LAMPIRAN

Lampiran 1. Kode Uji Korelasi

```
df_corr = df[['Open', 'High', 'Low', 'Close']].copy()
correlation_matrix = df_corr.corr(method='pearson')
```

Lampiran 2. Kode Normalisasi

```
FEATURES    = ['Open', 'High', 'Low', 'Close']
TARGET_IDX  = 3
SPLIT_RATIO = 0.8
WINDOW_SIZE = 30
VAL_RATIO   = 0.2
LR          = 0.001

data_raw  = df_clean[FEATURES].values
split_idx = int(len(data_raw) * SPLIT_RATIO)

train_raw, test_raw = data_raw[:split_idx], data_raw[split_idx:]

scaler       = MinMaxScaler()
train_scaled = scaler.fit_transform(train_raw)
test_scaled  = scaler.transform(test_raw)

close_min = scaler.data_min_[TARGET_IDX]
close_max = scaler.data_max_[TARGET_IDX]

def inverse_close(arr):
    return arr * (close_max - close_min) + close_min

print(f'Train: {len(train_raw)} | Test: {len(test_raw)}')
```

Lampiran 3. Kode Deteksi dan Penanganan Anomali

```
def detect_anomali_iqr(series, k=1.0):
    Q1, Q3 = series.quantile(0.25), series.quantile(0.75)
    IQR = Q3 - Q1
    return (series < Q1 - k * IQR) | (series > Q3 + k * IQR)

mask     = detect_anomali_iqr(df['Close'], k=1.0)
df_clean = df[~mask].reset_index(drop=True)
print(f'Data bersih : {len(df_clean)} baris | Anomali dihapus:
{mask.sum()}')
df_clean.head()
```

Lampiran 4. Kode *Windowing*

```
def create_sequences(data, window_size, target_idx):
    X, y = [], []
    for i in range(len(data) - window_size):
        X.append(data[i : i + window_size])
        y.append(data[i + window_size][target_idx])
    return np.array(X), np.array(y)

X_train, y_train = create_sequences(train_scaled, WINDOW_SIZE,
TARGET_IDX)
X_test,  y_test  = create_sequences(test_scaled,  WINDOW_SIZE,
TARGET_IDX)

val_split    = int(len(X_train) * (1 - VAL_RATIO))
```

```
X_tr,  X_val = X_train[:val_split], X_train[val_split:]
y_tr,  y_val = y_train[:val_split], y_train[val_split:]

print(f'X_tr   : {X_tr.shape}')
print(f'X_val  : {X_val.shape}')
print(f'X_test : {X_test.shape}')
```

Lampiran 5. Kode Mekanisme *Attention*

```
def attention_block(lstm_output):
    score   = Dense(1,
activation='tanh',  name='att_score')(lstm_output)
    weights = Activation('softmax',        name='att_weights')(score)
    context =
Multiply(                    name='att_multiply')([lstm_output,
weights])
    context = Lambda(lambda x: K.sum(x, axis=1),
name='att_sum')(context)
    return context
```

Lampiran 6. Kode CAE Conv1D

```
def build_cae(window_size, n_features):
    inp     = Input(shape=(window_size, n_features))
    x       = Conv1D(32, 3, activation='relu', padding='same')(inp)
    x       = MaxPooling1D(2, padding='same')(x)
    x       = Conv1D(16, 3, activation='relu', padding='same')(x)
    encoded = MaxPooling1D(2, padding='same', name='encoder_output')(x)
    x       = Conv1D(16, 3, activation='relu', padding='same')(encoded)
    x       = UpSampling1D(2)(x)
    x       = Conv1D(32, 3, activation='relu', padding='same')(x)
    x       = UpSampling1D(2)(x)
    x       = Lambda(lambda t: t[:, :window_size, :])(x)
    decoded = Conv1D(n_features, 3, activation='sigmoid',
padding='same')(x)
    return Model(inp, decoded, name='autoencoder'), Model(inp, encoded,
name='encoder')
```

Lampiran 7. Kode CAE-LSTM-*Attention*

```
def build_cae_lstm_attention(encoder, lstm_units=64, dense_units=32):
    enc_out_shape = encoder.output_shape[1:]
    inp     = Input(shape=enc_out_shape, name='m1_input')
    lstm    = LSTM(lstm_units, return_sequences=True,
name='m1_lstm')(inp)
    context = attention_block(lstm)
    x       = Dense(dense_units, activation='relu',
name='m1_dense')(context)
    out     = Dense(1, name='m1_output')(x)
    return Model(inputs=inp, outputs=out, name='CAE_LSTM_Attention')
```

Lampiran 8. Kode LSTM-*Attention*

```
def build_lstm_attention(window_size, n_features, lstm_units=64,
dense_units=32):
    inp     = Input(shape=(window_size, n_features), name='m2_input')
    lstm    = LSTM(lstm_units, return_sequences=True,
name='m2_lstm')(inp)
    context = attention_block(lstm)
```

```
    x       = Dense(dense_units, activation='relu',
name='m2_dense')(context)
    out     = Dense(1, name='m2_output')(x)
    return Model(inputs=inp, outputs=out, name='LSTM_Attention')
```

Lampiran 9. Kode LSTM

```
def build_lstm(window_size, n_features, lstm_units=64, dense_units=32):
    inp  = Input(shape=(window_size, n_features), name='m3_input')
    lstm = LSTM(lstm_units, return_sequences=False,
name='m3_lstm')(inp)
    x    = Dense(dense_units, activation='relu', name='m3_dense')(lstm)
    out  = Dense(1, name='m3_output')(x)
    return Model(inputs=inp, outputs=out, name='LSTM_Only')
```

Lampiran 10. Kode *Training* CAE

```
autoencoder, encoder = build_cae(WINDOW_SIZE, len(FEATURES))
autoencoder.compile(optimizer='adam', loss='mse')

history_cae = autoencoder.fit(
    X_tr, X_tr,
    epochs=100, batch_size=32,
    validation_data=(X_val, X_val),
    callbacks=[EarlyStopping(monitor='val_loss', patience=15,
                             restore_best_weights=True, verbose=1)],
    verbose=1
)

best_cae = np.argmin(history_cae.history['val_loss']) + 1
print(f'CAE selesai. Best epoch: {best_cae}')
```

Lampiran 11. Kode Training dan Evaluasi

```
def get_optimizer(name, lr=0.001):
    if name == 'Adam':
        return tf.keras.optimizers.Adam(learning_rate=lr, clipnorm=1.0)
    elif name == 'AdamW':
        return tf.keras.optimizers.AdamW(learning_rate=lr,
weight_decay=1e-4, clipnorm=1.0)
    elif name == 'Nadam':
        return tf.keras.optimizers.Nadam(learning_rate=lr,
clipnorm=1.0)

def get_callbacks():
    return [
        EarlyStopping(monitor='val_loss', patience=10,
                      restore_best_weights=True, min_delta=1e-5,
verbose=1),
        ReduceLROnPlateau(monitor='val_loss', factor=0.5,
                          patience=5, min_lr=1e-6, verbose=1)
    ]

def train_and_evaluate(model_name, optimizer_name, model,
                       X_tr_, y_tr_, X_val_, y_val_,
                       X_test_, y_test_,
                       epochs=200, batch_size=32):
    print(f'\n{"-"*55}')
    print(f'  {model_name} | {optimizer_name}')
    print(f'{"-"*55}')
```

```
    model.compile(
        optimizer=get_optimizer(optimizer_name),
        loss='mse', metrics=['mae']
    )

    history = model.fit(
        X_tr_, y_tr_,
        epochs=epochs, batch_size=batch_size,
        validation_data=(X_val_, y_val_),
        callbacks=get_callbacks(),
        shuffle=False, verbose=1
    )

    y_pred_s = model.predict(X_test_, verbose=0).flatten()
    y_actual = inverse_close(y_test_)
    y_pred   = inverse_close(y_pred_s)

    mae  = mean_absolute_error(y_actual, y_pred)
    rmse = np.sqrt(mean_squared_error(y_actual, y_pred))
    mape = np.mean(np.abs((y_actual - y_pred) / y_actual)) * 100
    da   = np.mean((np.diff(y_actual) > 0) == (np.diff(y_pred) > 0)) *
100
    best = np.argmin(history.history['val_loss']) + 1

    print(f'  MAE={mae:.4f} | RMSE={rmse:.4f} | MAPE={mape:.4f}% |
DA={da:.4f}% | Best epoch={best}')

    return {
        'model_name'    : model_name,
        'optimizer_name': optimizer_name,
        'history'       : history,
        'y_actual'      : y_actual,
        'y_pred'        : y_pred,
        'MAE'           : mae,
        'RMSE'          : rmse,
        'MAPE'          : mape,
        'DA'            : da,
        'best_epoch'    : best,
        'total_epoch'   : len(history.history['loss'])
    }
```

Lampiran 12. Kode *Training Optimizer* dan Model

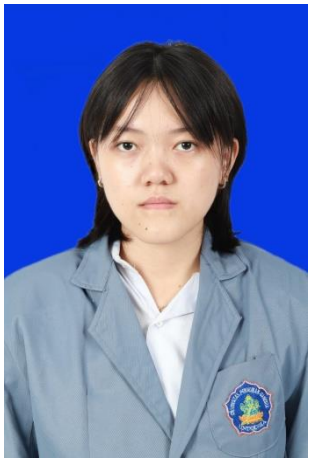
```
OPTIMIZERS = ['Adam', 'AdamW', 'Nadam']
results    = {}   # key: (model_name, optimizer_name)
```

```
    m1  = build_cae_lstm_attention(encoder)
    key = ('CAE-LSTM-Attention', opt)
    results[key] = train_and_evaluate(
        model_name     = 'CAE-LSTM-Attention',
        optimizer_name = opt,
        model          = m1,
        X_tr_          = X_tr_enc,
        y_tr_          = y_tr,
        X_val_         = X_val_enc,
        y_val_         = y_val,
        X_test_        = X_test_enc,
        y_test_        = y_test
    )
```

```
    m2  = build_lstm_attention(WINDOW_SIZE, len(FEATURES))
    key = ('LSTM-Attention', opt)
    results[key] = train_and_evaluate(
        model_name     = 'LSTM-Attention',
        optimizer_name = opt,
        model          = m2,
        X_tr_          = X_tr,
        y_tr_          = y_tr,
        X_val_         = X_val,
        y_val_         = y_val,
        X_test_        = X_test,
        y_test_        = y_test
    )
```

```
    m3  = build_lstm(WINDOW_SIZE, len(FEATURES))
    key = ('LSTM', opt)
    results[key] = train_and_evaluate(
        model_name     = 'LSTM',
        optimizer_name = opt,
        model          = m3,
        X_tr_          = X_tr,
        y_tr_          = y_tr,
        X_val_         = X_val,
        y_val_         = y_val,
        X_test_        = X_test,
        y_test_        = y_test
    )
```

RIWAYAT HIDUP



Penulis bernama Valensia Natalia, lahir di Surabaya pada tanggal 24 Desember 2002. Penulis merupakan warga negara Indonesia dan berdomisili di Surabaya, Jawa Timur. Penulis menyelesaikan pendidikan dasar di SD Santa Lorent Surabaya dan lulus pada tahun 2015, kemudian melanjutkan pendidikan di SMP Katolik Santa Agnes dan lulus pada tahun 2018, serta menyelesaikan pendidikan menengah di SMA Katolik Santa Agnes Surabaya pada tahun 2021. Pada tahun 2022, penulis melanjutkan pendidikan di Program Studi Ilmu Komputer, Fakultas Teknik dan Kejuruan, Universitas Pendidikan Ganesha. Sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer (S.Kom.), penulis menyusun skripsi yang berjudul "Implementasi dan Evaluasi Model CAE-LSTM dengan *Attention Mechanism* untuk Prediksi Harga Instrumen Keuangan Berbasis Data Historis".