|  |
| --- |
| **Pertemuan 11 dan 12 – Stacked-Bidirectional on Neural Network** |
|  |
| **Tujuan pembelajaran** |
| * Mahasiswa mampu memahami konsep timeseries pada cryptocurrency * Mahasiswa mampu memahami konsep stacked-bidirectional pada neural network |

Studi kasus: Model Prediksi Harga BTC-USD Menggunakan

Metode SBi-LSTM-RNN dan SBi-GRU-RNN

|  |
| --- |
| **C01\_data\_collection.py** |
|  |
| # lib data manipulation  import pandas as pd    # function load dataset  def data\_collection(df):      # load dataset    dataset = pd.read\_csv("dataset/"+df, parse\_dates=['Date'])    dataset = dataset[["Date", "Open", "High", "Low", "Close"]]      # return values    return dataset |

|  |
| --- |
| **C02\_visualization.py** |
|  |
| # lib data visualizations  import matplotlib.pyplot as plt  from matplotlib.dates import DateFormatter    # function of lineplot  def timeseries\_matplotlib(df, nm\_labels):        # create lineplot      fig, ax = plt.subplots(figsize = (8,4))      for x in range(len(nm\_labels)):        ax.plot(df.iloc[:, 0:1], df.iloc[:, x+1:x+2], label=nm\_labels[x], linewidth=2.5)        # set label-labels      ax.set\_title("", fontsize=12)      ax.set\_xlabel("", fontsize=10)      ax.set\_ylabel("", fontsize=10)      ax.legend(loc="best")      ax.grid(True)      return plt.show()    # visualisasi timeseries plot  def lineplot\_matplotlib1(x, y, label, title):      # membuat time series plot    fig, ax = plt.subplots(figsize = (8,4))    ax.plot(x, y, color="tab:blue", label=label, linewidth=2.5)      # membuat label-label    ax.xaxis.set\_major\_formatter(DateFormatter("%Y"))    ax.set\_title(title, fontsize=12)    ax.set\_xlabel("", fontsize=12)    ax.set\_ylabel("", fontsize=12)    ax.legend(loc="best")    ax.grid(True)      # return values    return plt.show()    # visualisasi timeseries plot  def lineplot\_matplotlib2(x1, y1, label1, x2, y2, label2, title):      # membuat time series plot    fig, ax = plt.subplots(figsize = (8,4))    ax.plot(x1, y1, color="tab:blue", label=label1, linewidth=2.5, linestyle="solid")    ax.plot(x2, y2, color="tab:red", label=label2, linewidth=2.5, linestyle="solid")      # membuat label-label    ax.xaxis.set\_major\_formatter(DateFormatter("%Y"))    ax.set\_title(title, fontsize=12)    ax.set\_xlabel("", fontsize=12)    ax.set\_ylabel("", fontsize=12)    ax.legend(loc="best")    ax.grid(True)      # return values    return plt.show()    # visualisasi timeseries plot  def lineplot\_matplotlib3(x1, y1, label1, x2, y2, label2, title):      # membuat time series plot    fig, ax = plt.subplots(figsize = (8,4))    ax.plot(x1, y1, color="tab:blue", label=label1, linewidth=2.5, linestyle="solid")    ax.plot(x2, y2, color="tab:orange", label=label2, linewidth=2.5, linestyle="solid")      # membuat label-label    ax.set\_title(title, fontsize=12)    ax.set\_xlabel("", fontsize=12)    ax.set\_ylabel("", fontsize=12)    ax.legend(loc="best")    ax.grid(True)      # return values    return plt.show() |

|  |
| --- |
| **C03\_preprocessing.py** |
|  |
| # lib manipulation data  import numpy as np  import pandas as pd    # lib data preprocessing  from sklearn.preprocessing import MinMaxScaler  from sklearn.model\_selection import train\_test\_split  # ----------------------------------------------------------------------------------------    # function for supervised learning  def create\_dataset(look\_back, dataset):        # declare variable X and Y      dataX = []      dataY = []        # for loop for create supervised learning      for i in range(look\_back, len(dataset)):          dataX.append(dataset[i-look\_back:i, 0])          dataY.append(dataset[i, 0])        # return value X and Y      return np.array(dataX), np.array(dataY)    # functions of data preprocessing  def preprocessing(dataset):      # 1. set feature    data = dataset.filter(['Close'])    data = data.values      # 2. normalize features    scaler = MinMaxScaler(feature\_range=(0, 1))    scaled = scaler.fit\_transform(np.array(data).reshape(-1,1))      # 3. traing testing    train\_data, test\_data = train\_test\_split(  scaled, train\_size=0.8, test\_size=0.2, shuffle=False)      # 4. supervised learning    x\_train, y\_train = create\_dataset(60, train\_data)    x\_test, y\_test = create\_dataset(60, test\_data)      # 5. reshape input to be [samples, time steps, features]    x\_train = np.reshape(x\_train, (x\_train.shape[0], x\_train.shape[1], 1))    x\_test = np.reshape(x\_test, (x\_test.shape[0], x\_test.shape[1], 1))      # return values    return scaler, scaled, x\_train, y\_train, x\_test, y\_test    # function for inverse normalized  def inverse(scaler, scaled):    return scaler.inverse\_transform(scaled.reshape(-1,1)) |

|  |
| --- |
| **C04\_model\_predictions.py** |
|  |
| # lib neural network algorithms  import tensorflow as tf  from keras.layers import LSTM  from keras.layers import GRU    # func model predictions  def get\_models(algorithm, x\_train, y\_train, x\_test, y\_test):      # 1. The LSTM architecture    if algorithm == "SBi-LSTM-RNN":      tf.keras.backend.clear\_session()      model = tf.keras.Sequential([        tf.keras.layers.Bidirectional(  LSTM(units=50, return\_sequences=True, input\_shape=(x\_train.shape[1], 1))),        tf.keras.layers.Bidirectional(LSTM(units=50, return\_sequences=False)),        tf.keras.layers.Dropout(0.05),        tf.keras.layers.Dense(1)      ])      # 2. The GRU-RNN architecture    if algorithm == "SBi-GRU-RNN":      tf.keras.backend.clear\_session()      model = tf.keras.Sequential([        tf.keras.layers.Bidirectional(  GRU(units=50, return\_sequences=True, input\_shape=(x\_train.shape[1], 1))),        tf.keras.layers.Bidirectional(GRU(units=50, return\_sequences=False)),        tf.keras.layers.Dropout(0.05),        tf.keras.layers.Dense(1)      ])      # 2. compile models    model.compile(optimizer='adamax', loss="mean\_squared\_error")      # 3. fitting models    history = model.fit(      x=x\_train, y=y\_train,      batch\_size=16, epochs=50, verbose="auto",      validation\_data=(x\_test, y\_test),      shuffle=False, use\_multiprocessing=False,    )      # 4. predict models    predictions = model.predict(x\_test, verbose=0)      # return values    return history, predictions |

|  |
| --- |
| **C05\_model\_evaluate.py** |
|  |
| # libs manipulations array  import numpy as np    # lib evaluate models  import scipy.stats as sc  from sklearn.metrics import mean\_absolute\_error  from sklearn.metrics import root\_mean\_squared\_error  from sklearn.metrics import mean\_absolute\_percentage\_error    # func evaluate models  def evaluate\_models(ytrue,ypred):      # calculate mae, rmse, mape    r     = sc.mstats.pearsonr(ytrue,ypred)[0]    p     = sc.mstats.pearsonr(ytrue,ypred)[1]    mae   = mean\_absolute\_error(ytrue,ypred)    rmse  = root\_mean\_squared\_error(ytrue,ypred)    mape  = mean\_absolute\_percentage\_error(ytrue,ypred)      # return values    return np.round(r,4), np.round(p,4), np.round(mae,4), np.round(rmse,4), np.round(mape,4) |

|  |
| --- |
| **K01-Volatilitas.ipynb** |
|  |
| # libs manipulations array  import pandas as pd  import numpy as np    # Muat data BTC-USD  btc\_data = pd.read\_csv('dataset/BTC-USD.csv', parse\_dates=['Date'])    # Hitung pengembalian harian  btc\_data['daily\_return'] = btc\_data['Close'].pct\_change()  # Tambahkan kolom tahun  btc\_data['Year'] = btc\_data['Date'].dt.year    # Hitung volatilitas tahunan  annual\_volatility = btc\_data.groupby('Year')['daily\_return'].std() \* np.sqrt(252)  annual\_volatility = annual\_volatility \* 100    # Tampilkan hasil  for year, volatility in annual\_volatility.items():      print(f"Volatilitas untuk tahun {year}: {volatility:.2f}") |
|  |
| Volatilitas untuk tahun 2015: 57.21  Volatilitas untuk tahun 2016: 39.90  Volatilitas untuk tahun 2017: 79.24  Volatilitas untuk tahun 2018: 67.36  Volatilitas untuk tahun 2019: 56.55  Volatilitas untuk tahun 2020: 59.86  Volatilitas untuk tahun 2021: 66.82  Volatilitas untuk tahun 2022: 52.80  Volatilitas untuk tahun 2023: 36.37  Volatilitas untuk tahun 2024: 49.34 |

|  |
| --- |
| **K01-Volatilitas.ipynb** |
|  |
| import pandas as pd    # Muat data BTC-USD  btc\_data = pd.read\_csv('dataset/BTC-USD.csv', parse\_dates=['Date'])    # Tambahkan kolom tahun  btc\_data['Year'] = btc\_data['Date'].dt.year    # Hitung harga tertinggi dan terendah setiap tahun  annual\_high = btc\_data.groupby('Year')['Close'].max().round(2)  annual\_low = btc\_data.groupby('Year')['Close'].min().round(2)    # Tampilkan hasil  annual\_high\_low = pd.DataFrame({      'Annual High': annual\_high,      'Annual Low': annual\_low  })    annual\_high\_low = annual\_high\_low.reset\_index()  print(annual\_high\_low) |
|  |
| Year Annual High Annual Low  0 2015 465.32 178.10  1 2016 975.92 364.33  2 2017 19497.40 777.76  3 2018 17527.00 3236.76  4 2019 13016.23 3399.47  5 2020 29001.72 4970.79  6 2021 67566.83 29374.15  7 2022 47686.81 15787.28  8 2023 44166.60 16625.08  9 2024 73083.50 39507.37 |

|  |
| --- |
| **K01-Volatilitas.ipynb** |
|  |
| import matplotlib.pyplot as plt    # Data  years = ['2015', '2016', '2017', '2018', '2019', '2020', '2021', '2022', '2023', '2024']  annual\_high = [465.32, 975.92, 19497.40, 17527.00, 13016.23, 29001.72, 67566.83,                 47686.81, 44166.60, 73083.50]  annual\_low  = [178.10, 364.33, 7777.76, 3236.76, 3399.47, 4970.79, 29374.15,                15787.28, 16625.08, 39507.37]  volatility  = [57.21, 39.90, 79.24, 67.36, 56.55, 59.86, 66.82, 52.80, 36.37, 49.34]    # Plotting  fig, ax1 = plt.subplots(figsize=(8,4))    # Bar plot for Annual High and Low  bar\_width = 0.35  r1 = np.arange(len(years))  r2 = [x + bar\_width for x in r1]    ax1.bar(r1, annual\_high, color='tab:blue', width=bar\_width, edgecolor='grey', label='Annual High')  ax1.bar(r2, annual\_low, color='tab:red', width=bar\_width, edgecolor='grey', label='Annual Low')  ax1.set\_xlabel('Years', fontweight='bold')  ax1.set\_ylabel('Value', fontweight='bold')  ax1.set\_xticks([r + bar\_width/2 for r in range(len(years))])  ax1.set\_xticklabels(years)  ax1.legend(loc='upper left')    # Creating another y-axis for Volatility  ax2 = ax1.twinx()  ax2.plot(years, volatility, color='tab:orange', marker='o', linestyle='-', linewidth=2, markersize=8, label='Volatility')  ax2.set\_ylabel('Volatility', color='g', fontweight='bold')  ax2.tick\_params(axis='y', labelcolor='black')  ax2.legend(loc='upper right')    plt.title('')  plt.xticks(rotation=0)  plt.tight\_layout()  plt.grid(True)  plt.show() |
|  |
|  |
| Gambar x. Output program |

|  |
| --- |
| **K02-Preprocessing.ipynb** |
|  |
| # lib manipulasi data  import pandas as pd  import numpy as np    # lib untuk visualisasi data  import seaborn as sns  import matplotlib.pyplot as plt    # lib praproses data  from sklearn.preprocessing import MinMaxScaler  from sklearn.model\_selection import train\_test\_split |
|  |
| # load dataset  dataset = pd.read\_csv("dataset/BTC-USD.csv", parse\_dates=["Date"])  dataset = dataset.set\_index("Date") |
|  |
| # show metadataset  print(np.round(    dataset[["Open","High","Low","Close"]].describe(),0  )) |
|  |
| Open High Low Close  count 3408.0 3408.0 3408.0 3408.0  mean 16502.0 16887.0 16089.0 16518.0  std 17823.0 18248.0 17348.0 17834.0  min 177.0 212.0 172.0 178.0  25% 1403.0 1464.0 1399.0 1445.0  50% 9028.0 9216.0 8809.0 9045.0  75% 27276.0 27799.0 26848.0 27300.0  max 73079.0 73750.0 71334.0 73084.0 |
|  |
| # round .3f  dataset = np.round(dataset[["Open","High","Low","Close"]],4) |
|  |
| # show dataset  print(dataset.tail()) |
|  |
| Open High Low Close  Date  2024-04-26 64485.3711 64789.6562 63322.3984 63755.3203  2024-04-27 63750.9883 63898.3633 62424.7188 63419.1406  2024-04-28 63423.5156 64321.4844 62793.5977 63113.2305  2024-04-29 63106.3633 64174.8789 61795.4570 63841.1211  2024-04-30 63839.4180 64703.3320 59120.0664 60636.8555 |
|  |
| # create frame  fig, ax = plt.subplots(figsize = (8,4))    # time series plot  ax.plot(dataset.index.values, dataset["Open"], color="tab:green", label="Open", linewidth=2)  ax.plot(dataset.index.values, dataset["High"], color="tab:orange", label="High", linewidth=2)  ax.plot(dataset.index.values, dataset["Low"], color="tab:red", label="Low", linewidth=2)  ax.plot(dataset.index.values, dataset["Close"], color="tab:blue", label="Close", linewidth=2)    # set label-labels  ax.set\_title("", fontsize=14)  ax.set\_xlabel("", fontsize=12)  ax.set\_ylabel("", fontsize=12)  ax.legend(loc="best")  ax.grid(True)  plt.show() |
|  |
|  |
| Gambar x. Output program |

|  |
| --- |
| # convert dataframe to series close price  data = dataset.values  np.round(data[:3],7) |
|  |
| array([[320.435, 320.435, 314.003, 314.249],  [314.079, 315.839, 313.565, 315.032],  [314.846, 315.15 , 281.082, 281.082]]) |
|  |
| # normalize features  scaler = MinMaxScaler(feature\_range=(0, 1))  scaled\_data = scaler.fit\_transform(np.array(data)) |
|  |
| # show normalize data  scaled\_data[:3] |
|  |
| array([[0.0019689 , 0.00147819, 0.00200236, 0.00186743],  [0.00188172, 0.0014157 , 0.0019962 , 0.00187817],  [0.00189224, 0.00140633, 0.00153974, 0.0014125 ]]) |
|  |
| # create frame  fig, ax = plt.subplots(figsize = (8,4))    # time series plot  ax.plot(dataset.index, scaled\_data[:,0:1], color="tab:green", label="Open", linewidth=2)  ax.plot(dataset.index, scaled\_data[:,1:2], color="tab:orange", label="High", linewidth=2)  ax.plot(dataset.index, scaled\_data[:,2:3], color="tab:red", label="Low", linewidth=2)  ax.plot(dataset.index, scaled\_data[:,3:4], color="tab:blue", label="Close", linewidth=2)    # set label-labels  ax.set\_title("",fontsize=14)  ax.set\_xlabel("",fontsize=12)  ax.set\_ylabel("",fontsize=12)  ax.legend(loc="best")  ax.grid(True)  plt.show() |
|  |
|  |
| Gambar x. Output program |

|  |
| --- |
| # results preprocessing of normalize data  df\_normalized = pd.concat([    pd.DataFrame(dataset.index.values, columns=["Date"]),    pd.DataFrame(scaled\_data, columns=["Open", "High", "Low", "Close"]),  ], axis=1) |
|  |
| # show normalize  print(df\_normalized.tail()) |
|  |
| Date Open High Low Close  3403 2024-04-26 0.882116 0.878153 0.887417 0.872051  3404 2024-04-27 0.872043 0.866033 0.874803 0.867440  3405 2024-04-28 0.867551 0.871787 0.879986 0.863244  3406 2024-04-29 0.863201 0.869793 0.865960 0.873228  3407 2024-04-30 0.873256 0.876979 0.828364 0.829277 |

|  |
| --- |
| # split data train and test  train\_data, test\_data = train\_test\_split(df\_normalized, train\_size=0.80, test\_size=0.20,  shuffle=False) |
|  |
| # create figure  fig, ax = plt.subplots(figsize = (8,4))    # create time series plot  ax.plot(    df\_normalized["Date"].iloc[0:len(train\_data)], train\_data["Close"],    color="tab:blue", label="training", linewidth=2  )  ax.plot(    df\_normalized["Date"].iloc[len(train\_data):], test\_data["Close"],    color="tab:red", label="testing", linewidth=2  )    # set labels  ax.set\_title("",fontsize=14)  ax.set\_xlabel("",fontsize=12)  ax.set\_ylabel("",fontsize=12)  ax.legend(loc="best")  ax.grid(True)    # show plot  plt.show() |
|  |
|  |
| Gambar x. Output program |

|  |
| --- |
| **K03-TimeSeries-Analysis.ipynb** |
|  |
| # lib manipulasi data  import pandas as pd  import numpy as np    # pustaka uji stasioneritas arch  from arch.unitroot import \*  from arch.unitroot import ADF  from arch.unitroot import PhillipsPerron  from arch.unitroot import KPSS |
|  |
| # load csv  dataset = pd.read\_csv("dataset/BTC-USD-norm.csv", parse\_dates=["Date"]) |
|  |
| # show metadata  print(dataset.info()) |
|  |
| <class 'pandas.core.frame.DataFrame'>  RangeIndex: 3408 entries, 0 to 3407  Data columns (total 5 columns):  # Column Non-Null Count Dtype  --- ------ -------------- -----  0 Date 3408 non-null datetime64[ns]  1 Open 3408 non-null float64  2 High 3408 non-null float64  3 Low 3408 non-null float64  4 Close 3408 non-null float64 |

|  |
| --- |
| adf = ADF(y=dataset["Close"], lags=60, trend="ct")  print(adf.summary().as\_text()) |
|  |
| Augmented Dickey-Fuller Results  =====================================  Test Statistic -2.783  P-value 0.203  Lags 60  -------------------------------------  Trend: Constant and Linear Time Trend  Critical Values: -3.96 (1%), -3.41 (5%), -3.13 (10%)  Null Hypothesis: The process contains a unit root.  Alternative Hypothesis: The process is weakly stationary. |
|  |
| pp = PhillipsPerron(y=dataset["Close"], lags=60, trend="ct")  print(pp.summary().as\_text()) |
| Phillips-Perron Test (Z-tau)  =====================================  Test Statistic -2.371  P-value 0.395  Lags 60  -------------------------------------  Trend: Constant and Linear Time Trend  Critical Values: -3.96 (1%), -3.41 (5%), -3.13 (10%)  Null Hypothesis: The process contains a unit root.  Alternative Hypothesis: The process is weakly stationary. |
|  |
| kpss = KPSS(y=dataset["Close"], lags=60, trend="ct")  print(kpss.summary().as\_text()) |
|  |
| KPSS Stationarity Test Results  =====================================  Test Statistic 0.241  P-value 0.006  Lags 60  -------------------------------------  Trend: Constant and Linear Time Trend  Critical Values: 0.22 (1%), 0.15 (5%), 0.12 (10%)  Null Hypothesis: The process is weakly stationary.  Alternative Hypothesis: The process contains a unit root. |

|  |
| --- |
| **K04-Model-Predictions.ipynb** |
|  |
| # lib manipulations time  import time as tm    # load all functions  from C01\_data\_collection import \*  from C02\_visualization import \*  from C03\_preprocessing import \*  from C04\_model\_predictions import \*  from C05\_model\_evaluate import \* |
|  |
| # set random number  import random as rm  rm.seed(1234)    # set random number  import numpy as np  np.random.seed(1234)    # set random number  import tensorflow as tf  tf.random.set\_seed(1234) |
|  |
| # load dataset  dataset = data\_collection("BTC-USD.csv") |
|  |
| # show dataset  print(dataset.tail()) |
|  |
| Date Open High Low Close  3403 2024-04-26 64485.371094 64789.656250 63322.398438 63755.320313  3404 2024-04-27 63750.988281 63898.363281 62424.718750 63419.140625  3405 2024-04-28 63423.515625 64321.484375 62793.597656 63113.230469  3406 2024-04-29 63106.363281 64174.878906 61795.457031 63841.121094  3407 2024-04-30 63839.417969 64703.332031 59120.066406 60636.855469 |

|  |
| --- |
| # EDA for timeseries  timeseries\_matplotlib(dataset, ["Open", "High", "Low", "Close"]) |
|  |
|  |
| Gambar x. Output program |
|  |
| # call func preprocessing  scaler, scaled, x\_train, y\_train, x\_test, y\_test = preprocessing(dataset) |
|  |
| # check results  print(x\_train.shape, y\_train.shape) |
|  |
| (2666, 60, 1) (2666,) |
|  |
| # check results  print(x\_test.shape, y\_test.shape) |
|  |
| (622, 60, 1) (622,) |
|  |
| # results preprocessing data  lineplot\_matplotlib1(    x=dataset[["Date"]].iloc[len(y\_train)+120:], y=y\_test, label="Close Price",  title="Results of Preprocessing Data",  ) |
|  |
|  |
| Gambar x. Output program |

|  |
| --- |
| #### 4.1 Algorithms SBi-LSTM-RNN |
|  |
| # measuring execution time  start\_time = tm.time()    # set algorithms  algorithms = "SBi-LSTM-RNN"    # results predictions  history, predictions = get\_models(algorithms, x\_train, y\_train, x\_test, y\_test)    # measuring execution time  end\_time = tm.time()    # calculating the total execution time  execution\_time = end\_time - start\_time |
|  |
| # show loss function  lineplot\_matplotlib3(    title="Results Loss Function Using SBi-LSTM-RNN",    x1=history.epoch, y1=history.history['loss'], label1="Training",    x2=history.epoch, y2=history.history['val\_loss'], label2="Validation",  ) |
|  |
|  |
| Gambar x. Output program |
|  |
| # show predictions  lineplot\_matplotlib2(    title="Results Prediction Using SBi-LSTM-RNN",    x1=dataset[["Date"]].iloc[len(y\_train)+120:], y1=inverse(scaler=scaler,scaled=y\_test),  label1="actual data",    x2=dataset[["Date"]].iloc[len(y\_train)+120:], y2=inverse(scaler=scaler,scaled=predictions),  label2="results predictions",  ) |
|  |
|  |
| Gambar x. Output program |
|  |
| # calculate eror  r, p\_value, mae, rmse, mape = evaluate\_models(    inverse(scaler=scaler,scaled=y\_test), inverse(scaler=scaler,scaled=predictions))    # show eror  print("Evaluate Models with : "+str(algorithms))  print("-------------------------------")  print("R       : "+str(r))  print("P-value : "+str(p\_value))  print("MAE     : "+str(mae))  print("RMSE    : "+str(rmse))  print("MAPE    : "+str(mape))  print("Time    : "+"{:,.2f}".format(execution\_time)) |
|  |
| Evaluate Models with : SBi-LSTM-RNN  -------------------------------  R : 0.9962  P-value : 0.0  MAE : 1149.0806  RMSE : 1696.8877  MAPE : 0.0351  Time : 263.30 |

|  |
| --- |
| #### 4.2 Algorithms SBi-GRU-RNN |
|  |
| # measuring execution time  start\_time = tm.time()    # set algorithms  algorithms = "SBi-GRU-RNN"    # results predictions  history, predictions = get\_models(algorithms, x\_train, y\_train, x\_test, y\_test)    # measuring execution time  end\_time = tm.time()    # calculating the total execution time  execution\_time = end\_time - start\_time |
|  |
| # show loss function  lineplot\_matplotlib3(    title="Results Loss Function Using SBi-GRU-RNN",    x1=history.epoch, y1=history.history['loss'], label1="Training",    x2=history.epoch, y2=history.history['val\_loss'], label2="Validation",  ) |
|  |
|  |
| Gambar x. Output program |
|  |
| # show predictions  lineplot\_matplotlib2(    title="Results Prediction Using SBi-GRU-RNN",    x1=dataset[["Date"]].iloc[len(y\_train)+120:], y1=inverse(scaler=scaler,scaled=y\_test),  label1="actual data",    x2=dataset[["Date"]].iloc[len(y\_train)+120:], y2=inverse(scaler=scaler,scaled=predictions),  label2="results predictions",  ) |
|  |
|  |
| Gambar x. Output program |
|  |
| # calculate eror  r, p\_value, mae, rmse, mape = evaluate\_models(    inverse(scaler=scaler,scaled=y\_test),    inverse(scaler=scaler,scaled=predictions)  )    # show eror  print("Evaluate Models with : "+str(algorithms))  print("-------------------------------")  print("R       : "+str(r))  print("P-value : "+str(p\_value))  print("MAE     : "+str(mae))  print("RMSE    : "+str(rmse))  print("MAPE    : "+str(mape))  print("Time    : "+"{:,.2f}".format(execution\_time)) |
|  |
| Evaluate Models with : SBi-GRU-RNN  -------------------------------  R : 0.9975  P-value : 0.0  MAE : 1077.2294  RMSE : 1575.4796  MAPE : 0.0299  Time : 268.31 |

|  |
| --- |
| # lib statistic  import scipy.stats as sc    # lib manipulation dataset  import pandas as pd  import numpy as np    # lib data visualization  import seaborn as sns  import matplotlib.pyplot as plt    # lib min-max scaler  from sklearn.preprocessing import MinMaxScaler |
|  |
| # load dataset  dataset = pd.read\_excel("dataset/hasil\_penelitian.xlsx", sheet\_name="hasil evaluasi")  print(np.round(dataset[:5],3)) |
|  |
| LSTM-R LSTM-MAE LSTM-RMSE LSTM-MAPE GRU-R GRU-MAE GRU-RMSE GRU-MAPE  0 0.996 0.013 0.019 0.031 0.997 0.016 0.025 0.033  1 0.996 0.015 0.020 0.036 0.998 0.014 0.021 0.028  2 0.997 0.012 0.020 0.026 0.998 0.014 0.022 0.028  3 0.997 0.013 0.021 0.026 0.997 0.009 0.014 0.022  4 0.996 0.015 0.020 0.037 0.998 0.012 0.018 0.025 |
|  |
| # define boxplot  fig, ax = plt.subplots(figsize=(8,4))  ax.boxplot(dataset[["LSTM-MAPE", "GRU-MAPE"]], labels=['LSTM', 'GRU'], patch\_artist=True, widths=(0.5, 0.5))    # set labels  ax.set\_title('Perbandingan MAPE Model LSTM dan GRU')  ax.set\_xlabel('')  ax.set\_ylabel('')  ax.grid(True)    # show boxplot  plt.show() |
|  |
|  |
| Gambar x. Output program |

|  |
| --- |
| # min-max scaler  scaler = MinMaxScaler(feature\_range=(0,1))    # process min-max  LSTM\_R = scaler.fit\_transform(np.array(dataset["LSTM-R"]).reshape(-1,1))  LSTM\_MAPE = scaler.fit\_transform(np.array(dataset["LSTM-MAPE"]).reshape(-1,1))    # process min-max  GRU\_R = scaler.fit\_transform(np.array(dataset["GRU-R"]).reshape(-1,1))  GRU\_MAPE = scaler.fit\_transform(np.array(dataset["GRU-MAPE"]).reshape(-1,1)) |
|  |
| - Normality Test with Shapiro-Wilk |
| # Hipotesa Awal  # H0 = Nilai R dan MAPE terdistribusi normal  # H2 = Nilai R dan MAPE TIDAK terdistribusi normal    # Interpretasi hasil:  # - p-value > 0.05: Data berdistribusi normal  # - p-value <= 0.05: Data tidak berdistribusi normal |
|  |
| # Hasil Shapiro-Wilk  print("Data tidak terdistribusi normal")  print("GRU-MAPE  :",np.round(sc.shapiro(GRU\_MAPE).pvalue,2))  print("LSTM-MAPE :",np.round(sc.shapiro(LSTM\_MAPE).pvalue,2)) |
|  |
| Data tidak terdistribusi normal  GRU-MAPE : 0.19  LSTM-MAPE : 0.31 |
|  |
| - Mann-Whitney Test |
| # Hipotesa Awal  # H0 = Maetode GRU lebih baik dari Metode LSTM  # H1 = Maetode GRU TIDAK lebih baik dari Metode LSTM    # Interpretasi hasil:  # p-value < 0.05 = Terima H0  # p-value > 0.05 = Terima H1 |
|  |
| # Mann-Whitney Test  MAPE = np.round(sc.mannwhitneyu(LSTM\_MAPE, GRU\_MAPE).pvalue,4)    # Intrepetasi hasil  if MAPE < 0.05:    print("Karena p-value:",MAPE,"maka Terima H0")  else :    print("Karena p-value:",MAPE,"maka Terima H1") |
| Karena p-value: [0.1857] maka Terima H1 |
|  |
|  |
| - Wilcoxon Rank Test |
| # Wilcoxon Rank Test  MAPE = np.round(sc.wilcoxon(LSTM\_MAPE, GRU\_MAPE).pvalue,4)    # Intrepetasi hasil  if MAPE < 0.05:    print("Karena p-value:",MAPE,"maka Terima H0")  else :    print("Karena p-value:",MAPE,"maka Terima H1") |
| Karena p-value: [0.3285] maka Terima H1 |
|  |
| - Kruskal-Wallis Test |
| # Kruskal-Wallis Test  MAPE = np.round(sc.kruskal(LSTM\_MAPE, GRU\_MAPE).pvalue,4)    # Intrepetasi hasil  if MAPE < 0.05:    print("Karena p-value:",MAPE,"maka Terima H0")  else :    print("Karena p-value:",MAPE,"maka Terima H1") |
| Karena p-value: [0.1833] maka Terima H1 |

**Selesai, Selamat Mencoba :3**