CHAPTER 4
DESIGN AND IMPLEMENTATION

4.1. Design

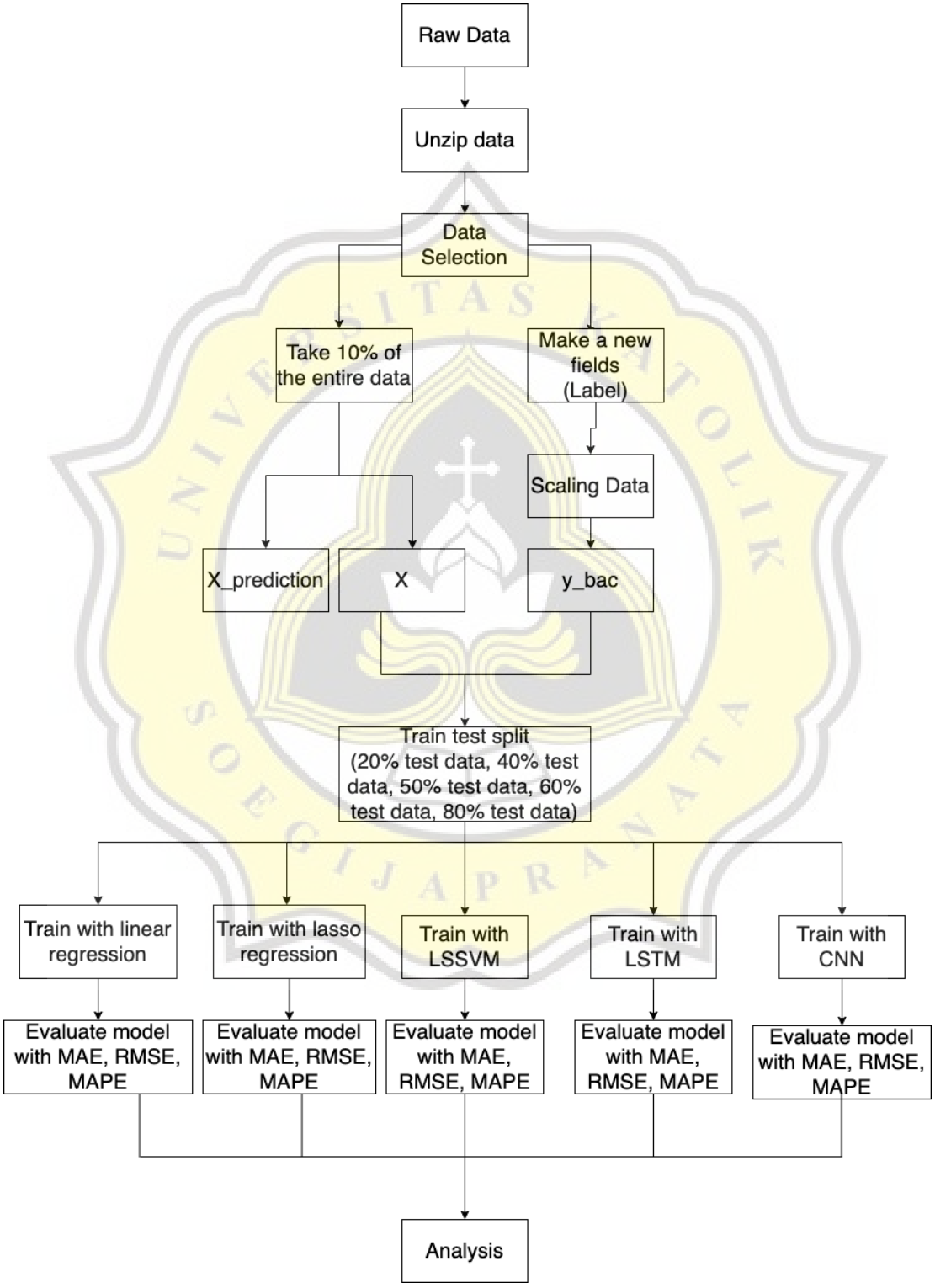

Figure 4.1 Flowchart

Figure 4.1 shows the flow of the proposed method applied in a single dataset. The flowchart is repeated three times with the same distribution of data for scenarios like those analyzed in this study, which contains three datasets. Following training and evaluation, comparison and analysis are performed. This stage informs the researcher on the most effective and correct model for this research.

4.2. Implementation

Preprocessing raw data is the first stage in constructing deep-learning and machine-learning approaches. The data preprocessing stage is when raw data is processed before being fed into machine-learning or deep-learning models.

4.2.1. Experiment Setup

This research employs Google Colab Pro with a RAM capacity of 4.81 GB, disk capacity of 22.92 GB, and Python version 3.8.15. Seaborn and matplotlib are used to visualize data, while tensorflow, keras, and scikit-learn are used for data processing such as calculating accuracy, constructing models, and scaling.

4.2.2. Data Visualization

```
1. bac = pd.read_csv('/content/Dataset/BAC.csv', parse_dates=['Date'],
   index_col='Date')
2. bac.head(10)
```

Date	Open	High	Low	Close	Adj Close	Volume
2017-01-03	22.600000	22.680000	22.200001	22.530001	20.000072	99298100
2017-01-04	22.719999	22.959999	22.600000	22.950001	20.372906	76875100
2017-01-05	22.820000	22.930000	22.350000	22.680000	20.133226	86826400
2017-01-06	22.780001	22.850000	22.559999	22.680000	20.133226	66281500
2017-01-09	22.510000	22.709999	22.400000	22.549999	20.017817	75901500
2017-01-10	22.590000	23.139999	22.540001	22.940001	20.364025	100977700
2017-01-11	22.940001	23.070000	22.719999	23.070000	20.479427	92385600
2017-01-12	23.010000	23.120001	22.610001	22.920000	20.346273	120474200
2017-01-13	23.209999	23.410000	22.799999	23.010000	20.426163	161930900
2017-01-17	22.680000	22.790001	22.010000	22.049999	19.573965	152495900

Figure 4.2 Dataset BAC

```
3. hdb = pd.read_csv('/content/Dataset/HDB.csv', parse_dates=['Date'],
   index_col='Date')
4. hdb.head(10)
```

Date	Open	High	Low	Close	Adj Close	Volume
2017-01-03	30.330000	30.434999	29.500000	29.590000	28.624722	3046600
2017-01-04	29.725000	29.940001	29.555000	29.594999	28.629559	2271600
2017-01-05	29.700001	30.495001	29.610001	30.450001	29.456667	3102400
2017-01-06	30.410000	30.545000	30.270000	30.340000	29.350258	1791400
2017-01-09	30.200001	30.545000	30.180000	30.365000	29.374441	1927400
2017-01-10	30.900000	31.135000	30.775000	30.945000	29.935520	1577000
2017-01-11	31.344999	32.099998	31.245001	31.990000	30.946428	3323600
2017-01-12	32.025002	32.290001	31.834999	32.060001	31.014149	2530800
2017-01-13	32.049999	32.195000	31.684999	32.060001	31.014149	2249200
2017-01-17	32.130001	32.610001	32.119999	32.445000	31.386585	2360200

Figure 4.3 Dataset HDB

```
5. ry = pd.read_csv('/content/Dataset/RY.csv', parse_dates=['Date'],
   index_col='Date')
6. ry.head(10)
```

Date	Open	High	Low	Close	Adj Close	Volume
2017-01-03	67.919998	68.419998	67.849998	68.139999	55.062710	853600
2017-01-04	68.519997	69.900002	68.519997	69.769997	56.379868	1719800
2017-01-05	69.760002	70.650002	69.760002	70.080002	56.630371	1183100
2017-01-06	70.139999	70.339996	69.860001	70.070000	56.622299	919000
2017-01-09	69.830002	70.089996	69.449997	69.940002	56.517262	712500
2017-01-10	70.099998	70.459999	70.059998	70.309998	56.816246	870900
2017-01-11	70.279999	71.330002	70.099998	71.290001	57.608154	1195400
2017-01-12	71.489998	71.519997	70.800003	71.410004	57.705135	1082300
2017-01-13	71.500000	72.239998	71.410004	71.959999	58.149582	1224600
2017-01-17	71.900002	72.169998	71.540001	71.650002	57.899075	1004600

Figure 4.4 Dataset RY

Figure 4.2, Figure 4.3, Figure 4.4 indicates that each dataset originated with just 7 columns: Volume, Adj Close, Close, Low, High, Open, and Date. The progression of the ups and downs in pricing for each field in each dataset can be seen in Figure 4.5, Figure 4.6, Figure 4.7. Lines 1,3,5 are used to import datasets, while lines 2,4,6 are used to view the top 10 data in each dataset.

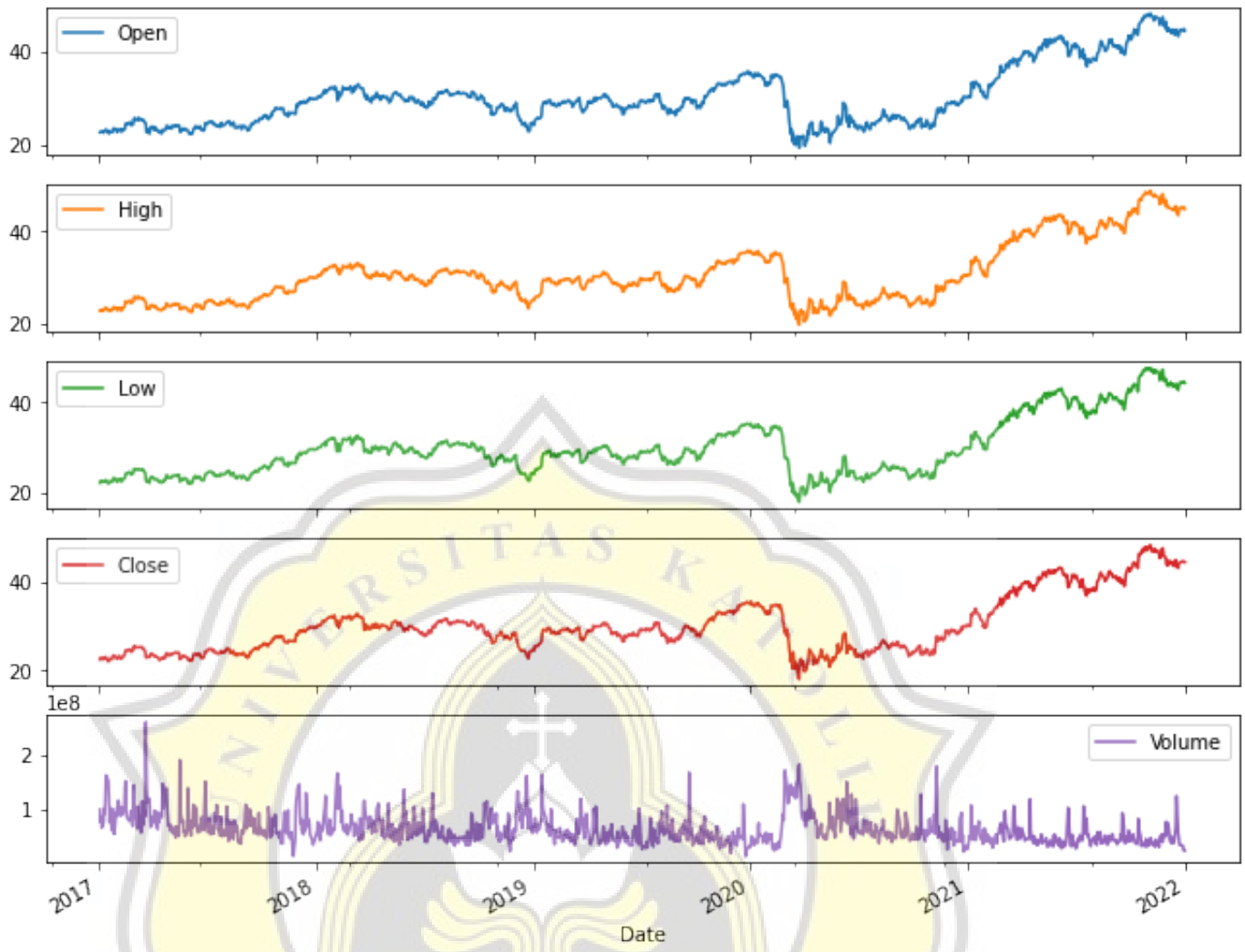

Figure 4.5 Graphic of BAC

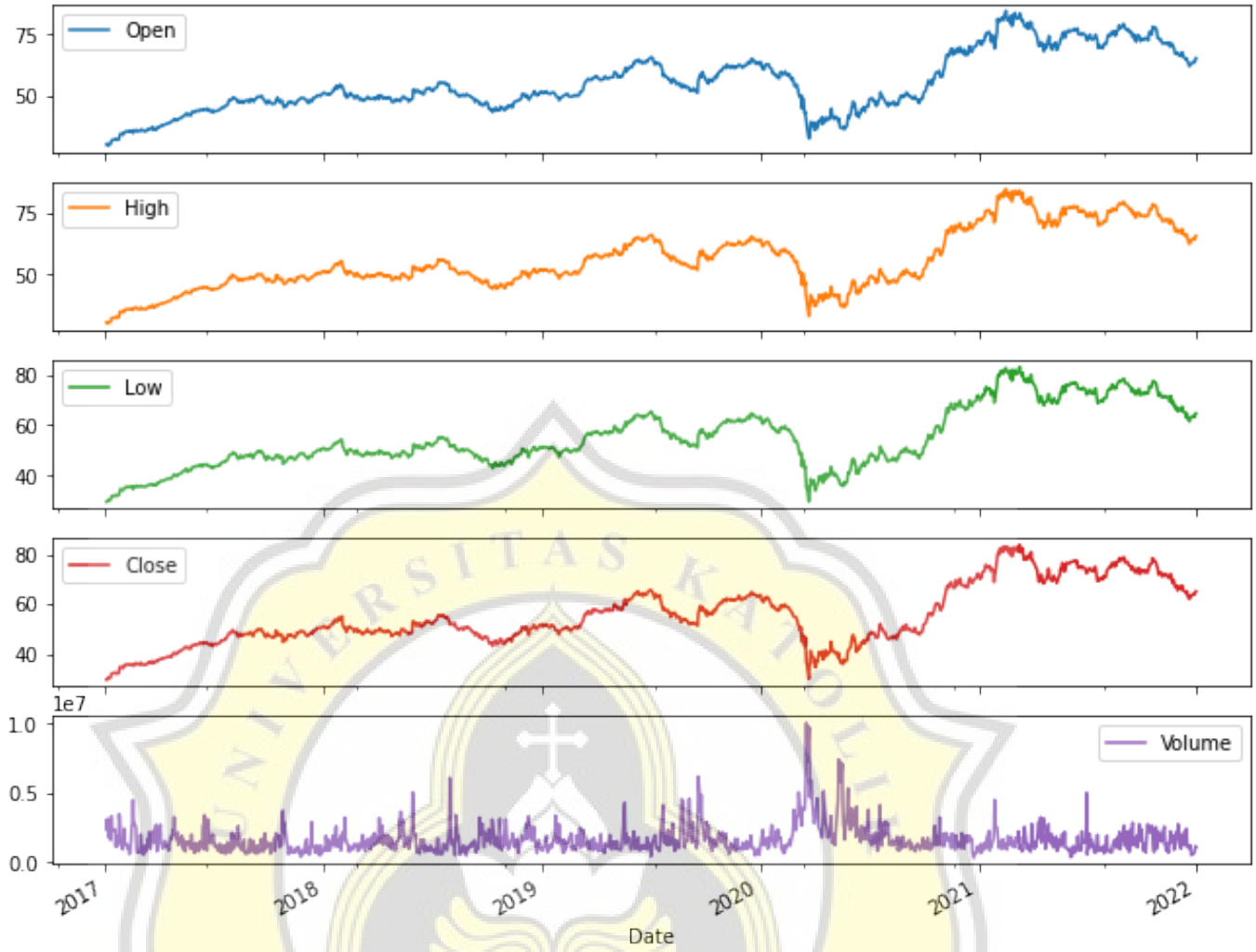

Figure 4.6 Graphic of HDB

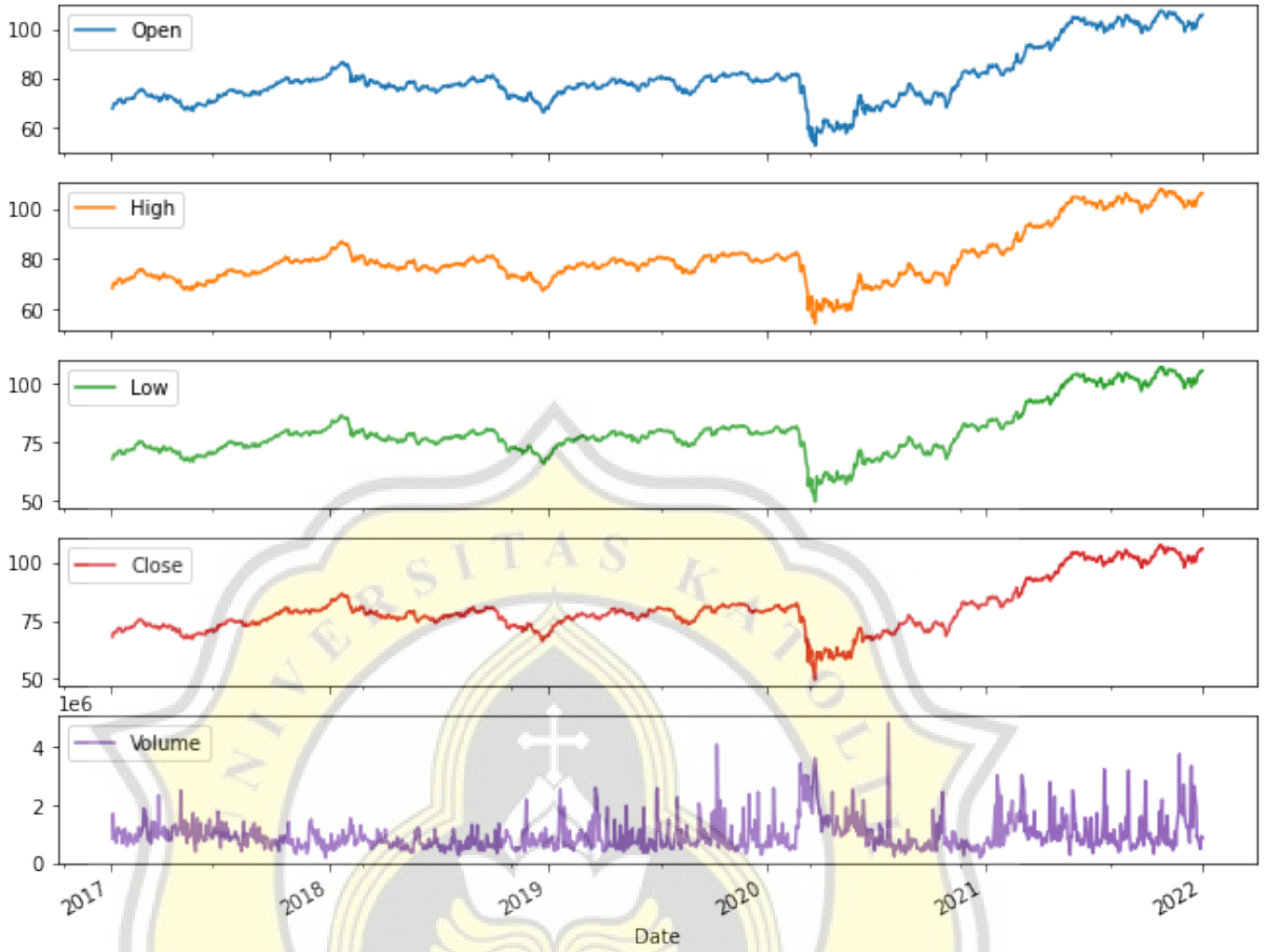

Figure 4.7 Graphic of RY

```
7. bac['Open:30 days rolling']=bac['Open'].rolling(30).mean()
8. bac['High:30 days rolling']=bac['High'].rolling(30).mean()
9. bac['Low:30 days rolling']=bac['Low'].rolling(30).mean()
10. bac['Close:30 days rolling']=bac['Close'].rolling(30).mean()
11. bac['Adj Close:30 days rolling']=bac['Adj Close'].rolling(30).mean()
12. bac['Volume:30 days rolling']=bac['Volume'].rolling(30).mean()
13.
14. bac[['Open','Open:30 days rolling']].plot(figsize=(12,5))
15. bac[['High','High:30 days rolling']].plot(figsize=(12,5))
16. bac[['Low','Low:30 days rolling']].plot(figsize=(12,5))
17. bac[['Close','Close:30 days rolling']].plot(figsize=(12,5))
18. bac[['Adj Close','Adj Close:30 days rolling']].plot(figsize=(12,5))
19. bac[['Volume','Volume:30 days rolling']].plot(figsize=(12,5))
```

```
20. hdb['Open:30 days rolling']=hdb['Open'].rolling(30).mean()
21. hdb['High:30 days rolling']=hdb['High'].rolling(30).mean()
22. hdb['Low:30 days rolling']=hdb['Low'].rolling(30).mean()
23. hdb['Close:30 days rolling']=hdb['Close'].rolling(30).mean()
24. hdb['Adj Close:30 days rolling']=hdb['Adj Close'].rolling(30).mean()
25. hdb['Volume:30 days rolling']=hdb['Volume'].rolling(30).mean()
26.
```

```
hdb[['Open','Open:30 days rolling']].plot(figsize=(12,5))
hdb[['High','High:30 days rolling']].plot(figsize=(12,5))
hdb[['Low','Low:30 days rolling']].plot(figsize=(12,5))
hdb[['Close','Close:30 days rolling']].plot(figsize=(12,5))
hdb[['Adj Close','Adj Close:30 days rolling']].plot(figsize=(12,5))
hdb[['Volume','Volume:30 days rolling']].plot(figsize=(12,5))
```

```
ry['Open:30 days rolling']=ry['Open'].rolling(30).mean()
ry['High:30 days rolling']=ry['High'].rolling(30).mean()
ry['Low:30 days rolling']=ry['Low'].rolling(30).mean()
ry['Close:30 days rolling']=ry['Close'].rolling(30).mean()
ry['Adj Close:30 days rolling']=ry['Adj Close'].rolling(30).mean()
ry['Volume:30 days rolling']=ry['Volume'].rolling(30).mean()

ry[['Open','Open:30 days rolling']].plot(figsize=(12,5))
ry[['High','High:30 days rolling']].plot(figsize=(12,5))
ry[['Low','Low:30 days rolling']].plot(figsize=(12,5))
ry[['Close','Close:30 days rolling']].plot(figsize=(12,5))
ry[['Adj Close','Adj Close:30 days rolling']].plot(figsize=(12,5))
ry[['Volume','Volume:30 days rolling']].plot(figsize=(12,5))
```

The results of averaging 30 of earlier periods in the time series are shown in Figure 4.8. The idea behind this rolling window is to determine the size k (must be consecutive at all times) all at once using many mathematical processes. This rolling window allows for the smoothing of minor fluctuation data into high fluctuation rates. Lines 7 to 12, 20 to 25, 33 to 38 are average shift calculations for the past 30 periods. Lines 14 to 19, 27 to 32, and 40 to 45 are utilized to display the results of the shift in each field.

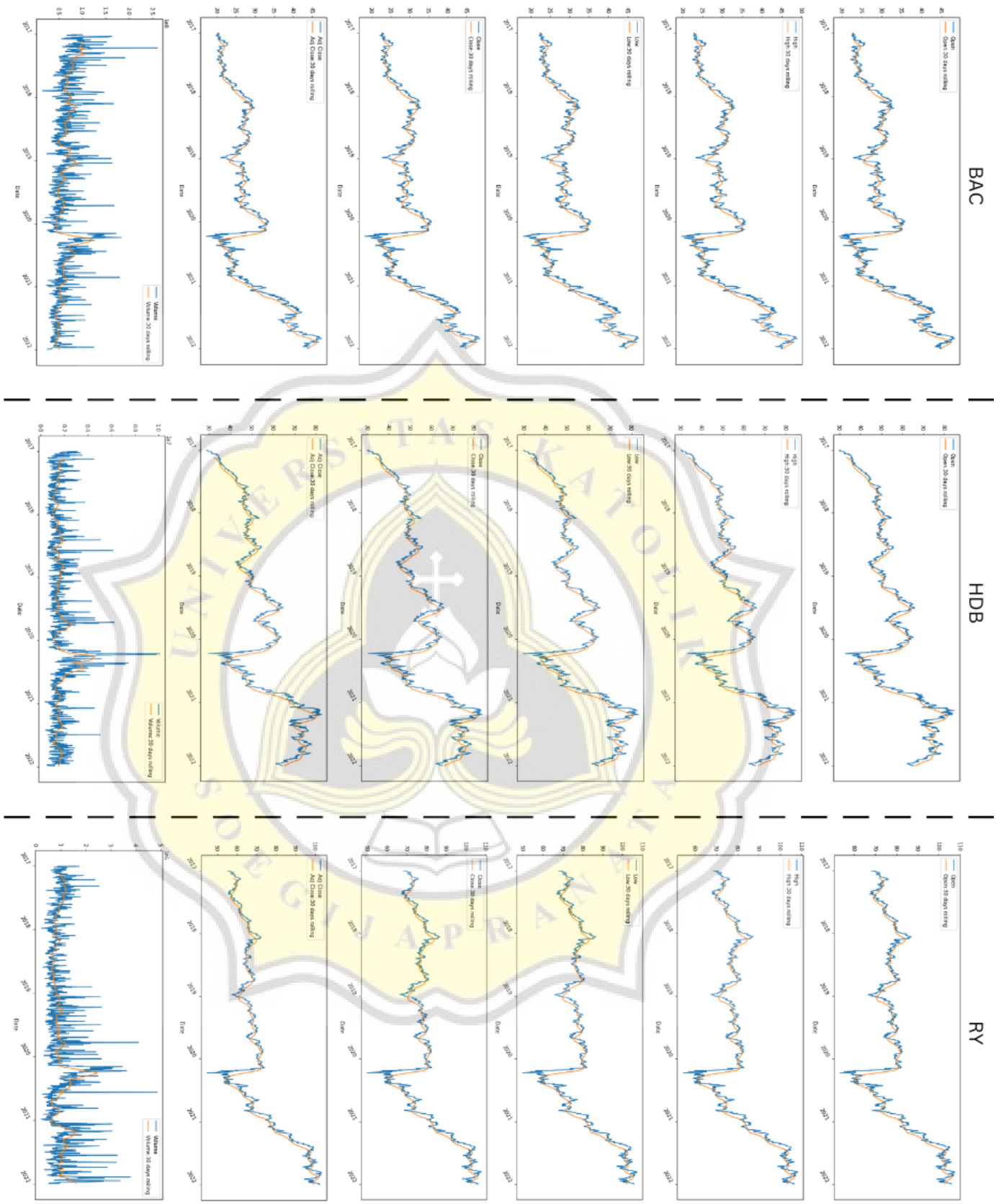

Figure 4.8 Rolling 30 days

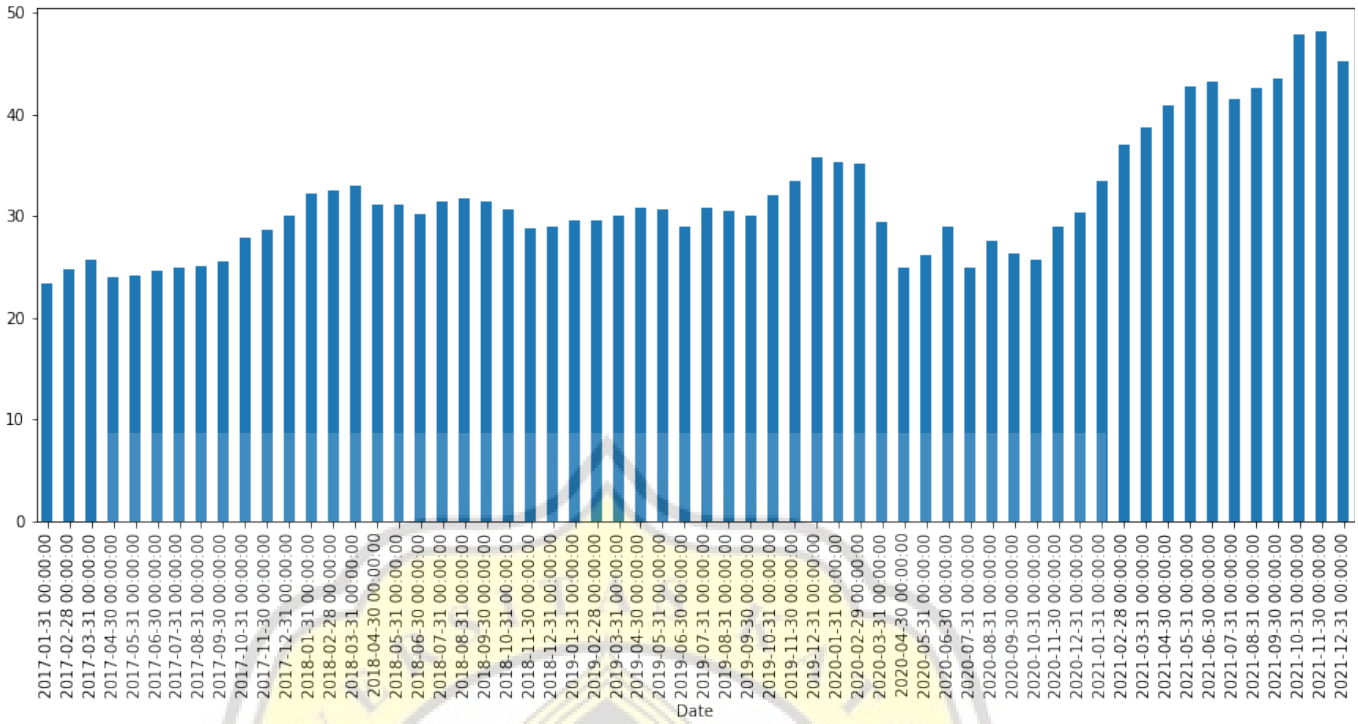

Figure 4.9 Open max value of BAC

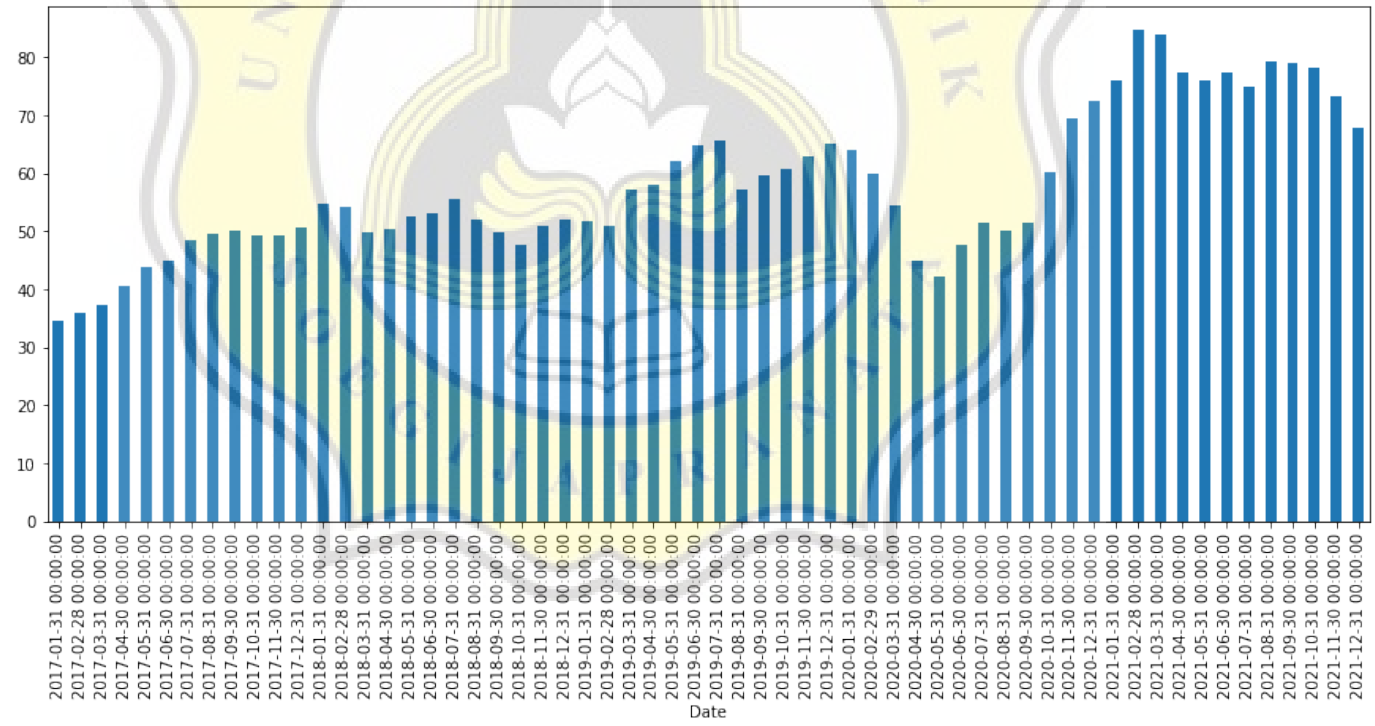

Figure 4.10 Open max value of HDB

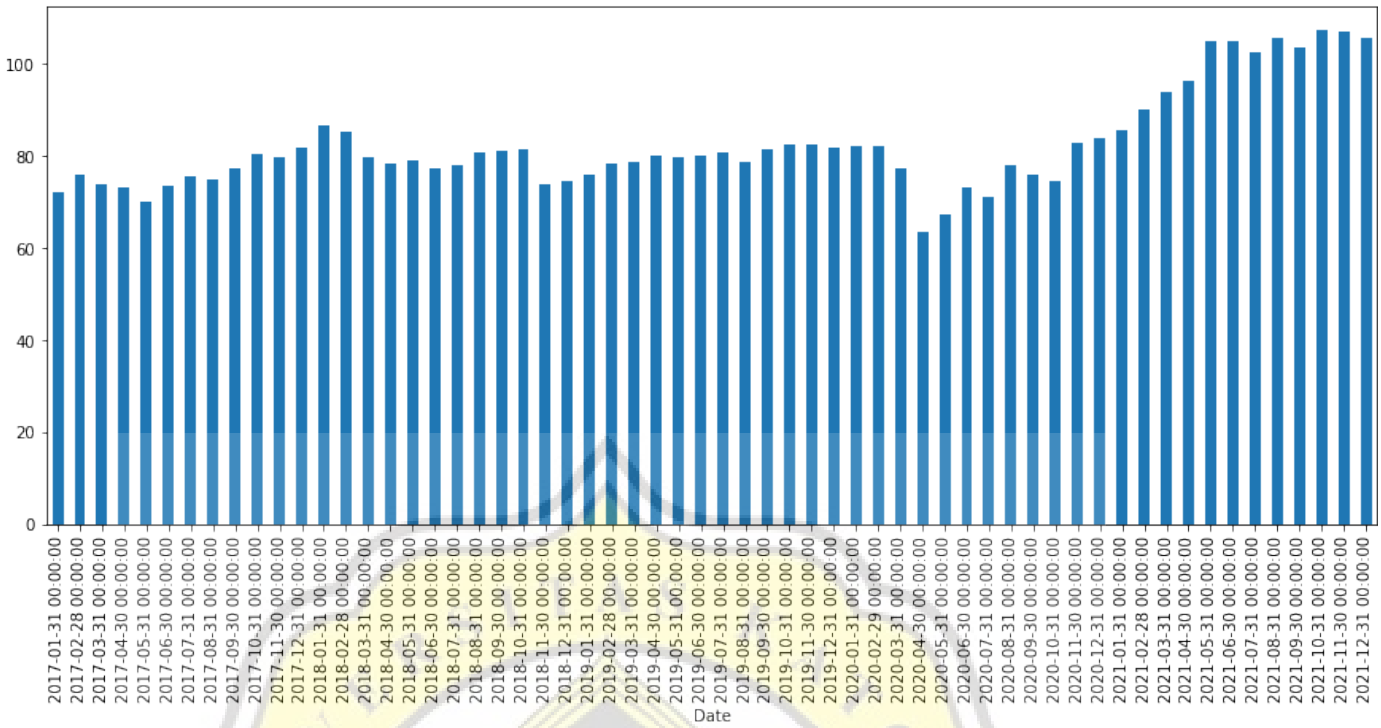

Figure 4.11 Open max value of RY

```
46. bac['Open'].resample(rule='M').max().plot(kind='bar',figsize=(15,6))
47. hdb['Open'].resample(rule='M').max().plot(kind='bar',figsize=(15,6))
48. ry['Open'].resample(rule='M').max().plot(kind='bar',figsize=(15,6))
```

According to the dataset presented in the previous section, Figure 4.9, Figure 4.10, Figure 4.11 shows the maximum value of the open price for each month from 2017 to 2021. Lines 46 to 48 display monthly open pricing data for each dataset.

```
49. bac['delta_open_close_day_before_%']=((bac['Open']-
   bac['Close'].shift(1))/bac['Close'].shift(1))*100
50. order = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"]
51. box_plot    =    sns.boxplot(x=pd.to_datetime(bac.index).day_name(),
   y=bac['delta_open_close_day_before_%'], order=order)
52.
53. ax = box_plot.axes
54. lines = ax.get_lines()
55. categories = ax.get_xticks()
56.
57. for cat in categories:
58.     y = round(lines[4+cat*6].get_ydata()[0],1)
59.
60.     ax.text(
61.         cat,
62.         y,
63.         f'{y}',
64.         ha='center',
65.         va='center',
66.         fontweight='bold',
67.         size=10,
68.         color='white',
```

```
        bbox=dict(facecolor='#445A64'))

box_plot.figure.tight_layout()


hdb['delta_open_close_day_before_%']=((hdb['Open']-
hdb['Close'].shift(1))/hdb['Close'].shift(1))*100
order = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"]
box_plot    =    sns.boxplot(x=pd.to_datetime(hdb.index).day_name(),
y=hdb['delta_open_close_day_before_%'], order=order)

ax = box_plot.axes
lines = ax.get_lines()
categories = ax.get_xticks()

for cat in categories:
    y = round(lines[4+cat*6].get_ydata()[0],1)

    ax.text(
        cat,
        y,
        f'{y}',
        ha='center',
        va='center',
        fontweight='bold',
        size=10,
        color='white',
        bbox=dict(facecolor='#445A64'))

box_plot.figure.tight_layout()


ry['delta_open_close_day_before_%']=((ry['Open']-
ry['Close'].shift(1))/ry['Close'].shift(1))*100
order = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"]
box_plot    =    sns.boxplot(x=pd.to_datetime(ry.index).day_name(),
y=ry['delta_open_close_day_before_%'], order=order)

ax = box_plot.axes
lines = ax.get_lines()
categories = ax.get_xticks()

for cat in categories:
    y = round(lines[4+cat*6].get_ydata()[0],1)

    ax.text(
        cat,
        y,
        f'{y}',
        ha='center',
        va='center',
        fontweight='bold',
        size=10,
        color='white',
        bbox=dict(facecolor='#445A64'))

box_plot.figure.tight_layout()
```

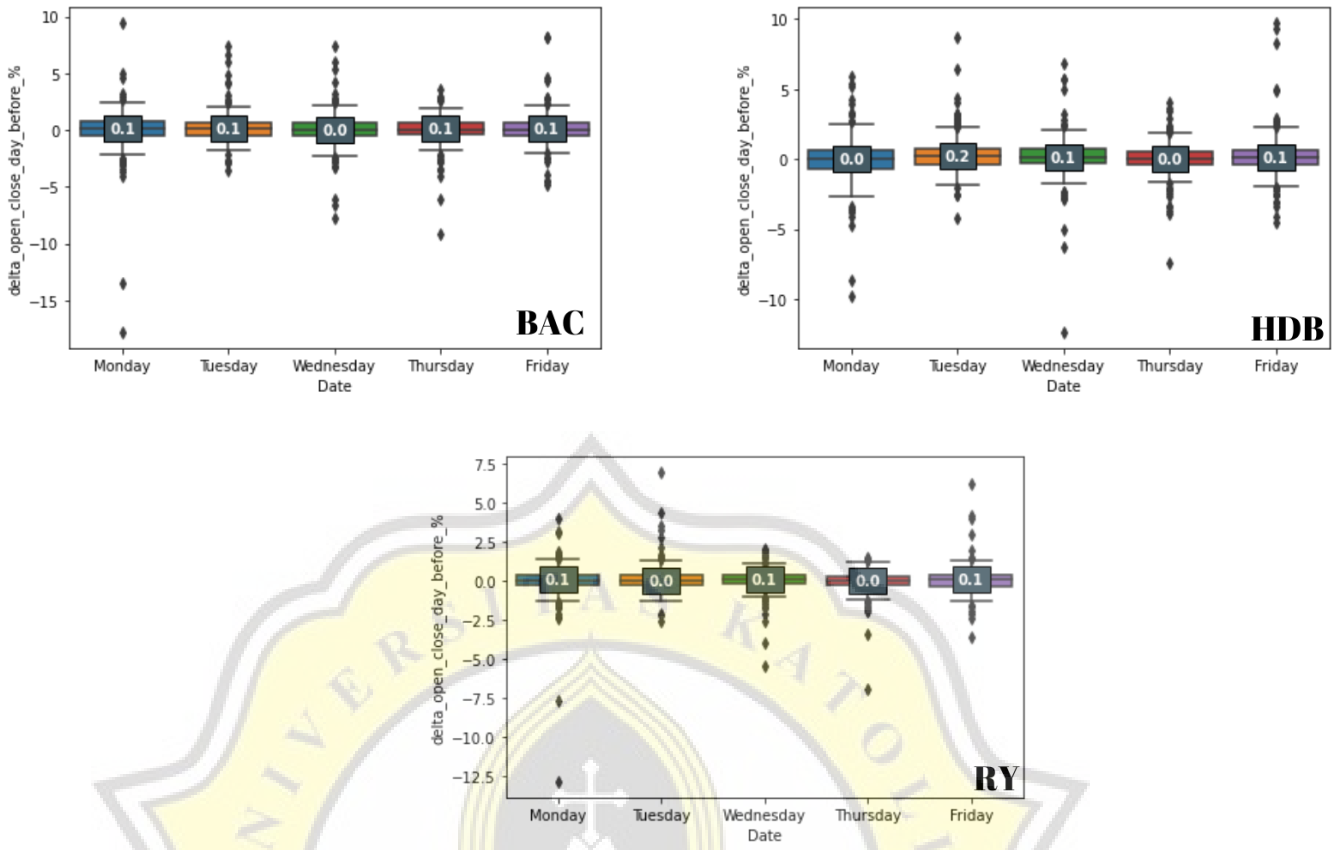

Figure 4.12 Gap between close price and open price

From Figure 4.12 depicts the difference between the close price and the open price the following day, categorized by day. As seen in the graphic above, the BAC dataset has the greatest value, 0.1, on Monday, Tuesday, Thursday, and Friday, implying that selling stocks at the start of that day may be more beneficial. Whereas Tuesdays may be more profitable days to sell shares in the HDB dataset. Mondays, Wednesdays, and Fridays may be more profitable in the RY dataset. Lines 49, 72, and 95 calculate the difference between the previous day's closing price and the first price the next day. Lines 51 to 71, 74 to 94, and 97 to 117, on the other hand, are used to depict the margin and are classified by day.

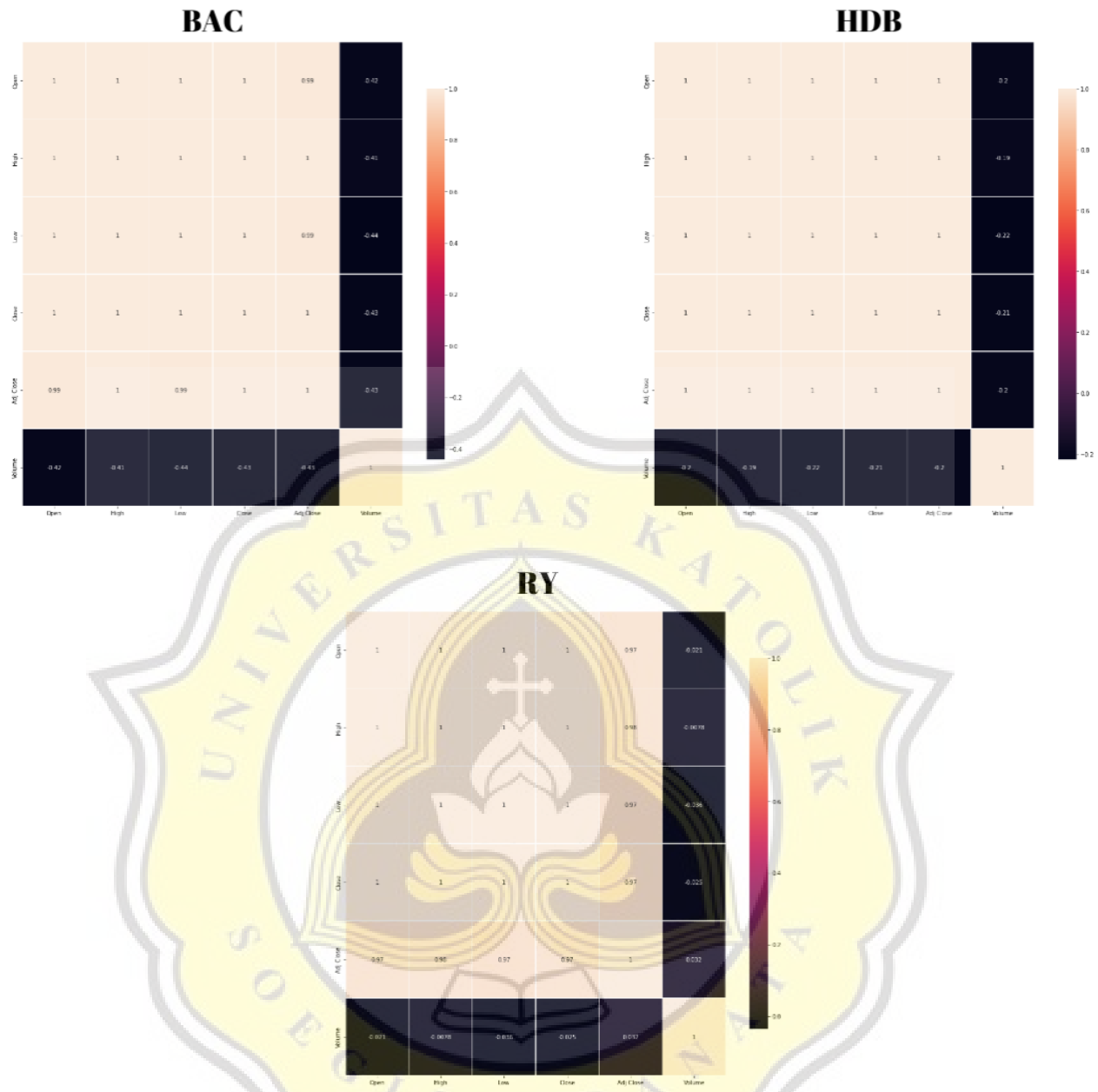

Figure 4.13 Heatmap of Dataset

Figure 4.13 shows how each variable is connected with one another; a score of 0 implies no correlation, whereas a score of 1 implies variable has a substantial positive link. This implies that when the value grows, so do the other values with other variables, however a negative value indicates that the value is negatively correlated with other values, which means that as the variable value increases, so do the other variable values since the correlation is negative.

4.2.3. Data Selection

```
118.bac['HL_PCT'] = (bac['High'] - bac['Low']) / bac['Low'] * 100.0
119.bac['PCT_change'] = (bac['Close'] - bac['Open']) / bac['Open'] * 100.0
```

```
120.hdb['HL_PCT'] = (hdb['High'] - hdb['Low']) / hdb['Low'] * 100.0
121.hdb['PCT_change'] = (hdb['Close'] - hdb['Open']) / hdb['Open'] * 100.0

122.ry['HL_PCT'] = (ry['High'] - ry['Low']) / ry['Low'] * 100.0
123.ry['PCT_change'] = (ry['Close'] - ry['Open']) / ry['Open'] * 100.0
```

Looking at the previous section's heatmap data, we can observe that Open, High, Low, Close, and Adj Close have the strongest correlation. As a result, the researcher devised two new variables: HL PCT (the proportion of high and low prices per day) and PCT Change (the percentage of open and closed prices per day). The fraction of high and low prices every day is calculated on lines 118, 120, and 122. Meanwhile, the percentages of open and closing prices every day are represented by 119, 121, and 123. As a result, the data will look like Figure 4.14.

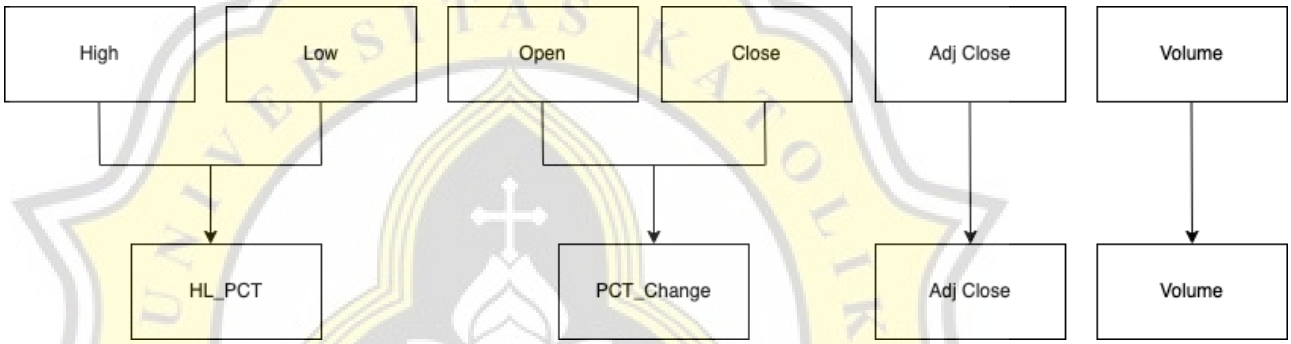

Figure 4.14 Data Selection

4.2.4. Feature Scaling

```
124.bac = bac[['HL_PCT', 'PCT_change', 'Adj Close','Volume']]
125.bac_forecast_out = int(math.ceil(0.1 * len(bac)))
126.bac['label'] = bac['Adj Close'].shift(-bac_forecast_out)

127.hdb = hdb[['HL_PCT', 'PCT_change', 'Adj Close','Volume']]
128.hdb_forecast_out = int(math.ceil(0.1 * len(hdb)))
129.hdb['label'] = hdb['Adj Close'].shift(-hdb_forecast_out)]

130.ry = ry[['HL_PCT', 'PCT_change', 'Adj Close','Volume']]
131.ry_forecast_out = int(math.ceil(0.1 * len(ry)))
132.ry['label'] = ry['Adj Close'].shift(-ry_forecast_out)

133.sc= MinMaxScaler(feature_range=(0,1))
134.bac_X = np.array(bac.drop(['label'], 1))
135.sc.fit(bac_X)
136.bac_X = sc.transform(bac_X)
137.
138.hdb_X = np.array(hdb.drop(['label'], 1))
139.sc.fit(hdb_X)
140.hdb_X = sc.transform(hdb_X)
141.
142.ry_X = np.array(ry.drop(['label'], 1))
143.sc.fit(ry_X)
```

```
144.ry_X = sc.transform(ry_X)
```

We took 10% of the data in each dataset from the selected data similar to line 125, 128, 131. We then added a new variable as seen on line 126, 129, 132 that is utilized to move the adj closing price as much as the dataset is taken. Following that, we scaled the data that we previously created because the preceding data will be inserted into the dataset in the same range as seen on line 133 to 144.

4.2.5. Split Data

```
145.bac20_X_train,   bac20_X_test,   bac20_y_train,   bac20_y_test   =
   train_test_split(X_bac, y_bac, test_size=0.2, random_state=42)
146.hdb20_X_train,   hdb20_X_test,   hdb20_y_train,   hdb20_y_test   =
   train_test_split(X_hdb, y_hdb, test_size=0.2, random_state=42)
147.ry20_X_train,    ry20_X_test,    ry20_y_train,    ry20_y_test    =
   train_test_split(X_ry, y_ry, test_size=0.2, random_state=42)

148.bac40_X_train,   bac40_X_test,   bac40_y_train,   bac40_y_test   =
   train_test_split(X_bac, y_bac, test_size=0.4, random_state=42)
149.hdb40_X_train,   hdb40_X_test,   hdb40_y_train,   hdb40_y_test   =
   train_test_split(X_hdb, y_hdb, test_size=0.4, random_state=42)
150.ry40_X_train,    ry40_X_test,    ry40_y_train,    ry40_y_test    =
   train_test_split(X_ry, y_ry, test_size=0.4, random_state=42)

151.bac60_X_train,   bac60_X_test,   bac60_y_train,   bac60_y_test   =
   train_test_split(X_bac, y_bac, test_size=0.6, random_state=42)
152.hdb60_X_train,   hdb60_X_test,   hdb60_y_train,   hdb60_y_test   =
   train_test_split(X_hdb, y_hdb, test_size=0.6, random_state=42)
153.ry60_X_train,    ry60_X_test,    ry60_y_train,    ry60_y_test    =
   train_test_split(X_ry, y_ry, test_size=0.6, random_state=42)

154.bac80_X_train,   bac80_X_test,   bac80_y_train,   bac80_y_test   =
   train_test_split(X_bac, y_bac, test_size=0.8, random_state=42)
155.hdb80_X_train,   hdb80_X_test,   hdb80_y_train,   hdb80_y_test   =
   train_test_split(X_hdb, y_hdb, test_size=0.8, random_state=42)
156.ry80_X_train,    ry80_X_test,    ry80_y_train,    ry80_y_test    =
   train_test_split(X_ry, y_ry, test_size=0.8, random_state=42)

157.bac50_X_train,   bac50_X_test,   bac50_y_train,   bac50_y_test   =
   train_test_split(X_bac, y_bac, test_size=0.5, random_state=42)
158.hdb50_X_train,   hdb50_X_test,   hdb50_y_train,   hdb50_y_test   =
   train_test_split(X_hdb, y_hdb, test_size=0.5, random_state=42)
159.ry50_X_train,    ry50_X_test,    ry50_y_train,    ry50_y_test    =
   train_test_split(X_ry, y_ry, test_size=0.5, random_state=42)
```

After successfully scaling the data, the data to be predicted is picked from 10% of the dataset, with the remainder being X data and the Y data containing array data from the adj closing price shift. The X and Y data will then be separated into five parts: 50% training data 50% test data as seen on line 157 to 159, 20% training data 80% is test data as seen on line 154 to 156, 40% is training data 60% is test data as seen on line 151 to 153, 60% is training data

40% is test data as seen on line 148 to 150, while the other 80% is training 20% is test data as seen on line 145 to 147. This is repeated for each dataset.

4.3. Deep-Learning Models

4.3.1. LSTM

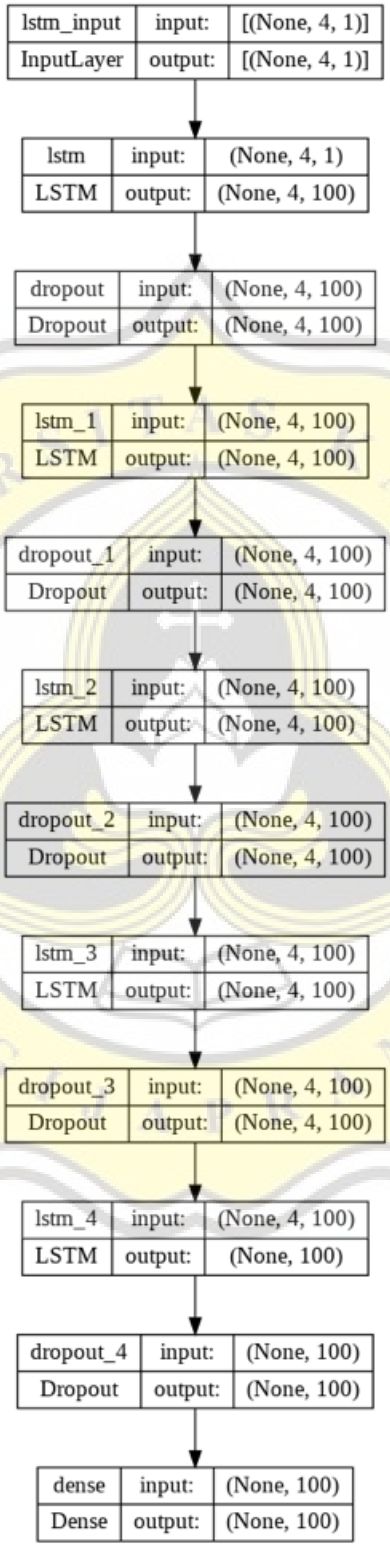

Figure 4.15 LSTM Model

```
model_bac_20= Sequential()
model_bac_20.add(LSTM(units=100,return_sequences=True,
   activation='relu', input_shape=(bac20_X_train.shape[1], 1)))
model_bac_20.add(Dropout(rate=0.2))

model_bac_20.add(LSTM(units=100,return_sequences=True,
   activation='relu'))
model_bac_20.add(Dropout(rate=0.01))

model_bac_20.add(LSTM(units=100,return_sequences=True,
   activation='relu'))
model_bac_20.add(Dropout(rate=0.002))

model_bac_20.add(LSTM(units=100,return_sequences=True,
   activation='relu'))
model_bac_20.add(Dropout(rate=0.02))

model_bac_20.add(LSTM(units=100, activation='relu'))
model_bac_20.add(Dropout(rate=0.02))

model_bac_20.add(Dense(units=100))
model_bac_20.compile(loss='mse',  optimizer='adam',  metrics=['mse',
   'mae', 'mape'])


model_bac_40= Sequential()

model_bac_40.add(LSTM(units=100,return_sequences=True,
   activation='relu', input_shape=(bac40_X_train.shape[1], 1)))
model_bac_40.add(Dropout(rate=0.2))

model_bac_40.add(LSTM(units=100,return_sequences=True,
   activation='relu'))
model_bac_40.add(Dropout(rate=0.01))

model_bac_40.add(LSTM(units=100,return_sequences=True,
   activation='relu'))
model_bac_40.add(Dropout(rate=0.002))

model_bac_40.add(LSTM(units=100,return_sequences=True,
   activation='relu'))
model_bac_40.add(Dropout(rate=0.02))

model_bac_40.add(LSTM(units=100, activation='relu'))
model_bac_40.add(Dropout(rate=0.02))

model_bac_40.add(Dense(units=100))

model_bac_40.compile(loss='mse',  optimizer='adam',  metrics=['mse',
   'mae', 'mape'])

model_bac_60= Sequential()

model_bac_60.add(LSTM(units=100,return_sequences=True,
   activation='relu', input_shape=(bac60_X_train.shape[1], 1)))
model_bac_60.add(Dropout(rate=0.2))

model_bac_60.add(LSTM(units=100,return_sequences=True,
   activation='relu'))
model_bac_60.add(Dropout(rate=0.01))

```

```
model_bac_60.add(LSTM(units=100,return_sequences=True,
   activation='relu'))
model_bac_60.add(Dropout(rate=0.002))

model_bac_60.add(LSTM(units=100,return_sequences=True,
   activation='relu'))
model_bac_60.add(Dropout(rate=0.02))

model_bac_60.add(LSTM(units=100, activation='relu'))
model_bac_60.add(Dropout(rate=0.02))

model_bac_60.add(Dense(units=100))

model_bac_60.compile(loss='mse',  optimizer='adam',  metrics=['mse',
   'mae', 'mape'])


model_bac_80= Sequential()

model_bac_80.add(LSTM(units=100,return_sequences=True,
   activation='relu', input_shape=(bac80_X_train.shape[1], 1)))
model_bac_80.add(Dropout(rate=0.2))

model_bac_80.add(LSTM(units=100,return_sequences=True,
   activation='relu'))
model_bac_80.add(Dropout(rate=0.01))

model_bac_80.add(LSTM(units=100,return_sequences=True,
   activation='relu'))
model_bac_80.add(Dropout(rate=0.002))

model_bac_80.add(LSTM(units=100,return_sequences=True,
   activation='relu'))
model_bac_80.add(Dropout(rate=0.02))

model_bac_80.add(LSTM(units=100, activation='relu'))
model_bac_80.add(Dropout(rate=0.02))

model_bac_80.add(Dense(units=4))

model_bac_80.compile(loss='mse',  optimizer='adam',  metrics=['mse',
   'mae', 'mape'])


model_bac_50= Sequential()

model_bac_50.add(LSTM(units=100,return_sequences=True,
   activation='relu', input_shape=(bac50_X_train.shape[1], 1)))
model_bac_50.add(Dropout(rate=0.2))

model_bac_50.add(LSTM(units=100,return_sequences=True,
   activation='relu'))
model_bac_50.add(Dropout(rate=0.01))

model_bac_50.add(LSTM(units=100,return_sequences=True,
   activation='relu'))
model_bac_50.add(Dropout(rate=0.002))

model_bac_50.add(LSTM(units=100,return_sequences=True,
   activation='relu'))
model_bac_50.add(Dropout(rate=0.02))
```

```
251.
252.model_bac_50.add(LSTM(units=100, activation='relu'))
253.model_bac_50.add(Dropout(rate=0.02))
254.
255.model_bac_50.add(Dense(units=100))
256.
257.model_bac_50.compile(loss='mse',  optimizer='adam',  metrics=['mse',
   'mae', 'mape'])
```

Figure 4.15 shows that there are 5 LSTM layers for one model as seen on line 160 to 177. Line 161, 180, 200, 220, 240 begins with the value "[(None, 4, 1)]" as input. This number 4 is derived from the value of the initial X train shape array, and the number of variables is steadily raised until an output with 100 neurons is found in line 173, 192, 212, 232, 252. This is performed for the remaining two datasets by separating the training and test data portions as indicated in the previous section.

4.3.2. CNN

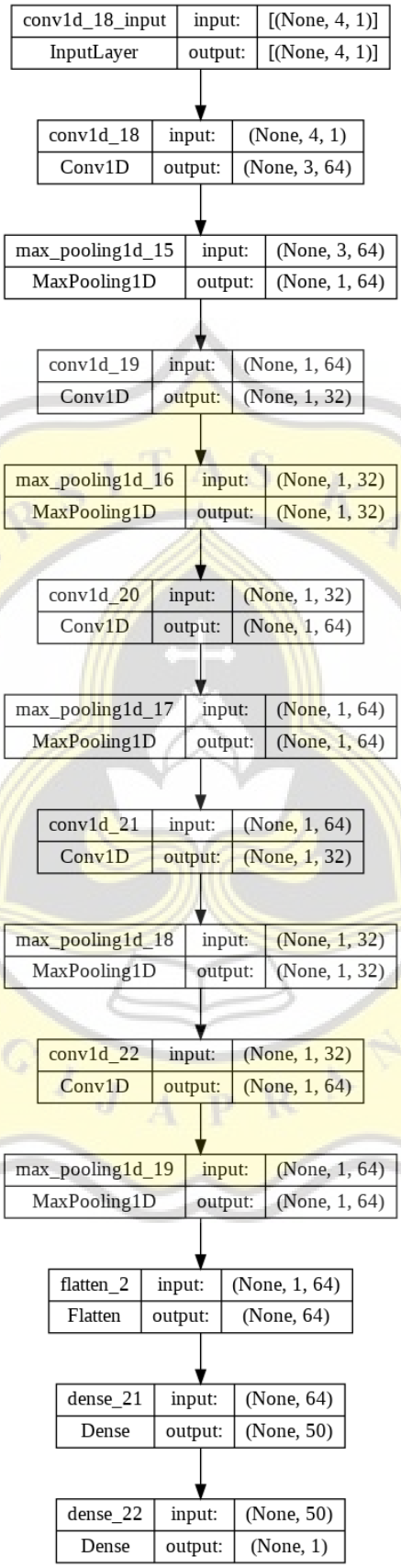

Figure 4.16 CNN Model

```
cnn_model_bac_20 = Sequential()
cnn_model_bac_20.add(Conv1D(filters=64,                kernel_size=2,
   activation='relu', input_shape=(bac20_X_train.shape[1], 1)))
cnn_model_bac_20.add(MaxPooling1D(pool_size=2))
cnn_model_bac_20.add(Conv1D(filters=32,                kernel_size=1,
   activation='relu'))
cnn_model_bac_20.add(MaxPooling1D(pool_size=1))
cnn_model_bac_20.add(Conv1D(filters=64,                kernel_size=1,
   activation='relu'))
cnn_model_bac_20.add(MaxPooling1D(pool_size=1))
cnn_model_bac_20.add(Conv1D(filters=32,                kernel_size=1,
   activation='relu'))
cnn_model_bac_20.add(MaxPooling1D(pool_size=1))
cnn_model_bac_20.add(Conv1D(filters=64,                kernel_size=1,
   activation='relu'))
cnn_model_bac_20.add(MaxPooling1D(pool_size=1))
cnn_model_bac_20.add(Flatten())
cnn_model_bac_20.add(Dense(50, activation='relu'))
cnn_model_bac_20.add(Dense(1))
cnn_model_bac_20.compile(loss='mse',              optimizer='adam',
   metrics=['mse', 'mae', 'mape'])


cnn_model_bac_40 = Sequential()
cnn_model_bac_40.add(Conv1D(filters=64,                kernel_size=2,
   activation='relu', input_shape=(bac40_X_train.shape[1], 1)))
cnn_model_bac_40.add(MaxPooling1D(pool_size=2))

cnn_model_bac_40.add(Conv1D(filters=32,                kernel_size=1,
   activation='relu'))
cnn_model_bac_40.add(MaxPooling1D(pool_size=1))

cnn_model_bac_40.add(Conv1D(filters=64,                kernel_size=1,
   activation='relu'))
cnn_model_bac_40.add(MaxPooling1D(pool_size=1))

cnn_model_bac_40.add(Conv1D(filters=32,                kernel_size=1,
   activation='relu'))
cnn_model_bac_40.add(MaxPooling1D(pool_size=1))

cnn_model_bac_40.add(Conv1D(filters=64,                kernel_size=1,
   activation='relu'))
cnn_model_bac_40.add(MaxPooling1D(pool_size=1))

cnn_model_bac_40.add(Flatten())
cnn_model_bac_40.add(Dense(50, activation='relu'))
cnn_model_bac_40.add(Dense(1))

cnn_model_bac_40.compile(loss='mse',              optimizer='adam',
   metrics=['mse', 'mae', 'mape'])


cnn_model_bac_60 = Sequential()
cnn_model_bac_60.add(Conv1D(filters=64,                kernel_size=2,
   activation='relu', input_shape=(bac60_X_train.shape[1], 1)))
cnn_model_bac_60.add(MaxPooling1D(pool_size=2))

cnn_model_bac_60.add(Conv1D(filters=32,                kernel_size=1,
   activation='relu'))
cnn_model_bac_60.add(MaxPooling1D(pool_size=1))
```

```
300.
301.cnn_model_bac_60.add(Conv1D(filters=64,                kernel_size=1,
   activation='relu'))
302.cnn_model_bac_60.add(MaxPooling1D(pool_size=1))
303.
304.cnn_model_bac_60.add(Conv1D(filters=32,                kernel_size=1,
   activation='relu'))
305.cnn_model_bac_60.add(MaxPooling1D(pool_size=1))
306.
307.cnn_model_bac_60.add(Conv1D(filters=64,                kernel_size=1,
   activation='relu'))
308.cnn_model_bac_60.add(MaxPooling1D(pool_size=1))
309.
310.cnn_model_bac_60.add(Flatten())
311.cnn_model_bac_60.add(Dense(50, activation='relu'))
312.cnn_model_bac_60.add(Dense(1))
313.
314.cnn_model_bac_60.compile(loss='mse',              optimizer='adam',
   metrics=['mse', 'mae', 'mape'])


315.cnn_model_bac_80 = Sequential()
316.cnn_model_bac_80.add(Conv1D(filters=64,                kernel_size=2,
   activation='relu', input_shape=(bac80_X_train.shape[1], 1)))
317.cnn_model_bac_80.add(MaxPooling1D(pool_size=2))
318.
319.cnn_model_bac_80.add(Conv1D(filters=32,                kernel_size=1,
   activation='relu'))
320.cnn_model_bac_80.add(MaxPooling1D(pool_size=1))
321.
322.cnn_model_bac_80.add(Conv1D(filters=64,                kernel_size=1,
   activation='relu'))
323.cnn_model_bac_80.add(MaxPooling1D(pool_size=1))
324.
325.cnn_model_bac_80.add(Conv1D(filters=32,                kernel_size=1,
   activation='relu'))
326.cnn_model_bac_80.add(MaxPooling1D(pool_size=1))
327.
328.cnn_model_bac_80.add(Conv1D(filters=64,                kernel_size=1,
   activation='relu'))
329.cnn_model_bac_80.add(MaxPooling1D(pool_size=1))
330.
331.cnn_model_bac_80.add(Flatten())
332.cnn_model_bac_80.add(Dense(50, activation='relu'))
333.cnn_model_bac_80.add(Dense(1))
334.
335.cnn_model_bac_80.compile(loss='mse',              optimizer='adam',
   metrics=['mse', 'mae', 'mape'])


336.cnn_model_bac_50 = Sequential()
337.cnn_model_bac_50.add(Conv1D(filters=64,                kernel_size=2,
   activation='relu', input_shape=(bac50_X_train.shape[1], 1)))
338.cnn_model_bac_50.add(MaxPooling1D(pool_size=2))
339.
340.cnn_model_bac_50.add(Conv1D(filters=32,                kernel_size=1,
   activation='relu'))
341.cnn_model_bac_50.add(MaxPooling1D(pool_size=1))
342.
```

```
343.cnn_model_bac_50.add(Conv1D(filters=64,                kernel_size=1,
   activation='relu'))
344.cnn_model_bac_50.add(MaxPooling1D(pool_size=1))
345.
346.cnn_model_bac_50.add(Conv1D(filters=32,                kernel_size=1,
   activation='relu'))
347.cnn_model_bac_50.add(MaxPooling1D(pool_size=1))
348.
349.cnn_model_bac_50.add(Conv1D(filters=64,                kernel_size=1,
   activation='relu'))
350.cnn_model_bac_50.add(MaxPooling1D(pool_size=1))
351.
352.cnn_model_bac_50.add(Flatten())
353.cnn_model_bac_50.add(Dense(50, activation='relu'))
354.cnn_model_bac_50.add(Dense(1))
355.
356.cnn_model_bac_50.compile(loss='mse',              optimizer='adam',
   metrics=['mse', 'mae', 'mape'])
```

This Figure 4.16 shows that the CNN model contains 5 layers as well. The input value is the same as the LSTM, "[(None,4,1)]," but the output value is different, namely if the CNN has 50 neurons as seen on 270, 290, 311, 332, 353.

4.4. Evaluation Models

```
357.earlyStop = EarlyStopping(monitor='loss', mode='min', verbose=1,
   patience=50)

358.def mae(y_test, y_pred):
359.   mae = metrics.mean_absolute_error(y_test, y_pred)
360.   return mae

361.def rmse(y_test, y_pred):
362.   rmse = np.sqrt(mean_squared_error(y_test, y_pred))
363.   return rmse

364.def mape(y_test, y_pred):
365.   mape = np.mean(np.abs((y_test - y_pred)/y_test))*100
366.   return mape
```

EarlyStopping is a Tensorflow library. Line 357 is used to reduce loss and thereby prevent data overfitting. Line 358 to 366 are used by the models in this study to evaluate data.