DAFTAR PUSTAKA

Abumohsen, M., Owda, A. Y., & Owda, M. (2023). Electrical load forecasting using LSTM, GRU, and RNN algorithms. *Energies*, *16*(5), 2283.

Brownlee, J. (2018). A Gentle Introduction to Dropout for Regularizing Deep Neural Networks. https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/

Brownlee, J. (2022). Difference Between a Batch and an Epoch in a Neural Network. https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Dimasti, D. (2022). Analisis Dampak Konflik Rusia-Ukraina Terhadap Harga Bahan Bakar Minyak Indonesia. *Cendekia (Jurnal Ilmu Pengetahuan),* 2(3), 261-269.

Fadillah, F., Wibowo, S. A., Biduman, F. T. Elektro., dan Telkom, U. (2020). Perancangan Dan Implementasi Prediksi Harga Saham Pada Aplikasi Berbasis Android Menggunakan Metode Support Vector Regression. *E-Peoceeding of Engineering*, 7(2), 3869-3876.

Fajri, M., Syafriandi, S., Vionanda, D., & Zilrahmi, Z. (2023). Prediksi Harga Minyak Mentah Dunia Menggunakan Metode Ensemble k-Nearest Neighbor. *Jurnal Pendidikan Tambusai, 7*(2), 17357-17368.

Fauzannissa, R. A., Yasin, H., dan Ispriyanti, D. (2015). Peramalan Harga Minyak Mentah Dunia Menggunkan Metode Radial Basis Function Neural Network. *Jurnal Gaussian*, 5(1), 193-202.

Ginting, L. M., Sigiro, M. MT., Manurung, E. D., dan Sinurat, J. J. P. (2021). Perbandingan Metode Algoritma Support Vector Regression dan Multiple Linear Regression Untuk Memprediksi Stok Obat. *Journal of Applied Techonology and Informatics*, 1(2), 2807-3347.

Hastomo, W., Karno, A.S., Kalbuana, N., Nisfiani, E., dan Lussiana. (2021). Optimasi Deep Learning untuk Prediksi Saham di Masa Pandemi Covid-19. *Jurnal Edukasi dan Penelitian Informatika*. 7(2), 133-140

Herjanto, E. (2019). *Sains Manajemen Analisis Kuantitatif untuk Pengambilan Keputusan*. Jakarta: PT Gramedia Widiarsana Indonesia.

Jange, B. (2022). Prediksi Indeks Harga Saham Gabungan (IHSG) Menggunakan Prophet. *Journal In Management And Entrepreneurship*, 1(2), 53-59.

Julpan, N. E. B. dan Zarlis, M. (2015). Bipolar Dalam Algoritma Backpropagation Pada. *Jurnal Teknovasi*, 2, 103-116.

Klein, B. (2022). Train and Test Sets by Splitting Learn an Test Data. https://pyhton-course.eu/machine-learning/train-and-test-sets-by-splitting-learn-and-test-data.php

Latif, F., Tambunan, N., dan R. D. Heryani, R. D. (2023). Kenaikan Harga Minyak Dunia dan Implikasinya Terhadap Perekonomian Indonesia di Masa Pandemi Covid-19. *Sinomika Journal*, 1(5), 1121–1126.

Maricar, M. A. (2019). Analisa Perbandingan Nilai Akurasi Moving Average dan Exponential Smoothing untuk Sistem Peramalan Pendapatan pada Perusahaan XYZ. *J. Sist. dan Inform*, 13(2), 36–45.

Miyanti, G. A. D. A., & Wiagustini, L. P. (2018). Pengaruh suku bunga The Fed, harga minyak, dan inflasi terhadap Indeks Harga Saham Gabungan (IHSG) di Bursa Efek Indonesia. *E-Jurnal Ekonomi Dan Bisnis Universitas Udayana*, *7*(5), 1261-1288.

Nabillah, I., dan Ranggadara, I. (2020). Mean Abdolute Percentage Error untuk Evaluasi Hasil Prediksi Komoditas Laut. *Journal of Information System*, 5(2), 250-255.

Nurlela, S., Fanani, A., dan Khaulasari, H. (2023). Prediksi Harga Minyak Mentah WTI Menggunakan Metode Fuzzy Time Series Markov Chain. *Jurnal Fourier*, 12(1), 10-19.

Oktaviani, A., dan Hustinawati, H. (2021). Prediksi rata-rata zat berbahaya di dki jakarta berdasarkan indeks standar pencemar udara menggunakan metode long short-term memory. *Jurnal Ilmiah Informatika Komputer*, *26*(1), 41-55.

Oud, M. A. A. (2014). *The dynamics of oil price and valuation of oil derivatives. School of Mathematics and Applied Statistics*, University of Wollongong, Australia.

Primartha & Rifkie, (2018). Belajar Machine Learning Teori dan Praktik. Bandung. Informatika Bandung.

Pu, H., Wu, Y., Hua, C., Song, Z., Zhang, W., Zhang, L., ... & Qiu, Y. (2022, August). Carbon Trading Based on Quadratic Modal Decomposition and Recurrent Neural Network Price Prediction Model. In *2022 5th International Conference on Pattern Recognition and Artificial Intelligence (PRAI)*, 26-35.

Putro, E. A. N., Rimawati, E., dan Vulandari, R. T. (2021). Prediksi Penjualan Kertas Menggunakan Metode Double Exponential Smoothing. *Jurnal Teknologi Informasi Dan Komunikasi (TIKomSiN),* 9(1), 60.

Rahman, M. (2008). Perilaku Harga Minyak Dunia. *Lembaran Publikasi Lemigas*, 42(3), 1-10.

Robial, A. M. (2018). Perbandingan Model Statistik Pada Analisis Metode Peramalan Time Series (Studi Kasus: Pt. Telekomunikasi Indonesia, Tbk Kandatel Sukabumi). *Jurnal Ilmiah SANTIKA*, 8(2), 1-17.

Singla, S. (2020). Introduction to Dropout to regularize Deep Neural Network. https://www.linkedin.com/pulse/introduction-dropout-regularize-deep-neural-network-saurav-singla

Tanudy, C., Handyani, T., dan Hendryli, J. (2023). Prediksi Harga Emas di Indonesia Menggunakan Gated Recurrent Unit. *Jurnal Fasilkom,* 13(3). 480-488.

Wardana, R. P. (2020). Penenrapan Model Gated Recurrent Unit Untuk Peramalan Jumlah Penumpang Kereta Api Di PT.KAI (Persero).

Zaccheaus, O. D. dan Ajuwon, O. S. (2019). Oil Price Dynamics and The Nigerian Banks Proitability. *J. Deveping Areas*, 53(2).

LAMPIRAN

Lampiran 1: Dataset Haga Minyak Mentah WTI

Berikut ini adalah dataset harga minyak mentah WTI tahun 2018. Untuk keseluruhan dataset bisa dilihat pada link di bawah ini.

https://finance.yahoo.com/quote/CL=F/

Lampiran 2: *Sorce Code* Model *Gated Recurrent Unit*

#Import Library

```
%tensorflow_version 2.8
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math
import random
import keras
import pandas as pd
import tensorflow as tf
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.utils import plot_model
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, GRU, Dropout
from sklearn.model_selection import train_test_split
from tabulate import tabulate
from sklearn.metrics import mean_squared_error,
mean_absolute_error, mean_absolute_percentage_error
```

```
#!pip install tensorflow==2.8
```

```
#!pip install keras==2.8
```

```
print(tf.__version__)
```

```
#Inisialisasi weight agar tiap run hasilnya tidak random
tf.random.set_seed(20)
```

#Preprocessing

```
# Reading the csv file
dataset = pd.read_excel("/content/1.xlsx")
```

```
dataset.head()
```

```
dataset.shape
```

```
dataset.dtypes
```

```
dataset['Date'] = dataset['Date'].astype('datetime64[ns]')
```

```
dataset.dtypes
```

```
dataset.isnull().sum()
```

```
dataset = pd.read_excel('/content/1.xlsx', index_col='Date',
parse_dates=['Date'])
dataset.head()
```

```
df = dataset.drop(['Adj Close', 'Volume'], axis=1)
df.head()
```

```
df.describe()
```

 # Visualisasi Data

```
# Split data into train (80%) and test (20%)
def dataset(dataframe):
    train_size = int(len(dataframe)*0.8)
    train_dataset, test_dataset = dataframe.iloc[:train_size],
dataframe.iloc[train_size:]
    return train_dataset, test_dataset

train_dataset, test_dataset = dataset(df)
```

 # Plot *Train and Test* Data *CLose*

```
# Plot train and test data
plt.figure(figsize = (20, 10))
plt.rcParams['figure.dpi'] = 200

plt.plot(train_dataset.Close, linestyle="-")
plt.plot(test_dataset.Close, linestyle="-")
plt.xlabel('Date', fontsize=18)
plt.ylabel('Close value ', fontsize=18)
plt.title('Plot Harga Close', fontsize=18)
```

```
plt.legend(['Train set', 'Test set'], loc='upper left',
prop={'size': 16})
print('Dimension of train data: ', train_dataset.shape)
print('Dimension of test data: ', test_dataset.shape)
```

 # Plot *Train and Test* Data *Open*

```
# Plot train and test data
plt.figure(figsize = (20, 10))
plt.rcParams['figure.dpi'] = 240

plt.plot(train_dataset.Open, linestyle="-")
plt.plot(test_dataset.Open, linestyle="-")
plt.xlabel('Date', fontsize=18)
plt.ylabel('Open value (USD $)', fontsize=18)
plt.title('Plot Harga Open', fontsize=18)
plt.legend(['Train set', 'Test set'], loc='upper left',
prop={'size': 16})
print('Dimension of train data: ', train_dataset.shape)
print('Dimension of test data: ', test_dataset.shape)
```

 # Plot *Train and Test* Data *High*

```
# Plot train and test data
plt.figure(figsize = (20, 10))
plt.rcParams['figure.dpi'] = 240

plt.plot(train_dataset.High, linestyle="-")
plt.plot(test_dataset.High, linestyle="-")
plt.xlabel('Date', fontsize=18)
plt.ylabel('High value (USD $)', fontsize=18)
plt.title('Plot Harga High', fontsize=18)
plt.legend(['Train set', 'Test set'], loc='upper left',
prop={'size': 16})
print('Dimension of train data: ', train_dataset.shape)
print('Dimension of test data: ', test_dataset.shape)
```

 # Plot *Train and Test* Data *Low*

```
# Plot train and test data
plt.figure(figsize = (20, 10))
plt.rcParams['figure.dpi'] = 240

plt.plot(train_dataset.Low, linestyle="-")
plt.plot(test_dataset.Low, linestyle="-")
plt.xlabel('Date', fontsize=18)
plt.ylabel('Low value', fontsize=18)
```

```
plt.title('Plot Harga Low', fontsize=18)
plt.legend(['Train set', 'Test set'], loc='upper left',
prop={'size': 16})
print('Dimension of train data: ', train_dataset.shape)
print('Dimension of test data: ', test_dataset.shape)
```

 # Normalisasi Data

```
#Normalisasi
def scaled(trainds, testds):
    scaler = MinMaxScaler(feature_range=(0,1))

    # Identify numeric columns
    numeric_columns =
trainds.select_dtypes(include=['float64', 'int64']).columns

    # Fit and transform the scaler on numeric columns
    trainds[numeric_columns] =
scaler.fit_transform(trainds[numeric_columns])
    testds[numeric_columns] =
scaler.transform(testds[numeric_columns])

    return trainds, testds

train_scaled, test_scaled = scaled(train_dataset,
test_dataset)
```

 # Segmentasi Data

```
# input dataset
def create_dataset(X, look_back=1):
    Xs, ys = [], []

    for i in range(len(X) - look_back):
        v = X.iloc[i:i + look_back, :]
        Xs.append(v.values)  # Convert the sliced DataFrame to
a NumPy array
        ys.append(X.iloc[i + look_back].values)

    return np.array(Xs), np.array(ys)

LOOK_BACK = 60

X_train, y_train = create_dataset(train_scaled, LOOK_BACK)
X_test, y_test = create_dataset(test_scaled, LOOK_BACK)

# Print data shape
```

```
print('X_train.shape: ', X_train.shape)
print('y_train.shape: ', y_train.shape)
print('X_test.shape: ', X_test.shape)
print('y_test.shape: ', y_test.shape)
```

 # Memunculkan Hasil Normalisasi Data

```
print(train_scaled)
```

```
print(df)
```

```
train_df = pd.DataFrame(train_scaled, columns = ['Close',
'Open', 'High', 'Low'])
train_df.head(10)
```

```
print(test_scaled)
```

```
print(df)
```

```
test_df = pd.DataFrame(test_scaled, columns = ['Close',
'Open', 'High', 'Low'])
test_df.head(10)
```

 # Mendefinisikan Model

```
# Fit Model
def fit_model(model, X, y):
    early_stop = keras.callbacks.EarlyStopping(monitor =
'val_loss', patience = 10)
    history = model.fit(X, y, epochs = 10, validation_split =
0.2, batch_size = 32,
                        shuffle = False, callbacks =
[early_stop])
    return history
```

 # Plot model Prediksi, Plot Data, dan Plot *Loss*

```
# prediction Model
def prediction(model_predict, x_test):
    prediction = model_predict.predict(x_test)
    return prediction

# Plot test data vs prediction Model
def plot_future(prediction, model_name, y_test):
    plt.figure(figsize=(20, 6))
```

```
    range_future = len(prediction)
    plt.plot(np.arange(range_future), np.array(y_test),
             label='Test data')
    plt.plot(np.arange(range_future), np.array(prediction),
             label='Prediction')
    plt.title('Test data vs prediction for ' + model_name,
fontsize=18)
    plt.legend(loc='upper left', prop={'size': 16})
    plt.xlabel('Time (day)', fontsize=18)
    plt.ylabel('Price (USD $)', fontsize=18)

# Plot Loss data
def plot_loss(history, model_name):
    plt.figure(figsize = (28, 12))
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model Train vs Validation Loss for ' +
model_name, fontsize=18)
    plt.ylabel('Loss', fontsize=18)
    plt.xlabel('epoch', fontsize=18)
    plt.legend(['Train loss', 'Validation loss'], loc='upper
right', prop={'size': 16})
```

 # Mendefinikan Metode Evaluasi

```
# Pembuatan Method Root Mean Square Error (RMSE) berdasarkan
library sklearn
import numbers
import warnings

import numpy as np
from scipy.special import xlogy

from sklearn.exceptions import UndefinedMetricWarning
from sklearn.utils.validation import (
    check_array,
    check_consistent_length,
    check_scalar,
    _num_samples,
    column_or_1d,
    _check_sample_weight,
)
from sklearn.utils.stats import _weighted_percentile

def _check_reg_targets(y_true, y_pred, multioutput,
dtype="numeric"):
```

```
    """Check that y_true and y_pred belong to the same
regression task.
    Parameters
    ----------
    y_true : array-like
    y_pred : array-like
    multioutput : array-like or string in ['raw_values',
uniform_average',
        'variance_weighted'] or None
        None is accepted due to backward compatibility of
r2_score().
    dtype : str or list, default="numeric"
        the dtype argument passed to check_array.
    Returns
    -------
    type_true : one of {'continuous', continuous-multioutput'}
        The type of the true target data, as output by
        'utils.multiclass.type_of_target'.
    y_true : array-like of shape (n_samples, n_outputs)
        Ground truth (correct) target values.
    y_pred : array-like of shape (n_samples, n_outputs)
        Estimated target values.
    multioutput : array-like of shape (n_outputs) or string in
['raw_values',
        uniform_average', 'variance_weighted'] or None
        Custom output weights if ``multioutput`` is array-like
or
        just the corresponding argument if ``multioutput`` is
a
        correct keyword.
    """
    check_consistent_length(y_true, y_pred)
    y_true = check_array(y_true, ensure_2d=False, dtype=dtype)
    y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)

    if y_true.ndim == 1:
        y_true = y_true.reshape((-1, 1))

    if y_pred.ndim == 1:
        y_pred = y_pred.reshape((-1, 1))

    if y_true.shape[1] != y_pred.shape[1]:
        raise ValueError(
            "y_true and y_pred have different number of output
({0}!={1})".format(
                y_true.shape[1], y_pred.shape[1]
            )
```

```
        )

    n_outputs = y_true.shape[1]
    allowed_multioutput_str = ("raw_values",
"uniform_average", "variance_weighted")
    if isinstance(multioutput, str):
        if multioutput not in allowed_multioutput_str:
            raise ValueError(
                "Allowed 'multioutput' string values are {}. "
                "You provided multioutput={!r}".format(
                    allowed_multioutput_str, multioutput
                )
            )
    elif multioutput is not None:
        multioutput = check_array(multioutput,
ensure_2d=False)
        if n_outputs == 1:
            raise ValueError("Custom weights are useful only
in multi-output cases.")
        elif n_outputs != len(multioutput):
            raise ValueError(
                "There must be equally many custom weights
(%d) as outputs (%d)."
                % (len(multioutput), n_outputs)
            )
    y_type = "continuous" if n_outputs == 1 else "continuous-
multioutput"

    return y_type, y_true, y_pred, multioutput

def root_mean_squared_percentage_error(y_true, y_pred, *,
sample_weight=None, multioutput="uniform_average",
squared=True):
    """Root Mean squared percentage error regression loss.
    Read more in the :ref:`User Guide <mean_squared_error>`.
    Parameters
    ----------
    y_true : array-like of shape (n_samples,) or (n_samples,
n_outputs)
        Ground truth (correct) target values.
    y_pred : array-like of shape (n_samples,) or (n_samples,
n_outputs)
        Estimated target values.
    sample_weight : array-like of shape (n_samples,),
default=None
        Sample weights.
```

```
    multioutput : {'raw_values', 'uniform_average'} or array-
like of shape \
            (n_outputs,), default='uniform_average'
        Defines aggregating of multiple output values.
        Array-like value defines weights used to average
errors.
        'raw_values' :
            Returns a full set of errors in case of
multioutput input.
        'uniform_average' :
            Errors of all outputs are averaged with uniform
weight.
    squared : bool, default=True
        If True returns MSE value, if False returns RMSE
value.
    Returns
    -------
    loss : float or ndarray of floats
        A non-negative floating point value (the best value is
0.0), or an
        array of floating point values, one for each
individual target.
    """
    y_type, y_true, y_pred, multioutput =
_check_reg_targets(y_true, y_pred, multioutput)
    check_consistent_length(y_true, y_pred, sample_weight)
    output_errors = np.average(((y_true - y_pred) / y_true) **
2, axis=0, weights=sample_weight)

    if not squared:
        output_errors = np.sqrt(output_errors)

    if isinstance(multioutput, str):
        if multioutput == "raw_values":
            return output_errors
        elif multioutput == "uniform_average":
            # pass None as weights to np.average: uniform mean
            multioutput = None

    return np.average(output_errors, weights=multioutput)
```

 # Memanggil *Dataframe* Setiap *Variable*

```
#Dataframe untuk setiap variable
df_Close = df.drop(['Open', 'High', 'Low'], axis=1)
df_Open = df.drop(['Close', 'High', 'Low'], axis=1)
df_High = df.drop(['Close', 'Open', 'Low'], axis=1)
```

```
df_Low = df.drop(['Close', 'Open', 'High'], axis=1)

df_Close.head()
df_Close.tail()
df_Open.head()
df_High.head()
df_Low.head()
```

 # Menampilkan Pembagian Data Segmentasi Setiap *Variable*

```
train_dataset_Close, test_dataset_Close = dataset(df_Close)
train_dataset_Open, test_dataset_Open = dataset(df_Open)
train_dataset_High, test_dataset_High = dataset(df_High)
train_dataset_Low, test_dataset_Low = dataset(df_Low)
```

```
train_scaled_Close, test_scaled_Close =
scaled(train_dataset_Close, test_dataset_Close)
train_scaled_Open, test_scaled_Open =
scaled(train_dataset_Open, test_dataset_Open)
train_scaled_High, test_scaled_High =
scaled(train_dataset_High, test_dataset_High)
train_scaled_Low, test_scaled_Low = scaled(train_dataset_Low,
test_dataset_Low)
```

```
X_train_Close, y_train_Close =
create_dataset(train_scaled_Close, LOOK_BACK)
X_test_Close, y_test_Close = create_dataset(test_scaled_Close,
LOOK_BACK)

print('X_train_Close.shape: ', X_train_Close.shape)
print('y_train_Close.shape: ', y_train_Close.shape)
print('X_test_Close.shape: ', X_test_Close.shape)
print('y_test_Close.shape: ', y_test_Close.shape)
```

```
X_train_Open, y_train_Open = create_dataset(train_scaled_Open,
LOOK_BACK)
X_test_Open, y_test_Open = create_dataset(test_scaled_Open,
LOOK_BACK)

print('X_train_Open.shape: ', X_train_Open.shape)
print('y_train_Open.shape: ', y_train_Open.shape)
print('X_test_Open.shape: ', X_test_Open.shape)
print('y_test_Open.shape: ', y_test_Open.shape)
```

```
X_train_High, y_train_High = create_dataset(train_scaled_High,
LOOK_BACK)
X_test_High, y_test_High = create_dataset(test_scaled_High,
LOOK_BACK)

print('X_train_High.shape: ', X_train_High.shape)
print('y_train_High.shape: ', y_train_High.shape)
print('X_test_High.shape: ', X_test_High.shape)
print('y_test_High.shape: ', y_test_High.shape)
```

```
X_train_Low, y_train_Low = create_dataset(train_scaled_Low,
LOOK_BACK)
X_test_Low, y_test_Low = create_dataset(test_scaled_Low,
LOOK_BACK)

print('X_train_Low.shape: ', X_train_Low.shape)
print('y_train_Low.shape: ', y_train_Low.shape)
print('X_test_Low.shape: ', X_test_Low.shape)
print('y_test_Low.shape: ', y_test_Low.shape)
```

 # Model GRU

```
# GRU Model
def create_gru(units, x_train):
    model = Sequential()
    # Input layer
    model.add(GRU(units = units, return_sequences = True,
                  input_shape = ([x_train.shape[1],
x_train.shape[2]])))
    model.add(Dropout(0.3))
    # Hidden layer
    model.add(GRU (units = units*2))
    model.add(Dropout(0.3))
    # Output Layer
    model.add(Dense(units = 1))
    #Compile model
    model.compile(optimizer='adam',loss='mse')
    return model
```

 # GRU *Close*

```
# GRU CLose
model_gru_Close = create_gru(32, X_train_Close)
```

```
print(model_gru_Close.summary())
```

 # Menampilkan Arsitektur Model

```
plot_model(model_gru_Close, to_file='model_gru_Close.png',
show_shapes=True, show_layer_names=True)
```

 # Menampilkan Hasil *Epoch*

```
history_gru_Close = fit_model(model_gru_Close, X_train_Close,
y_train_Close)
```

```
evaluation_GRU_Close = model_gru_Close.evaluate(X_test_Close,
y_test_Close)
print(evaluation_GRU_Close)
```

 # Menampilkan Hasil Plot *Loss*

```
plot_loss(history_gru_Close, 'GRU Close')
```

 # Menampilkan Hasil Peramalan

```
prediction_gru_Close = prediction(model_gru_Close,
X_test_Close)

plot_future(prediction_gru_Close, 'GRU Close', y_test_Close)
```

 # Mendenormalisasikan Hasil Peramalan

```
scaler = MinMaxScaler(feature_range=(0,1)).fit(df_Close)
pred_inv_gru_Close =
scaler.inverse_transform(prediction_gru_Close)
print(pred_inv_gru_Close)
```

```
# Inverse transform y_test_Close dan prediksi
y_test_Close_inv = scaler.inverse_transform(y_test_Close)
pred_inv_gru_Close =
scaler.inverse_transform(prediction_gru_Close)
```

```
plot_future(pred_inv_gru_Close, 'Close', y_test_Close_inv)
```

 # GRU *Open*

```
# GRU Open
model_gru_Open = create_gru(32, X_train_Open)
```

```
print(model_gru_Open.summary())
```

 # Menampilkan Arsitektur Model

```
plot_model(model_gru_Open, to_file='model_gru_Open.png',
show_shapes=True, show_layer_names=True)
```

 # Menampilkan Hasil *Epoch*

```
history_gru_Open = fit_model(model_gru_Open, X_train_Open,
y_train_Open)
```

```
evaluation_GRU_Open = model_gru_Open.evaluate(X_test_Open,
y_test_Open)
print(evaluation_GRU_Open)
```

 # Menampilkan Hasil Plot *Loss*

```
plot_loss(history_gru_Open, 'GRU Open')
```

 # Menampilkan Hasil Peramalan

```
prediction_gru_Open = prediction(model_gru_Open, X_test_Open)
plot_future(prediction_gru_Open, 'GRU Open', y_test_Open)
```

 # Mendenormalisasikan Hasil Peramalan

```
scaler = MinMaxScaler(feature_range=(0,1)).fit(df_Open)
pred_inv_gru_Open =
scaler.inverse_transform(prediction_gru_Open)
```

```
# Menggunakan MinMaxScaler
y_test_Open_inv =
scaler.inverse_transform(np.array([y_test_Open]).reshape(-1,
1)).flatten()
```

```
plot_future(pred_inv_gru_Open, 'GRU Open', y_test_Open_inv)
```

 # GRU *High*

```
# GRU High
model_gru_High = create_gru(32, X_train_High)
```

```
print(model_gru_High.summary())
```

 # Menampilkan Arsitektur Model

```
plot_model(model_gru_High, to_file='model_gru_High.png',
show_shapes=True, show_layer_names=True)
```

 # Menampilkan Hasil *Epoch*

```
history_gru_High = fit_model(model_gru_High, X_train_High,
y_train_High)
```

```
evaluation_GRU_High = model_gru_High.evaluate(X_test_High,
y_test_High)
print(evaluation_GRU_High)
```

 # Menampilkan Hasil Plot *Loss*

```
plot_loss(history_gru_High, 'GRU High')
```

 # Menampilkan Hasil Peramalan

```
prediction_gru_High = prediction(model_gru_High, X_test_High)
plot_future(prediction_gru_High, 'GRU High', y_test_High)
```

 # Mendenormalisasikan Hasil Peramalan

```
scaler = MinMaxScaler(feature_range=(0,1)).fit(df_High)
pred_inv_gru_High =
scaler.inverse_transform(prediction_gru_High)
```

```
y_test_High_inv =
scaler.inverse_transform(y_test_High.reshape(-1, 1)).flatten()
```

```
plot_future(pred_inv_gru_High, 'GRU High', y_test_High_inv)
```

 # GRU *Low*

```
# GRU Low
model_gru_Low = create_gru(32, X_train_Low)
```

```
print(model_gru_Low.summary())
```

 # Menampilkan Arsitektur Model

```
plot_model(model_gru_Low, to_file='model_gru_Low.png',
show_shapes=True, show_layer_names=True)
```

 # Menampilkan Hasil *Epoch*

```
history_gru_Low = fit_model(model_gru_Low, X_train_Low,
y_train_Low)
```

```
evaluation_GRU_Low = model_gru_Low.evaluate(X_test_Low,
y_test_Low)
print(evaluation_GRU_Low)
```

 # Menampilkan Hasil Plot *Loss*

```
plot_loss(history_gru_Low, 'GRU Low')
```

 # Menampilkan Hasil Peramalan

```
prediction_gru_Low = prediction(model_gru_Low, X_test_Low)
plot_future(prediction_gru_Low, 'GRU Low', y_test_Low)
```

 # Mendenormalisasikan Hasil Peramalan

```
scaler = MinMaxScaler(feature_range=(0,1)).fit(df_Low)
pred_inv_gru_Low =
scaler.inverse_transform(prediction_gru_Low)
```

```
y_test_Low_inv = scaler.inverse_transform(y_test_Low.reshape(-
1, 1)).flatten()
```

```
plot_future(pred_inv_gru_Low, 'GRU Low', y_test_Low_inv)
```

 # Ukuran Kinerja GRU Setiap *Variable*

```
rmse = mean_squared_error(y_test_Close_inv,
pred_inv_gru_Close, squared=False, multioutput='raw_values')
mape = mean_absolute_percentage_error(y_test_Close_inv,
pred_inv_gru_Close, multioutput='raw_values')
print('Prediksi Metode GRU Close')
print('RMSE Close = %.4f'%(rmse[0]))
print()
print('MAPE Close = %.4f'%((mape[0])*100), '%')
```

```
rmse = mean_squared_error(y_test_Open_inv, pred_inv_gru_Open,
squared=False, multioutput='raw_values')
mape = mean_absolute_percentage_error(y_test_Open_inv,
pred_inv_gru_Open, multioutput='raw_values')
print('Prediksi Metode GRU Open')
```

```
print('RMSE Open = %.4f'%(rmse[0]))
print()
print('MAPE Open = %.4f'%((mape[0])*100), '%')
```

```
rmse = mean_squared_error(y_test_High_inv, pred_inv_gru_High,
squared=False, multioutput='raw_values')
mape = mean_absolute_percentage_error(y_test_High_inv,
pred_inv_gru_High, multioutput='raw_values')
print('Prediksi Metode GRU High')
print('RMSE High = %.4f'%(rmse[0]))
print()
print('MAPE High = %.4f'%((mape[0])*100), '%')
print()
```

```
rmse = mean_squared_error(y_test_Low_inv, pred_inv_gru_Low,
squared=False, multioutput='raw_values')
mape = mean_absolute_percentage_error(y_test_Low_inv,
pred_inv_gru_Low, multioutput='raw_values')
print('Prediksi Metode GRU Low')
print('RMSE Low = %.4f'%(rmse[0]))
print()
print('MAPE Low = %.4f'%((mape[0])*100), '%')
print()
```