# LAMPIRAN

Lampiran 1. Souce Code GRNN

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error, r2_score
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsRegressor

# Fungsi untuk membuat dataset menjadi sequence
def create_sequences(data, n_steps_in, n_steps_out):
    X, y = [], []
    for i in range(len(data)):
        end_ix = i + n_steps_in
        out_end_ix = end_ix + n_steps_out
        if out_end_ix > len(data):
            break
        seq_x, seq_y = data[i:end_ix, :-1],
data[end_ix:out_end_ix, -1]
        X.append(seq_x)
        y.append(seq_y)
    return np.array(X), np.array(y)

# Fungsi untuk visualisasi actual vs predicted
def plot_actual_vs_predicted(y_actual, y_pred, dataset_name,
n_steps, data_type):
    plt.figure(figsize=(10, 6))
    plt.plot(y_actual, label='Actual', marker='o')
    plt.plot(y_pred, label='Predicted', marker='x')
    plt.title(f'Actual vs. Predicted Values for {dataset_name}
with {n_steps} Timestep ({data_type} Data)')
    plt.xlabel('Data Point')
    plt.ylabel('Value')
    plt.legend()
    plt.show()

# Fungsi untuk membangun model GRNN
def build_grnn_model():
    return KNeighborsRegressor(n_neighbors=1)

# Contoh data
datasets = {
    'Yen': 'Dataaset Kurs JYP1.csv',
    'Yuan': 'Dataset Kurs CNY.csv',
    'Dollar': 'Dataset Kurs USD1.csv'
}
```

```
# Time step options untuk setiap dataset
time_steps_options = [1, 3, 5, 7, 9, 11]

# Menyimpan hasil
results = []

for name, filepath in datasets.items():
    df = pd.read_csv(filepath)
    df['suku_bunga_acuan'] =
df['suku_bunga_acuan'].str.replace('%', '').astype(float)
    df['inflasi'] = df['inflasi'].str.replace('%',
'').astype(float)
    df['tanggal'] = pd.to_datetime(df['tanggal'])

    # Pisahkan kolom tanggal
    dates = df['tanggal'].values
    data_numeric = df[['suku_bunga_acuan', 'inflasi',
'kurs_beli', 'kurs_jual']].values

    # Lakukan scaling hanya pada data numerik
    scaler = MinMaxScaler()
    data_scaled = scaler.fit_transform(data_numeric)

    for n_steps in time_steps_options:
        X, y = create_sequences(data_scaled, n_steps, 1)

        X_train, X_test, y_train, y_test, dates_train,
dates_test = train_test_split(X, y, dates[n_steps:],
test_size=0.2, random_state=42)

        # Bentuk ulang X_train dan X_test untuk GRNN
        X_train = X_train.reshape(X_train.shape[0], -1)
        X_test = X_test.reshape(X_test.shape[0], -1)

        model = build_grnn_model()
        model.fit(X_train, y_train)

        y_pred_train = model.predict(X_train)
        y_pred_test = model.predict(X_test)

        # Denormalisasi hasil prediksi dan data aktual
        # Tambahkan kolom dummy untuk denormalisasi
        dummy_train = np.zeros((y_train.shape[0],
data_numeric.shape[1] - 1))
        dummy_test = np.zeros((y_test.shape[0],
data_numeric.shape[1] - 1))
```

```
        y_train_combined = np.concatenate((dummy_train,
y_train), axis=1)
        y_pred_train_combined = np.concatenate((dummy_train,
y_pred_train), axis=1)
        y_test_combined = np.concatenate((dummy_test, y_test),
axis=1)
        y_pred_test_combined = np.concatenate((dummy_test,
y_pred_test), axis=1)

        # Denormalisasi data
        y_train_denorm =
scaler.inverse_transform(y_train_combined)[:, -1]
        y_pred_train_denorm =
scaler.inverse_transform(y_pred_train_combined)[:, -1]
        y_test_denorm =
scaler.inverse_transform(y_test_combined)[:, -1]
        y_pred_test_denorm =
scaler.inverse_transform(y_pred_test_combined)[:, -1]

        # Menghitung metrik evaluasi pada skala asli untuk
data training
        mae_train = mean_absolute_error(y_train_denorm,
y_pred_train_denorm)
        r2_train = r2_score(y_train_denorm,
y_pred_train_denorm)
        bias_train = np.mean(y_pred_train_denorm -
y_train_denorm)
        mape_train = np.mean(np.abs((y_train_denorm -
y_pred_train_denorm) / y_train_denorm)) * 100

        # Menyimpan hasil untuk timestep saat ini dengan data
denormalisasi (training)
        results.append({
            'Dataset': name,
            'Timestep': n_steps,
            'Data Type': 'Train',
            'MAE': mae_train,
            'R-square': r2_train,
            'Bias': bias_train,
            'MAPE': mape_train
        })

        # Menghitung metrik evaluasi pada skala asli untuk
data testing
        mae_test = mean_absolute_error(y_test_denorm,
y_pred_test_denorm)
```

```
        r2_test = r2_score(y_test_denorm, y_pred_test_denorm)
        bias_test = np.mean(y_pred_test_denorm -
y_test_denorm)
        mape_test = np.mean(np.abs((y_test_denorm -
y_pred_test_denorm) / y_test_denorm)) * 100

        # Menyimpan hasil untuk timestep saat ini dengan data
denormalisasi (testing)
        results.append({
            'Dataset': name,
            'Timestep': n_steps,
            'Data Type': 'Test',
            'MAE': mae_test,
            'R-square': r2_test,
            'Bias': bias_test,
            'MAPE': mape_test
        })

        # Visualisasi actual vs predicted untuk data train dan
test
        plot_actual_vs_predicted(y_train_denorm,
y_pred_train_denorm, name, n_steps, 'Train')
        plot_actual_vs_predicted(y_test_denorm,
y_pred_test_denorm, name, n_steps, 'Test')

        # Membuat DataFrame untuk data actual dan predicted
test dengan tanggal
        test_results_df = pd.DataFrame({
            'Date': dates_test,
            'Actual': y_test_denorm,
            'Predicted': y_pred_test_denorm
        })

        # Simpan DataFrame ke file CSV
        test_results_df.to_csv(f'{name}_test_results_timestep_
{n_steps}.csv', index=False)

        print(f"Test results for {name} with {n_steps}
timestep saved to {name}_test_results_timestep_{n_steps}.csv")

# Menampilkan hasil
results_df = pd.DataFrame(results)
print(results_df)

# Pilih nilai timestep dengan MAE terendah sebagai nilai
optimal untuk data test
```

```
optimal_timestep = results_df[results_df['Data Type'] ==
'Test'].loc[results_df[results_df['Data Type'] ==
'Test']['MAE'].idxmin()]['Timestep']
print(f"Optimal Timestep: {optimal_timestep}")

# Plot MAE terhadap timestep untuk setiap dataset
def plot_timestep_vs_metric(results_df, metric):
    plt.figure(figsize=(12, 6))
    for dataset in results_df['Dataset'].unique():
        data = results_df[(results_df['Dataset'] == dataset) &
(results_df['Data Type'] == 'Test')]
        plt.plot(data['Timestep'], data[metric], marker='o',
label=dataset)

    plt.title(f'Timestep vs {metric}')
    plt.xlabel('Timestep')
    plt.ylabel(metric)
    plt.legend()
    plt.grid(True)
    plt.show()

# Plot MAE terhadap timestep untuk setiap dataset
plot_timestep_vs_metric(results_df, 'MAE')
```

Lampiran 2 Source Code LSTM

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error, r2_score
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# Fungsi untuk membuat dataset menjadi sequence
def create_sequences(data, n_steps_in, n_steps_out):
    X, y = [], []
    for i in range(len(data)):
        end_ix = i + n_steps_in
        out_end_ix = end_ix + n_steps_out
        if out_end_ix > len(data):
            break
        seq_x, seq_y = data[i:end_ix, :-1],
data[end_ix:out_end_ix, -1]
        X.append(seq_x)
```

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

# Fungsi untuk visualisasi actual vs predicted
def plot_actual_vs_predicted(y_actual, y_pred, dataset_name,
n_steps, data_type):
    plt.figure(figsize=(10, 6))
    plt.plot(y_actual, label='Actual', marker='o')
    plt.plot(y_pred, label='Predicted', marker='x')
    plt.title(f'Actual vs. Predicted Values for {dataset_name}
with {n_steps} Timestep ({data_type} Data)')
    plt.xlabel('Data Point')
    plt.ylabel('Value')
    plt.legend()
    plt.show()

# Fungsi untuk membangun model LSTM
def build_lstm_model(input_shape):
    model = Sequential()
    model.add(LSTM(50, activation='relu',
input_shape=input_shape))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse')
    return model

# Contoh data
datasets = {
    'Yen': 'Dataaset Kurs JYP1.csv',
    'Yuan': 'Dataset Kurs CNY.csv',
    'Dollar': 'Dataset Kurs USD1.csv'
}

# Time step options untuk setiap dataset
time_steps_options = [1, 3, 5, 7, 9, 11]

# Menyimpan hasil
results = []

for name, filepath in datasets.items():
    df = pd.read_csv(filepath)
    df['suku_bunga_acuan'] =
df['suku_bunga_acuan'].str.replace('%', '').astype(float)
    df['inflasi'] = df['inflasi'].str.replace('%',
'').astype(float)
    df['tanggal'] = pd.to_datetime(df['tanggal'])

    # Pisahkan kolom tanggal
```

```
    dates = df['tanggal'].values
    data_numeric = df[['suku_bunga_acuan', 'inflasi',
'kurs_beli', 'kurs_jual']].values

    # Lakukan scaling hanya pada data numerik
    scaler = MinMaxScaler()
    data_scaled = scaler.fit_transform(data_numeric)

    for n_steps in time_steps_options:
        X, y = create_sequences(data_scaled, n_steps, 1)

        X_train, X_test, y_train, y_test, dates_train,
dates_test = train_test_split(X, y, dates[n_steps:],
test_size=0.2, random_state=42)

        model = build_lstm_model((X_train.shape[1],
X_train.shape[2]))
        model.fit(X_train, y_train, epochs=50, batch_size=32,
verbose=0)

        y_pred_train = model.predict(X_train)
        y_pred_test = model.predict(X_test)

        # Denormalisasi hasil prediksi dan data aktual
        # Menyusun kembali data yang akan didenormalisasi agar
memiliki shape yang sesuai
        y_train_combined = np.concatenate((X_train[:, -1, :-
1], y_train), axis=1)
        y_pred_train_combined = np.concatenate((X_train[:, -1,
:-1], y_pred_train), axis=1)
        y_test_combined = np.concatenate((X_test[:, -1, :-1],
y_test), axis=1)
        y_pred_test_combined = np.concatenate((X_test[:, -1,
:-1], y_pred_test), axis=1)

        # Tambahkan kolom dummy
        y_train_combined = np.concatenate([y_train_combined,
np.zeros((y_train_combined.shape[0], 1))], axis=1)
        y_pred_train_combined =
np.concatenate([y_pred_train_combined,
np.zeros((y_pred_train_combined.shape[0], 1))], axis=1)
        y_test_combined = np.concatenate([y_test_combined,
np.zeros((y_test_combined.shape[0], 1))], axis=1)
        y_pred_test_combined =
np.concatenate([y_pred_test_combined,
np.zeros((y_pred_test_combined.shape[0], 1))], axis=1)
```

```
        y_train_denorm =
scaler.inverse_transform(y_train_combined)[:, -2]
        y_pred_train_denorm =
scaler.inverse_transform(y_pred_train_combined)[:, -2]
        y_test_denorm =
scaler.inverse_transform(y_test_combined)[:, -2]
        y_pred_test_denorm =
scaler.inverse_transform(y_pred_test_combined)[:, -2]

        # Menghitung metrik evaluasi pada skala asli untuk
data training
        mae_train = mean_absolute_error(y_train_denorm,
y_pred_train_denorm)
        r2_train = r2_score(y_train_denorm,
y_pred_train_denorm)
        bias_train = np.mean(y_pred_train_denorm -
y_train_denorm)
        mape_train = np.mean(np.abs((y_train_denorm -
y_pred_train_denorm) / y_train_denorm)) * 100

        # Menyimpan hasil untuk timestep saat ini dengan data
denormalisasi (training)
        results.append({
            'Dataset': name,
            'Timestep': n_steps,
            'Data Type': 'Train',
            'MAE': mae_train,
            'R-square': r2_train,
            'Bias': bias_train,
            'MAPE': mape_train
        })

        # Menghitung metrik evaluasi pada skala asli untuk
data testing
        mae_test = mean_absolute_error(y_test_denorm,
y_pred_test_denorm)
        r2_test = r2_score(y_test_denorm, y_pred_test_denorm)
        bias_test = np.mean(y_pred_test_denorm -
y_test_denorm)
        mape_test = np.mean(np.abs((y_test_denorm -
y_pred_test_denorm) / y_test_denorm)) * 100

        # Menyimpan hasil untuk timestep saat ini dengan data
denormalisasi (testing)
        results.append({
            'Dataset': name,
            'Timestep': n_steps,
```

```
            'Data Type': 'Test',
            'MAE': mae_test,
            'R-square': r2_test,
            'Bias': bias_test,
            'MAPE': mape_test
        })

        # Visualisasi actual vs predicted untuk data train dan
test
        plot_actual_vs_predicted(y_train_denorm,
y_pred_train_denorm, name, n_steps, 'Train')
        plot_actual_vs_predicted(y_test_denorm,
y_pred_test_denorm, name, n_steps, 'Test')

        # Membuat DataFrame untuk data actual dan predicted
test dengan tanggal
        test_results_df = pd.DataFrame({
            'Date': dates_test,
            'Actual': y_test_denorm,
            'Predicted': y_pred_test_denorm
        })

        # Simpan DataFrame ke file CSV
        test_results_df.to_csv(f'{name}_test_results_timestep_
{n_steps}.csv', index=False)

        print(f"Test results for {name} with {n_steps}
timestep saved to {name}_test_results_timestep_{n_steps}.csv")

# Menampilkan hasil
results_df = pd.DataFrame(results)
print(results_df)

# Pilih nilai timestep dengan MAE terendah sebagai nilai
optimal untuk data test
optimal_timestep = results_df[results_df['Data Type'] ==
'Test'].loc[results_df[results_df['Data Type'] ==
'Test']['MAE'].idxmin()]['Timestep']
print(f"Optimal Timestep: {optimal_timestep}")

# Plot MAE terhadap timestep untuk setiap dataset
def plot_timestep_vs_metric(results_df, metric):
    plt.figure(figsize=(12, 6))
    for dataset in results_df['Dataset'].unique():
        data = results_df[(results_df['Dataset'] == dataset) &
(results_df['Data Type'] == 'Test')]
```

```
        plt.plot(data['Timestep'], data[metric], marker='o',
label=dataset)

    plt.title(f'Timestep vs {metric}')
    plt.xlabel('Timestep')
    plt.ylabel(metric)
    plt.legend()
    plt.grid(True)
    plt.show()

# Plot MAE terhadap timestep untuk setiap dataset
plot_timestep_vs_metric(results_df, 'MAE')
```