DAFTAR PUSTAKA

Arfan, A., & ETP, L. (2019). Prediksi harga saham di Indonesia menggunakan algoritma long short-term memory. *SeNTIK*, *3*(1), 225–230.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, 1724–1734. https://doi.org/10.3115/v1/d14-1179

Dwiyanto, M. A., Djamal, C. E., & Maspupah, A. (2019). Prediksi Harga Saham menggunakan Metode Recurrent Neural Network. *Seminar Nasional Aplikasi Teknologi Informasi (SNATI)*, 33–38.

Fauzi, A. (2019). Forecasting Saham Syariah Dengan Menggunakan Lstm. *Al-Masraf : Jurnal Lembaga Keuangan Dan Perbankan*, *4*(1), 65. https://doi.org/10.15548/al-masraf.v4i1.235

Gao, P., Zhang, R., & Yang, X. (2020). The Application of Stock Index Price *Predict*ion with Neural Network. *Mathematical and Computational Applications*, *25*(3), 53. https://doi.org/10.3390/mca25030053

Gao, Y., Wang, R., & Zhou, E. (2021). Stock *Predict*ion Based on Optimized LSTM and GRU Models. *Scientific Programming*, *2021*. https://doi.org/10.1155/2021/4055281

Ghudafa, M., Akbar, T., Panggabean, S., & Noor, M. (2022). *Perbandingan Prediksi Harga Saham Dengan Menggunakan LSTM GRU Dengan Transformer*. *11*(1), 2020–2023.

Hastomo, W., Karno, A. S. B., Kalbuana, N., Nisfiani, E., ETP, L. (2021). Optimasi Deep Learning untuk Prediksi Saham di. *... (Jurnal Edukasi Dan ...*, *7*(2), 133–140. https://jurnal.untan.ac.id/index.php/jepin/article/view/47411

Herdianto. (2013). Prediksi Kerusakan Motor Induksi Menggunakan Tesis Oleh Herdianto Fakultas Teknik. *Fakultas Teknik, Universitas Sumatera Utara, Medan*.

Huang, J., Zhang, Y., Zhang, J., & Zhang, X. (2018). A tensor-based sub-mode coordinate algorithm for stock prediction. *Proceedings - 2018 IEEE 3rd International Conference on Data Science in Cyberspace, DSC 2018*, 716–721. https://doi.org/10.1109/DSC.2018.00114

Islam, M. S., & Hossain, E. (2021). Foreign exchange currency rate *predict*ion using a GRU-LSTM hybrid network. *Soft Computing Letters*, *3*(October 2020), 100009. https://doi.org/10.1016/j.socl.2020.100009

Karno, A. S. B. (2020). Prediksi Data *Time series* Saham Bank BRI Dengan Mesin Belajar LSTM (Long ShortTerm Memory). *Journal of Informatic and*

Information Security, *1*(1), 1–8. https://doi.org/10.31599/jiforty.v1i1.133

Khalis Sofi, Aswan Supriyadi Sunge, Sasmitoh Rahmad Riady, & Antika Zahrotul Kamalia. (2021). Perbandingan Algoritma Linear Regression, Lstm, Dan Gru Dalam Memprediksi Harga Saham Dengan Model *Time series*. *Seminastika*, *3*(1), 39–46. https://doi.org/10.47002/seminastika.v3i1.275

Müller, F. (2020). Stock Market *Predict*ion using *Multivariate Time series* and Recurrent Neural Networks in Python. https://www.relataly.com/stock-market-*predict*ion-using-*multivariate*-time-series-in-python/

Obaidur Rahman, M., Sabir Hossain, M., Kamal Hossen, M., Junaid, T.-S., & Shafiul Alam Forhad, M. (2019). *Predict*ing Prices of Stock Market using *Gated Recurrent Unit*s (GRUs) Neural Networks Pattern Matching View project Fish Observatory and Census System (FOCS) View project *Predict*ing Prices of Stock Market using *Gated Recurrent Unit*s (GRUs) Neural Networks. *IJCSNS International Journal of Computer Science and Network Security*, *19*(1). https://www.researchgate.net/publication/331385031

Rahmi, A., Mahmudy, W. F., & Setiawan, B. D. (2015). Prediksi Harga Saham Berdasarkan Data Historis Menggunakan Model Regresi (Stock Price Prediction Based on Historical Data using Genetic Algorithm). *DORO : Repository Jurnal Mahasiswa PTIIK Universitas Brawijaya*, *5*(12), 1–9.

Rochman, E. M. S., & Djunaidy, A. (2014). Prediksi Harga Saham Yang Mempertimbangkan Faktor Eksternal Menggunakan Jaringan Saraf Tiruan. *Jurnal Ilmiah NERO*, *1*(2), 5–11.

Santi, N., & Widodo, S. (2021). Algoritma Neural Network Backpropagation Untuk Prediksi Harga Saham Pada Tiga Golongan Perusahaan Berdasarkan Kapitalisasinya. *Faktor Exacta*, *14*(3), 131. https://doi.org/10.30998/faktorexacta.v14i3.9365

Satyo, A., Karno, B., Hastomo, W., Nisfiani, E., & Lukman, S. (2020). Optimais Deep Learning untuk Prediksi Data Saham Di Era Pandemi Covid -19. *Santei*, 43–54.

Singh, P. (2021). Deploy Machine Learning Models to Production. In *Deploy Machine Learning Models to Production*. https://doi.org/10.1007/978-1-4842-6546-8

Siringoringo, Z. (2021). *Prediksi Tingkat Inflasi Nasional Menggunakan Metode Gated Recurrent Unit*.

Tanwar, S., Patel, N. P., Patel, S. N., Patel, J. R., Sharma, G., & Davidson, I. E. (2021). Deep Learning-Based Cryptocurrency Price *Predict*ion Scheme with Inter-Dependent Relations. *IEEE Access*, *9*, 138633–138646. https://doi.org/10.1109/ACCESS.2021.3117848

Thakur, A. (2021). What's the Optimal *batch size* to *Train* a Neural Network?. https://wandb.ai/ayush-thakur/dl-question-bank/reports/What-s-the-Optimal-Batch-Size-to-*Train*-a-Neural-Network---VmlldzoyMDkyNDU

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

Wojtkiewicz, J., Hosseini, M., Gottumukkala, R., & Chambers, T. L. (2019). Hour-ahead solar irradiance forecasting using *multivariate Gated Recurrent Unit*s. *Energies*, *12*(21), 1–13. https://doi.org/10.3390/en12214055

Zaman, L., Sumpeno, S., & Hariadi, M. (2019). Analisis Kinerja LSTM dan GRU sebagai Model Generatif untuk Tari Remo. *Jurnal Nasional Teknik Elektro Dan Teknologi Informasi (JNTETI)*, *8*(2), 142. https://doi.org/10.22146/jnteti.v8i2.503

Zayini Anwar, M., & Habibi, S. (2020). *Analisis Prediksi Performasi Arah Pergerakan Saham Apple (Appl) Menggukan Metode Recurrent Neural Networks/Long Short Term Memory Networks (Rnn/Lstm)*. *February*.

LAMPIRAN

Berikut lampiran *source code* .ipynb dalam membuat arsitektur model *multivariate Gated Recurrent Unit*:

- *Library* yang digunakan dan pengambilan data

```
## Import Libraries and set information
from pandas_datareader import data as pdr
from datetime import date

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from numpy import array
import math
from sklearn.metrics import mean_squared_error,
mean_absolute_error
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, GRU, Dropout,
concatenate
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.backend import square, mean
from keras.utils.vis_utils import plot_model
import yfinance as yf

yf.pdr_override()

# Get Current Date
today = date.today()

# Set Info
start_date = '2010-01-01'
end_date = '2022-02-08'

rates = ['MXNUSD=X', 'EURUSD=X']
assets = {'AAPL':'USD', 'AAPL.MX':'MXN', 'APC.F':'EUR'}
tickers = yf.Tickers(' '.join(rates))
```

- Konversi data

```
#Convert Low price

exchange_low = []

for i in tickers.tickers:

       exchange_low.append(tickers.tickers[i].history(start=sta
rt_date, end=end_date).Low)


exlow_df = pd.DataFrame(exchange_low).T
```

```
exlow_df.columns = rates
exlow_df['USDUSD=X'] = 1.0


low_df = pd.DataFrame()
for k,v in assets.items():
    data = yf.download(k, start=start_date, end=end_date,
progress=False).Low.to_frame()
    data['ticker'] = k
    if v[:3] == 'USD':
        data['rating'] = exlow_df['USDUSD=X']
        data['xlow'] = data['Low'] * data['rating']
    elif v[:3] == 'MXN':
        data['rating'] = exlow_df['MXNUSD=X']
        data['xlow'] = data['Low'] * data['rating']
    elif v[:3] == 'EUR':
        data['rating'] = exlow_df['EURUSD=X']
        data['xlow'] = data['Low'] * data['rating']
    else:
        data['rating'] = np.NaN
        data['xlow'] = np.NaN
    low_df = pd.concat([low_df, data], axis=0)

#Convert Open price
exchange_open = []
for i in tickers.tickers:
       exchange_open.append(tickers.tickers[i].history(start=st
art_date, end=end_date).Open)


exop_df = pd.DataFrame(exchange_open).T
exop_df.columns = rates
exop_df['USDUSD=X'] = 1.0


open_df = pd.DataFrame()
for k,v in assets.items():
    data = yf.download(k, start=start_date, end=end_date,
progress=False).Open.to_frame()
    data['ticker'] = k
    if v[:3] == 'USD':
        data['rating'] = exop_df['USDUSD=X']
```

```
        data['xopen'] = data['Open'] * data['rating']
    elif v[:3] == 'MXN':
        data['rating'] = exop_df['MXNUSD=X']
        data['xopen'] = data['Open'] * data['rating']
    elif v[:3] == 'EUR':
        data['rating'] = exop_df['EURUSD=X']
        data['xopen'] = data['Open'] * data['rating']
    else:
        data['rating'] = np.NaN
        data['xopen'] = np.NaN
    open_df = pd.concat([open_df, data], axis=0)

#Convert Close price
exchange_close = []
for i in tickers.tickers:
       exchange_close.append(tickers.tickers[i].history(start=s
tart_date, end=end_date).Close)

excl_df = pd.DataFrame(exchange_close).T
excl_df.columns = rates
excl_df['USDUSD=X'] = 1.0

close_df = pd.DataFrame()
for k,v in assets.items():
    data = yf.download(k, start=start_date, end=end_date,
progress=False).Close.to_frame()
    data['ticker'] = k
    if v[:3] == 'USD':
        data['rating'] = excl_df['USDUSD=X']
        data['xclose'] = data['Close'] * data['rating']
    elif v[:3] == 'MXN':
        data['rating'] = excl_df['MXNUSD=X']
        data['xclose'] = data['Close'] * data['rating']
    elif v[:3] == 'EUR':
        data['rating'] = excl_df['EURUSD=X']
        data['xclose'] = data['Close'] * data['rating']
    else:
        data['rating'] = np.NaN
```

```
        data['xclose'] = np.NaN
    close_df = pd.concat([close_df, data], axis=0)

#Convert High price
exchange_high = []
for i in tickers.tickers:
       exchange_high.append(tickers.tickers[i].history(start=st
art_date, end=end_date).High)

exhigh_df = pd.DataFrame(exchange_high).T
exhigh_df.columns = rates
exhigh_df['USDUSD=X'] = 1.0

high_df = pd.DataFrame()
for k,v in assets.items():
    data = yf.download(k, start=start_date, end=end_date,
progress=False).High.to_frame()
    data['ticker'] = k
    if v[:3] == 'USD':
        data['rating'] = exhigh_df['USDUSD=X']
        data['xhigh'] = data['High'] * data['rating']
    elif v[:3] == 'MXN':
        data['rating'] = exhigh_df['MXNUSD=X']
        data['xhigh'] = data['High'] * data['rating']
    elif v[:3] == 'EUR':
        data['rating'] = exhigh_df['EURUSD=X']
        data['xhigh'] = data['High'] * data['rating']
    else:
        data['rating'] = np.NaN
        data['xhigh'] = np.NaN
    high_df = pd.concat([high_df, data], axis=0)

#Volume
volume_df = pd.DataFrame()
for k,v in assets.items():
    data = yf.download(k, start=start_date, end=end_date,
progress=False).Volume.to_frame()
    data['ticker'] = k
    volume_df = pd.concat([volume_df, data], axis=0)
```

- Buat *dataset* masing – masing data saham

```
stock_aapl = stock_df.loc[stock_df['ticker'] == 'AAPL']
stock_f = stock_df.loc[stock_df['ticker'] == 'APC.F']
stock_mx = stock_df.loc[stock_df['ticker'] == 'AAPL.MX']
```

- Visualisasi data masing – masing data saham

```
#AAPL
plt.figure(figsize=(16,8))
plt.title('AAPL Price')
plt.plot(stock_aapl['xlow'])
plt.plot(stock_aapl['xopen'])
plt.plot(stock_aapl['xclose'])
plt.plot(stock_aapl['xhigh'])

plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend(['low','open','close','high'])
plt.grid()

currentFig = plt.gcf()
currentFig.set_facecolor('white')
plt.show()

#APC.F
plt.figure(figsize=(16,8))
plt.title('APC.F Price')
plt.plot(stock_f['xlow'])
plt.plot(stock_f['xopen'])
plt.plot(stock_f['xclose'])
plt.plot(stock_f['xhigh'])

plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend(['low','open','close','high'])
plt.grid()

currentFig = plt.gcf()
currentFig.set_facecolor('white')
plt.show()

#AAPL.MX
plt.figure(figsize=(16,8))
plt.title('AAPL.MX Price')
plt.plot(stock_mx['xlow'])
plt.plot(stock_mx['xopen'])
plt.plot(stock_mx['xclose'])
plt.plot(stock_mx['xhigh'])

plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend(['low','open','close','high'])
```

```
plt.grid()

currentFig = plt.gcf()
currentFig.set_facecolor('white')
plt.show()
```

- Normalisasi data masing – masing data saham

```
# Extract price
data_aapl =
stock_aapl.filter(['xlow','xopen','xclose','xhigh','Volume'])
dataset_aapl = data_aapl.values

# Preprocess the data
normalizer = MinMaxScaler(feature_range=(0,1)) # instantiate
scaler
normalizedData_aapl = normalizer.fit_transform(dataset_aapl) #
values between 0,1
print(normalizedData_aapl)

# Extract price
data_f =
stock_f.filter(['xlow','xopen','xclose','xhigh','Volume'])
dataset_f = data_f.values

# Preprocess the data
normalizer = MinMaxScaler(feature_range=(0,1))
normalizedData_f = normalizer.fit_transform(dataset_f) # values
between 0,1
print(normalizedData_f)
# Extract price
data_mx = stock_mx.filter(['xlow', 'xopen', 'xclose',
'xhigh','Volume'])
dataset_mx = data_mx.values

# Preprocess the data
normalizer = MinMaxScaler(feature_range=(0,1)) # instantiate
scaler
normalizedData_mx = normalizer.fit_transform(dataset_mx) #
values between 0,1
print(normalizedData_mx)
```

- Visualisasi data setelah normalisasi

```
# Visualization data normalized

# AAPL
plt.figure(figsize=(16,8))
plt.title('Data Normalized AAPL')
plt.plot(normalizedData_aapl[:,:4])

plt.xlabel('Date')
plt.ylabel('Stock Price')
```

```
plt.legend(['low','open','close','high'])
plt.grid()

currentFig = plt.gcf()
currentFig.set_facecolor('white')
plt.show()

plt.figure(figsize=(16,8))
plt.title('Data Normalized Volume AAPL')
plt.plot(normalizedData_aapl[:,4:])

plt.xlabel('Date')
plt.ylabel('Volume')
plt.legend(['volume'])
plt.grid()

currentFig = plt.gcf()
currentFig.set_facecolor('white')
plt.show()
```

- Segmentasi data dan *set shape output*

```
# Storing the number of data points in the array
num_data_aapl = len(normalizedData_aapl)
num_days_used = 40
data_used_aapl = np.array([normalizedData_aapl[i : i +
num_days_used].copy() for i in range(num_data_aapl -
num_days_used)])

# Creating a numpy array that contains the value(s) to predict
# Currently, trying to fit the 4 outputs
data_to_predict_aapl =
np.array(normalizedData_aapl[(num_days_used):, :4])

# Creating a dates array for the dates that were used by
data_used
dates_used_aapl = stock_aapl.index[num_days_used:num_data_aapl]

# Storing the scaler object for prediction later
y_normaliser_aapl = MinMaxScaler()
y_normaliser_aapl.fit(stock_aapl[['xlow', 'xopen', 'xclose',
'xhigh']].to_numpy()[num_days_used:])

display(normalizedData_aapl.shape,
data_used_aapl.shape,data_to_predict_aapl.shape,
dates_used_aapl.shape)
```

- *Split* data

```
train_split = 0.8
data_size_aapl = data_used_aapl.shape[0]
num_features_aapl = data_used_aapl.shape[2]
```

```
train_size_aapl = int(data_size_aapl * train_split)
test_size_aapl = data_size_aapl - train_size_aapl

# Splitting the dataset up into train and test sets
X_train_aapl = data_used_aapl[0:train_size_aapl, :, :]
y_train_aapl = data_to_predict_aapl[0:train_size_aapl, :]
dates_train_aapl = dates_used_aapl[0:train_size_aapl]
X_test_aapl = data_used_aapl[train_size_aapl:, :, :]
y_test_aapl = data_to_predict_aapl[train_size_aapl:, :]
dates_test_aapl = dates_used_aapl[train_size_aapl:]

unscaled_y_train_aapl = stock_aapl[['xlow', 'xopen', 'xclose',
'xhigh']].to_numpy()[(num_days_used):][0:train_size_aapl, :]
unscaled_y_test_aapl = stock_aapl[['xlow', 'xopen', 'xclose',
'xhigh']].to_numpy()[(num_days_used):][train_size_aapl:, :]

display("X_train shape:", X_train_aapl.shape, "y_train shape:",
y_train_aapl.shape,
        "X_test shape:", X_test_aapl.shape, "y_test shape:",
y_test_aapl.shape,
        "unscaled_y_train shape:", unscaled_y_train_aapl.shape,
"unscaled_y_test shape:", unscaled_y_test_aapl.shape)
```

- Membuat model arsitektur *multivariate* GRU

```
# Creating the input layer, whose shape is (num_days_used,
num_features) because it is
# the open, high, low, close, volume for the past num_days_used
days
input = Input(shape=(num_days_used, num_features_aapl), name =
'input')

x = GRU(200, return_sequences=True, name='gru_1')(input)
x = Dropout(0.5)(x)
x = Dense(4, name='dense_2')(x)

# Branching out
output1 = GRU(200, return_sequences=True, name='low_0')(x)
output1 = Dropout(0.5)(output1)
output1 = GRU(200, name='low_1')(output1)
output1 = Dropout(0.5)(output1)
output1 = Dense(1, name='low_final')(output1)

output2 = GRU(200, return_sequences=True, name='open_0')(x)
output2 = Dropout(0.5)(output2)
output2 = GRU(200, name='open_1')(output2)
output2 = Dropout(0.5)(output2)
output2 = Dense(1, name='open_final')(output2)

output3 = GRU(200, return_sequences=True, name='close_0')(x)
output3 = Dropout(0.5)(output3)
output3 = GRU(200, name='close_1')(output3)
output3 = Dropout(0.5)(output3)
```

```
output3 = Dense(1, name='close_final')(output3)

output4 = GRU(200, return_sequences=True, name='high_0')(x)
output4 = Dropout(0.5)(output4)
output4 = GRU(200, name='high_1')(output4)
output4 = Dropout(0.5)(output4)
output4 = Dense(1, name='high_final')(output4)

model_aapl = Model(inputs = input, outputs = [output1, output2,
output3, output4])

# Choosing Adam as the optimizer
# Adam is an optimizer of hyperparameters
adam = Adam(learning_rate=0.001)

# model.compile is where you define the type of optimizer,
loss, etc.
model_aapl.compile(optimizer=adam, loss='mae')
model_aapl.summary()
```

- Visualisasi arsitektur model

```
# Displaying the structure of the final model
plot_model(model_aapl, show_shapes=True)
```

- Proses *training* model

```
history = model_aapl.fit(x=X_train_aapl, y=y_train_aapl,
batch_size=32, epochs=20, validation_split=0.2)
evaluation = model_aapl.evaluate(X_test_aapl, y_test_aapl)
print(evaluation)
```

- Visualisasi hasil *loss training* model

```
loss = history.history['loss']
val_loss = history.history['val_loss']
low_loss = history.history['low_final_loss']
val_low_loss = history.history['val_low_final_loss']
open_loss = history.history['open_final_loss']
val_open_loss = history.history['val_open_final_loss']
close_loss = history.history['close_final_loss']
val_close_loss = history.history['val_close_final_loss']
high_loss = history.history['high_final_loss']
val_high_loss = history.history['val_high_final_loss']
epochs = range(1, len(loss) + 1)
plt.figure()

#Train and validation loss
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and Validation loss')
```

```
plt.legend()
plt.show()

plt.plot(epochs, low_loss, 'b', label='Training loss')
plt.plot(epochs, val_low_loss, 'r', label='Validation loss')
plt.title('Training and Validation loss of low')
plt.legend()
plt.show()

plt.plot(epochs, open_loss, 'b', label='Training loss')
plt.plot(epochs, val_open_loss, 'r', label='Validation loss')
plt.title('Training and Validation loss of open')
plt.legend()
plt.show()

plt.plot(epochs, close_loss, 'b', label='Training loss')
plt.plot(epochs, val_close_loss, 'r', label='Validation loss')
plt.title('Training and Validation loss of close')
plt.legend()
plt.show()

plt.plot(epochs, high_loss, 'b', label='Training loss')
plt.plot(epochs, val_high_loss, 'r', label='Validation loss')
plt.title('Training and Validation loss of high')
plt.legend()
plt.show()
```

- Menampilkan visualisasi hasil prediksi data *training*

```
y_train_low_aapl_pred, y_train_open_aapl_pred,
y_train_close_aapl_pred, y_train_high_aapl_pred =
model_aapl.predict(X_train_aapl)
trainpreds_arr_aapl = np.hstack((y_train_low_aapl_pred,
y_train_open_aapl_pred, y_train_close_aapl_pred,
y_train_high_aapl_pred))
y_train_aapl_pred =
y_normaliser_aapl.inverse_transform(trainpreds_arr_aapl)

plt.gcf().set_size_inches(22, 15, forward=True)
# real values plotted
plt.plot(dates_train_aapl, unscaled_y_train_aapl[:,low],
label='real train low price', color='g')

# predicted values plotted
plt.plot(dates_train_aapl, y_train_aapl_pred[:,low],
label='predicted train low', color='r', linestyle='dashed')

currentFig.set_facecolor('white')
plt.legend()
plt.xlabel('Year', fontsize=20)
plt.ylabel('Price (USD)', fontsize=20)
plt.title('Real and Predicted Low AAPL Price on the Train Set',
fontsize=30)
```

```
plt.show()

plt.gcf().set_size_inches(22, 15, forward=True)
# real values plotted
plt.plot(dates_train_aapl, unscaled_y_train_aapl[:,open],
label='real train open price', color='g')

# predicted values plotted
plt.plot(dates_train_aapl, y_train_aapl_pred[:,open],
label='predicted train open', color='r', linestyle='dashed')

currentFig.set_facecolor('white')
plt.legend()
plt.xlabel('Year', fontsize=20)
plt.ylabel('Price (USD)', fontsize=20)
plt.title('Real and Predicted Open AAPL Price on the Train Set',
fontsize=30)

plt.show()

plt.gcf().set_size_inches(22, 15, forward=True)
# real values plotted
plt.plot(dates_train_aapl, unscaled_y_train_aapl[:,close],
label='real train close price', color='g')

# predicted values plotted
plt.plot(dates_train_aapl, y_train_aapl_pred[:,close],
label='predicted train close', color='r', linestyle='dashed')

currentFig.set_facecolor('white')
plt.legend()
plt.xlabel('Year', fontsize=20)
plt.ylabel('Price (USD)', fontsize=20)
plt.title('Real and Predicted Close AAPL Price on the Train Set',
fontsize=30)

plt.show()

plt.gcf().set_size_inches(22, 15, forward=True)
# real values plotted
plt.plot(dates_train_aapl, unscaled_y_train_aapl[:,high],
label='real train high price', color='g')

# predicted values plotted
plt.plot(dates_train_aapl, y_train_aapl_pred[:,high],
label='predicted train high', color='r', linestyle='dashed')

currentFig.set_facecolor('white')
plt.legend()
plt.xlabel('Year', fontsize=20)
plt.ylabel('Price (USD)', fontsize=20)
plt.title('Real and Predicted High AAPL Price on the Train Set',
fontsize=30)
```

```
plt.show()
```

- Melakukan prediksi dan visualisasi hasil prediksi terhadap data *testing*

```
plt.gcf().set_size_inches(22, 15, forward=True)
# real values plotted
plt.plot(dates_test_aapl, unscaled_y_test_aapl)

# predicted values plotted
plt.plot(dates_test_aapl, y_test_aapl_pred, linestyle='dashed')

currentFig.set_facecolor('white')
plt.legend(['real low','real open','real close','real
high','predict low','predict open','predict close','predict
high'])
plt.xlabel('Year', fontsize=20)
plt.ylabel('Price (USD)', fontsize=20)
plt.title('Real and Predicted AAPL Price on the Test Set',
fontsize=30)

plt.show()
```

- Visualisasi hasil prediksi model terhadap data *training* dan *testing*

```
y_test_low_aapl_pred, y_test_open_aapl_pred,
y_test_close_aapl_pred, y_test_high_aapl_pred =
model_aapl.predict(X_test_aapl)
testpreds_arr_aapl = np.hstack((y_test_low_aapl_pred,
y_test_open_aapl_pred, y_test_close_aapl_pred,
y_test_high_aapl_pred))
y_test_aapl_pred =
y_normaliser_aapl.inverse_transform(testpreds_arr_aapl)

plt.gcf().set_size_inches(22, 15, forward=True)
# real values plotted
plt.plot(stock_aapl['xlow'], label='real low price', color='g')

# predicted values plotted
plt.plot(dates_train_aapl, y_train_aapl_pred[:,low],
label='predicted train low', color='lightgreen')
plt.plot(dates_test_aapl, y_test_aapl_pred[:,low],
label='predicted test low', color='r', linestyle='dashed')

currentFig.set_facecolor('white')
plt.legend()
plt.xlabel('Year', fontsize=20)
plt.ylabel('Price (USD)', fontsize=20)
plt.title('Real and Predicted Low AAPL Price on the Train and
Test Set', fontsize=30)

plt.show()
```

```
plt.gcf().set_size_inches(22, 15, forward=True)
# real values plotted
plt.plot(stock_aapl['xopen'], label='real open price',
color='g')

# predicted values plotted
plt.plot(dates_train_aapl, y_train_aapl_pred[:,open],
label='predicted train open', color='lightgreen')
plt.plot(dates_test_aapl, y_test_aapl_pred[:,open],
label='predicted test open', color='r', linestyle='dashed')

currentFig.set_facecolor('white')
plt.legend()
plt.xlabel('Year', fontsize=20)
plt.ylabel('Price (USD)', fontsize=20)
plt.title('Real and Predicted Open AAPL Price on the Train and
Test Set', fontsize=30)

plt.show()

plt.gcf().set_size_inches(22, 15, forward=True)
# real values plotted
plt.plot(stock_aapl['xclose'], label='real close price',
color='g')

# predicted values plotted
plt.plot(dates_train_aapl, y_train_aapl_pred[:,close],
label='predicted train close', color='lightgreen')
plt.plot(dates_test_aapl, y_test_aapl_pred[:,close],
label='predicted test close', color='r', linestyle='dashed')

currentFig.set_facecolor('white')
plt.legend()
plt.xlabel('Year', fontsize=20)
plt.ylabel('Price (USD)', fontsize=20)
plt.title('Real and Predicted Close AAPL Price on the Train and
Test Set', fontsize=30)

plt.show()

plt.gcf().set_size_inches(22, 15, forward=True)
# real values plotted
plt.plot(stock_aapl['xhigh'], label='real high price',
color='g')

# predicted values plotted
plt.plot(dates_train_aapl, y_train_aapl_pred[:,high],
label='predicted train high', color='lightgreen')
plt.plot(dates_test_aapl, y_test_aapl_pred[:,high],
label='predicted test high', color='r', linestyle='dashed')

currentFig.set_facecolor('white')
plt.legend()
```

```
plt.xlabel('Year', fontsize=20)
plt.ylabel('Price (USD)', fontsize=20)
plt.title('Real and Predicted High AAPL Price on the Train and
Test Set', fontsize=30)

plt.show()
```

- Evaluasi model dengan MAE, RMSE, MAPE dan RMSPE

```
#Evaluation MAE, RMSE, MAPE, RMSPE of High Price
def evaluation(unscaled_y_aapl, y_aapl_pred, indexc, x_aapl):
    mae = mean_absolute_error(unscaled_y_aapl[:,indexc],
y_aapl_pred[:,indexc])
    rmse =
math.sqrt(mean_squared_error(unscaled_y_aapl[:,indexc],
y_aapl_pred[:,indexc]))
    mape = np.mean(np.abs((unscaled_y_aapl[:,indexc] -
y_aapl_pred[:,indexc])/unscaled_y_aapl[:,indexc]))*100

    prermspe = ((y_aapl_pred[:,indexc] -
unscaled_y_aapl[:,indexc])/unscaled_y_aapl[:,indexc])
    rmspe_data = np.array([None]*len(x_aapl))
    for i in range(len(x_aapl)):
        rmspe_data[i] = math.pow(prermspe[i],2)
    rmspe = math.sqrt(np.mean(rmspe_data))*100
    print('MAE:', mae, '  \nRMSE:', rmse, '  \nMAPE:', mape, '
\nRMSPE:', rmspe)

# Low
# Calculating Train data prediction performance metrics
print('Evaluation Low Train Data Prediction')
evaluation(unscaled_y_train_aapl, y_train_aapl_pred, low,
X_train_aapl)
# Calculating Test data prediction performance metrics
print('Evaluation Low Test Data Prediction')
evaluation(unscaled_y_test_aapl, y_test_aapl_pred, low,
X_test_aapl)

# Open
# Calculating Train data prediction performance metrics
print('\nEvaluation Open Train Data Prediction')
evaluation(unscaled_y_train_aapl, y_train_aapl_pred, open,
X_train_aapl)
# Calculating Test data prediction performance metrics
print('Evaluation Open Test Data Prediction')
evaluation(unscaled_y_test_aapl, y_test_aapl_pred, open,
X_test_aapl)

# Close
# Calculating Train data prediction performance metrics
print('\nEvaluation Close Train Data Prediction')
```

```
evaluation(unscaled_y_train_aapl, y_train_aapl_pred, close,
X_train_aapl)
# Calculating Test data prediction performance metrics
print('Evaluation Close Test Data Prediction')
evaluation(unscaled_y_test_aapl, y_test_aapl_pred, close,
X_test_aapl)

# High
# Calculating Train data prediction performance metrics
print('\nEvaluation High Train Data Prediction')
evaluation(unscaled_y_train_aapl, y_train_aapl_pred, high,
X_train_aapl)
# Calculating Test data prediction performance metrics
print('Evaluation High Test Data Prediction')
evaluation(unscaled_y_test_aapl, y_test_aapl_pred, high,
X_test_aapl)
```

- Menyimpan model

```
model_aapl.save('modelaapl.h5')
```

- Proses data untuk melakukan prediksi 1 hari bursa selanjutnya

```
data_pred_aapl = normalizedData_aapl[num_data_aapl-
200:,:5].reshape(5,40,5)
data_pred_aapl

# x_input_aapl = data_used_aapl[train_size_aapl+test_size_aapl-
1:, :, :]
ylow_aapl,yopen_aapl,yclose_aapl,yhigh_aapl =
model_aapl.predict(data_pred_aapl[[4]])
yhat_aapl = ylow_aapl,yopen_aapl,yclose_aapl,yhigh_aapl
nextday_pred_aapl =
np.hstack((ylow_aapl,yopen_aapl,yclose_aapl,yhigh_aapl))
nextday_pred_aapl =
y_normaliser_aapl.inverse_transform(nextday_pred_aapl)
nextday_pred_aapl
```

Berikut lampiran *source code* .py dalam membuat rancangan *website*:

- File *utils.py*

```
import streamlit as st
from plotly import graph_objs as go
import math
from sklearn.metrics import mean_squared_error,
mean_absolute_error
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import load_model
import base64
```

```
low = 0
open = 1
close = 2
high = -1

def plot_actual_data(x_stock, stock, option):
    fig = go.Figure()
    fig.add_trace(go.Scatter(x=x_stock, y=stock['Low'],
name='Low'))
    fig.add_trace(go.Scatter(x=x_stock, y=stock['High'],
name='High'))
    fig.add_trace(go.Scatter(x=x_stock, y=stock['Open'],
name='Open'))
    fig.add_trace(go.Scatter(x=x_stock, y=stock['Close'],
name='Close'))
    fig.layout.update(title_text=option,
xaxis_rangeslider_visible=True, hovermode = 'x')
    figv = go.Figure()
    figv.add_trace(go.Scatter(x=x_stock, y=stock['Volume']))
    figv.layout.update(title_text='Volume',
xaxis_rangeslider_visible=True)
    st.plotly_chart(fig)
    st.plotly_chart(figv)

def plot_predict_data(dates_used, unscaled_y, y_pred):
    figl = go.Figure()
    figo = go.Figure()
    figc = go.Figure()
    figh = go.Figure()

    figl.layout.update(title_text=('Actual and Predicted on Low
Price'), xaxis_rangeslider_visible=True, hovermode = 'x')
    figl.add_trace(go.Scatter(x=dates_used,
y=unscaled_y[:,low], name='Actual Low Price'))
    figl.add_trace(go.Scatter(x=dates_used, y=y_pred[:,low],
name='Predicted Low Price'))
    figo.layout.update(title_text=('Actual and Predicted on
Open Price'), xaxis_rangeslider_visible=True, hovermode = 'x')
    figo.add_trace(go.Scatter(x=dates_used,
y=unscaled_y[:,open], name='Actual Open Price'))
    figo.add_trace(go.Scatter(x=dates_used, y=y_pred[:,open],
name='Predicted Open Price'))
    figc.layout.update(title_text=('Actual and Predicted on
Close Price'), xaxis_rangeslider_visible=True, hovermode = 'x')
    figc.add_trace(go.Scatter(x=dates_used,
y=unscaled_y[:,close], name='Actual Close Price'))
    figc.add_trace(go.Scatter(x=dates_used, y=y_pred[:,close],
name='Predicted Close Price'))
    figh.layout.update(title_text=('Actual and Predicted on
High Price'), xaxis_rangeslider_visible=True, hovermode = 'x')
    figh.add_trace(go.Scatter(x=dates_used,
y=unscaled_y[:,high], name='Actual High Price'))
```

```
    figh.add_trace(go.Scatter(x=dates_used, y=y_pred[:,high],
name='Predicted High Price'))

    st.plotly_chart(figl)
    st.plotly_chart(figo)
    st.plotly_chart(figc)
    st.plotly_chart(figh)

def convert_df(df):
    return df.to_csv()

def csvdata(option, stock, y_pred):
    datapred = pd.DataFrame(y_pred)
    datapred.rename(columns =
{0:'Low',1:'Open',2:'Close',3:'High'}, inplace = True)
    data = pd.concat([stock,datapred], axis=1)
    csv = convert_df(data)
    b64 = base64.b64encode(csv.encode('utf-8')).decode()
    file_name = 'Data_{}.csv'.format(option)
       # st.markdown("#### Download File ###")
    href = f'<a href="data:file/csv;base64,{b64}"
download="{file_name}">Download data actual and prediction as
CSV</a>'
    st.markdown(href,unsafe_allow_html=True)

def evaluation(unscaled_y, y_pred, indexc, len_data):
    mae = mean_absolute_error(unscaled_y[:,indexc],
y_pred[:,indexc])
    rmse = math.sqrt(mean_squared_error(unscaled_y[:,indexc],
y_pred[:,indexc]))
    mape = np.mean(np.abs((unscaled_y[:,indexc] -
y_pred[:,indexc])/unscaled_y[:,indexc]))*100

    prermspe = ((y_pred[:,indexc] -
unscaled_y[:,indexc])/unscaled_y[:,indexc])
    rmspe_data = np.array([None]*len(len_data))
    for i in range(len(len_data)):
        rmspe_data[i] = math.pow(prermspe[i],2)
    rmspe = math.sqrt(np.mean(rmspe_data))*100
    st.write('MAE:', mae, '  \nRMSE:', rmse, '  \nMAPE:', mape,
'  \nRMSPE:', rmspe)

def forcast(model, normalizedData, num_data, y_normaliser):
    data_pred = normalizedData[num_data-
200:,:5].reshape(5,40,5)
    ylow,yopen,yclose,yhigh = model.predict(data_pred[[4]])
    yhat = ylow,yopen,yclose,yhigh
    nextday_pred = np.hstack((yhat))
    nextday_pred = y_normaliser.inverse_transform(nextday_pred)
    nextday_pred = {'Low':nextday_pred[0][0],
'Open':nextday_pred[0][1], 'Close':nextday_pred[0][2],
'High':nextday_pred[0][3]}
    st.write('🙏Next day stock price forecast:', nextday_pred)
```

```
def write_low():
    st.write('Evaluation of low price')
def write_open():
    st.write('Evaluation of open price')
def write_close():
    st.write('Evaluation of close price')
def write_high():
    st.write('Evaluation of high price')
def data_act():
    st.subheader('Data Actual')
def visual_data():
    st.subheader('Visualizations Data')
def visual_actpred_data():
    st.subheader('Visualizations Actual and Prediction Data')
def write_evaluation():
    st.subheader('Evaluation')

def proccess(dataset, stock, option):
    # Preprocess the data
    num_days_used = 40
    normalizer = MinMaxScaler(feature_range=(0,1)) #
instantiate scaler
    normalizedData = normalizer.fit_transform(dataset) # values
between 0,1

    # Storing the number of data points in the array
    num_data = len(normalizedData)
    data_used = np.array([normalizedData[i : i +
num_days_used].copy() for i in range(num_data -
num_days_used)])
    dates_used = stock.index[num_days_used:num_data]
    y_normaliser = MinMaxScaler()
    y_normaliser.fit(stock[['Low', 'Open', 'Close',
'High']].to_numpy()[num_days_used:])

    # Splitting the dataset up into train and test sets
    X_test = data_used[0:, :, :]
    unscaled_y = stock[['Low', 'Open', 'Close',
'High']].to_numpy()[(num_days_used):][0:, :]

    # load model
    if option == 'AAPL':
        model = load_model(modelaapl.h5')
    elif option == 'APC.F':
        model = load_model(modelf.h5')
    else:
        model = load_model(modelmx.h5')

    # modelpredict
    y_low_pred, y_open_pred, y_close_pred, y_high_pred =
model.predict(X_test)
    testpreds_arr = np.hstack((y_low_pred, y_high_pred,
y_open_pred, y_close_pred))
```

```
    y_pred = y_normaliser.inverse_transform(testpreds_arr)
    visual_actpred_data()
    plot_predict_data(dates_used, unscaled_y, y_pred)

    # data to csv
    csvdata(option, stock, y_pred)
    st.info('📝Note: The amount of data is not the same because
there are 40 initial data used as model input.👆')

    #evaluation
    write_evaluation()
    col1, col2 = st.columns(2)
    col3, col4 = st.columns(2)
    with col1:
        write_low()
        evaluation(unscaled_y, y_pred, low, data_used)
    with col2:
        write_open()
        evaluation(unscaled_y, y_pred, open, data_used)
    with col3:
        write_close()
        evaluation(unscaled_y, y_pred, close, data_used)
    with col4:
        write_high()
        evaluation(unscaled_y, y_pred, high, data_used)

    #forcasting
    forcast(model, normalizedData, num_data, y_normaliser)
```

- File *main.py*

```
import streamlit as st
import pandas as pd
from datetime import date, timedelta
from utils import *
import yfinance as yf

st.set_page_config(
    page_title='Apple Stock Prediction',
    page_icon='📈',
    layout='centered')

st.title('Apple Inc Stock Prediction based on Historical Data')

# sidebar #
st.sidebar.write('Data as on [Yahoo
Finance](https://finance.yahoo.com) exchange in USD')
option = st.sidebar.selectbox('What would you like to
predicted?',('AAPL', 'APC.F', 'AAPL.MX'), key = 'tick')
st.sidebar.write('You selected:', st.session_state.tick)
today = date.today()
before = today - timedelta(days=4500)
start_date = st.sidebar.date_input('Start date', before)
```

```
end_date = st.sidebar.date_input('End date', today)
dates = timedelta(days=60)
predict = st.sidebar.button('Predict')
st.sidebar.info('''This Project is used for only learning and
development process. I don't encourage anyone
to invest in stock based on any data represented here.😊🤦🤖''')

# proccess #
rates = ['MXNUSD=X', 'EURUSD=X']
tickers = yf.Tickers(' '.join(rates))

low = 0
open = 1
close = 2
high = -1

exchange_low = []
for i in tickers.tickers:

exchange_low.append(tickers.tickers[i].history(start=start_date,
end=end_date).Low)
exchange_open = []
for i in tickers.tickers:

exchange_open.append(tickers.tickers[i].history(start=start_date,
end=end_date).Open)
exchange_close = []
for i in tickers.tickers:

exchange_close.append(tickers.tickers[i].history(start=start_date,
end=end_date).Close)
exchange_high = []
for i in tickers.tickers:

exchange_high.append(tickers.tickers[i].history(start=start_date,
end=end_date).High)

try:
    if predict:
        if start_date < end_date:
            if option == 'AAPL':
                # get data
                stock_aapl = yf.download(option, start_date,
end_date)
                stock_aapl.reset_index(inplace=True)

                # show data
                data_act()
                st.write(stock_aapl)

                # visualizations
                visual_data()
                plot_actual_data(stock_aapl['Date'], stock_aapl,
option)
```

```
                # Extract price
                data_aapl =
stock_aapl.filter(['Low','Open','Close','High','Volume'])
                dataset_aapl = data_aapl.values

                # Proccess prediction
                proccess(dataset_aapl, stock_aapl, option)

            elif option == 'APC.F':
                #Convert Low price
                exlow_df = pd.DataFrame(exchange_low).T
                exlow_df.columns = rates
                low_df = pd.DataFrame()

                data = yf.download(option, start=start_date,
end=end_date, progress=False).Low.to_frame()
                data['ticker'] = option
                data['rating'] = exlow_df['EURUSD=X']
                data['Low'] = data['Low'] * data['rating']
                low_df = pd.concat([low_df, data], axis=0)

                #Convert Open price
                exop_df = pd.DataFrame(exchange_open).T
                exop_df.columns = rates
                open_df = pd.DataFrame()

                data = yf.download(option, start=start_date,
end=end_date, progress=False).Open.to_frame()
                data['ticker'] = option
                data['rating'] = exop_df['EURUSD=X']
                data['Open'] = data['Open'] * data['rating']
                open_df = pd.concat([open_df, data], axis=0)

                #Convert Close price
                excl_df = pd.DataFrame(exchange_close).T
                excl_df.columns = rates
                close_df = pd.DataFrame()

                data = yf.download(option, start=start_date,
end=end_date, progress=False).Close.to_frame()
                data['ticker'] = option
                data['rating'] = excl_df['EURUSD=X']
                data['Close'] = data['Close'] * data['rating']
                close_df = pd.concat([close_df, data], axis=0)

                #Convert High price
                exhigh_df = pd.DataFrame(exchange_high).T
                exhigh_df.columns = rates
                high_df = pd.DataFrame()

                data = yf.download(option, start=start_date,
end=end_date, progress=False).High.to_frame()
                data['ticker'] = option
```

```
                data['rating'] = exhigh_df['EURUSD=X']
                data['High'] = data['High'] * data['rating']
                high_df = pd.concat([high_df, data], axis=0)

                #Volume
                volume_df = pd.DataFrame()
                data = yf.download(option, start=start_date,
end=end_date, progress=False).Volume.to_frame()
                data['ticker'] = option
                volume_df = pd.concat([volume_df, data], axis=0)

                #concat
                stock_f = pd.DataFrame()
                stock_f = pd.concat([stock_f, low_df['ticker'],
low_df['Low'], open_df['Open'], close_df['Close'],
high_df['High'], volume_df['Volume']], axis=1)
                stock_f = stock_f.ffill()
                stock_f.reset_index(inplace=True)

                # show data
                data_act()
                st.write(stock_f)

                # visualizations
                visual_data()
                plot_actual_data(stock_f['index'], stock_f,
option)

                # Extract price
                data_f =
stock_f.filter(['Low','Open','Close','High','Volume'])
                dataset_f = data_f.values

                # Proccess prediction
                proccess(dataset_f, stock_f, option)

            else:
                #Convert Low price
                exlow_df = pd.DataFrame(exchange_low).T
                exlow_df.columns = rates
                low_df = pd.DataFrame()

                data = yf.download(option, start=start_date,
end=end_date, progress=False).Low.to_frame()
                data['ticker'] = option
                data['rating'] = exlow_df['MXNUSD=X']
                data['Low'] = data['Low'] * data['rating']
                low_df = pd.concat([low_df, data], axis=0)

                #Convert Open price
                exop_df = pd.DataFrame(exchange_open).T
                exop_df.columns = rates
                open_df = pd.DataFrame()
```

```
                data = yf.download(option, start=start_date,
end=end_date, progress=False).Open.to_frame()
                data['ticker'] = option
                data['rating'] = exop_df['MXNUSD=X']
                data['Open'] = data['Open'] * data['rating']
                open_df = pd.concat([open_df, data], axis=0)

                #Convert Close price
                excl_df = pd.DataFrame(exchange_close).T
                excl_df.columns = rates
                close_df = pd.DataFrame()

                data = yf.download(option, start=start_date,
end=end_date, progress=False).Close.to_frame()
                data['ticker'] = option
                data['rating'] = excl_df['MXNUSD=X']
                data['Close'] = data['Close'] * data['rating']
                close_df = pd.concat([close_df, data], axis=0)

                #Convert High price
                exhigh_df = pd.DataFrame(exchange_high).T
                exhigh_df.columns = rates
                high_df = pd.DataFrame()

                data = yf.download(option, start=start_date,
end=end_date, progress=False).High.to_frame()
                data['ticker'] = option
                data['rating'] = exhigh_df['MXNUSD=X']
                data['High'] = data['High'] * data['rating']
                high_df = pd.concat([high_df, data], axis=0)

                #Volume
                volume_df = pd.DataFrame()
                data = yf.download(option, start=start_date,
end=end_date, progress=False).Volume.to_frame()
                data['ticker'] = option
                volume_df = pd.concat([volume_df, data], axis=0)

                #concat
                stock_mx = pd.DataFrame()
                stock_mx = pd.concat([stock_mx, low_df['ticker'],
low_df['Low'], open_df['Open'], close_df['Close'],
high_df['High'], volume_df['Volume']], axis=1)
                stock_mx = stock_mx.ffill()
                stock_mx.reset_index(inplace=True)

                # show data
                data_act()
                st.write(stock_mx)

                # visualizations
                visual_data()
```

```
                plot_actual_data(stock_mx['index'], stock_mx,
option)

                # Extract price
                data_mx =
stock_mx.filter(['Low','Open','Close','High','Volume'])
                dataset_mx = data_mx.values

                # Proccess prediction
                proccess(dataset_mx, stock_mx, option)
        else:
            st.info('Please, end date must fall after start
date🙏')
    else:
        st.info('Please, choose the available options then click
predict button🙏')
except:
    st.error('Unable process data to prediction because it does
not match system requirements, please choose again with more data
the available options then click predict button🙏')
```

Berikut *link repository source code*:

ce3tnia/ProjectStockPredictStreamlit: Project Skripsi (github.com)