DAFTAR PUSTAKA

Aldi, M. W. P., Jondri, J., & Aditsania, A. (2018). Analisis Dan Implementasi Long Short Term Memory Neural Network Untuk Prediksi Harga Bitcoin. *eProceedings of Engineering*, *5*(2), Article 2. https://openlibrarypublications.telkomuniversity.ac.id/index.php/engineering/article/view/6739

Arfan, A., & Etp, L. (2020). Perbandingan Algoritma Long Short-Term Memory dengan SVR Pada Prediksi Harga Saham di Indonesia. *PETIR*, *13*(1), Article 1. https://doi.org/10.33322/petir.v13i1.858

Aulia, N. (2020). *Prediksi Harga Ethereum Berdasarkan Informasi Blockchain Menggunakan Metode Long Short Term Memory* [Thesis, Universitas Islam Indonesia]. https://dspace.uii.ac.id/handle/123456789/23610

CoinMarketCap. (2022). https://coinmarketcap.com/. Diakses pada 1 Agustus 2022

Darmawan, H. & S.Ds. (2017, February 23). *Mengenal MACD (Moving Average Convergence Divergence) Pada Saham*. Perencana Keuangan Pertama Yang Tercatat OJK. https://www.finansialku.com/mengenal-indikator-macd-moving-average-convergence-divergence-dalam-trading-saham/

Hayat, N. M., Prasetijo, A. B., & Septiana, R. (2019). Analisis Kinerja Algoritma J48 Decision Tree untuk Pengambilan Keputusan Beli/Jual pada Saham PT Harum Energi Tbk. (HRUM). *JTIM : Jurnal Teknologi Informasi Dan Multimedia*, *1*(3), Article 3. https://doi.org/10.35746/jtim.v1i3.43

Howell, J. (2022, March 15). An Introduction to Binance Smart Chain (BSC). *101 Blockchains*. https://101blockchains.com/binance-smart-chain/

Huda, N., & Hambali, R. (2020). *Risiko dan Tingkat Keuntungan Investasi Cryptocurrency*. *17*, 72–84. https://doi.org/10.29313/performa.v17i1.7236

Karno, A. S. B. (2020). *Prediksi Data Time Series Saham Bank BRI Dengan Mesin Belajar LSTM (Long ShortTerm Memory)* (No. 1). *1*(1), Article 1. https://doi.org/10.31599/jiforty.v1i1.133

Kharisma, D. (2021, April 20). *Bollinger Band*. https://blog.pluang.com/cerdascuan/bollinger-band-adalah/

Lapian, E., Osmond, A. B., & Saputra, R. E. (2018). Recurrent Neural Network Untuk Pengenalan Ucapan Pada Dialek Manado. *eProceedings of Engineering*, *5*(3). https://openlibrarypublications.telkomuniversity.ac.id/index.php/engineering/article/view/8157

Larasati, K. D. (2020). *Prediksi Harga Bitcoin Berdasarkan Informasi Blockchain Menggunakan Metode Long-Short Term Memory* [Thesis, Universitas Islam Indonesia]. https://dspace.uii.ac.id/handle/123456789/23614

Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. 9.

Putri, R. N. M. H., Primasari, N. S., & Khusnah, H. (2022). Return Analisis Teknikal Moving Average, Bollinger Band, dan Relative Strength Index pada Cryptocurrency. *Jurnal Ilmiah Akuntansi Dan Keuangan*, *11*(1), Article 1. https://doi.org/10.32639/jiak.v11i1.25

Riyantoko, P. A., Fahruddin, T. M., Hindrayani, K. M., & Safitri, E. M. (2020). ANALISIS PREDIKSI HARGA SAHAM SEKTOR PERBANKAN

MENGGUNAKAN ALGORITMA LONG-SHORT TERMS MEMORY (LSTM). *Seminar Nasional Informatika (SEMNASIF)*, *1*(1), Article 1.

Rizkilloh, M. F., & Widyanesti, S. (2022). *Prediksi Harga Cryptocurrency Menggunakan Algoritma Long Short Term Memory (LSTM) | Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*. http://www.jurnal.iaii.or.id/index.php/RESTI/article/view/3630

Roy, G. W. (2016). ANALISIS TEKNIKAL SAHAM MENGGUNAKAN INDIKATOR BOLLINGER BANDS DAN RELATIVE STRENGTH INDEX UNTUK PENGAMBILAN KEPUTUSAN INVESTASI. *Jurnal Manajemen*, *6*(1), Article 1. https://doi.org/10.26460/jm.v6i1.202

Sherpa, D. (2021, March 16). Univariate, Bivariate and Multivariate Analysis. *Analytics Vidhya*. https://medium.com/analytics-vidhya/univariate-bivariate-and-multivariate-analysis-8b4fc3d8202c. Diakses pada 3 Agustus 2022

Xu, Y. (2020). *Bitcoin Price Forecast Using LSTM and GRU Recurrent networks, and Hidden Markov Model* [UCLA]. https://escholarship.org/uc/item/70d9n5sd

Zahara, S., Sugianto, & Ilmiddafiq, M. B. (2019). *Prediksi Indeks Harga Konsumen Menggunakan Metode Long Short Term Memory (LSTM) Berbasis Cloud Computing | Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*. https://www.jurnal.iaii.or.id/index.php/RESTI/article/view/1086

LAMPIRAN

Lampiran 1 *Source code* model prediksi menggunakan LSTM

btc-1m.py

```
# %%
import datetime
import pytz
from urllib.request import urlopen
import json
import pandas as pd
import pandas_ta as ta
import time
import datetime
import math

crrency = "BTCBUSD"
interval = "1m"
len_data = "1000"

def unix2utc(timestamp):
    your_dt = datetime.datetime.fromtimestamp(int(timestamp)/1000)  # using the local timezone
    utc_dt= your_dt.astimezone(pytz.UTC)
    return(utc_dt.strftime("%Y-%m-%d %H:%M:%S"))  # 2018-04-07 20:48:08, YMMV

def getDataCrypto(symbol,interval,limit,timeStart,timeEnd):

    # store the URL in url as
    # parameter for urlopen
    if((timeStart and timeEnd) is None):
        url = "https://api.binance.me/api/v3/klines?symbol="+str(symbol)+"&interval="+str(interval)+"&limit="+limit
    else:
        url =
"https://api.binance.me/api/v3/klines?symbol="+str(symbol)+"&interval="+str(interval)+"&limit="+limit+"&startTime="+s
tr(timeStart)+"&endTime="+str(timeEnd)

    response = urlopen(url)

    # storing the JSON response
    # from url in data
    data_json = json.loads(response.read())

    # print the json response
#     print(data_json)
    tm,op,hi,lo,cl,v = [],[],[],[],[],[]

    for x in range(len(data_json)):
    #     print(data_json[x][0])
        tm.append(data_json[x][0])
        op.append(data_json[x][1])
        hi.append(data_json[x][2])
        lo.append(data_json[x][3])
        cl.append(data_json[x][4])
        v.append(data_json[x][5])
    data = {"time": tm, "open":op,"high":hi,"low":lo,"close":cl,"volume":v}
    df = pd.DataFrame(data)
    df['time'] = df['time'].transform(lambda x: unix2utc(x))
    df["open"] = pd.to_numeric(df["open"], downcast="float")
    df["high"] = pd.to_numeric(df["high"], downcast="float")
    df["low"] = pd.to_numeric(df["low"], downcast="float")
    df["close"] = pd.to_numeric(df["close"], downcast="float")
    df["volume"] = pd.to_numeric(df["volume"], downcast="float")
    return df


df  = getDataCrypto(currency,interval,len_data,None,None)
# df.drop(columns=df.columns[0], axis=1,inplace=True)

# Calculate Indicator Technical
import pandas_ta as ta

sma5 = ta.sma(df['close'], length=5)
sma10 = ta.sma(df['close'], length=10)
sma20 = ta.sma(df['close'], length=20)
rsi = ta.rsi(df['close'].astype(float), length = 10)
macd = ta.macd(df['close'].astype(float), length = 10)
bbands = ta.bbands(df['close'].astype(float), length = 10)


df['ma5']=sma5
df['ma10'] = sma10
df['ma20'] = sma20
```

```
df['rsi']=rsi
df['bb_upper'] = bbands['BBL_10_2.0']
df['bb_lower'] = bbands['BBU_10_2.0']
df['bb_middle'] = bbands['BBM_10_2.0']
df['macd_l'] = macd['MACD_12_26_9']
df['macd_s'] = macd['MACDs_12_26_9']



df = df.dropna()



df = df[['time','close','volume','ma5','ma10','ma20','rsi','bb_upper','bb_lower','macd_l','macd_s']]

from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.models import load_model
from keras.layers import Dense
from keras.layers import LSTM

import pandas as pd

# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg

dataset = df
dataset = dataset.set_index('time')
# df = df[['time','close','volume','ma5','ma10','ma20','rsi','bb_upper','bb_lower','macd_l','macd_s']]
# dataset = dataset.drop(columns=['ma5','ma10','ma20','bb_upper','bb_lower','macd_l','macd_s'])

# %%
from datetime import datetime
from datetime import timedelta
predict_data = dataset.tail(1)
time_predict_data = predict_data.index.values[0]
if (interval=="1m"):
    interval_time = 1
elif (interval == "5m"):
    interval_time = 5
elif (interval == "15m"):
    interval_time = 15
time_predict_data = pd.to_datetime(time_predict_data)+timedelta(minutes=interval_time)

predict_data = predict_data.values
predict_data = predict_data.astype('float32')

values = dataset.values
values = values.astype('float32')
# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
scaled_predict = scaler.transform(predict_data)
# frame as supervised learning
reframed = series_to_supervised(scaled, 1, 1)
# drop columns we don't want to predict
# reframed.drop(reframed.columns[[9,10,11,12,13,14,15]], axis=1, inplace=True)
reframed = reframed.iloc[:, 0:(len(dataset.columns)+1)]
scaled_predict = DataFrame(scaled_predict)
scaled_predict.columns = reframed.columns[range(0,(len(dataset.columns)))]
```

```
# split into train and test sets
values = reframed.values
n_train_hours = 871
train = values[:n_train_hours, :]
test = values[n_train_hours:, :]
# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]
# predict_test_X, predict_test_y = predict_test[:, :-1], predict_test[:, -1]

# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
# predict_test_X = predict_test_X.reshape((predict_test_X.shape[0], 1, predict_test_X.shape[1]))
values_predict = scaled_predict.values
predict_data_x = values_predict.reshape((values_predict.shape[0],1,values_predict.shape[1]))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
print(predict_data_x.shape)

# %%
# design network
from keras.layers import GRU
model = Sequential()
model.add(GRU(25,input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')
history = model.fit(train_X, train_y, epochs=100, batch_size=32, validation_data=(test_X, test_y), verbose=1,
shuffle=False)



yhat_predict = model.predict(predict_data_x)
# predict_data_x
predict_data_x_new= predict_data_x.reshape((predict_data_x.shape[0], predict_data_x.shape[2]))

inv_yhat_predict_new = concatenate((yhat_predict, predict_data_x_new[:, 1:]), axis=1)
inv_yhat_predict_new = scaler.inverse_transform(inv_yhat_predict_new)
inv_yhat_predict_new = inv_yhat_predict_new[:,0].round(2)
# inv_yhat_predict_new = round(inv_yhat_predict_new,2)
print(time_predict_data)
print("%.2f" %inv_yhat_predict_new[0])

row_contents = [time_predict_data,inv_yhat_predict_new[0]]

from csv import writer
def append_list_as_row(file_name, list_of_elem):
    # Open file in append mode
    with open(file_name, 'a+', newline='') as write_obj:
        # Create a writer object from csv module
        csv_writer = writer(write_obj)
        # Add contents of list as last row in the csv file
        csv_writer.writerow(list_of_elem)

# Append a list as new line to an old csv file
append_list_as_row('result/btc-1m.csv', row_contents)

from deta import Deta

# 2) initialize with a project key
deta = Deta("c0fws2vv_pM7b7NMZEYyaCpSJBNv3W1tsmpGSVFYX")

# 3) create and use as many DBs as you want!
post = deta.Base("btc1m_gru_db")

post.insert({
    "close_predict": str(inv_yhat_predict_new[0]),
    "time": str(time_predict_data)
})
```

btc-5m.py

```
# %%
import datetime
import pytz

from urllib.request import urlopen
import json
import pandas as pd
import pandas_ta as ta
import time
import datetime
import math

currency = "BTCBUSD"
```

```
interval = "5m"
len_data = "1000"

def unix2utc(timestamp):
    your_dt = datetime.datetime.fromtimestamp(int(timestamp)/1000)  # using the local timezone
    utc_dt= your_dt.astimezone(pytz.UTC)
    return(utc_dt.strftime("%Y-%m-%d %H:%M:%S"))  # 2018-04-07 20:48:08, YMMV

def getDataCrypto(symbol,interval,limit,timeStart,timeEnd):

    # store the URL in url as
    # parameter for urlopen
    if((timeStart and timeEnd) is None):
        url = "https://api.binance.me/api/v3/klines?symbol="+str(symbol)+"&interval="+str(interval)+"&limit="+limit
    else:
        url =
"https://api.binance.me/api/v3/klines?symbol="+str(symbol)+"&interval="+str(interval)+"&limit="+limit+"&startTime="+s
tr(timeStart)+"&endTime="+str(timeEnd)

    response = urlopen(url)

    # storing the JSON response
    # from url in data
    data_json = json.loads(response.read())

    # print the json response
#     print(data_json)
    tm,op,hi,lo,cl,v = [],[],[],[],[],[]

    for x in range(len(data_json)):
    #     print(data_json[x][0])
        tm.append(data_json[x][0])
        op.append(data_json[x][1])
        hi.append(data_json[x][2])
        lo.append(data_json[x][3])
        cl.append(data_json[x][4])
        v.append(data_json[x][5])
    data = {"time": tm, "open":op,"high":hi,"low":lo,"close":cl,"volume":v}
    df = pd.DataFrame(data)
    df['time'] = df['time'].transform(lambda x: unix2utc(x))
    df["open"] = pd.to_numeric(df["open"], downcast="float")
    df["high"] = pd.to_numeric(df["high"], downcast="float")
    df["low"] = pd.to_numeric(df["low"], downcast="float")
    df["close"] = pd.to_numeric(df["close"], downcast="float")
    df["volume"] = pd.to_numeric(df["volume"], downcast="float")
    return df


df  = getDataCrypto(currency,interval,len_data,None,None)
# df.drop(columns=df.columns[0], axis=1,inplace=True)

# Calculate Indicator Technical
import pandas_ta as ta

sma5 = ta.sma(df['close'], length=5)
sma10 = ta.sma(df['close'], length=10)
sma20 = ta.sma(df['close'], length=20)
rsi = ta.rsi(df['close'].astype(float), length = 10)
macd = ta.macd(df['close'].astype(float), length = 10)
bbands = ta.bbands(df['close'].astype(float), length = 10)



df['ma5']=sma5
df['ma10'] = sma10
df['ma20'] = sma20
df['rsi']=rsi
df['bb_upper'] = bbands['BBL_10_2.0']
df['bb_lower'] = bbands['BBU_10_2.0']
df['bb_middle'] = bbands['BBM_10_2.0']
df['macd_l'] = macd['MACD_12_26_9']
df['macd_s'] = macd['MACDs_12_26_9']

df = df.dropna()



df = df[['time','close','volume','ma5','ma10','ma20','rsi','bb_upper','bb_lower','macd_l','macd_s']]

from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
```

```
from keras.models import load_model
from keras.layers import Dense
from keras.layers import LSTM

import pandas as pd

# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg

dataset = df
dataset = dataset.set_index('time')
# dataset = dataset.drop(columns=['ma5','ma10','ma20','bb_upper','bb_lower','macd_l','macd_s'])

# %%
from datetime import datetime
from datetime import timedelta
predict_data = dataset.tail(1)
time_predict_data = predict_data.index.values[0]
if (interval=="1m"):
    interval_time = 1
elif (interval == "5m"):
    interval_time = 5
elif (interval == "15m"):
    interval_time = 15
time_predict_data = pd.to_datetime(time_predict_data)+timedelta(minutes=interval_time)

predict_data = predict_data.values
predict_data = predict_data.astype('float32')

values = dataset.values
values = values.astype('float32')
# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
scaled_predict = scaler.transform(predict_data)
# frame as supervised learning
reframed = series_to_supervised(scaled, 1, 1)
# drop columns we don't want to predict
# reframed.drop(reframed.columns[[9,10,11,12,13,14,15]], axis=1, inplace=True)
reframed = reframed.iloc[:, 0:(len(dataset.columns)+1)]
scaled_predict = DataFrame(scaled_predict)
scaled_predict.columns = reframed.columns[range(0,(len(dataset.columns)))]

# split into train and test sets
values = reframed.values
n_train_hours = 871
train = values[:n_train_hours, :]
test = values[n_train_hours:, :]
# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]
# predict_test_X, predict_test_y = predict_test[:, :-1], predict_test[:, -1]

# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
# predict_test_X = predict_test_X.reshape((predict_test_X.shape[0], 1, predict_test_X.shape[1]))
values_predict = scaled_predict.values
predict_data_x = values_predict.reshape((values_predict.shape[0],1,values_predict.shape[1]))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
print(predict_data_x.shape)

# %%
# design network
from keras.layers import GRU
model = Sequential()
model.add(GRU(25,input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
```

```
model.compile(loss='mae', optimizer='adam')
history = model.fit(train_X, train_y, epochs=100, batch_size=32, validation_data=(test_X, test_y), verbose=1,
shuffle=False)



yhat_predict = model.predict(predict_data_x)
# predict_data_x
predict_data_x_new= predict_data_x.reshape((predict_data_x.shape[0], predict_data_x.shape[2]))

inv_yhat_predict_new = concatenate((yhat_predict, predict_data_x_new[:, 1:]), axis=1)
inv_yhat_predict_new = scaler.inverse_transform(inv_yhat_predict_new)
inv_yhat_predict_new = inv_yhat_predict_new[:,0].round(2)
# inv_yhat_predict_new = round(inv_yhat_predict_new,2)
print(time_predict_data)
print("%.2f" %inv_yhat_predict_new[0])

row_contents = [time_predict_data,inv_yhat_predict_new[0]]

from csv import writer
def append_list_as_row(file_name, list_of_elem):
    # Open file in append mode
    with open(file_name, 'a+', newline='') as write_obj:
        # Create a writer object from csv module
        csv_writer = writer(write_obj)
        # Add contents of list as last row in the csv file
        csv_writer.writerow(list_of_elem)

# Append a list as new line to an old csv file
append_list_as_row('result/btc-5m.csv', row_contents)

from deta import Deta

# 2) initialize with a project key
deta = Deta("c0fws2vv_pM7b7NMZEYyaCpSJBNv3W1tsmpGSVFYX")

# 3) create and use as many DBs as you want!
post = deta.Base("btc5m_gru_db")

post.insert({
    "close_predict": str(inv_yhat_predict_new[0]),
    "time": str(time_predict_data)
})
```

btc-15m.py

```
# %%
import datetime
import pytz

from urllib.request import urlopen
import json
import pandas as pd
import pandas_ta as ta
import time
import datetime
import math

currency = "BTCBUSD"
interval = "15m"
len_data = "1000"

def unix2utc(timestamp):
    your_dt = datetime.datetime.fromtimestamp(int(timestamp)/1000)  # using the local timezone
    utc_dt= your_dt.astimezone(pytz.UTC)
    return(utc_dt.strftime("%Y-%m-%d %H:%M:%S"))  # 2018-04-07 20:48:08, YMMV

def getDataCrypto(symbol,interval,limit,timeStart,timeEnd):

    # store the URL in url as
    # parameter for urlopen
    if((timeStart and timeEnd) is None):
        url = "https://api.binance.me/api/v3/klines?symbol="+str(symbol)+"&interval="+str(interval)+"&limit="+limit
    else:
        url =
"https://api.binance.me/api/v3/klines?symbol="+str(symbol)+"&interval="+str(interval)+"&limit="+limit+"&startTime="+s
tr(timeStart)+"&endTime="+str(timeEnd)

    response = urlopen(url)

    # storing the JSON response
    # from url in data
    data_json = json.loads(response.read())
```

```
    # print the json response
#     print(data_json)
    tm,op,hi,lo,cl,v = [],[],[],[],[],[]

    for x in range(len(data_json)):
    #     print(data_json[x][0])
        tm.append(data_json[x][0])
        op.append(data_json[x][1])
        hi.append(data_json[x][2])
        lo.append(data_json[x][3])
        cl.append(data_json[x][4])
        v.append(data_json[x][5])
    data = {"time": tm, "open":op,"high":hi,"low":lo,"close":cl,"volume":v}
    df = pd.DataFrame(data)
    df['time'] = df['time'].transform(lambda x: unix2utc(x))
    df["open"] = pd.to_numeric(df["open"], downcast="float")
    df["high"] = pd.to_numeric(df["high"], downcast="float")
    df["low"] = pd.to_numeric(df["low"], downcast="float")
    df["close"] = pd.to_numeric(df["close"], downcast="float")
    df["volume"] = pd.to_numeric(df["volume"], downcast="float")
    return df


df  = getDataCrypto(currency,interval,len_data,None,None)
# df.drop(columns=df.columns[0], axis=1,inplace=True)

# Calculate Indicator Technical
import pandas_ta as ta

sma5 = ta.sma(df['close'], length=5)
sma10 = ta.sma(df['close'], length=10)
sma20 = ta.sma(df['close'], length=20)
rsi = ta.rsi(df['close'].astype(float), length = 10)
macd = ta.macd(df['close'].astype(float), length = 10)
bbands = ta.bbands(df['close'].astype(float), length = 10)



df['ma5']=sma5
df['ma10'] = sma10
df['ma20'] = sma20
df['rsi']=rsi
df['bb_upper'] = bbands['BBL_10_2.0']
df['bb_lower'] = bbands['BBU_10_2.0']
df['bb_middle'] = bbands['BBM_10_2.0']
df['macd_l'] = macd['MACD_12_26_9']
df['macd_s'] = macd['MACDs_12_26_9']



df = df.dropna()



df = df[['time','close','volume','ma5','ma10','ma20','rsi','bb_upper','bb_lower','macd_l','macd_s']]

from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.models import load_model
from keras.layers import Dense
from keras.layers import LSTM

import pandas as pd

# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
```

```
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg

dataset = df
dataset = dataset.set_index('time')
# dataset = dataset.drop(columns=['ma5','ma10','ma20','bb_upper','bb_lower','macd_l','macd_s'])

# %%
from datetime import datetime
from datetime import timedelta
predict_data = dataset.tail(1)
time_predict_data = predict_data.index.values[0]
if (interval=="1m"):
    interval_time = 1
elif (interval == "5m"):
    interval_time = 5
elif (interval == "15m"):
    interval_time = 15
time_predict_data = pd.to_datetime(time_predict_data)+timedelta(minutes=interval_time)

predict_data = predict_data.values
predict_data = predict_data.astype('float32')

values = dataset.values
values = values.astype('float32')
# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
scaled_predict = scaler.transform(predict_data)
# frame as supervised learning
reframed = series_to_supervised(scaled, 1, 1)
# drop columns we don't want to predict
# reframed.drop(reframed.columns[[9,10,11,12,13,14,15]], axis=1, inplace=True)
reframed = reframed.iloc[:, 0:(len(dataset.columns)+1)]
scaled_predict = DataFrame(scaled_predict)
scaled_predict.columns = reframed.columns[range(0,(len(dataset.columns)))]

# split into train and test sets
values = reframed.values
n_train_hours = 871
train = values[:n_train_hours, :]
test = values[n_train_hours:, :]
# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]
# predict_test_X, predict_test_y = predict_test[:, :-1], predict_test[:, -1]

# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
# predict_test_X = predict_test_X.reshape((predict_test_X.shape[0], 1, predict_test_X.shape[1]))
values_predict = scaled_predict.values
predict_data_x = values_predict.reshape((values_predict.shape[0],1,values_predict.shape[1]))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
print(predict_data_x.shape)

# %%
# design network
from keras.layers import GRU
model = Sequential()
model.add(GRU(25,input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')
history = model.fit(train_X, train_y, epochs=100, batch_size=32, validation_data=(test_X, test_y), verbose=1,
shuffle=False)



yhat_predict = model.predict(predict_data_x)
# predict_data_x
predict_data_x_new= predict_data_x.reshape((predict_data_x.shape[0], predict_data_x.shape[2]))

inv_yhat_predict_new = concatenate((yhat_predict, predict_data_x_new[:, 1:]), axis=1)
inv_yhat_predict_new = scaler.inverse_transform(inv_yhat_predict_new)
inv_yhat_predict_new = inv_yhat_predict_new[:,0].round(2)
# inv_yhat_predict_new = round(inv_yhat_predict_new,2)
print(time_predict_data)
print("%.2f" %inv_yhat_predict_new[0])

row_contents = [time_predict_data,inv_yhat_predict_new[0]]

from csv import writer
def append_list_as_row(file_name, list_of_elem):
    # Open file in append mode
    with open(file_name, 'a+', newline='') as write_obj:
        # Create a writer object from csv module
        csv_writer = writer(write_obj)
```

```
        # Add contents of list as last row in the csv file
        csv_writer.writerow(list_of_elem)

# Append a list as new line to an old csv file
append_list_as_row('result/btc-15m.csv', row_contents)

from deta import Deta

# 2) initialize with a project key
deta = Deta("c0fws2vv_pM7b7NMZEYyaCpSJBNv3W1tsmpGSVFYX")

# 3) create and use as many DBs as you want!
post = deta.Base("btc15m_gru_db")

post.insert({
    "close_predict": str(inv_yhat_predict_new[0]),
    "time": str(time_predict_data)
}
```

Lampiran 2 *Source code* model prediksi menggunakan GRU

btc-1m.py

```
# %%
import datetime
import pytz

from urllib.request import urlopen
import json
import pandas as pd
import pandas_ta as ta
import time
import datetime
import math

currency = "BTCBUSD"
interval = "1m"
len_data = "1000"

def unix2utc(timestamp):
    your_dt = datetime.datetime.fromtimestamp(int(timestamp)/1000)  # using the local timezone
    utc_dt= your_dt.astimezone(pytz.UTC)
    return(utc_dt.strftime("%Y-%m-%d %H:%M:%S"))  # 2018-04-07 20:48:08, YMMV

def getDataCrypto(symbol,interval,limit,timeStart,timeEnd):

    # store the URL in url as
    # parameter for urlopen
    if((timeStart and timeEnd) is None):
        url = "https://api.binance.me/api/v3/klines?symbol="+str(symbol)+"&interval="+str(interval)+"&limit="+limit
    else:
        url =
"https://api.binance.me/api/v3/klines?symbol="+str(symbol)+"&interval="+str(interval)+"&limit="+limit+"&startTime="+s
tr(timeStart)+"&endTime="+str(timeEnd)

    response = urlopen(url)

    # storing the JSON response
    # from url in data
    data_json = json.loads(response.read())

    # print the json response
#     print(data_json)
    tm,op,hi,lo,cl,v = [],[],[],[],[],[]

    for x in range(len(data_json)):
    #     print(data_json[x][0])
        tm.append(data_json[x][0])
        op.append(data_json[x][1])
        hi.append(data_json[x][2])
        lo.append(data_json[x][3])
        cl.append(data_json[x][4])
        v.append(data_json[x][5])
    data = {"time": tm, "open":op,"high":hi,"low":lo,"close":cl,"volume":v}
    df = pd.DataFrame(data)
    df['time'] = df['time'].transform(lambda x: unix2utc(x))
    df["open"] = pd.to_numeric(df["open"], downcast="float")
    df["high"] = pd.to_numeric(df["high"], downcast="float")
    df["low"] = pd.to_numeric(df["low"], downcast="float")
```

```
    df["close"] = pd.to_numeric(df["close"], downcast="float")
    df["volume"] = pd.to_numeric(df["volume"], downcast="float")
    return df



df  = getDataCrypto(currency,interval,len_data,None,None)
# df.drop(columns=df.columns[0], axis=1,inplace=True)

# Calculate Indicator Technical
import pandas_ta as ta

sma5 = ta.sma(df['close'], length=5)
sma10 = ta.sma(df['close'], length=10)
sma20 = ta.sma(df['close'], length=20)
rsi = ta.rsi(df['close'].astype(float), length = 10)
macd = ta.macd(df['close'].astype(float), length = 10)
bbands = ta.bbands(df['close'].astype(float), length = 10)



df['ma5']=sma5
df['ma10'] = sma10
df['ma20'] = sma20
df['rsi']=rsi
df['bb_upper'] = bbands['BBL_10_2.0']
df['bb_lower'] = bbands['BBU_10_2.0']
df['bb_middle'] = bbands['BBM_10_2.0']
df['macd_l'] = macd['MACD_12_26_9']
df['macd_s'] = macd['MACDs_12_26_9']



df = df.dropna()



df = df[['time','close','volume','ma5','ma10','ma20','rsi','bb_upper','bb_lower','macd_l','macd_s']]

from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.models import load_model
from keras.layers import Dense
from keras.layers import LSTM

import pandas as pd

# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg

dataset = df
dataset = dataset.set_index('time')
# df = df[['time','close','volume','ma5','ma10','ma20','rsi','bb_upper','bb_lower','macd_l','macd_s']]
# dataset = dataset.drop(columns=['ma5','ma10','ma20','bb_upper','bb_lower','macd_l','macd_s'])

# %%
from datetime import datetime
from datetime import timedelta
predict_data = dataset.tail(1)
time_predict_data = predict_data.index.values[0]
if (interval=="1m"):
    interval_time = 1
```

```
elif (interval == "5m"):
    interval_time = 5
elif (interval == "15m"):
    interval_time = 15
time_predict_data = pd.to_datetime(time_predict_data)+timedelta(minutes=interval_time)

predict_data = predict_data.values
predict_data = predict_data.astype('float32')

values = dataset.values
values = values.astype('float32')
# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
scaled_predict = scaler.transform(predict_data)
# frame as supervised learning
reframed = series_to_supervised(scaled, 1, 1)
# drop columns we don't want to predict
# reframed.drop(reframed.columns[[9,10,11,12,13,14,15]], axis=1, inplace=True)
reframed = reframed.iloc[:, 0:(len(dataset.columns)+1)]
scaled_predict = DataFrame(scaled_predict)
scaled_predict.columns = reframed.columns[range(0,(len(dataset.columns)))]

# split into train and test sets
values = reframed.values
n_train_hours = 871
train = values[:n_train_hours, :]
test = values[n_train_hours:, :]
# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]
# predict_test_X, predict_test_y = predict_test[:, :-1], predict_test[:, -1]

# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
# predict_test_X = predict_test_X.reshape((predict_test_X.shape[0], 1, predict_test_X.shape[1]))
values_predict = scaled_predict.values
predict_data_x = values_predict.reshape((values_predict.shape[0],1,values_predict.shape[1]))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
print(predict_data_x.shape)

# %%
# design network
from keras.layers import GRU
model = Sequential()
model.add(GRU(25,input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')
history = model.fit(train_X, train_y, epochs=100, batch_size=32, validation_data=(test_X, test_y), verbose=1,
shuffle=False)



yhat_predict = model.predict(predict_data_x)
# predict_data_x
predict_data_x_new= predict_data_x.reshape((predict_data_x.shape[0], predict_data_x.shape[2]))

inv_yhat_predict_new = concatenate((yhat_predict, predict_data_x_new[:, 1:]), axis=1)
inv_yhat_predict_new = scaler.inverse_transform(inv_yhat_predict_new)
inv_yhat_predict_new = inv_yhat_predict_new[:,0].round(2)
# inv_yhat_predict_new = round(inv_yhat_predict_new,2)
print(time_predict_data)
print("%.2f" %inv_yhat_predict_new[0])

row_contents = [time_predict_data,inv_yhat_predict_new[0]]

from csv import writer
def append_list_as_row(file_name, list_of_elem):
    # Open file in append mode
    with open(file_name, 'a+', newline='') as write_obj:
        # Create a writer object from csv module
        csv_writer = writer(write_obj)
        # Add contents of list as last row in the csv file
        csv_writer.writerow(list_of_elem)

# Append a list as new line to an old csv file
append_list_as_row('result/btc-1m.csv', row_contents)

from deta import Deta

# 2) initialize with a project key
deta = Deta("c0fws2vv_pM7b7NMZEYyaCpSJBNv3W1tsmpGSVFYX")

# 3) create and use as many DBs as you want!
post = deta.Base("btc1m_gru_db")

post.insert({
    "close_predict": str(inv_yhat_predict_new[0]),
    "time": str(time_predict_data)
```

```
})
```

btc-5m.py

```
# %%
import datetime
import pytz

from urllib.request import urlopen
import json
import pandas as pd
import pandas_ta as ta
import time
import datetime
import math

currency = "BTCBUSD"
interval = "5m"
len_data = "1000"

def unix2utc(timestamp):
    your_dt = datetime.datetime.fromtimestamp(int(timestamp)/1000)  # using the local timezone
    utc_dt= your_dt.astimezone(pytz.UTC)
    return(utc_dt.strftime("%Y-%m-%d %H:%M:%S"))  # 2018-04-07 20:48:08, YMMV

def getDataCrypto(symbol,interval,limit,timeStart,timeEnd):

    # store the URL in url as
    # parameter for urlopen
    if((timeStart and timeEnd) is None):
        url = "https://api.binance.me/api/v3/klines?symbol="+str(symbol)+"&interval="+str(interval)+"&limit="+limit
    else:
        url =
"https://api.binance.me/api/v3/klines?symbol="+str(symbol)+"&interval="+str(interval)+"&limit="+limit+"&startTime="+s
tr(timeStart)+"&endTime="+str(timeEnd)

    response = urlopen(url)

    # storing the JSON response
    # from url in data
    data_json = json.loads(response.read())

    # print the json response
#     print(data_json)
    tm,op,hi,lo,cl,v = [],[],[],[],[],[]

    for x in range(len(data_json)):
    #     print(data_json[x][0])
        tm.append(data_json[x][0])
        op.append(data_json[x][1])
        hi.append(data_json[x][2])
        lo.append(data_json[x][3])
        cl.append(data_json[x][4])
        v.append(data_json[x][5])
    data = {"time": tm, "open":op,"high":hi,"low":lo,"close":cl,"volume":v}
    df = pd.DataFrame(data)
    df['time'] = df['time'].transform(lambda x: unix2utc(x))
    df["open"] = pd.to_numeric(df["open"], downcast="float")
    df["high"] = pd.to_numeric(df["high"], downcast="float")
    df["low"] = pd.to_numeric(df["low"], downcast="float")
    df["close"] = pd.to_numeric(df["close"], downcast="float")
    df["volume"] = pd.to_numeric(df["volume"], downcast="float")
    return df


df  = getDataCrypto(currency,interval,len_data,None,None)
# df.drop(columns=df.columns[0], axis=1,inplace=True)

# Calculate Indicator Technical
import pandas_ta as ta

sma5 = ta.sma(df['close'], length=5)
sma10 = ta.sma(df['close'], length=10)
sma20 = ta.sma(df['close'], length=20)
rsi = ta.rsi(df['close'].astype(float), length = 10)
macd = ta.macd(df['close'].astype(float), length = 10)
bbands = ta.bbands(df['close'].astype(float), length = 10)



df['ma5']=sma5
df['ma10'] = sma10
df['ma20'] = sma20
df['rsi']=rsi
df['bb_upper'] = bbands['BBL_10_2.0']
```

```
df['bb_lower'] = bbands['BBU_10_2.0']
df['bb_middle'] = bbands['BBM_10_2.0']
df['macd_l'] = macd['MACD_12_26_9']
df['macd_s'] = macd['MACDs_12_26_9']

df = df.dropna()


df = df[['time','close','volume','ma5','ma10','ma20','rsi','bb_upper','bb_lower','macd_l','macd_s']]

from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.models import load_model
from keras.layers import Dense
from keras.layers import LSTM

import pandas as pd

# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg

dataset = df
dataset = dataset.set_index('time')
# dataset = dataset.drop(columns=['ma5','ma10','ma20','bb_upper','bb_lower','macd_l','macd_s'])

# %%
from datetime import datetime
from datetime import timedelta
predict_data = dataset.tail(1)
time_predict_data = predict_data.index.values[0]
if (interval=="1m"):
    interval_time = 1
elif (interval == "5m"):
    interval_time = 5
elif (interval == "15m"):
    interval_time = 15
time_predict_data = pd.to_datetime(time_predict_data)+timedelta(minutes=interval_time)

predict_data = predict_data.values
predict_data = predict_data.astype('float32')

values = dataset.values
values = values.astype('float32')
# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
scaled_predict = scaler.transform(predict_data)
# frame as supervised learning
reframed = series_to_supervised(scaled, 1, 1)
# drop columns we don't want to predict
# reframed.drop(reframed.columns[[9,10,11,12,13,14,15]], axis=1, inplace=True)
reframed = reframed.iloc[:, 0:(len(dataset.columns)+1)]
scaled_predict = DataFrame(scaled_predict)
scaled_predict.columns = reframed.columns[range(0,(len(dataset.columns)))]

# split into train and test sets
values = reframed.values
n_train_hours = 871
train = values[:n_train_hours, :]
test = values[n_train_hours:, :]
```

```
# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]
# predict_test_X, predict_test_y = predict_test[:, :-1], predict_test[:, -1]

# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
# predict_test_X = predict_test_X.reshape((predict_test_X.shape[0], 1, predict_test_X.shape[1]))
values_predict = scaled_predict.values
predict_data_x = values_predict.reshape((values_predict.shape[0],1,values_predict.shape[1]))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
print(predict_data_x.shape)

# %%
# design network
from keras.layers import GRU
model = Sequential()
model.add(GRU(25,input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')
history = model.fit(train_X, train_y, epochs=100, batch_size=32, validation_data=(test_X, test_y), verbose=1,
shuffle=False)



yhat_predict = model.predict(predict_data_x)
# predict_data_x
predict_data_x_new= predict_data_x.reshape((predict_data_x.shape[0], predict_data_x.shape[2]))

inv_yhat_predict_new = concatenate((yhat_predict, predict_data_x_new[:, 1:]), axis=1)
inv_yhat_predict_new = scaler.inverse_transform(inv_yhat_predict_new)
inv_yhat_predict_new = inv_yhat_predict_new[:,0].round(2)
# inv_yhat_predict_new = round(inv_yhat_predict_new,2)
print(time_predict_data)
print("%.2f" %inv_yhat_predict_new[0])

row_contents = [time_predict_data,inv_yhat_predict_new[0]]

from csv import writer
def append_list_as_row(file_name, list_of_elem):
    # Open file in append mode
    with open(file_name, 'a+', newline='') as write_obj:
        # Create a writer object from csv module
        csv_writer = writer(write_obj)
        # Add contents of list as last row in the csv file
        csv_writer.writerow(list_of_elem)

# Append a list as new line to an old csv file
append_list_as_row('result/btc-5m.csv', row_contents)

from deta import Deta

# 2) initialize with a project key
deta = Deta("c0fws2vv_pM7b7NMZEYyaCpSJBNv3W1tsmpGSVFYX")

# 3) create and use as many DBs as you want!
post = deta.Base("btc5m_gru_db")

post.insert({
    "close_predict": str(inv_yhat_predict_new[0]),
    "time": str(time_predict_data)
})
```

btc-15m.py

```
# %%
import datetime
import pytz

from urllib.request import urlopen
import json
import pandas as pd
import pandas_ta as ta
import time
import datetime
import math

currency = "BTCBUSD"
interval = "15m"
len_data = "1000"

def unix2utc(timestamp):
    your_dt = datetime.datetime.fromtimestamp(int(timestamp)/1000)  # using the local timezone
    utc_dt= your_dt.astimezone(pytz.UTC)
    return(utc_dt.strftime("%Y-%m-%d %H:%M:%S"))  # 2018-04-07 20:48:08, YMMV
```

```
def getDataCrypto(symbol,interval,limit,timeStart,timeEnd):

    # store the URL in url as
    # parameter for urlopen
    if((timeStart and timeEnd) is None):
        url = "https://api.binance.me/api/v3/klines?symbol="+str(symbol)+"&interval="+str(interval)+"&limit="+limit
    else:
        url =
"https://api.binance.me/api/v3/klines?symbol="+str(symbol)+"&interval="+str(interval)+"&limit="+limit+"&startTime="+s
tr(timeStart)+"&endTime="+str(timeEnd)

    response = urlopen(url)

    # storing the JSON response
    # from url in data
    data_json = json.loads(response.read())

    # print the json response
#     print(data_json)
    tm,op,hi,lo,cl,v = [],[],[],[],[],[]

    for x in range(len(data_json)):
    #     print(data_json[x][0])
        tm.append(data_json[x][0])
        op.append(data_json[x][1])
        hi.append(data_json[x][2])
        lo.append(data_json[x][3])
        cl.append(data_json[x][4])
        v.append(data_json[x][5])
    data = {"time": tm, "open":op,"high":hi,"low":lo,"close":cl,"volume":v}
    df = pd.DataFrame(data)
    df['time'] = df['time'].transform(lambda x: unix2utc(x))
    df["open"] = pd.to_numeric(df["open"], downcast="float")
    df["high"] = pd.to_numeric(df["high"], downcast="float")
    df["low"] = pd.to_numeric(df["low"], downcast="float")
    df["close"] = pd.to_numeric(df["close"], downcast="float")
    df["volume"] = pd.to_numeric(df["volume"], downcast="float")
    return df


df  = getDataCrypto(currency,interval,len_data,None,None)
# df.drop(columns=df.columns[0], axis=1,inplace=True)

# Calculate Indicator Technical
import pandas_ta as ta

sma5 = ta.sma(df['close'], length=5)
sma10 = ta.sma(df['close'], length=10)
sma20 = ta.sma(df['close'], length=20)
rsi = ta.rsi(df['close'].astype(float), length = 10)
macd = ta.macd(df['close'].astype(float), length = 10)
bbands = ta.bbands(df['close'].astype(float), length = 10)



df['ma5']=sma5
df['ma10'] = sma10
df['ma20'] = sma20
df['rsi']=rsi
df['bb_upper'] = bbands['BBL_10_2.0']
df['bb_lower'] = bbands['BBU_10_2.0']
df['bb_middle'] = bbands['BBM_10_2.0']
df['macd_l'] = macd['MACD_12_26_9']
df['macd_s'] = macd['MACDs_12_26_9']



df = df.dropna()



df = df[['time','close','volume','ma5','ma10','ma20','rsi','bb_upper','bb_lower','macd_l','macd_s']]

from math import sqrt
from numpy import concatenate
from matplotlib import pyplot
from pandas import read_csv
from pandas import DataFrame
from pandas import concat
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.models import load_model
from keras.layers import Dense
from keras.layers import LSTM

import pandas as pd
```

```
# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg

dataset = df
dataset = dataset.set_index('time')
# dataset = dataset.drop(columns=['ma5','ma10','ma20','bb_upper','bb_lower','macd_l','macd_s'])

# %%
from datetime import datetime
from datetime import timedelta
predict_data = dataset.tail(1)
time_predict_data = predict_data.index.values[0]
if (interval=="1m"):
    interval_time = 1
elif (interval == "5m"):
    interval_time = 5
elif (interval == "15m"):
    interval_time = 15
time_predict_data = pd.to_datetime(time_predict_data)+timedelta(minutes=interval_time)

predict_data = predict_data.values
predict_data = predict_data.astype('float32')

values = dataset.values
values = values.astype('float32')
# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
scaled_predict = scaler.transform(predict_data)
# frame as supervised learning
reframed = series_to_supervised(scaled, 1, 1)
# drop columns we don't want to predict
# reframed.drop(reframed.columns[[9,10,11,12,13,14,15]], axis=1, inplace=True)
reframed = reframed.iloc[:, 0:(len(dataset.columns)+1)]
scaled_predict = DataFrame(scaled_predict)
scaled_predict.columns = reframed.columns[range(0,(len(dataset.columns)))]

# split into train and test sets
values = reframed.values
n_train_hours = 871
train = values[:n_train_hours, :]
test = values[n_train_hours:, :]
# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]
# predict_test_X, predict_test_y = predict_test[:, :-1], predict_test[:, -1]

# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
# predict_test_X = predict_test_X.reshape((predict_test_X.shape[0], 1, predict_test_X.shape[1]))
values_predict = scaled_predict.values
predict_data_x = values_predict.reshape((values_predict.shape[0],1,values_predict.shape[1]))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
print(predict_data_x.shape)

# %%
# design network
from keras.layers import GRU
model = Sequential()
model.add(GRU(25,input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam')
history = model.fit(train_X, train_y, epochs=100, batch_size=32, validation_data=(test_X, test_y), verbose=1,
shuffle=False)



yhat_predict = model.predict(predict_data_x)
```

```
# predict_data_x
predict_data_x_new= predict_data_x.reshape((predict_data_x.shape[0], predict_data_x.shape[2]))

inv_yhat_predict_new = concatenate((yhat_predict, predict_data_x_new[:, 1:]), axis=1)
inv_yhat_predict_new = scaler.inverse_transform(inv_yhat_predict_new)
inv_yhat_predict_new = inv_yhat_predict_new[:,0].round(2)
# inv_yhat_predict_new = round(inv_yhat_predict_new,2)
print(time_predict_data)
print("%.2f" %inv_yhat_predict_new[0])

row_contents = [time_predict_data,inv_yhat_predict_new[0]]

from csv import writer
def append_list_as_row(file_name, list_of_elem):
    # Open file in append mode
    with open(file_name, 'a+', newline='') as write_obj:
        # Create a writer object from csv module
        csv_writer = writer(write_obj)
        # Add contents of list as last row in the csv file
        csv_writer.writerow(list_of_elem)

# Append a list as new line to an old csv file
append_list_as_row('result/btc-15m.csv', row_contents)

from deta import Deta

# 2) initialize with a project key
deta = Deta("c0fws2vv_pM7b7NMZEYyaCpSJBNv3W1tsmpGSVFYX")

# 3) create and use as many DBs as you want!
post = deta.Base("btc15m_gru_db")

post.insert({
    "close_predict": str(inv_yhat_predict_new[0]),
    "time": str(time_predict_data)
})
```

Lampiran 3 *Source code* scheduler model realtime

Scheduler-1m.py

```
import time
import os

from datetime import datetime
import time
# starttime = datetime.now()
# print(starttime)
while True:
    os.system("python bnb-1m.py")
    starttime = datetime.now()
    time.sleep(63.0 - starttime.second)
```

Scheduler-5m.py

```
import time
import os

from datetime import datetime
import time
# starttime = datetime.now()
# print(starttime)
while True:
    os.system("python bnb-5m.py")
    starttime = datetime.now()
    time.sleep((5 - starttime.minute%5)*60)
    time.sleep(3)
```

Scheduler-15m.py

```
import time
import os

from datetime import datetime
import time
# starttime = datetime.now()
# print(starttime)
while True:
    os.system("python bnb-15m.py")
    starttime = datetime.now()
    time.sleep((15 - starttime.minute%15)*60)
    time.sleep(3)
```

Lampiran 4 *Source code* evaluasi model

tester.py

```
# %%
import pandas as pd

# %%
df_1m_day1 = pd.read_csv('bnb/gru/1m/bnb-day1.csv')
df_1m_day2 = pd.read_csv('bnb/gru/1m/bnb-day2.csv')
df_1m_day3 = pd.read_csv('bnb/gru/1m/bnb-day3.csv')
df_1m_day4 = pd.read_csv('bnb/gru/1m/bnb-day4.csv')
df_1m_day5 = pd.read_csv('bnb/gru/1m/bnb-day5.csv')
df_1m_day6 = pd.read_csv('bnb/gru/1m/bnb-day6.csv')
df_1m_day7 = pd.read_csv('bnb/gru/1m/bnb-day7.csv')

df_1m_day1['predict'].round(1)
df_1m_day2['predict'].round(1)
df_1m_day3['predict'].round(1)
df_1m_day4['predict'].round(1)
df_1m_day5['predict'].round(1)
df_1m_day6['predict'].round(1)
df_1m_day7['predict'].round(1)

# %%
#library
import pandas as pd
import time
import datetime
from datetime import timedelta
import math
import pytz
import pandas_ta as ta

# import urllib library
from urllib.request import urlopen
# import json
import json

def utcToUnix(timeData):
    timeData = datetime.datetime.timestamp(timeData)*1000
    timeData = math.trunc(timeData)
#     print(timeData)
    return timeData

def getData(timeStart,timeEnd,coin):
    # store the URL in url as
    # parameter for urlopen
    url =
"https://api.binance.me/api/v3/klines?symbol="+str(coin)+"BUSD&interval=1m&limit=1000&startTime="+str(timeStart)+"&en
dTime="+str(timeEnd)
    # store the response of URL
    response = urlopen(url)

    # storing the JSON response
    # from url in data
    data_json = json.loads(response.read())

    # print the json response
#     print(data_json)
    tm,op,hi,lo,cl,v = [],[],[],[],[],[]

    for x in range(len(data_json)):
    #     print(data_json[x][0])
        tm.append(data_json[x][0])
```

```
        op.append(data_json[x][1])
        hi.append(data_json[x][2])
        lo.append(data_json[x][3])
        cl.append(data_json[x][4])
        v.append(data_json[x][5])
    data = {"time": tm, "open":op,"high":hi,"low":lo,"close":cl,"volume":v}
    df = pd.DataFrame(data)
    df['time'] = df['time'].transform(lambda x: unix2utc(x))
    # df["open"] = pd.to_numeric(df["open"], downcast="float")
    # df["high"] = pd.to_numeric(df["high"], downcast="float")
    # df["low"] = pd.to_numeric(df["low"], downcast="float")
    df["close"] = pd.to_numeric(df["close"], downcast="float")
    # df["volume"] = pd.to_numeric(df["volume"], downcast="float")
    return df

def unix2utc(timestamp):
    your_dt = datetime.datetime.fromtimestamp(int(timestamp)/1000)  # using the local timezone
    utc_dt= your_dt.astimezone(pytz.UTC)
    return(utc_dt.strftime("%Y-%m-%d %H:%M:%S"))

def timeFrameOne(startTime,loop,timeFrame,interval,coin):
    dfall = pd.DataFrame()
    openTime = startTime
    for x in range(loop):
        openTime = openTime + datetime.timedelta(minutes = timeFrame)
        closeTime = openTime + datetime.timedelta(hours = interval)
        df = getData(utcToUnix(openTime),utcToUnix(closeTime),coin)
#         df = pd.concat(df)
#         print(len(df))
        dfall = dfall.append(df)
#         print(len(dfall))
        print(openTime)
        print(closeTime)
        openTime = closeTime
    print(len(dfall))
    json2data = json.loads(dfall.to_json(orient='records'))
    return json2data

day1 = datetime.datetime(2022, 7, 1, 16, 0, 0) + timedelta(hours=8)# format (yyyy, mm, dd, hh, mm, ss)
day2 = datetime.datetime(2022, 7, 2, 16, 0, 0) + timedelta(hours=8)# format (yyyy, mm, dd, hh, mm, ss)
day3 = datetime.datetime(2022, 7, 3, 16, 0, 0) + timedelta(hours=8)# format (yyyy, mm, dd, hh, mm, ss)
day4 = datetime.datetime(2022, 7, 4, 16, 0, 0) + timedelta(hours=8)# format (yyyy, mm, dd, hh, mm, ss)
day5 = datetime.datetime(2022, 7, 5, 16, 0, 0) + timedelta(hours=8)# format (yyyy, mm, dd, hh, mm, ss)
day6 = datetime.datetime(2022, 7, 6, 16, 0, 0) + timedelta(hours=8)# format (yyyy, mm, dd, hh, mm, ss)
day7 = datetime.datetime(2022, 7, 7, 16, 0, 0) + timedelta(hours=8)# format (yyyy, mm, dd, hh, mm, ss)

df_getday1 = timeFrameOne(startTime = day1, loop = 2,interval = 16, timeFrame = 1, coin = "BNB")
df_getday2 = timeFrameOne(startTime = day2, loop = 2,interval = 16, timeFrame = 1, coin = "BNB")
df_getday3 = timeFrameOne(startTime = day3, loop = 2,interval = 16, timeFrame = 1, coin = "BNB")
df_getday4 = timeFrameOne(startTime = day4, loop = 2,interval = 16, timeFrame = 1, coin = "BNB")
df_getday5 = timeFrameOne(startTime = day5, loop = 2,interval = 16, timeFrame = 1, coin = "BNB")
df_getday6 = timeFrameOne(startTime = day6, loop = 2,interval = 16, timeFrame = 1, coin = "BNB")
df_getday7 = timeFrameOne(startTime = day7, loop = 2,interval = 16, timeFrame = 1, coin = "BNB")

data_day1 = pd.DataFrame(df_getday1)
data_day1 = data_day1.drop(columns=['open','high','low','volume'])
# data_day1['time'] = pd.to_datetime(data_day1['time'])

data_day2 = pd.DataFrame(df_getday2)
data_day2 = data_day2.drop(columns=['open','high','low','volume'])
# data_day2['time'] = pd.to_datetime(data_day2['time'])

data_day3 = pd.DataFrame(df_getday3)
data_day3 = data_day3.drop(columns=['open','high','low','volume'])
# data_day3['time'] = pd.to_datetime(data_day3['time'])

data_day4 = pd.DataFrame(df_getday4)
data_day4 = data_day4.drop(columns=['open','high','low','volume'])
# data_day4['time'] = pd.to_datetime(data_day4['time'])

data_day5 = pd.DataFrame(df_getday5)
data_day5 = data_day5.drop(columns=['open','high','low','volume'])
# data_day5['time'] = pd.to_datetime(data_day5['time'])

data_day6 = pd.DataFrame(df_getday6)
data_day6 = data_day6.drop(columns=['open','high','low','volume'])
# data_day5['time'] = pd.to_datetime(data_day5['time'])

data_day7 = pd.DataFrame(df_getday7)
data_day7 = data_day7.drop(columns=['open','high','low','volume'])
# data_day5['time'] = pd.to_datetime(data_day5['time'])

# %%
new_data_day1 = pd.merge(data_day1,df_1m_day1, on='time')
new_data_day1['predict'] = new_data_day1['predict'].transform(lambda x: round(x,1))
new_data_day1['close'] = new_data_day1['close'].transform(lambda x: round(x,1))
new_data_day1['time'] = pd.to_datetime(new_data_day1['time'])

new_data_day2 = pd.merge(data_day2,df_1m_day2, on='time')
```

```
new_data_day2['predict'] = new_data_day2['predict'].transform(lambda x: round(x,1))
new_data_day2['close'] = new_data_day2['close'].transform(lambda x: round(x,1))
new_data_day2['time'] = pd.to_datetime(new_data_day2['time'])

new_data_day3 = pd.merge(data_day3,df_1m_day3, on='time')
new_data_day3['predict'] = new_data_day3['predict'].transform(lambda x: round(x,1))
new_data_day3['close'] = new_data_day3['close'].transform(lambda x: round(x,1))
new_data_day3['time'] = pd.to_datetime(new_data_day3['time'])

new_data_day4 = pd.merge(data_day4,df_1m_day4, on='time')
new_data_day4['predict'] = new_data_day4['predict'].transform(lambda x: round(x,1))
new_data_day4['close'] = new_data_day4['close'].transform(lambda x: round(x,1))
new_data_day4['time'] = pd.to_datetime(new_data_day4['time'])

new_data_day5 = pd.merge(data_day5,df_1m_day5, on='time')
new_data_day5['predict'] = new_data_day5['predict'].transform(lambda x: round(x,1))
new_data_day5['close'] = new_data_day5['close'].transform(lambda x: round(x,1))
new_data_day5['time'] = pd.to_datetime(new_data_day5['time'])

new_data_day6 = pd.merge(data_day6,df_1m_day6, on='time')
new_data_day6['predict'] = new_data_day6['predict'].transform(lambda x: round(x,1))
new_data_day6['close'] = new_data_day6['close'].transform(lambda x: round(x,1))
new_data_day6['time'] = pd.to_datetime(new_data_day6['time'])

new_data_day7 = pd.merge(data_day7,df_1m_day7, on='time')
new_data_day7['predict'] = new_data_day7['predict'].transform(lambda x: round(x,1))
new_data_day7['close'] = new_data_day7['close'].transform(lambda x: round(x,1))
new_data_day7['time'] = pd.to_datetime(new_data_day7['time'])

frames = [new_data_day1,new_data_day2,new_data_day3,new_data_day4,new_data_day5,new_data_day6,new_data_day7]
all_data = pd.concat(frames)

# %%
all_data

# %%
import numpy as np
from sklearn.metrics import mean_squared_error
from math import sqrt

def mape(actual, pred):
    actual, pred = np.array(actual), np.array(pred)
    return np.mean(np.abs((actual - pred) / actual)) * 100

rmse_day1 = sqrt(mean_squared_error(new_data_day1['close'], new_data_day1['predict']))
mape_day1 = mape(new_data_day1['predict'], new_data_day1['close'])



rmse_day2 = sqrt(mean_squared_error(new_data_day2['close'], new_data_day2['predict']))
mape_day2 = mape(new_data_day2[close], new_data_day2[predict])

rmse_day3 = sqrt(mean_squared_error(new_data_day3['close'], new_data_day3['predict’]))
mape_day3 = mape(new_data_day3['close'], new_data_day3['predict’])

rmse_day4 = sqrt(mean_squared_error(new_data_day4['close'], new_data_day4['predict']))
mape_day4 = mape(new_data_day4['close'], new_data_day4['predict']

rmse_day5 = sqrt(mean_squared_error(new_data_day5['predict'], new_data_day5['close']))
mape_day5 = mape(new_data_day5['close'], new_data_day5['predict'])

rmse_day6 = sqrt(mean_squared_error(new_data_day6['close'], new_data_day6['predict']))
mape_day6 = mape(new_data_day6['close'], new_data_day6['predict'])

rmse_day7 = sqrt(mean_squared_error(new_data_day7['close'], new_data_day7['predict']))
mape_day7 = mape(new_data_day7[‘close'], new_data_day7['predict'])

rmse_all = sqrt(mean_squared_error(all_data['close'], all_data['predict']))
mape_all = mape(all_data['close'], all_data['predict'])

# %%
import matplotlib.pyplot as plt



plt.figure(figsize=(10, 5))
# plt.figure()
plt.title("[GRU] BNB timeframe 1m day 1\nRMSE : %.3f\nMAPE : %.3f" %(rmse_day1 ,mape_day1))
plt.plot(new_data_day1['time'],new_data_day1['close'],label = 'actual')
plt.plot(new_data_day1['time'],new_data_day1['predict'],label = 'predict')
plt.legend()

plt.figure(figsize=(10, 5))
# plt.figure()
plt.title("[GRU] BNB timeframe 1m day 2\nRMSE : %.3f\nMAPE : %.3f" %(rmse_day2 ,mape_day2))
plt.plot(new_data_day2['time'],new_data_day2['close'],label = 'actual')
plt.plot(new_data_day2['time'],new_data_day2['predict'],label = 'predict')
plt.legend()

plt.figure(figsize=(10, 5))
```

```
# plt.figure()
plt.title("[GRU] BNB timeframe 1m day 3\nRMSE : %.3f\nMAPE : %.3f" %(rmse_day3 ,mape_day3))
plt.plot(new_data_day3['time'],new_data_day3['close'],label = 'actual')
plt.plot(new_data_day3['time'],new_data_day3['predict'],label = 'predict')
plt.legend()

plt.figure(figsize=(10, 5))
# plt.figure()
plt.title("[GRU] BNB timeframe 1m day 4\nRMSE : %.3f\nMAPE : %.3f" %(rmse_day4 ,mape_day4))
plt.plot(new_data_day4['time'],new_data_day4['close'],label = 'actual')
plt.plot(new_data_day4['time'],new_data_day4['predict'],label = 'predict')
plt.legend()

plt.figure(figsize=(10, 5))
# plt.figure()
plt.title("[GRU] BNB timeframe 1m day 5\nRMSE : %.3f\nMAPE : %.3f" %(rmse_day5 ,mape_day5))
plt.plot(new_data_day5['time'],new_data_day5['close'],label = 'actual')
plt.plot(new_data_day5['time'],new_data_day5['predict'],label = 'predict')
plt.legend()

plt.figure(figsize=(10, 5))
# plt.figure()
plt.title("[GRU] BNB timeframe 1m day 6\nRMSE : %.3f\nMAPE : %.3f" %(rmse_day6 ,mape_day6))
plt.plot(new_data_day6['time'],new_data_day6['close'],label = 'actual')
plt.plot(new_data_day6['time'],new_data_day6['predict'],label = 'predict')
plt.legend()

plt.figure(figsize=(10, 5))
# plt.figure()
plt.title("[GRU] BNB timeframe 1m day 7\nRMSE : %.3f\nMAPE : %.3f" %(rmse_day7 ,mape_day7))
plt.plot(new_data_day7['time'],new_data_day7['close'],label = 'actual')
plt.plot(new_data_day7['time'],new_data_day7['predict'],label = 'predict')
plt.legend()

plt.figure(figsize=(10, 5))
# plt.figure()
plt.title("[GRU] BNB timeframe 1m 7 Days\nRMSE : %.3f\nMAPE : %.3f" %(rmse_all ,mape_all))
plt.plot(all_data['time'],all_data['close'],label = 'actual')
plt.plot(all_data['time'],all_data['predict'],label = 'predict')
plt.legend()

# %%
all_data.to_csv("csv/gru_bnb_1m.csv")

# %%
```

Lampiran 5 Sourcecode Microservice

```
from doctest import Example
from fastapi import FastAPI,Body
from fastapi.middleware.cors import CORSMiddleware
from deta import Deta
import pandas as pd
from typing import Optional
import json
import math
import datetime
from datetime import timedelta
from urllib.request import urlopen
import pandas as pd
import pytz
from cryptomodule import unix2utc,getDataCrypto,utcToUnix

import crypto_module as cm

app = FastAPI()
origins = ["*"]

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

def utcToUnix(timeData):
    timeData = datetime.datetime.timestamp(timeData)*1000
    timeData = math.trunc(timeData)
    return timeData

deta = Deta("c0fws2vv_pM7b7NMZEYyaCpSJBNv3W1tsmpGSVFYX")

def unix2utc(timestamp):
    your_dt = datetime.datetime.fromtimestamp(int(timestamp)/1000)  # using the local timezone
    utc_dt= your_dt.astimezone(pytz.UTC)
```

```
    return(utc_dt.strftime("%Y-%m-%d %H:%M:%S"))  # 2018-04-07 20:48:08, YMMV


@app.get("/")
async def root():
    return {"test" :"OK","docs" :"https://fanfan-api.deta.dev/docs"}

@app.get("/bnb")
async def bnb(timeframe:str,limit:Optional[str] = None):

    tf = "bnb"+ timeframe + "_db"
    tf2 = "bnb"+ timeframe + "_gru_db"
    bnb = deta.Base(tf)
    bnb_gru = deta.Base(tf2)

    if(timeframe == '1m'):
        datetimenow = datetime.datetime.now() - timedelta(hours=2)
    else:
        datetimenow = datetime.datetime.now() - timedelta(hours=20)

    datetimenow1 = datetime.datetime.now()+timedelta(minutes=20)

    if(timeframe == '1m'):
        res = bnb.fetch(query=[{"key?r": [str(utcToUnix(datetimenow)),str(utcToUnix(datetimenow1))]}],limit=1000)
        res_gru = bnb_gru.fetch(query=[{"key?r":
[str(utcToUnix(datetimenow)),str(utcToUnix(datetimenow1))]}],limit=1000)
    else:
        res = bnb.fetch(query=[{"time?r": [str(datetimenow),str(datetimenow1)]}],limit=1000)
        res_gru = bnb_gru.fetch(query=[{"time?r": [str(datetimenow),str(datetimenow1)]}],limit=1000)

    all_item = res.items
    all_item_gru = res_gru.items
    # while res.last:
    #     res = bnb.fetch(last=res.last)
    #     all_item+=res.items

    # while res_gru.last:
    #     res_gru = bnb_gru.fetch(last=res_gru.last)
    #     all_item_gru+=res_gru.items

    data = pd.DataFrame(all_item)
    data_gru = pd.DataFrame(all_item_gru)

    data['time'] = data['time'].transform(lambda x: datetime.datetime.strptime(x, '%Y-%m-%d %H:%M:%S'))
    data_gru['time'] = data_gru['time'].transform(lambda x: datetime.datetime.strptime(x, '%Y-%m-%d %H:%M:%S'))

    data.rename(columns ={'close_predict':'close_predict_lstm'}, inplace = True)
    data_gru.rename(columns ={'close_predict':'close_predict_gru'}, inplace = True)

    data =data[['time','close_predict_lstm']]
    data_gru =data_gru[['time','close_predict_gru']]

    data = data.sort_values(by="time")
    data['time'] = data['time'].transform(lambda x: str(x))
    data = data.reset_index()
        # data = data.set_index('time')
    data = data.iloc[:,1:]

    data_gru = data_gru.sort_values(by="time")
    data_gru['time'] = data_gru['time'].transform(lambda x: str(x))
    data_gru = data_gru.reset_index()
        # data = data.set_index('time')
    data_gru = data_gru.iloc[:,1:]

    if(limit is None):
        data = data.tail(5)
        data_gru = data_gru.tail(5)
    elif(int(limit) < 1000):
        data = data.tail(int(limit))
        data_gru = data_gru.tail(int(limit))

    data = pd.merge(data,data_gru, on='time',how='outer')
    currency = "BNBBUSD"
    interval = "1m"
    len_data = "1000"
    start_time = data.head(1)
    start_time = start_time['time'].values[0]
    print(start_time)
    start_time_dt = datetime.datetime.strptime(start_time, "%Y-%m-%d %H:%M:%S") + timedelta(hours=8)
    start_time_dt = str(utcToUnix(start_time_dt))
    df_scrap = cm.getDataCrypto(currency,interval,len_data,start_time_dt,None)
    df_scrap = df_scrap[['time','close']]
    df_scrap = pd.DataFrame(df_scrap)
    df_scrap['time'] = df_scrap['time'].transform(lambda x: unix2utc(x))
    df_scrap['close'] = df_scrap['close'].transform(lambda x: round(x,2))
    new_data = pd.merge(data,df_scrap, on='time',how='left')
    new_data = new_data.fillna('TBD')
    new_data['close'] = new_data['close'].transform(lambda x: str(x))
    json2data = json.loads(new_data.to_json(orient='records'))
```

```
    json2scrap = json.loads(df_scrap.to_json(orient='records'))
    # json2data = json.loads(data.to_json(orient='records'))
    return { "data": json2data}


@app.get("/eth")
async def eth(timeframe:str,limit:Optional[str] = None):
    tf = "eth"+ timeframe + "_db"
    tf2 = "eth"+ timeframe + "_gru_db"
    bnb = deta.Base(tf)
    bnb_gru = deta.Base(tf2)

    if(timeframe == '1m'):
        datetimenow = datetime.datetime.now() - timedelta(hours=2)
    else:
        datetimenow = datetime.datetime.now() - timedelta(hours=20)

    datetimenow1 = datetime.datetime.now()+timedelta(minutes=20)

    if(timeframe == '1m'):
        res = bnb.fetch(query=[{"key?r": [str(utcToUnix(datetimenow)),str(utcToUnix(datetimenow1))]}],limit=1000)
        res_gru = bnb_gru.fetch(query=[{"key?r":
[str(utcToUnix(datetimenow)),str(utcToUnix(datetimenow1))]}],limit=1000)
    else:
        res = bnb.fetch(query=[{"time?r": [str(datetimenow),str(datetimenow1)]}],limit=1000)
        res_gru = bnb_gru.fetch(query=[{"time?r": [str(datetimenow),str(datetimenow1)]}],limit=1000)

    all_item = res.items
    all_item_gru = res_gru.items
    # while res.last:
    #     res = bnb.fetch(last=res.last)
    #     all_item+=res.items

    # while res_gru.last:
    #     res_gru = bnb_gru.fetch(last=res_gru.last)
    #     all_item_gru+=res_gru.items

    data = pd.DataFrame(all_item)
    data_gru = pd.DataFrame(all_item_gru)

    data['time'] = data['time'].transform(lambda x: datetime.datetime.strptime(x, '%Y-%m-%d %H:%M:%S'))
    data_gru['time'] = data_gru['time'].transform(lambda x: datetime.datetime.strptime(x, '%Y-%m-%d %H:%M:%S'))

    data.rename(columns ={'close_predict':'close_predict_lstm'}, inplace = True)
    data_gru.rename(columns ={'close_predict':'close_predict_gru'}, inplace = True)

    data =data[['time','close_predict_lstm']]
    data_gru =data_gru[['time','close_predict_gru']]

    data = data.sort_values(by="time")
    data['time'] = data['time'].transform(lambda x: str(x))
    data = data.reset_index()
        # data = data.set_index('time')
    data = data.iloc[:,1:]

    data_gru = data_gru.sort_values(by="time")
    data_gru['time'] = data_gru['time'].transform(lambda x: str(x))
    data_gru = data_gru.reset_index()
        # data = data.set_index('time')
    data_gru = data_gru.iloc[:,1:]

    if(limit is None):
        data = data.tail(5)
        data_gru = data_gru.tail(5)
    elif(int(limit) < 1000):
        data = data.tail(int(limit))
        data_gru = data_gru.tail(int(limit))

    data = pd.merge(data,data_gru, on='time',how='outer')
    currency = "ETHBUSD"
    interval = "1m"
    len_data = "1000"
    start_time = data.head(1)
    start_time = start_time['time'].values[0]
    print(start_time)
    start_time_dt = datetime.datetime.strptime(start_time, "%Y-%m-%d %H:%M:%S") + timedelta(hours=8)
    start_time_dt = str(utcToUnix(start_time_dt))
    df_scrap = cm.getDataCrypto(currency,interval,len_data,start_time_dt,None)
    df_scrap = df_scrap[['time','close']]
    df_scrap = pd.DataFrame(df_scrap)
    df_scrap['time'] = df_scrap['time'].transform(lambda x: unix2utc(x))
    df_scrap['close'] = df_scrap['close'].transform(lambda x: round(x,2))
    new_data = pd.merge(data,df_scrap, on='time',how='left')
    new_data = new_data.fillna('TBD')
    new_data['close'] = new_data['close'].transform(lambda x: str(x))
    json2data = json.loads(new_data.to_json(orient='records'))
    json2scrap = json.loads(df_scrap.to_json(orient='records'))
    # json2data = json.loads(data.to_json(orient='records'))
    return { "data": json2data}
```

```python
@app.get("/btc")
async def btc(timeframe:str,limit:Optional[str] = None):
    tf = "btc"+ timeframe + "_db"
    tf2 = "btc"+ timeframe + "_gru_db"
    bnb = deta.Base(tf)
    bnb_gru = deta.Base(tf2)

    if(timeframe == '1m'):
        datetimenow = datetime.datetime.now() - timedelta(hours=2)
    else:
        datetimenow = datetime.datetime.now() - timedelta(hours=20)

    datetimenow1 = datetime.datetime.now()+timedelta(minutes=20)

    if(timeframe == '1m'):
        res = bnb.fetch(query=[{"key?r": [str(utcToUnix(datetimenow)),str(utcToUnix(datetimenow1))]}],limit=1000)
        res_gru = bnb_gru.fetch(query=[{"key?r":
[str(utcToUnix(datetimenow)),str(utcToUnix(datetimenow1))]}],limit=1000)
    else:
        res = bnb.fetch(query=[{"time?r": [str(datetimenow),str(datetimenow1)]}],limit=1000)
        res_gru = bnb_gru.fetch(query=[{"time?r": [str(datetimenow),str(datetimenow1)]}],limit=1000)

    all_item = res.items
    all_item_gru = res_gru.items
    # while res.last:
    #     res = bnb.fetch(last=res.last)
    #     all_item+=res.items

    # while res_gru.last:
    #     res_gru = bnb_gru.fetch(last=res_gru.last)
    #     all_item_gru+=res_gru.items

    data = pd.DataFrame(all_item)
    data_gru = pd.DataFrame(all_item_gru)

    data['time'] = data['time'].transform(lambda x: datetime.datetime.strptime(x, '%Y-%m-%d %H:%M:%S'))
    data_gru['time'] = data_gru['time'].transform(lambda x: datetime.datetime.strptime(x, '%Y-%m-%d %H:%M:%S'))

    data.rename(columns ={'close_predict':'close_predict_lstm'}, inplace = True)
    data_gru.rename(columns ={'close_predict':'close_predict_gru'}, inplace = True)

    data =data[['time','close_predict_lstm']]
    data_gru =data_gru[['time','close_predict_gru']]

    data = data.sort_values(by="time")
    data['time'] = data['time'].transform(lambda x: str(x))
    data = data.reset_index()
        # data = data.set_index('time')
    data = data.iloc[:,1:]

    data_gru = data_gru.sort_values(by="time")
    data_gru['time'] = data_gru['time'].transform(lambda x: str(x))
    data_gru = data_gru.reset_index()
        # data = data.set_index('time')
    data_gru = data_gru.iloc[:,1:]

    if(limit is None):
        data = data.tail(5)
        data_gru = data_gru.tail(5)
    elif(int(limit) < 1000):
        data = data.tail(int(limit))
        data_gru = data_gru.tail(int(limit))

    data = pd.merge(data,data_gru, on='time',how='outer')
    currency = "BTCBUSD"
    interval = "1m"
    len_data = "1000"
    start_time = data.head(1)
    start_time = start_time['time'].values[0]
    print(start_time)
    start_time_dt = datetime.datetime.strptime(start_time, "%Y-%m-%d %H:%M:%S") + timedelta(hours=8)
    start_time_dt = str(utcToUnix(start_time_dt))
    df_scrap = cm.getDataCrypto(currency,interval,len_data,start_time_dt,None)
    df_scrap = df_scrap[['time','close']]
    df_scrap = pd.DataFrame(df_scrap)
    df_scrap['time'] = df_scrap['time'].transform(lambda x: unix2utc(x))
    df_scrap['close'] = df_scrap['close'].transform(lambda x: round(x,2))
    new_data = pd.merge(data,df_scrap, on='time',how='left')
    new_data = new_data.fillna('TBD')
    new_data['close'] = new_data['close'].transform(lambda x: str(x))
    json2data = json.loads(new_data.to_json(orient='records'))
    json2scrap = json.loads(df_scrap.to_json(orient='records'))
    # json2data = json.loads(data.to_json(orient='records'))
    return { "data": json2data}
@app.get("/testing")
```

```
async def testing():
    currency = "BTCBUSD"
    interval = "1m"
    len_data = "100"
    start_time = "2022-07-20 00:41:00"
    print(start_time)
    start_time_dt = datetime.datetime.strptime(start_time, "%Y-%m-%d %H:%M:%S") + timedelta(hours=8)
    start_time_dt = str(utcToUnix(start_time_dt))
    # df_scrap  = getDataCrypto(currency,interval,len_data,start_time_dt)
    df_scrap = cm.getDataCrypto(currency,interval,len_data,start_time_dt,None)
    df_scrap = df_scrap[['time','close']]
    json2scrap = json.loads(df_scrap.to_json(orient='records'))
    return {"scrap":json2scrap}
```

Lampiran 6 Sourcecode Dashboard BTC

```
<!doctype html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <meta name="description" content="">
  <meta name="author" content="Mark Otto, Jacob Thornton, and Bootstrap contributors">
  <meta name="generator" content="Hugo 0.84.0">
  <title>Fanfan Crypto</title>
  <script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>
  <link rel="canonical" href="https://getbootstrap.com/docs/5.0/examples/product/">
  <!-- Bootstrap core CSS -->
  <link href="assets/dist/css/bootstrap.min.css" rel="stylesheet">
  <style>
    .bd-placeholder-img {
      font-size: 1.125rem;
      text-anchor: middle;
      -webkit-user-select: none;
      -moz-user-select: none;
      user-select: none;
    }

    @media (min-width: 768px) {
      .bd-placeholder-img-lg {
        font-size: 3.5rem;
      }
    }
  </style>


  <!-- Custom styles for this template -->
  <link href="product.css" rel="stylesheet">
</head>
<body>
  <header class="site-header sticky-top py-1">
    <nav class="container d-flex flex-column flex-md-row justify-content-between">
      <a class="py-2" href="#" aria-label="Product">
        <svg xmlns="http://www.w3.org/2000/svg" width="24" height="24" fill="none" stroke="currentColor"
          stroke-linecap="round" stroke-linejoin="round" stroke-width="2" class="d-block mx-auto" role="img"
          viewBox="0 0 24 24">
          <title>Product</title>
          <circle cx="12" cy="12" r="10" />
          <path
            d="M14.31 8l5.74 9.94M9.69 8h11.48M7.38 12l5.74-9.94M9.69 16L3.95 6.06M14.31 16H2.83m13.79-4l-5.74 9.94"
/>
        </svg>
      </a>
      <a class="py-2 d-none d-md-inline-block" href="index.html">BTC/BUSD</a>
      <a class="py-2 d-none d-md-inline-block" href="eth.html">ETH/BUSD</a>
      <a class="py-2 d-none d-md-inline-block" href="bnb.html">BNB/BUSD</a>
    </nav>
  </header>

  <main>
    <div class="position-relative overflow-hidden p-3 p-md-5 m-md-3 text-center bg-light">
      <div class="col-md-5 p-lg-5 mx-auto my-5">
        <h1 class="display-4 fw-normal">fanfanmyid</h1>
        <p class="lead fw-normal">Prediksi Nilai Tukar Cryptocurrency menggunakan LSTM dan GRU</p>
      </div>
    </div>
    <div class="d-md-flex flex-md-equal w-100 my-md-3 ps-md-3">
    <div class="bg-light me-md-3 pt-3 px-3 pt-md-5 px-md-5 text-center overflow-hidden">
      <div class="my-3 py-3" id="btc_banner">
        <!-- <p class="lead btc_time" id="time_btc">19.20.</p>
        <p class="lead" id="predict_btc">And an even wittier subheading.</p> -->
      </div>
```

```
  </div>
  <div class="bg-light me-md-3 pt-3 px-3 pt-md-5 px-md-5 text-center overflow-hidden">
    <div class="my-3 p-3" id="eth_banner">
      <!-- <h5 class="display-8">ETH/BUSD</h5>
      <p class="lead">And an even wittier subheading.</p> -->
    </div>
  </div>
  <div class="bg-light me-md-3 pt-3 px-3 pt-md-5 px-md-5 text-center overflow-hidden">
    <div class="my-3 p-3" id="bnb_banner">
      <!-- <h5 class="display-8">BNB/BUSD</h5>
      <p class="lead">And an even wittier subheading.</p> -->
    </div>
  </div>
</div>
  <div class="d-md-flex flex-md-equal w-100 my-md-3 ps-md-3">
    <div class="bg-light me-md-3 pt-3 px-3 pt-md-5 px-md-5 text-center overflow-hidden">
      <div class="my-3 py-3">
        <H4>BTC/BUSD 1m</H4>
        <table class="table">
          <thead>
            <tr>
              <th scope="col">Time</th>
              <th scope="col">Predict LSTM</th>
              <th scope="col">Predict GRU</th>
              <th scope="col">Actual</th>
            </tr>
          </thead>
          <tbody id="btcdata">
          </tbody>
        </table>
      </div>

    </div>

    <div class="bg-light me-md-3 pt-3 px-3 pt-md-5 px-md-5 text-center overflow-hidden">
      <div class="my-3 p-3">
        <H4>BTC/BUSD 5m</H4>
        <table class="table">
          <thead>
            <tr>
              <th scope="col">Time</th>
              <th scope="col">Predict LSTM</th>
              <th scope="col">Predict GRU</th>
              <th scope="col">Actual</th>
            </tr>
          </thead>
          <tbody id="ethdata">
          </tbody>
        </table>
      </div>

    </div>

    <div class="bg-light me-md-3 pt-3 px-3 pt-md-5 px-md-5 text-center overflow-hidden">
      <div class="my-3 p-3">
        <H4>BTC/BUSD 15m</H4>
        <table class="table">
          <thead>
            <tr>
              <th scope="col">Time</th>
              <th scope="col">Predict LSTM</th>
              <th scope="col">Predict GRU</th>
              <th scope="col">Actual</th>
            </tr>
          </thead>
          <tbody id="bnbdata">
          </tbody>
        </table>
      </div>

    </div>
  </div>
  <script>
    function onBtcGet() {
      const url = "https://fanfan-api.deta.dev/btc?timeframe=1m";
      var headers = {}

      fetch(url)
        .then((response) => {
          if (!response.ok) {
            throw new Error(response.error)
          }
          return response.json();
        })
        .then(data => {
          var jsonData = data.data
          var temp = ""
          var banner = ""
```

```
        for (let x in jsonData) {
          temp += "<tr>";
            date = new Date(Date.parse(jsonData[x]['time']+" UTC"))
            temp += "<td>" + date.toLocaleString() + "</td>";
          temp += "<td>" + jsonData[x]['close_predict_lstm'] + "</td>";
          temp += "<td>" + jsonData[x]['close_predict_gru'] + "</td>";
          temp += "<td>" + jsonData[x]['close'] + "</td>";
          if (x == 4){
            console.log("last data")
            banner+="<h5 class='display-8'>BTC/BUSD 1m</h5>"
            banner += "<p class='lead'>" + "Time :" + "</p>"
            date = new Date(Date.parse(jsonData[x]['time']+" UTC"))
            banner += "<p class='lead'>" + date.toLocaleString() + "</p>"
            banner += "<p class='lead'>" + "Predict LSTM :" + "</p>"
            banner += "<p class='lead'>" + jsonData[x]['close_predict_lstm'] + "</p>"
            banner += "<p class='lead'>" + "Predict GRU :" + "</p>"
            banner += "<p class='lead'>" + jsonData[x]['close_predict_gru'] + "</p>"
            document.getElementById('btc_banner').innerHTML = banner
          }
        }
        document.getElementById('btcdata').innerHTML = temp;
      })
      .catch(function (error) {
        console.log(error)
      });
  }
  function onEthGet() {
    const url = "https://fanfan-api.deta.dev/btc?timeframe=5m";
    var headers = {}

    fetch(url)
      .then((response) => {
        if (!response.ok) {
          throw new Error(response.error)
        }
        return response.json();
      })
      .then(data => {
        var jsonData = data.data
        var temp = ""
        var banner = ""
        for (let x in jsonData) {
          temp += "<tr>";
            date = new Date(Date.parse(jsonData[x]['time']+" UTC"))
            temp += "<td>" + date.toLocaleString() + "</td>";
          temp += "<td>" + jsonData[x]['close_predict_lstm'] + "</td>";
          temp += "<td>" + jsonData[x]['close_predict_gru'] + "</td>";
          temp += "<td>" + jsonData[x]['close'] + "</td>";
          if (x == 4){
            banner+="<h5 class='display-8'>BTC/BUSD 5m</h5>"
            banner += "<p class='lead'>" + "Time :" + "</p>"
            date = new Date(Date.parse(jsonData[x]['time']+" UTC"))
            banner += "<p class='lead'>" + date.toLocaleString() + "</p>"
            banner += "<p class='lead'>" + "Predict LSTM :" + "</p>"
            banner += "<p class='lead'>" + jsonData[x]['close_predict_lstm'] + "</p>"
            banner += "<p class='lead'>" + "Predict GRU :" + "</p>"
            banner += "<p class='lead'>" + jsonData[x]['close_predict_gru'] + "</p>"
            document.getElementById('eth_banner').innerHTML = banner
          }
        }
        document.getElementById('ethdata').innerHTML = temp;
      })
      .catch(function (error) {
        console.log(error)
      });
  }

  function onBnbGet() {
    const url = "https://fanfan-api.deta.dev/btc?timeframe=15m";
    var headers = {}

    fetch(url)
      .then((response) => {
        if (!response.ok) {
          throw new Error(response.error)
        }
        return response.json();
      })
      .then(data => {
        var jsonData = data.data
        var temp = ""
        var banner = ""
        console.log(jsonData.length)
        for (let x in jsonData) {
          temp += "<tr>";
            date = new Date(Date.parse(jsonData[x]['time']+" UTC"))
            temp += "<td>" + date.toLocaleString() + "</td>";
          temp += "<td>" + jsonData[x]['close_predict_lstm'] + "</td>";
          temp += "<td>" + jsonData[x]['close_predict_gru'] + "</td>";
```

```
            temp += "<td>" + jsonData[x]['close'] + "</td>";
            if (x == 4){
              banner+="<h5 class='display-8'>BTC/BUSD 15m</h5>"
              banner += "<p class='lead'>" + "Time :" + "</p>"
              date = new Date(Date.parse(jsonData[x]['time']+" UTC"))
              banner += "<p class='lead'>" + date.toLocaleString() + "</p>"
              banner += "<p class='lead'>" + "Predict LSTM :" + "</p>"
              banner += "<p class='lead'>" + jsonData[x]['close_predict_lstm'] + "</p>"
              banner += "<p class='lead'>" + "Predict GRU :" + "</p>"
              banner += "<p class='lead'>" + jsonData[x]['close_predict_gru'] + "</p>"
              document.getElementById('bnb_banner').innerHTML = banner
            }
          }
          document.getElementById('bnbdata').innerHTML = temp;
        })
        .catch(function (error) {
          console.log(error)
        });
    }
    const interval = setInterval(function () {
      // method to be executed;
      onBtcGet();
      onEthGet();
      onBnbGet();
    }, 5000);
  </script>
</main>

<footer class="container py-5">
  <div class="mypanel"></div>
  <div class="row">
    <div class="col-12 col-md">
      <small class="d-block mb-3 text-muted">&copy; Fanfanmyid-2022</small>
    </div>

  </div>
</footer>
<script src="assets/dist/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

Lampiran 7 Sourcecode Dashboard ETH

```
<!doctype html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <meta name="description" content="">
  <meta name="author" content="Mark Otto, Jacob Thornton, and Bootstrap contributors">
  <meta name="generator" content="Hugo 0.84.0">
  <title>Fanfan Crypto</title>
  <script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>
  <link rel="canonical" href="https://getbootstrap.com/docs/5.0/examples/product/">
  <!-- Bootstrap core CSS -->
  <link href="assets/dist/css/bootstrap.min.css" rel="stylesheet">

  <style>
    .bd-placeholder-img {
      font-size: 1.125rem;
      text-anchor: middle;
      -webkit-user-select: none;
      -moz-user-select: none;
      user-select: none;
    }

    @media (min-width: 768px) {
      .bd-placeholder-img-lg {
        font-size: 3.5rem;
      }
    }
  </style>


  <!-- Custom styles for this template -->
  <link href="product.css" rel="stylesheet">
</head>

<body>

  <header class="site-header sticky-top py-1">
    <nav class="container d-flex flex-column flex-md-row justify-content-between">
      <a class="py-2" href="#" aria-label="Product">
        <svg xmlns="http://www.w3.org/2000/svg" width="24" height="24" fill="none" stroke="currentColor"
```

```
          stroke-linecap="round" stroke-linejoin="round" stroke-width="2" class="d-block mx-auto" role="img"
          viewBox="0 0 24 24">
          <title>Product</title>
          <circle cx="12" cy="12" r="10" />
          <path
            d="M14.31 8l5.74 9.94M9.69 8h11.48M7.38 12l5.74-9.94M9.69 16L3.95 6.06M14.31 16H2.83m13.79-4l-5.74 9.94"
/>
        </svg>
      </a>
      <a class="py-2 d-none d-md-inline-block" href="index.html">BTC/BUSD</a>
      <a class="py-2 d-none d-md-inline-block" href="eth.html">ETH/BUSD</a>
      <a class="py-2 d-none d-md-inline-block" href="bnb.html">BNB/BUSD</a>
    </nav>
  </header>
  <main>
    <div class="position-relative overflow-hidden p-3 p-md-5 m-md-3 text-center bg-light">
      <div class="col-md-5 p-lg-5 mx-auto my-5">
        <h1 class="display-4 fw-normal">fanfanmyid</h1>
        <p class="lead fw-normal">Prediksi Nilai Tukar Cryptocurrency menggunakan LSTM dan GRU</p>

      </div>

    </div>

    <div class="d-md-flex flex-md-equal w-100 my-md-3 ps-md-3">
    <div class="bg-light me-md-3 pt-3 px-3 pt-md-5 px-md-5 text-center overflow-hidden">
      <div class="my-3 py-3" id="btc_banner">
        <!-- <p class="lead btc_time" id="time_btc">19.20.</p>
        <p class="lead" id="predict_btc">And an even wittier subheading.</p> -->
      </div>
    </div>
    <div class="bg-light me-md-3 pt-3 px-3 pt-md-5 px-md-5 text-center overflow-hidden">
      <div class="my-3 p-3" id="eth_banner">
        <!-- <h5 class="display-8">ETH/BUSD</h5>
        <p class="lead">And an even wittier subheading.</p> -->
      </div>

    </div>
    <div class="bg-light me-md-3 pt-3 px-3 pt-md-5 px-md-5 text-center overflow-hidden">
      <div class="my-3 p-3" id="bnb_banner">
        <!-- <h5 class="display-8">BNB/BUSD</h5>
        <p class="lead">And an even wittier subheading.</p> -->
      </div>
    </div>
  </div>

    <div class="d-md-flex flex-md-equal w-100 my-md-3 ps-md-3">
      <div class="bg-light me-md-3 pt-3 px-3 pt-md-5 px-md-5 text-center overflow-hidden">
        <div class="my-3 py-3">
          <H4>ETH/BUSD 1m</H4>
          <table class="table">
            <thead>
              <tr>
                <th scope="col">Time</th>
                <th scope="col">Predict LSTM</th>
                <th scope="col">Predict GRU</th>
                <th scope="col">Actual</th>
              </tr>
            </thead>
            <tbody id="btcdata">
            </tbody>
          </table>
        </div>
      </div>
      <div class="bg-light me-md-3 pt-3 px-3 pt-md-5 px-md-5 text-center overflow-hidden">
        <div class="my-3 p-3">
          <H4>ETH/BUSD 5m</H4>
          <table class="table">
            <thead>
              <tr>
                <th scope="col">Time</th>
                <th scope="col">Predict LSTM</th>
                <th scope="col">Predict GRU</th>
                <th scope="col">Actual</th>
              </tr>
            </thead>
            <tbody id="ethdata">
            </tbody>
          </table>
        </div>
      </div>
      <div class="bg-light me-md-3 pt-3 px-3 pt-md-5 px-md-5 text-center overflow-hidden">
        <div class="my-3 p-3">
          <H4>ETH/BUSD 15m</H4>
          <table class="table">
            <thead>
              <tr>
                <th scope="col">Time</th>
                <th scope="col">Predict LSTM</th>
```

```
              <th scope="col">Predict GRU</th>
              <th scope="col">Actual</th>
            </tr>
          </thead>
          <tbody id="bnbdata">
          </tbody>
        </table>
      </div>
    </div>
  </div>
<script>
  function onBtcGet() {
    const url = "https://fanfan-api.deta.dev/eth?timeframe=1m";
    var headers = {}
    fetch(url)
      .then((response) => {
        if (!response.ok) {
          throw new Error(response.error)
        }
        return response.json();
      })
      .then(data => {
        var jsonData = data.data
        var temp = ""
        var banner = ""
        for (let x in jsonData) {
          temp += "<tr>";
          date = new Date(Date.parse(jsonData[x]['time']+" UTC"))
          temp += "<td>" + date.toLocaleString() + "</td>";
          temp += "<td>" + jsonData[x]['close_predict_lstm'] + "</td>";
          temp += "<td>" + jsonData[x]['close_predict_gru'] + "</td>";
          temp += "<td>" + jsonData[x]['close'] + "</td>";
          if (x == 4){
            console.log("last data")
            banner+="<h5 class='display-8'>ETH/BUSD 1m</h5>"
            banner += "<p class='lead'>" + "Time :" + "</p>"
            date = new Date(Date.parse(jsonData[x]['time']+" UTC"))
            banner += "<p class='lead'>" + date.toLocaleString() + "</p>"
            banner += "<p class='lead'>" + "Predict LSTM :" + "</p>"
            banner += "<p class='lead'>" + jsonData[x]['close_predict_lstm'] + "</p>"
            banner += "<p class='lead'>" + "Predict GRU :" + "</p>"
            banner += "<p class='lead'>" + jsonData[x]['close_predict_gru'] + "</p>"
            document.getElementById('btc_banner').innerHTML = banner
          }
        }
        document.getElementById('btcdata').innerHTML = temp;
      })
      .catch(function (error) {
        console.log(error)
      });
  }
  function onEthGet() {
    const url = "https://fanfan-api.deta.dev/eth?timeframe=5m";
    var headers = {}
    fetch(url)
      .then((response) => {
        if (!response.ok) {
          throw new Error(response.error)
        }
        return response.json();
      })
      .then(data => {
        var jsonData = data.data
        var temp = ""
        var banner = ""
        for (let x in jsonData) {
          temp += "<tr>";
          date = new Date(Date.parse(jsonData[x]['time']+" UTC"))
          temp += "<td>" + date.toLocaleString() + "</td>";
          temp += "<td>" + jsonData[x]['close_predict_lstm'] + "</td>";
          temp += "<td>" + jsonData[x]['close_predict_gru'] + "</td>";
          temp += "<td>" + jsonData[x]['close'] + "</td>";
          if (x == 4){
            banner+="<h5 class='display-8'>ETH/BUSD 5m</h5>"
            banner += "<p class='lead'>" + "Time :" + "</p>"
            date = new Date(Date.parse(jsonData[x]['time']+" UTC"))
            banner += "<p class='lead'>" + date.toLocaleString() + "</p>"
            banner += "<p class='lead'>" + "Predict LSTM :" + "</p>"
            banner += "<p class='lead'>" + jsonData[x]['close_predict_lstm'] + "</p>"
            banner += "<p class='lead'>" + "Predict GRU :" + "</p>"
            banner += "<p class='lead'>" + jsonData[x]['close_predict_gru'] + "</p>"
            document.getElementById('eth_banner').innerHTML = banner
          }
        }
        document.getElementById('ethdata').innerHTML = temp;
      })
      .catch(function (error) {
        console.log(error)
      });
```

```
    }
    function onBnbGet() {
      const url = "https://fanfan-api.deta.dev/eth?timeframe=15m";
      var headers = {}
      fetch(url)
        .then((response) => {
          if (!response.ok) {
            throw new Error(response.error)
          }
          return response.json();
        })
        .then(data => {
          var jsonData = data.data
          var temp = ""
          var banner = ""
          console.log(jsonData.length)
          for (let x in jsonData) {
            temp += "<tr>";
            date = new Date(Date.parse(jsonData[x]['time']+" UTC"))
            temp += "<td>" + date.toLocaleString() + "</td>";
            temp += "<td>" + jsonData[x]['close_predict_lstm'] + "</td>";
            temp += "<td>" + jsonData[x]['close_predict_gru'] + "</td>";
            temp += "<td>" + jsonData[x]['close'] + "</td>";
            if (x == 4){
              banner+="<h5 class='display-8'>ETH/BUSD 15m</h5>"
              banner += "<p class='lead'>" + "Time :" + "</p>"
              date = new Date(Date.parse(jsonData[x]['time']+" UTC"))
              banner += "<p class='lead'>" + date.toLocaleString() + "</p>"
              banner += "<p class='lead'>" + "Predict LSTM :" + "</p>"
              banner += "<p class='lead'>" + jsonData[x]['close_predict_lstm'] + "</p>"
              banner += "<p class='lead'>" + "Predict GRU :" + "</p>"
              banner += "<p class='lead'>" + jsonData[x]['close_predict_gru'] + "</p>"
              document.getElementById('bnb_banner').innerHTML = banner
            }
          }
          document.getElementById('bnbdata').innerHTML = temp;
        })
        .catch(function (error) {
          console.log(error)
        });
    }

    const interval = setInterval(function () {
      // method to be executed;
      onBtcGet();
      onEthGet();
      onBnbGet();
    }, 5000);
  </script>
</main>
<footer class="container py-5">
  <div class="mypanel"></div>
  <div class="row">
    <div class="col-12 col-md">
      <small class="d-block mb-3 text-muted">&copy; Fanfanmyid-2022</small>
    </div>

  </div>
</footer>
<script src="assets/dist/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

Lampiran 8 Sourcecode Dashboard BNB

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <meta name="description" content="">
  <meta name="author" content="Mark Otto, Jacob Thornton, and Bootstrap contributors">
  <meta name="generator" content="Hugo 0.84.0">
  <title>Fanfan Crypto</title>
  <script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>
  <link rel="canonical" href="https://getbootstrap.com/docs/5.0/examples/product/">
  <link href="assets/dist/css/bootstrap.min.css" rel="stylesheet">
  <style>
    .bd-placeholder-img {
      font-size: 1.125rem;
      text-anchor: middle;
      -webkit-user-select: none;
      -moz-user-select: none;
      user-select: none;
    }
```

```
    @media (min-width: 768px) {
      .bd-placeholder-img-lg {
        font-size: 3.5rem;
      }
    }
  </style>
  <!-- Custom styles for this template -->
  <link href="product.css" rel="stylesheet">
</head>
<body>
  <header class="site-header sticky-top py-1">
    <nav class="container d-flex flex-column flex-md-row justify-content-between">
      <a class="py-2" href="#" aria-label="Product">
        <svg xmlns="http://www.w3.org/2000/svg" width="24" height="24" fill="none" stroke="currentColor"
          stroke-linecap="round" stroke-linejoin="round" stroke-width="2" class="d-block mx-auto" role="img"
          viewBox="0 0 24 24">
          <title>Product</title>
          <circle cx="12" cy="12" r="10" />
          <path
            d="M14.31 8l5.74 9.94M9.69 8h11.48M7.38 12l5.74-9.94M9.69 16L3.95 6.06M14.31 16H2.83m13.79-4l-5.74 9.94"
/>
        </svg>
      </a>
      <a class="py-2 d-none d-md-inline-block" href="index.html">BTC/BUSD</a>
      <a class="py-2 d-none d-md-inline-block" href="eth.html">ETH/BUSD</a>
      <a class="py-2 d-none d-md-inline-block" href="bnb.html">BNB/BUSD</a>
    </nav>
  </header>
  <main>
    <div class="position-relative overflow-hidden p-3 p-md-5 m-md-3 text-center bg-light">
      <div class="col-md-5 p-lg-5 mx-auto my-5">
        <h1 class="display-4 fw-normal">fanfanmyid</h1>
        <p class="lead fw-normal">Prediksi Nilai Tukar Cryptocurrency menggunakan LSTM dan GRU</p>
      </div>
    </div>
    <div class="d-md-flex flex-md-equal w-100 my-md-3 ps-md-3">
    <div class="bg-light me-md-3 pt-3 px-3 pt-md-5 px-md-5 text-center overflow-hidden">
      <div class="my-3 py-3" id="btc_banner">
        <!-- <p class="lead btc_time" id="time_btc">19.20.</p>
        <p class="lead" id="predict_btc">And an even wittier subheading.</p> -->
      </div>
    </div>
    <div class="bg-light me-md-3 pt-3 px-3 pt-md-5 px-md-5 text-center overflow-hidden">
      <div class="my-3 p-3" id="eth_banner">
        <!-- <h5 class="display-8">ETH/BUSD</h5>
        <p class="lead">And an even wittier subheading.</p> -->
      </div>
    </div>
    <div class="bg-light me-md-3 pt-3 px-3 pt-md-5 px-md-5 text-center overflow-hidden">
      <div class="my-3 p-3" id="bnb_banner">
        <!-- <h5 class="display-8">BNB/BUSD</h5>
        <p class="lead">And an even wittier subheading.</p> -->
      </div>
    </div>
  </div>
    <div class="d-md-flex flex-md-equal w-100 my-md-3 ps-md-3">
      <div class="bg-light me-md-3 pt-3 px-3 pt-md-5 px-md-5 text-center overflow-hidden">
        <div class="my-3 py-3">
          <H4>BNB/BUSD 1m</H4>
          <table class="table">
            <thead>
              <tr>
                <th scope="col">Time</th>
                <th scope="col">Predict LSTM</th>
                <th scope="col">Predict GRU</th>
                <th scope="col">Actual</th>
              </tr>
            </thead>
            <tbody id="btcdata">
            </tbody>
          </table>
        </div>
      </div>
      <div class="bg-light me-md-3 pt-3 px-3 pt-md-5 px-md-5 text-center overflow-hidden">
        <div class="my-3 p-3">
          <H4>BNB/BUSD 5m</H4>
          <table class="table">
            <thead>
              <tr>
                <th scope="col">Time</th>
                <th scope="col">Predict LSTM</th>
                <th scope="col">Predict GRU</th>
                <th scope="col">Actual</th>
              </tr>
            </thead>
            <tbody id="ethdata">
            </tbody>
          </table>
        </div>
```

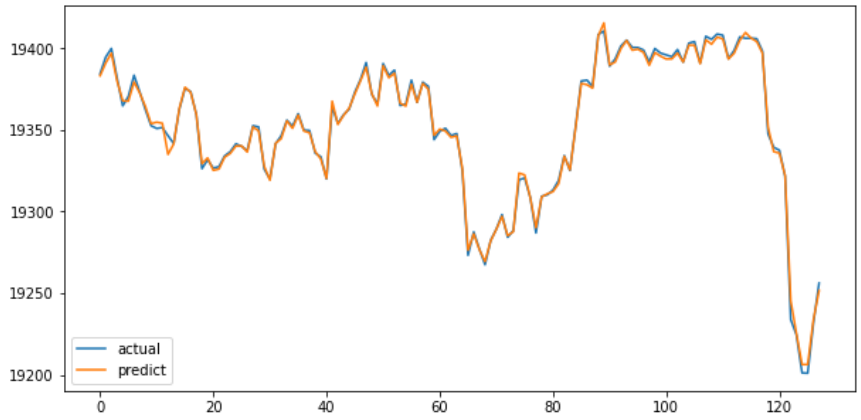
```
    </div>
    <div class="bg-light me-md-3 pt-3 px-3 pt-md-5 px-md-5 text-center overflow-hidden">
      <div class="my-3 p-3">
        <H4>BNB/BUSD 15m</H4>
        <table class="table">
          <thead>
            <tr>
              <th scope="col">Time</th>
              <th scope="col">Predict LSTM</th>
              <th scope="col">Predict GRU</th>
              <th scope="col">Actual</th>
            </tr>
          </thead>
          <tbody id="bnbdata">
          </tbody>
        </table>
      </div>
    </div>
  </div>
  <script>
    function onBtcGet() {
      const url = "https://fanfan-api.deta.dev/bnb?timeframe=1m";
      var headers = {}
      fetch(url)
        .then((response) => {
          if (!response.ok) {
            throw new Error(response.error)
          }
          return response.json();
        })
        .then(data => {
          var jsonData = data.data
          var temp = ""
          var banner = ""
          for (let x in jsonData) {
            temp += "<tr>";
              date = new Date(Date.parse(jsonData[x]['time']+" UTC"))
              temp += "<td>" + date.toLocaleString() + "</td>";
            temp += "<td>" + jsonData[x]['close_predict_lstm'] + "</td>";
            temp += "<td>" + jsonData[x]['close_predict_gru'] + "</td>";
            temp += "<td>" + jsonData[x]['close'] + "</td>";
            if (x == 4){
              console.log("last data")
              banner+="<h5 class='display-8'>BNB/BUSD 1m</h5>"
              banner += "<p class='lead'>" + "Time :" + "</p>"
              date = new Date(Date.parse(jsonData[x]['time']+" UTC"))
              banner += "<p class='lead'>" + date.toLocaleString() + "</p>"
              banner += "<p class='lead'>" + "Predict LSTM :" + "</p>"
              banner += "<p class='lead'>" + jsonData[x]['close_predict_lstm'] + "</p>"
              banner += "<p class='lead'>" + "Predict GRU :" + "</p>"
              banner += "<p class='lead'>" + jsonData[x]['close_predict_gru'] + "</p>"
              document.getElementById('btc_banner').innerHTML = banner
            }

          }
          document.getElementById('btcdata').innerHTML = temp;
        })
        .catch(function (error) {
          console.log(error)
        });
    }
    function onEthGet() {
      const url = "https://fanfan-api.deta.dev/bnb?timeframe=5m";
      var headers = {}
      fetch(url)
        .then((response) => {
          if (!response.ok) {
            throw new Error(response.error)
          }
          return response.json();
        })
        .then(data => {
          var jsonData = data.data
          var temp = ""
          var banner = ""
          for (let x in jsonData) {
            temp += "<tr>";
              date = new Date(Date.parse(jsonData[x]['time']+" UTC"))
              temp += "<td>" + date.toLocaleString() + "</td>";
            temp += "<td>" + jsonData[x]['close_predict_lstm'] + "</td>";
            temp += "<td>" + jsonData[x]['close_predict_gru'] + "</td>";
            temp += "<td>" + jsonData[x]['close'] + "</td>";
            if (x == 4){
              banner+="<h5 class='display-8'>BNB/BUSD 5m</h5>"
              banner += "<p class='lead'>" + "Time :" + "</p>"
              date = new Date(Date.parse(jsonData[x]['time']+" UTC"))
              banner += "<p class='lead'>" + date.toLocaleString() + "</p>"
              banner += "<p class='lead'>" + "Predict LSTM :" + "</p>"
              banner += "<p class='lead'>" + jsonData[x]['close_predict_lstm'] + "</p>"
```

```
                banner += "<p class='lead'>" + "Predict GRU :" + "</p>"
                banner += "<p class='lead'>" + jsonData[x]['close_predict_gru'] + "</p>"
                document.getElementById('eth_banner').innerHTML = banner
              }
            }
            document.getElementById('ethdata').innerHTML = temp;
          })
          .catch(function (error) {
            console.log(error)
          });
      }
      function onBnbGet() {
        const url = "https://fanfan-api.deta.dev/bnb?timeframe=15m";
        var headers = {}
        fetch(url)
          .then((response) => {
            if (!response.ok) {
              throw new Error(response.error)
            }
            return response.json();
          })
          .then(data => {
            var jsonData = data.data
            var temp = ""
            var banner = ""
            console.log(jsonData.length)
            for (let x in jsonData) {
              temp += "<tr>";
                date = new Date(Date.parse(jsonData[x]['time']+" UTC"))
                temp += "<td>" + date.toLocaleString() + "</td>";
              temp += "<td>" + jsonData[x]['close_predict_lstm'] + "</td>";
              temp += "<td>" + jsonData[x]['close_predict_gru'] + "</td>";
              temp += "<td>" + jsonData[x]['close'] + "</td>";
              if (x == 4){
                banner+="<h5 class='display-8'>BNB/BUSD 15m</h5>"
                banner += "<p class='lead'>" + "Time :" + "</p>"
                date = new Date(Date.parse(jsonData[x]['time']+" UTC"))
                banner += "<p class='lead'>" + date.toLocaleString() + "</p>"
                banner += "<p class='lead'>" + "Predict LSTM :" + "</p>"
                banner += "<p class='lead'>" + jsonData[x]['close_predict_lstm'] + "</p>"
                banner += "<p class='lead'>" + "Predict GRU :" + "</p>"
                banner += "<p class='lead'>" + jsonData[x]['close_predict_gru'] + "</p>"
                document.getElementById('bnb_banner').innerHTML = banner
              }
            }
            document.getElementById('bnbdata').innerHTML = temp;
          })
          .catch(function (error) {
            console.log(error)
          });
      }
      const interval = setInterval(function () {
        // method to be executed;
        onBtcGet();
        onEthGet();
        onBnbGet();
      }, 5000);
    </script>
  </main>
  <footer class="container py-5">
    <div class="mypanel"></div>
    <div class="row">
      <div class="col-12 col-md">
        <small class="d-block mb-3 text-muted">&copy; Fanfanmyid-2022</small>
      </div>
    </div>
  </footer>
  <script src="assets/dist/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

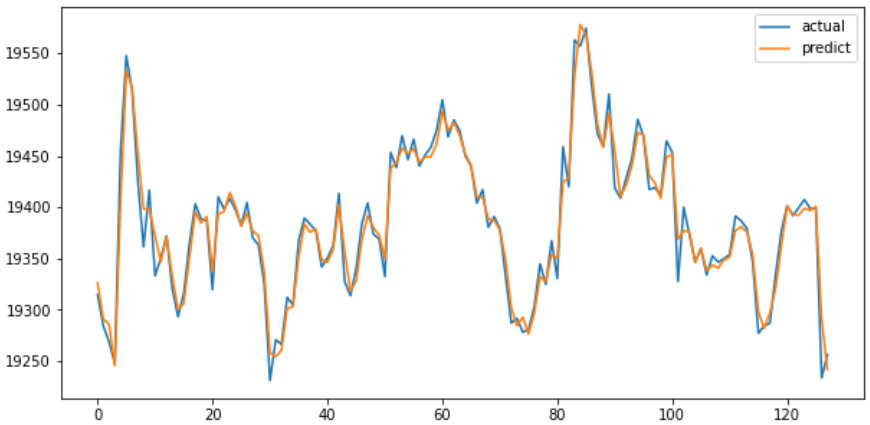
Lampiran 9 Grafik Pengujian Hasil Pengujian Model

BTC

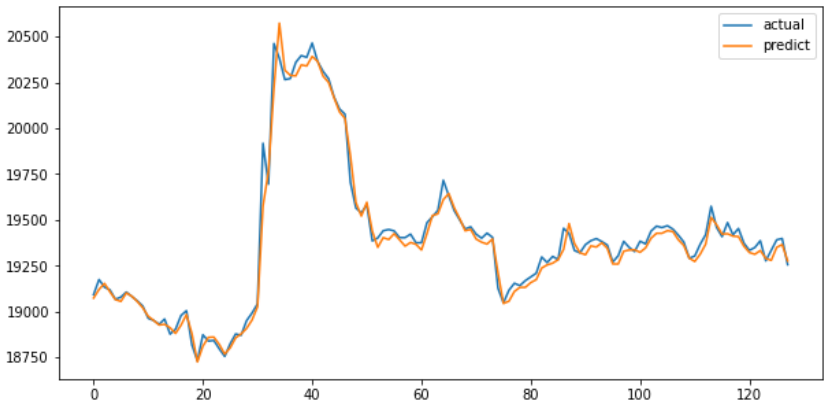
BTC 1m
Test RMSE: 2.419
Test Mape: 0.009



BTC 5m
Test RMSE: 14.733
Test Mape: 0.055

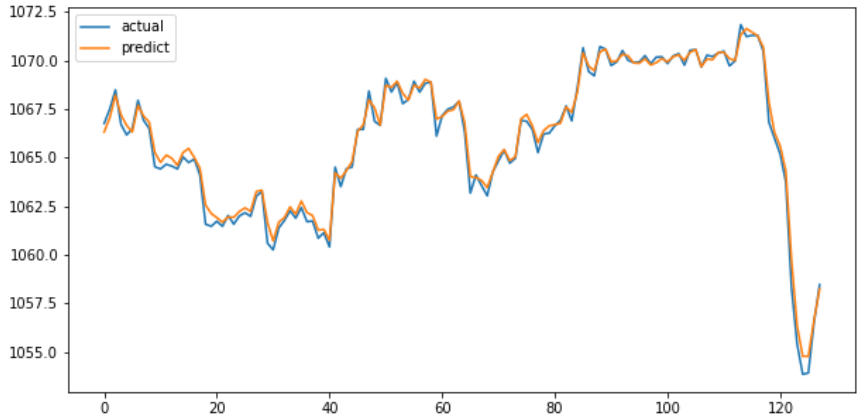


BTC 15m
Test RMSE: 56.585
Test Mape: 0.186

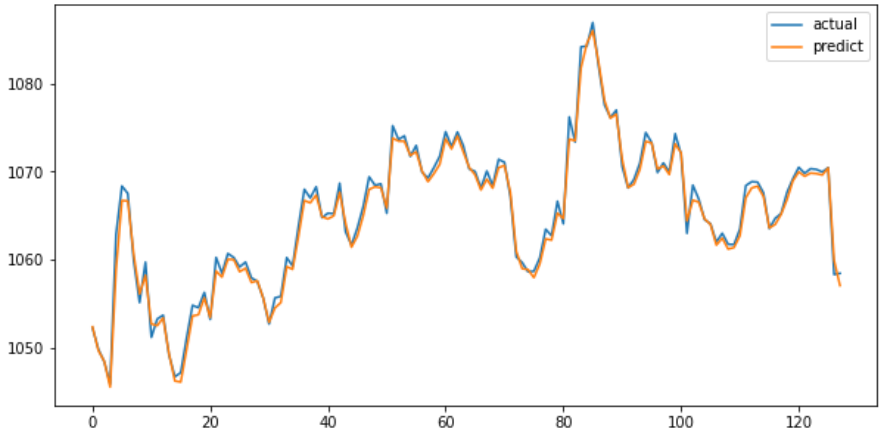


ETH

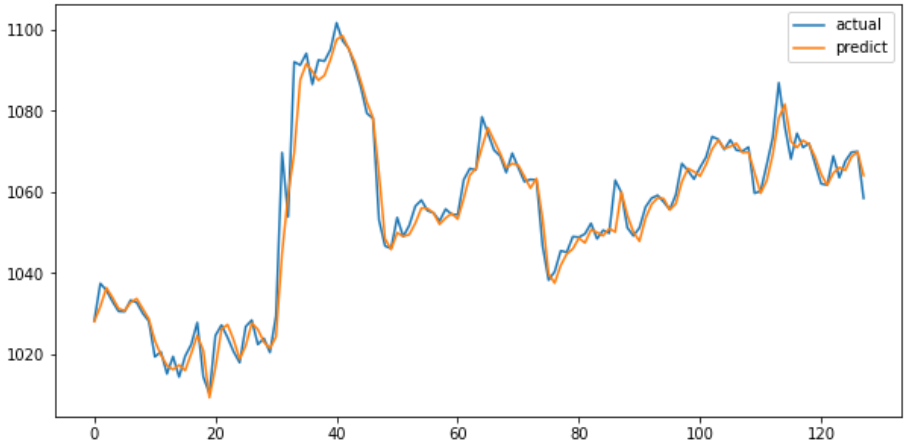
ETH 1m
Test RMSE: 0.389
Test Mape: 0.027



ETH 5m
Test RMSE: 0.876
Test Mape: 0.063

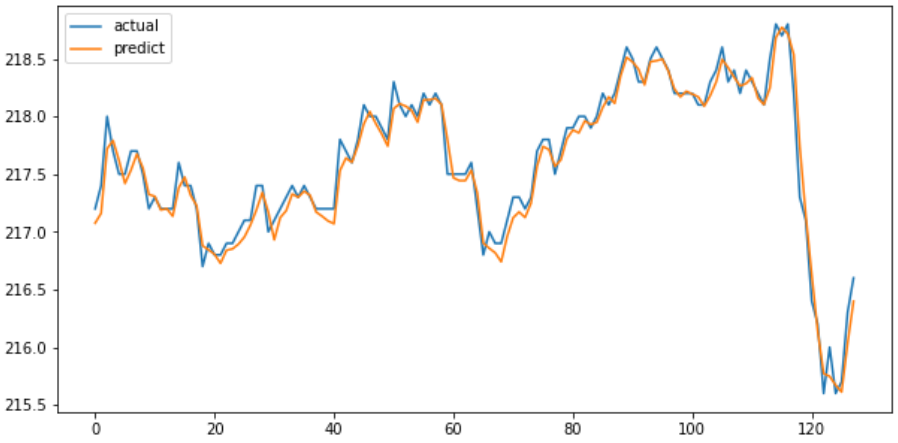


ETH 15m
Test RMSE: 4.361
Test Mape: 0.255

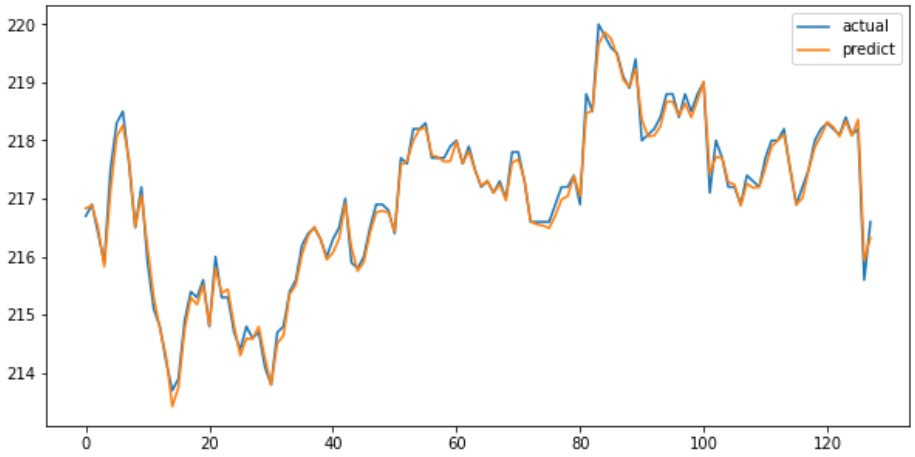


BNB

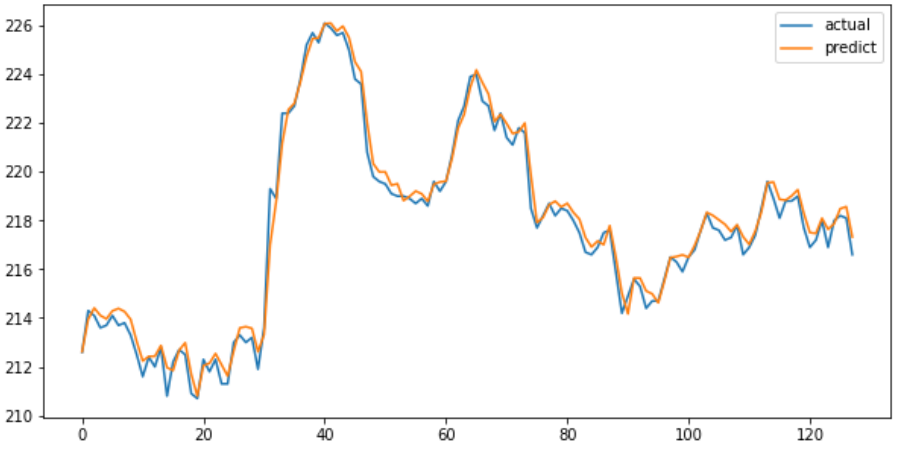
BNB 1m
Test RMSE: 0.121
Test Mape: 0.043



BNB 5m
Test RMSE: 0.140
Test Mape: 0.049

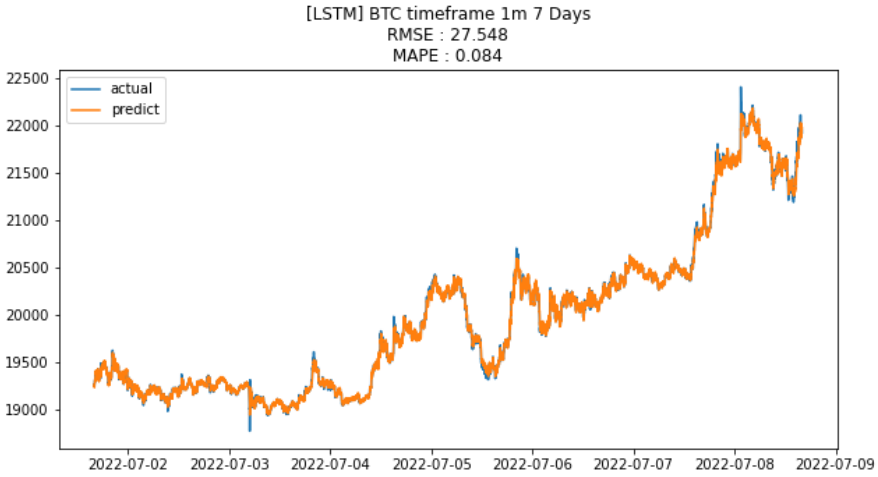


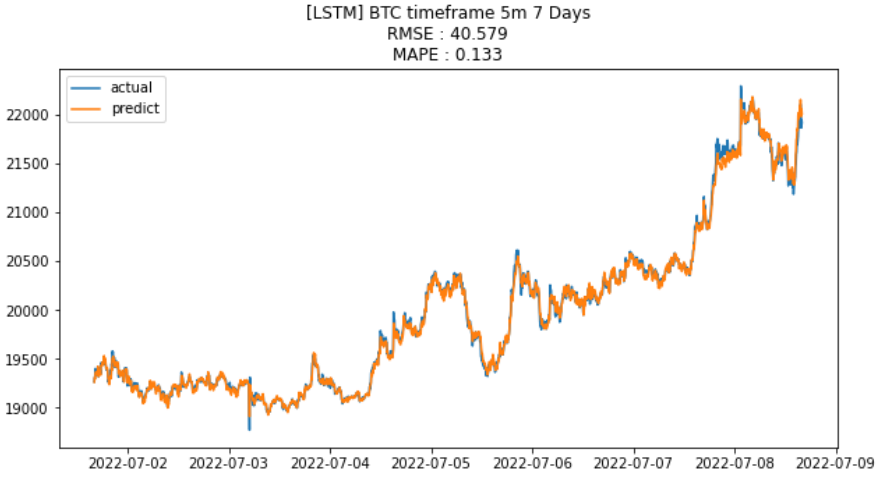
BNB 15m
Test RMSE: 0.502
Test Mape: 0.177

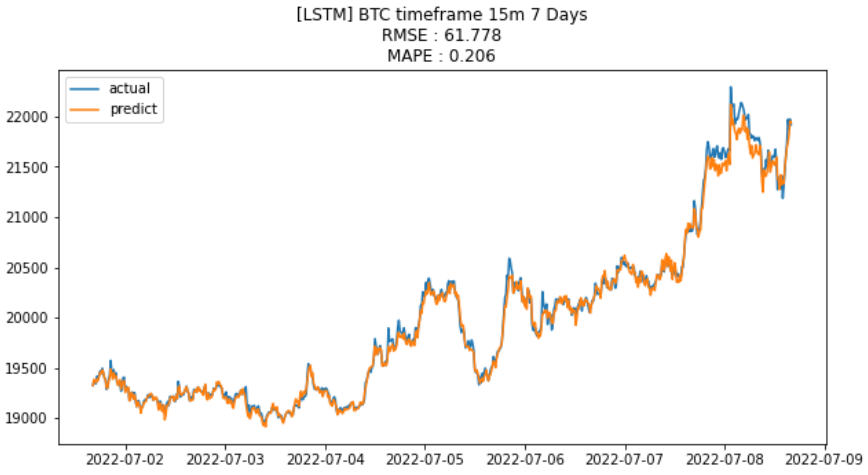


Lampiran 10 Grafik Pengujian Realtime

BTC

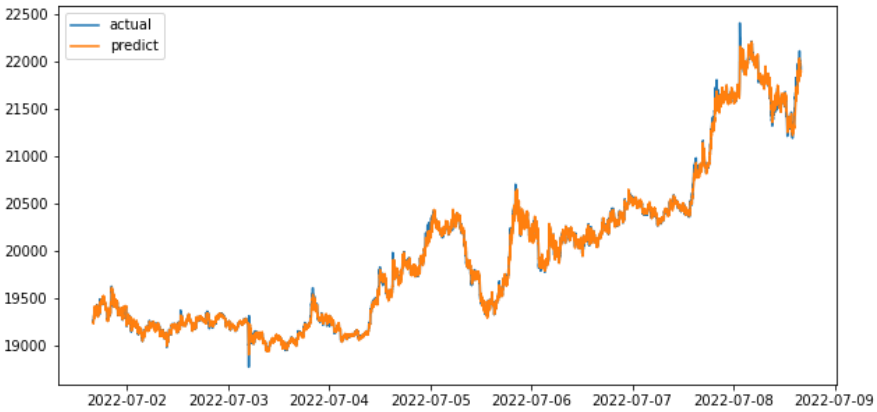






[GRU] BTC timeframe 15m 7 Days
RMSE : 78.772
MAPE : 0.276

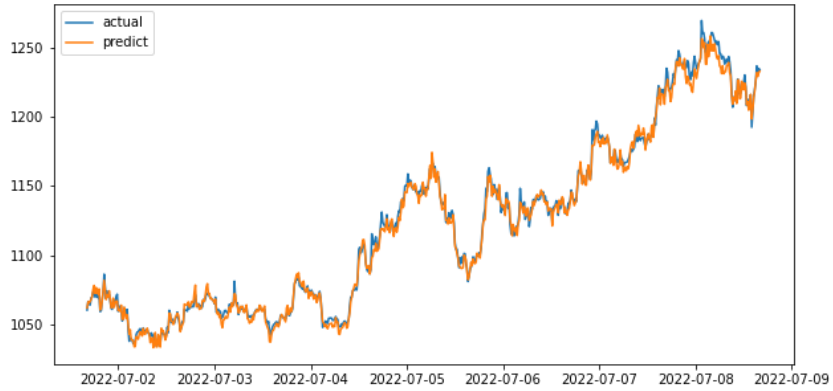
[GRU] BTC timeframe 1m 7 Days
RMSE : 31.093
MAPE : 0.096



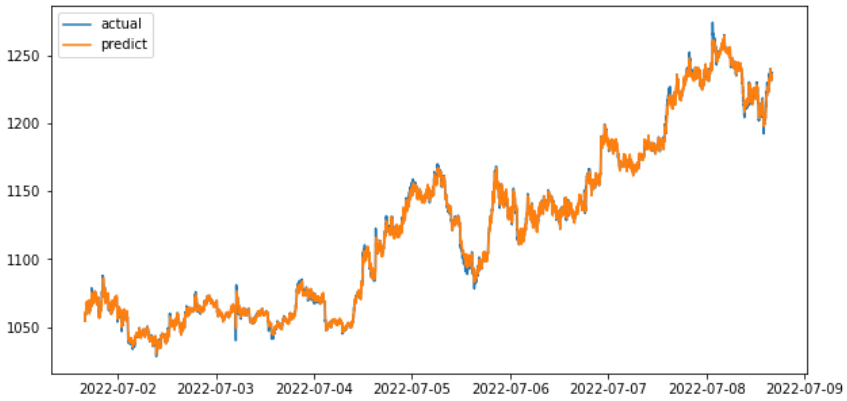
[GRU] BTC timeframe 5m 7 Days
RMSE : 53.597
MAPE : 0.175

ETH

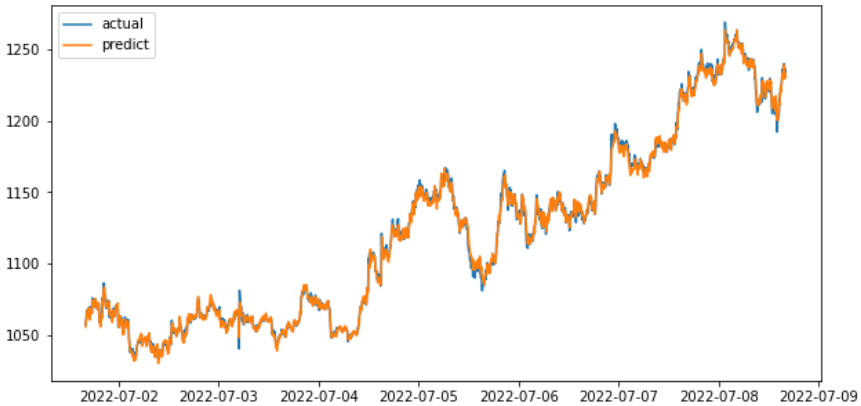
[LSTM] ETH timeframe 15m 7 Days
RMSE : 4.106
MAPE : 0.274



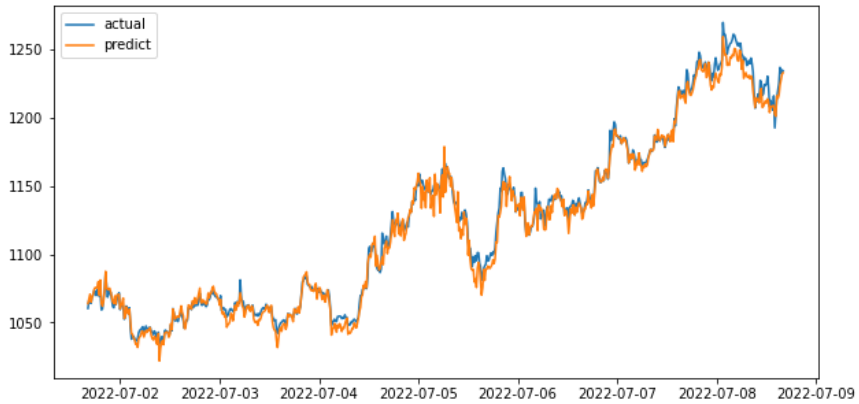
[LSTM] ETH timeframe 1m 7 Days
RMSE : 1.839
MAPE : 0.112



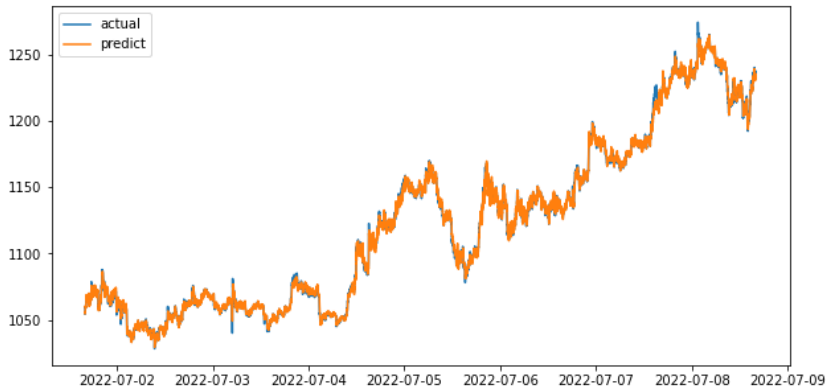
[LSTM] ETH timeframe 5m 7 Days
RMSE : 2.521
MAPE : 0.166



[GRU] ETH timeframe 15m 7 Days
RMSE : 5.989
MAPE : 0.392



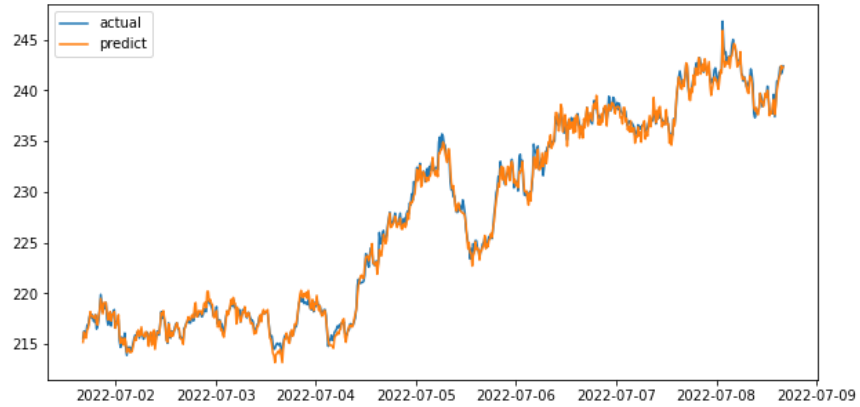
[GRU] ETH timeframe 1m 7 Days
RMSE : 2.220
MAPE : 0.133



[GRU] ETH timeframe 5m 7 Days
RMSE : 3.080
MAPE : 0.201

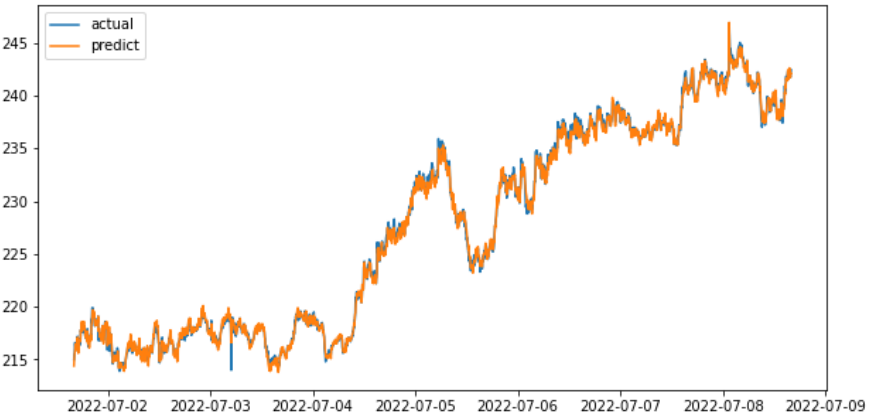
BNB

[LSTM] BNB timeframe 15m 7 Days
RMSE : 0.538
MAPE : 0.184



[LSTM] BNB timeframe 1m 7 Days
RMSE : 0.270
MAPE : 0.083

[LSTM] BNB timeframe 5m 7 Days
RMSE : 0.391
MAPE : 0.129



[GRU] BNB timeframe 1m 7 Days
RMSE : 0.300
MAPE : 0.094

[GRU] BNB timeframe 5m 7 Days
RMSE : 0.486
MAPE : 0.156

[GRU] BNB timeframe 15m 7 Days
RMSE : 0.837
MAPE : 0.287

Lampiran 11 *Riwayat Transaksi*

BTC

2022-07-21 18:41:52	BTCBUSD Perpetual	Sell	22,943.3	23.0 BUSD	0.00688299 BUSD	0.02540000 BUSD
2022-07-21 17:24:15	BTCBUSD Perpetual	Buy	22,917.9	23.0 BUSD	0.00687537 BUSD	0.00000000 BUSD
2022-07-21 18:53:25	BTCBUSD Perpetual	Buy	22,963.2	23.0 BUSD	0.00688896 BUSD	-0.02130000 BUSD
2022-07-21 18:42:01	BTCBUSD Perpetual	Sell	22,941.9	23.0 BUSD	0.00688257 BUSD	0.00000000 BUSD
2022-07-21 19:19:58	BTCBUSD Perpetual	Sell	22,726.0	22.8 BUSD	0.00681780 BUSD	-0.13470000 BUSD
2022-07-21 18:59:34	BTCBUSD Perpetual	Buy	22,860.7	22.9 BUSD	0.00685821 BUSD	0.00000000 BUSD
2022-07-21 21:27:40	BTCBUSD Perpetual	Sell	22,695.4	22.7 BUSD	0.00680862 BUSD	-0.06160000 BUSD
2022-07-21 20:23:31	BTCBUSD Perpetual	Buy	22,757.0	22.8 BUSD	0.00273084 BUSD	0.00000000 BUSD
2022-07-22 06:04:32	BTCBUSD Perpetual	Sell	23,145.7	23.2 BUSD	0.00694371 BUSD	0.14570000 BUSD
2022-07-22 00:49:07	BTCBUSD Perpetual	Buy	23,000.0	23.0 BUSD	0.00690000 BUSD	0.00000000 BUSD
2022-07-21 19:30:17	BTCBUSD Perpetual	Buy	22,552.9	22.6 BUSD	0.00676587 BUSD	0.18700000 BUSD
2022-07-21 19:20:23	BTCBUSD Perpetual	Sell	22,739.9	22.8 BUSD	0.00682197 BUSD	0.00000000 BUSD
2022-07-22 07:09:37	BTCBUSD Perpetual	Sell	23,035.5	23.1 BUSD	0.00691065 BUSD	0.03570000 BUSD
2022-07-22 06:54:11	BTCBUSD Perpetual	Buy	22,999.8	23.0 BUSD	0.00689994 BUSD	0.00000000 BUSD
2022-07-22 11:00:22	BTCBUSD Perpetual	Buy	22,994.8	23.0 BUSD	0.00689844 BUSD	0.05790000 BUSD
2022-07-22 10:20:38	BTCBUSD Perpetual	Sell	23,052.7	23.1 BUSD	0.00691581 BUSD	0.00000000 BUSD

2022-07-22 11:28:39	BTCBUSD Perpetual	Buy	22,889.1	22.9 BUSD	0.00686673 BUSD	0.13510000 BUSD
2022-07-22 11:15:01	BTCBUSD Perpetual	Sell	23,024.2	23.1 BUSD	0.00690726 BUSD	0.00000000 BUSD
2022-07-21 20:20:49	BTCBUSD Perpetual	Sell	22,733.5	22.8 BUSD	0.00682005 BUSD	0.12770000 BUSD
2022-07-21 19:30:47	BTCBUSD Perpetual	Buy	22,605.8	22.7 BUSD	0.00678174 BUSD	0.00000000 BUSD

ETH

2022-07-20 19:54:50	ETHBUSD Perpetual	Buy	1,600.84	140.88 BUSD	0.04226217 BUSD	-0.96800000 BUSD
2022-07-20 19:48:55	ETHBUSD Perpetual	Sell	1,589.84	139.91 BUSD	0.04197177 BUSD	0.00000000 BUSD
2022-07-20 19:41:19	ETHBUSD Perpetual	Sell	1,592.83	138.58 BUSD	0.04157286 BUSD	-0.08874000 BUSD
2022-07-20 19:39:30	ETHBUSD Perpetual	Buy	1,593.85	132.29 BUSD	0.03968686 BUSD	0.00000000 BUSD
2022-07-20 19:29:11	ETHBUSD Perpetual	Buy	1,589.46	127.16 BUSD	0.03814704 BUSD	0.19040000 BUSD
2022-07-20 19:29:11	ETHBUSD Perpetual	Buy	1,589.46	12.72 BUSD	0.00381470 BUSD	0.01904000 BUSD
2022-07-20 19:28:15	ETHBUSD Perpetual	Sell	1,591.84	7.96 BUSD	0.00238776 BUSD	0.00000000 BUSD
2022-07-20 19:28:15	ETHBUSD Perpetual	Sell	1,591.84	6.37 BUSD	0.00191020 BUSD	0.00000000 BUSD
2022-07-20 19:28:15	ETHBUSD Perpetual	Sell	1,591.84	20.70 BUSD	0.00620817 BUSD	0.00000000 BUSD
2022-07-20 19:28:15	ETHBUSD Perpetual	Sell	1,591.84	105.07 BUSD	0.03151843 BUSD	0.00000000 BUSD
2022-07-20 19:18:08	ETHBUSD Perpetual	Buy	1,594.03	39.86 BUSD	0.01195522 BUSD	0.07550000 BUSD
2022-07-20 19:13:42	ETHBUSD Perpetual	Sell	1,597.05	39.93 BUSD	0.01197787 BUSD	0.00000000 BUSD
2022-07-20 19:12:34	ETHBUSD Perpetual	Buy	1,603.41	19.25 BUSD	0.00577227 BUSD	0.02472000 BUSD
2022-07-20 19:11:12	ETHBUSD Perpetual	Sell	1,605.47	19.27 BUSD	0.00577969 BUSD	0.00000000 BUSD

2022-07-20 19:32:52	ETHBUSD Perpetual	Sell	1,592.44	140.14 BUSD	0.04204041 BUSD	0.30536000 BUSD
2022-07-20 19:29:28	ETHBUSD Perpetual	Buy	1,588.97	139.83 BUSD	0.04194880 BUSD	0.00000000 BUSD
2022-07-20 19:32:52	ETHBUSD Perpetual	Sell	1,592.44	140.14 BUSD	0.04204041 BUSD	0.30536000 BUSD
2022-07-20 19:29:28	ETHBUSD Perpetual	Buy	1,588.97	139.83 BUSD	0.04194880 BUSD	0.00000000 BUSD
2022-07-20 19:10:07	ETHBUSD Perpetual	Sell	1,605.15	19.27 BUSD	0.00577854 BUSD	0.13068000 BUSD
2022-07-20 19:01:15	ETHBUSD Perpetual	Buy	1,594.26	6.38 BUSD	0.00191311 BUSD	0.00000000 BUSD
2022-07-21 06:19:47	ETHBUSD Perpetual	Sell	1,543.78	961.78 BUSD	0.28853248 BUSD	0.95319000 BUSD
2022-07-21 06:18:52	ETHBUSD Perpetual	Buy	1,542.25	960.83 BUSD	0.28824652 BUSD	0.00000000 BUSD
2022-07-20 19:32:52	ETHBUSD Perpetual	Sell	1,592.44	140.14 BUSD	0.04204041 BUSD	0.30536000 BUSD
2022-07-20 19:29:28	ETHBUSD Perpetual	Buy	1,588.97	139.83 BUSD	0.04194880 BUSD	0.00000000 BUSD

BNB

2022-07-22 13:12:46	BNBBUSD Perpetual	Sell	263.660	21.10 BUSD	0.00632784 BUSD	-0.08240000 BUSD
2022-07-22 13:08:12	BNBBUSD Perpetual	Buy	264.690	21.18 BUSD	0.00635256 BUSD	0.00000000 BUSD
2022-07-22 16:43:57	BNBBUSD Perpetual	Buy	267.160	18.71 BUSD	0.00561036 BUSD	0.03500000 BUSD
2022-07-22 16:40:40	BNBBUSD Perpetual	Sell	267.660	18.74 BUSD	0.00562086 BUSD	0.00000000 BUSD
2022-07-22 16:39:35	BNBBUSD Perpetual	Sell	267.730	18.75 BUSD	0.00562233 BUSD	0.02660000 BUSD
2022-07-22 14:10:58	BNBBUSD Perpetual	Buy	267.350	18.72 BUSD	0.00224574 BUSD	0.00000000 BUSD
2022-07-22 14:10:22	BNBBUSD Perpetual	Buy	267.470	18.73 BUSD	0.00561687 BUSD	0.00070000 BUSD
2022-07-22 14:07:08	BNBBUSD Perpetual	Sell	267.480	18.73 BUSD	0.00561708 BUSD	0.00000000 BUSD

2022-07-22 14:05:51	**BNBBUSD** Perpetual	Sell	267.370	18.72 BUSD	0.00561477 BUSD	0.01750000 BUSD
2022-07-22 14:01:31	**BNBBUSD** Perpetual	Buy	267.120	18.70 BUSD	0.00560952 BUSD	0.00000000 BUSD
2022-07-22 13:58:07	**BNBBUSD** Perpetual	Sell	266.440	18.66 BUSD	0.00559524 BUSD	0.02450000 BUSD
2022-07-22 13:55:33	**BNBBUSD** Perpetual	Buy	266.090	18.63 BUSD	0.00558789 BUSD	0.00000000 BUSD
2022-07-22 13:46:02	**BNBBUSD** Perpetual	Sell	265.540	21.25 BUSD	0.00637296 BUSD	-0.01600000 BUSD
2022-07-22 13:39:01	**BNBBUSD** Perpetual	Buy	265.740	21.26 BUSD	0.00637776 BUSD	0.00000000 BUSD
2022-07-22 13:33:44	**BNBBUSD** Perpetual	Sell	265.550	21.25 BUSD	0.00637320 BUSD	0.04560000 BUSD
2022-07-22 13:26:12	**BNBBUSD** Perpetual	Buy	264.980	21.20 BUSD	0.00635952 BUSD	0.00000000 BUSD
2022-07-22 13:24:46	**BNBBUSD** Perpetual	Sell	264.910	21.20 BUSD	0.00635784 BUSD	0.05060000 BUSD
2022-07-22 13:17:54	**BNBBUSD** Perpetual	Buy	264.280	15.86 BUSD	0.00475703 BUSD	0.00000000 BUSD

LEMBAR PERBAIKAN SKRIPSI

"PREDIKSI NILAI TUKAR *CRPTOCURRENCY* JANGKA PENDEK DENGAN MENGGUNAKAN *LONG SHORT TERM MEMORY* (LSTM)

OLEH:

MUHAMMAD FANDLY FADLURACHMAN
D121181701

Skripsi ini telah dipertahankan pada Ujian Akhir Sarjana tanggal 26 Oktober 2022.

Telah dilakukan perbaikan penulisan dan isi skripsi berdasarkan usulan dari penguji dan pembimbing skripsi.

Persetujuan perbaikan oleh tim penguji:

	Nama	Tanda Tangan
Ketua	Dr. Ir. Indrabayu, M.T., M.Bus.Sys.,IPM	
Sekretaris	A. Ais Prayogi Alimuddin, S.T., M.Eng.	
Anggota	Hasniaty A, S.T., M.T.,Ph.D.	
	Elly Warni, S.T., M.T.	

Persetujuan Perbaikan oleh pembimbing:

Pembimbing	Nama	Tanda Tangan
I	Dr. Ir.Indrabayu, M.T., M.Bus.Sys.,IPM	
II	A. Ais Prayogi Alimuddin, S.T., M.Eng.	