0.0186 - val_mae: 0.1024
Epoch 10/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 20s 17ms/step - loss: 0.0189 - mae:
0.0908 - val_loss: 0.0195 - val_mae: 0.1050
Epoch 11/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 19s 16ms/step - loss: 0.0186 - mae:
0.0902 - val_loss: 0.0184 - val_mae: 0.1004
Epoch 12/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 21s 18ms/step - loss: 0.0179 - mae:
0.0891 - val_loss: 0.0187 - val_mae: 0.1012
Epoch 13/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 19s 16ms/step - loss: 0.0176 - mae:
0.0885 - val_loss: 0.0197 - val_mae: 0.1039
Epoch 14/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 21s 16ms/step - loss: 0.0174 - mae:
0.0880 - val_loss: 0.0203 - val_mae: 0.1056
148/148 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step
Learning rate: 0.001 | MSE: 0.01405 | MAE: 0.09193 | R2: 0.93488
Epoch 1/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 25s 17ms/step - loss: 0.1076 - mae:
0.2256 - val_loss: 0.0307 - val_mae: 0.1362
Epoch 2/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 20s 17ms/step - loss: 0.0319 - mae:
0.1225 - val_loss: 0.0240 - val_mae: 0.1189
Epoch 3/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 19s 16ms/step - loss: 0.0261 - mae:
0.1086 - val_loss: 0.0228 - val_mae: 0.1155
Epoch 4/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 20s 17ms/step - loss: 0.0240 - mae:
0.1019 - val_loss: 0.0196 - val_mae: 0.1054
Epoch 5/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 18s 15ms/step - loss: 0.0228 - mae:
0.0988 - val_loss: 0.0178 - val_mae: 0.0998
Epoch 6/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 19s 16ms/step - loss: 0.0222 - mae:
0.0964 - val_loss: 0.0189 - val_mae: 0.1037
Epoch 7/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 19s 16ms/step - loss: 0.0211 - mae:
0.0941 - val_loss: 0.0186 - val_mae: 0.1027
Epoch 8/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 20s 16ms/step - loss: 0.0209 - mae:
0.0935 - val_loss: 0.0191 - val_mae: 0.1047
Epoch 9/100
```

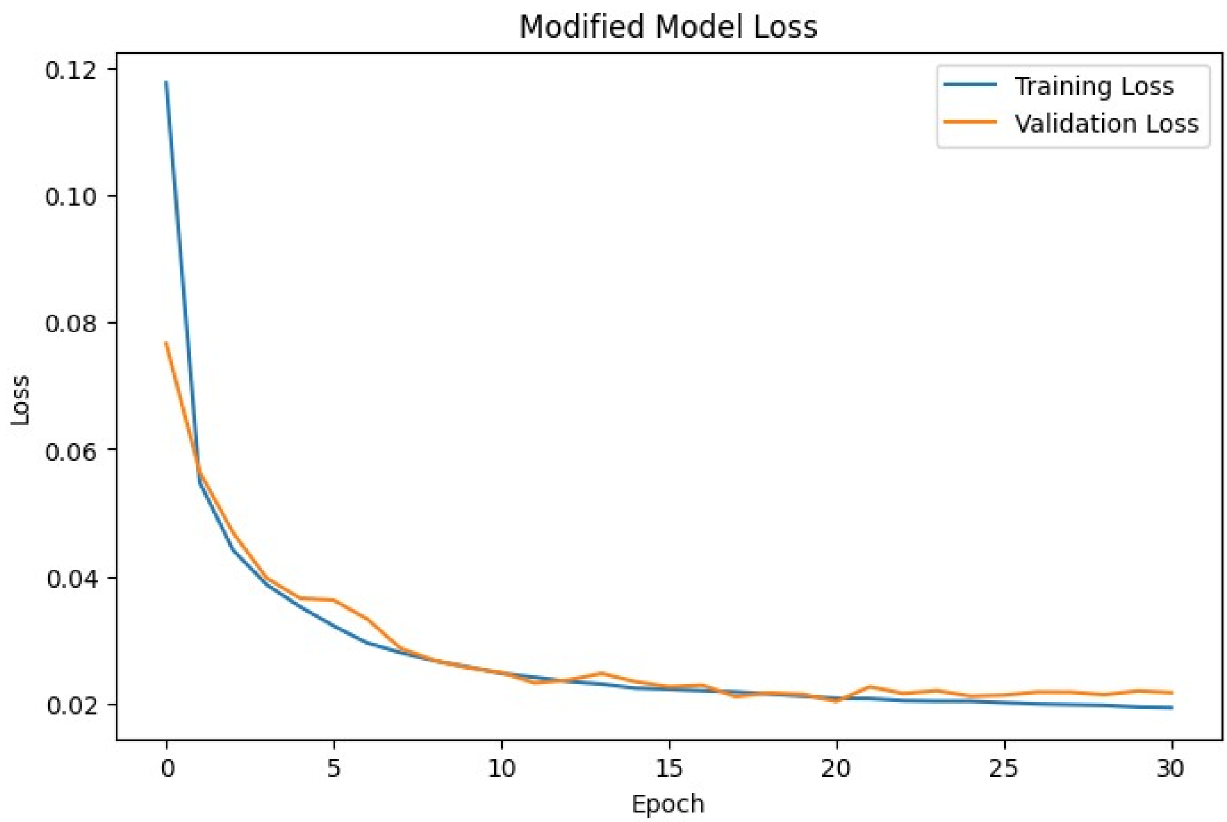
```
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 20s 17ms/step - loss: 0.0202 - mae:
0.0922 - val_loss: 0.0178 - val_mae: 0.1002
Epoch 10/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 21s 18ms/step - loss: 0.0199 - mae:
0.0912 - val_loss: 0.0191 - val_mae: 0.1045
Epoch 11/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 19s 16ms/step - loss: 0.0194 - mae:
0.0904 - val_loss: 0.0203 - val_mae: 0.1082
Epoch 12/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 21s 18ms/step - loss: 0.0192 - mae:
0.0896 - val_loss: 0.0191 - val_mae: 0.1044
Epoch 13/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 18s 15ms/step - loss: 0.0187 - mae:
0.0890 - val_loss: 0.0192 - val_mae: 0.1049
Epoch 14/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 20s 17ms/step - loss: 0.0183 - mae:
0.0885 - val_loss: 0.0188 - val_mae: 0.1037
Epoch 15/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 18s 16ms/step - loss: 0.0177 - mae:
0.0874 - val_loss: 0.0196 - val_mae: 0.1054
Epoch 16/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 19s 16ms/step - loss: 0.0176 - mae:
0.0868 - val_loss: 0.0200 - val_mae: 0.1070
Epoch 17/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 19s 16ms/step - loss: 0.0173 - mae:
0.0865 - val_loss: 0.0209 - val_mae: 0.1104
Epoch 18/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 20s 16ms/step - loss: 0.0173 - mae:
0.0865 - val_loss: 0.0204 - val_mae: 0.1078
Epoch 19/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 19s 16ms/step - loss: 0.0171 - mae:
0.0864 - val_loss: 0.0202 - val_mae: 0.1068
148/148 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step
Learning rate: 0.0005 | MSE: 0.02415 | MAE: 0.12629 | R2: 0.88807
Epoch 1/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 23s 17ms/step - loss: 0.2111 - mae:
0.3263 - val_loss: 0.0766 - val_mae: 0.2183
Epoch 2/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 18s 15ms/step - loss: 0.0580 - mae:
0.1695 - val_loss: 0.0564 - val_mae: 0.1855
Epoch 3/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 19s 16ms/step - loss: 0.0449 - mae:
0.1476 - val_loss: 0.0468 - val_mae: 0.1683
Epoch 4/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 18s 16ms/step - loss: 0.0386 - mae:
0.1366 - val_loss: 0.0397 - val_mae: 0.1550
Epoch 5/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 19s 16ms/step - loss: 0.0347 - mae:
0.1286 - val_loss: 0.0365 - val_mae: 0.1477
```

```
Epoch 6/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 18s 15ms/step - loss: 0.0318 - mae:
0.1234 - val_loss: 0.0362 - val_mae: 0.1477
Epoch 7/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 18s 15ms/step - loss: 0.0293 - mae:
0.1170 - val_loss: 0.0332 - val_mae: 0.1416
Epoch 8/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 19s 16ms/step - loss: 0.0275 - mae:
0.1134 - val_loss: 0.0286 - val_mae: 0.1306
Epoch 9/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 19s 15ms/step - loss: 0.0263 - mae:
0.1095 - val_loss: 0.0267 - val_mae: 0.1259
Epoch 10/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 19s 16ms/step - loss: 0.0253 - mae:
0.1074 - val_loss: 0.0256 - val_mae: 0.1227
Epoch 11/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 18s 15ms/step - loss: 0.0241 - mae:
0.1038 - val_loss: 0.0248 - val_mae: 0.1209
Epoch 12/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 20s 17ms/step - loss: 0.0234 - mae:
0.1018 - val_loss: 0.0232 - val_mae: 0.1162
Epoch 13/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 18s 16ms/step - loss: 0.0229 - mae:
0.1004 - val_loss: 0.0236 - val_mae: 0.1177
Epoch 14/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 18s 15ms/step - loss: 0.0224 - mae:
0.0991 - val_loss: 0.0246 - val_mae: 0.1206
Epoch 15/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 19s 16ms/step - loss: 0.0217 - mae:
0.0970 - val_loss: 0.0234 - val_mae: 0.1168
Epoch 16/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 18s 15ms/step - loss: 0.0215 - mae:
0.0961 - val_loss: 0.0226 - val_mae: 0.1148
Epoch 17/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 19s 16ms/step - loss: 0.0214 - mae:
0.0955 - val_loss: 0.0228 - val_mae: 0.1150
Epoch 18/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 18s 15ms/step - loss: 0.0213 - mae:
0.0947 - val_loss: 0.0210 - val_mae: 0.1098
Epoch 19/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 19s 16ms/step - loss: 0.0208 - mae:
0.0936 - val_loss: 0.0216 - val_mae: 0.1116
Epoch 20/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 18s 15ms/step - loss: 0.0205 - mae:
0.0926 - val_loss: 0.0214 - val_mae: 0.1113
Epoch 21/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 18s 15ms/step - loss: 0.0202 - mae:
0.0918 - val_loss: 0.0203 - val_mae: 0.1076
Epoch 22/100
```

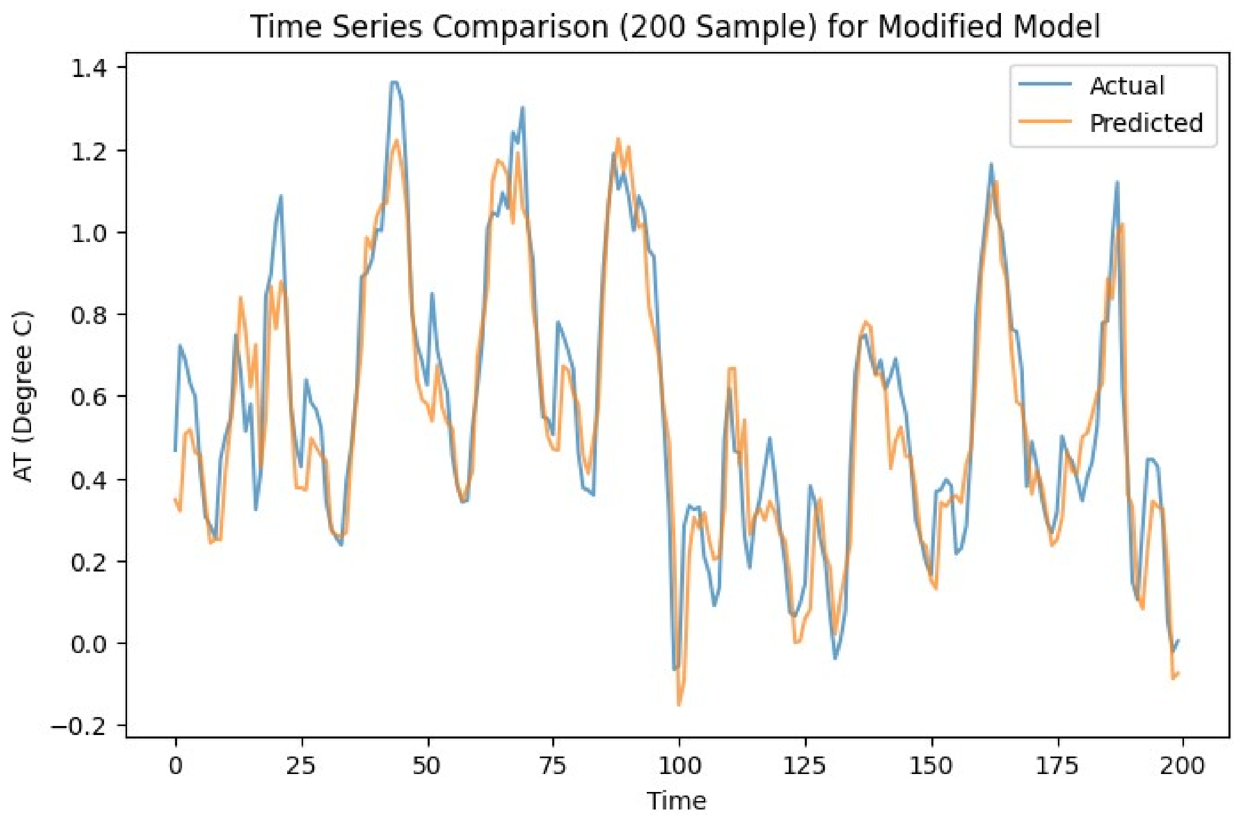
```
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 19s 16ms/step - loss: 0.0202 - mae: 
0.0919 - val_loss: 0.0225 - val_mae: 0.1147
Epoch 23/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 18s 15ms/step - loss: 0.0198 - mae: 
0.0912 - val_loss: 0.0215 - val_mae: 0.1114
Epoch 24/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 19s 16ms/step - loss: 0.0198 - mae: 
0.0903 - val_loss: 0.0219 - val_mae: 0.1123
Epoch 25/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 19s 16ms/step - loss: 0.0197 - mae: 
0.0904 - val_loss: 0.0211 - val_mae: 0.1101
Epoch 26/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 20s 17ms/step - loss: 0.0196 - mae: 
0.0902 - val_loss: 0.0212 - val_mae: 0.1105
Epoch 27/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 18s 15ms/step - loss: 0.0192 - mae: 
0.0894 - val_loss: 0.0217 - val_mae: 0.1119
Epoch 28/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 19s 16ms/step - loss: 0.0191 - mae: 
0.0891 - val_loss: 0.0217 - val_mae: 0.1119
Epoch 29/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 21s 18ms/step - loss: 0.0190 - mae: 
0.0887 - val_loss: 0.0213 - val_mae: 0.1107
Epoch 30/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 18s 16ms/step - loss: 0.0188 - mae: 
0.0884 - val_loss: 0.0219 - val_mae: 0.1125
Epoch 31/100
1182/1182 ━━━━━━━━━━━━━━━━━━━━ 20s 16ms/step - loss: 0.0187 - mae: 
0.0878 - val_loss: 0.0216 - val_mae: 0.1115
148/148 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step
Learning rate: 0.0001 | MSE: 0.03033 | MAE: 0.13874 | R2: 0.85941
Best learning rate untuk model LSTM yang sudah dimodifikasi: 0.001
```

```
plt.figure(figsize=(8, 5))
plt.plot(history_modified.history['loss'], label='Training Loss')
plt.plot(history_modified.history['val_loss'], label='Validation 
Loss')
plt.title('Modified Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7bab13287320>
```




```
y_pred_modified_unscaled =
rs_y.inverse_transform(best_model.predict(x_test))
y_test_unscaled = rs_y.inverse_transform(y_test.reshape(-1, 1))

plt.figure(figsize=(8, 5))
sample_size = min(200, len(y_test))
plt.plot(y_test[:sample_size], label='Actual', alpha=0.7)
plt.plot(y_pred_modified_unscaled[:sample_size], label='Predicted',
alpha=0.7)
plt.title('Time Series Comparison (200 Sample) for Modified Model')
plt.xlabel('Time')
plt.ylabel('AT (Degree C)')
plt.legend()

148/148 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step

<matplotlib.legend.Legend at 0x7bab13eb12e0>
```



Conclusion

```
modified_mse = mean_squared_error(y_test_unscaled,
y_pred_modified_unscaled)
modified_mae = mean_absolute_error(y_test_unscaled,
y_pred_modified_unscaled)
modified_r2 = r2_score(y_test_unscaled, y_pred_modified_unscaled)

base_mse = mean_squared_error(y_test_unscaled, y_pred_base_unscaled)
base_mae = mean_absolute_error(y_test_unscaled, y_pred_base_unscaled)
base_r2 = r2_score(y_test_unscaled, y_pred_base_unscaled)

comparison_data = {
    'Model': ['Baseline Model', f'Modified Model (LR={best_lr})'],
    'MSE': [base_mse, modified_mse],
    'MAE': [base_mae, modified_mae],
    'R2 Score': [base_r2, modified_r2]
}

comparison_df = pd.DataFrame(comparison_data)
display(comparison_df)
```


```
{"summary":"{\n  \"name\": \"comparison_df\",\n  \"rows\": 2,\n
\"fields\": [\n    {\n      \"column\": \"Model\",\n
```

```
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n
\"Modified Model (LR=0.001)\",\n          \"Baseline Model\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"MSE\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0.0030169704123174223,\n        \"min\":
0.014046893513132423,\n        \"max\": 0.01831353398751007,\n
\"num_unique_values\": 2,\n        \"samples\": [\n
0.014046893513132423,\n          0.01831353398751007\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"MAE\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.009237740578163314,\n
\"min\": 0.09193407050748409,\n        \"max\": 0.10499820851880692,\n
\"num_unique_values\": 2,\n        \"samples\": [\n
0.09193407050748409,\n          0.10499820851880692\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"R2 Score\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.013985432470542115,\n        \"min\": 0.9151059977805471,\n
\"max\": 0.9348843860560409,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          0.9348843860560409,\n
0.9151059977805471\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"comparison_df"}
```

Berdasarkan hasil evaluasi, dapat diketahui bahwa nilai MSE dari 0.018314 (baseline) ke 0.014047 (modified) dan MAE dari 0.104998 (baseline) ke 0.091934 (modified) menunjukkan bahwa model yang telah dimodifikasi memberikan hasil prediksi yang lebih akurat dan mendekati nilai 0. Selain itu, untuk nilai R2-nya, dapat diketahui bahwa Modified Model memiliki nilai yang lebih tinggi dan mendekati 1, yaitu sebesar 0.934884. Hal ini dapat diaratikan bawa kemampuan Modified Model lebih unggul dalam menjelaskan variabilitas data.

Dapat disimpulkan bahwa Modified Model (dengan best learning rate: 0.001) mengungguli Baseline Model (learning rate juga 0.001) di ketiga metric tersebut dan merupakan model yang lebih baik.

Keberhasilan Modified Model untuk mengungguli Baseline Model ini kemungkinan besar dikarenakan oleh pembuatan arsitektur model yang lebih kompleks, yaitu menambahkan Dropout dan Dense layer.