CHAPTER 4
RESEARCH IMPLEMENTATION AND RESULTS

In a study titled Evaluating Coal Stocks with ANN, Random Forest and ARIMA Model,researcher try to coal stocks and produce signal that will be beneficial for investor.Because coal is a commodity therefore it is considered as cyclical stock which we need to know the cycle.By studying the past data using machine learning,it will provide insight which model has the least error by evaluating these three models (ANN,Random Forest, and ARIMA).

4.1. Experimental Setup

The author uses Google Colaboratory to run machine learning.It is much convenient to process the dataset with the library in Google Colaboratory.The dataset will be connected through the Google Drive and it is much easier to modify if the future experiment needed.The writer uses Google Chrome to run Google Colaboratory that uses python 3.8.Libraries Used to create machine learning models and neural network

1. Numpy : 1.22.4
2. Pandas :1.4.4
3. Math :1.3.0
4. Datetime
5. Matplotlib : 3.7.1
6. Tensorflow : 2.11.0
7. Keras : 2.11.0
8. Tensorflow : 2.11.0
9. Scikit-learn : 1.2.2
10. Scikeras : 1.2.2

4.2. Implementation

4.2.1. Library Setup

```
1 from google.colab import drive
2 %cd /content/drive/MyDrive/IHSG
```

In line 1-2,this section declares how the dataset taken from which is the address from the Google Drive

```
# General Library
3 import numpy as np
4 import pandas as pd
5 import math
6 import datetime
7 import matplotlib.pyplot as plt

# Machine Learning Library Keras
8 from keras.models import Sequential
9 from keras.layers.core import Dense,Activation,Dropout

# Machine Learning Library  scikit-Learn
10 from sklearn.model_selection import train_test_split
11 from sklearn.ensemble import RandomForestClassifier
12 from sklearn.metrics import accuracy_score
13 from sklearn.metrics import confusion_matrix
14 from sklearn.metrics import mean_squared_error
15 from sklearn.metrics import mean_absolute_error
16 from sklearn.metrics import mean_absolute_percentage_error
17 from sklearn import preprocessing
18 from sklearn import utils
```

In line 3-18 is used to import all the libraries needed for this project.The general library is used to make the dataset compatible in the software.In line 8-9 is specifically used to import Keras Library that enable the software to create Neural Network.And the last library is scikit-Learn whereby it allow the author to preprocess the data and run algorithm.

4.2.2. Import Dataset

```
19 df = pd.read_csv("ADRO.JK.csv")
```

In line 19 is shown how to read the dataset file from the Google Drive folder.

```
20 df.head()
21 col_names = ['Date','Open','High','Low','Close','Volume','Adj Close']
22 stocks = pd.read_csv("ADRO.JK.csv",header=0,names=col_names)
23 df = pd.DataFrame(stocks)
24 date_split = df['Date'].str.split('-').str
25 df['Year'],df['Month'],df['Day'] = date_split
26 df["Volume"]=df["Volume"]/10000
27 df.describe()
```

In line 20-27 is shown how to define all the details from the dataset.

```
28 df.drop(df.columns[[0,3]],axis=1,inplace=True)
29 df.head()
30 df.describe()
```

```
31 df.dtypes
32 df.shape
33 df.info()
```

In line 28-33 is to describe how to create another dataset that will be trained including the details of data from the dataset.

```
34 nan_value_index = []
35 High = def.High.isnull()
36 for i in range(0, len(High)):
       37 if High[i] == 1:
              38 nan_value_index.append(i)
              39 df['High'][i] = 0
40 Open = df.Open.isnull()
41 for i in range(0, len(Open)):
         42 if Open[i] == 1:
              43 nan_value_index.append(i)
              44 df['Open'][i] = 0
45 Volume = df.Volume.isnull()
46 for i in range(0, len(Volume)):
         47 if Volume[i] == 1:
              48 nan_value_index.append(i)
              49 df['Volume'][i] = 0
50 Close = df.Close.isnull()
51 for i in range(0, len(Close)):
         52 if Close[i] == 1:
              53 nan_value_index.append(i)
              54 df['Close'][i] = 0
55 openMax= df['Open'].max()
56 openMin= df['Open'].min()
57 highMax= df['High'].max()
58 highMin= df['High'].min()
59 lowMax= df['Low'].max()
60 lowMin= df['Low'].min()
61 volumeMax= df['Volume'].max()
62 volumeMin= df['Volume'].min()
63 closeMax= df['Close'].max()
64 closeMin= df['Close'].min()
65 df['Open'] = (df['Open']-openMin)/(openMax-openMin)
66 df['High'] = (df['High']-highMin)/(highMax-highMin)
67 df['Low'] = (df['Low']-lowMin)/(lowMax-lowMin)
68 df['Volume'] = (df['Volume']-volumeMin)/(volumeMax-volumeMin)
69 df['Close'] = (df['Close']-closeMin)/(closeMax-closeMin)
70  df['Adj  Close']  =  (df['Adj  Close']-df['Adj  Close'].min())/(df['Adj
   71Close'].max()-df['Adj Close'].min())
72 print(df)
```

In line 34-72 is shown how to cleaning and normalize the data with Min-Max normalization.

```
73 df.head()
74 X = df[['Open','High','Adj Close','Volume']]
75 Y = df[['Close']]
76 X.head()
77 Y.head()

78 X = df[['High','Open', 'Volume']]
79 Y = df[['Close']]
80 factor = 0.80
81 length = X.shape[0]
82 total_for_train = int(length*factor)
83 X_train = X[:total_for_train]
84 Y_train = Y[:total_for_train]
85 X_test = X[total_for_train:]
86 Y_test = Y[total_for_train:]
87 X_train.head()
88 X_test.head()
89 Y_train.head()
90 Y_test.head()
91 X_train.describe()
92 print(X_train.shape,"   ",Y_train.shape)
93 X_test.size
```

In line 73-93 is describing how the dataset is used for trained and test model.Line 80 shows that X_train are the training part of the first sequence and Y_train are the training part of the second sequence

4.2.3. ANN Part

```
94 def create_model():
  95 ann = Sequential()
  96
   ann.add(Dense(units=128,kernel_initializer='uniform',activation='relu',inp
   ut_dim=4))
  97 ann.add(Dense(units=64,kernel_initializer='uniform',activation='relu'))
  98 ann.add(Dense(units=32,kernel_initializer='uniform',activation='relu'))
  99 ann.add(Dense(units=1,kernel_initializer='uniform',activation='linear'))
  100 ann.compile(optimizer='adam',loss='mean_squared_error',metrics=[‘mse])
  101 return ann

102 from sklearn.model_selection import GridSearchCV
103 from scikeras.wrappers import KerasRegressor
104 from tensorflow import keras
105  model  =  KerasRegressor(model=create_model,  batch_size=  10,  epochs=
   400,verbose=0, validation_split=0.3, metrics=['mse'])
106 learn_rate = [0.0001, 0.001, 0.01, 0.1, 0.2, 0.5]
107 param_grid = dict(optimizer__learning_rate=learn_rate)
108  grid=  GridSearchCV(estimator=model,  param_grid=param_grid,  n_jobs=-1,
   cv=3)
109 grid_result = grid.fit(X_train,Y_train)
```

```
110    print("Best:    %f    using    %s"    %    (grid_result.best_score_,
   grid_result.best_params_))
111 means = grid_result.cv_results_['mean_test_score']*-1
112 stds = grid_result.cv_results_['std_test_score']
113 params = grid_result.cv_results_['params']
114 for mean, stdev, param in zip(means, stds, params):
    115 print("%f (%f) with: %r" % (mean, stdev, param))
```

In line 94-115 are code to get the optimizer learning rate.

```
# GridSearch to find the best batch and epochs
116      model      =      KerasRegressor(model=create_model,      verbose=0,
   validation_split=0.3, optimizer__learning_rate=0.0001,metrics=['mse'])
117 batch_size= [10, 50, 100, 150, 200]
118 epochs = [100, 200, 300, 400, 500]
119 param_grid= dict(batch_size= batch_size, epochs=epochs)
120  grid = GridSearchCV(estimator=model, param_grid= param_grid, n_jobs=-1,
   cv=3)
121 grid_result = grid.fit(X_train,Y_train)
122    print("Best:    %f    using    %s"    %    (grid_result.best_score_,
   111grid_result.best_params_))
123 means = grid_result.cv_results_['mean_test_score']
124 stds = grid_result.cv_results_['std_test_score']
125 params = grid_result.cv_results_['params']
126 for mean, stdev, param in zip(means, stds, params):
    127 print("%f (%f) with: %r" % (mean, stdev, param))
```

In line 116-127 are code for Grid Search find the best batch and epochs

```
128 ann = Sequential()
129
   ann.add(Dense(units=32,kernel_initializer='uniform',activation='relu',inpu
   t_dim=3))
130 ann.add(Dense(units=16,kernel_initializer='uniform',activation='relu'))
131 ann.add(Dense(units=1,kernel_initializer='uniform',activation='linear'))
132 ann.compile(optimizer='adam',loss='mean_squared_error',metrics=[])
133 ann.fit(X_train,Y_train,batch_size=200,epochs=300,validation_split=0.05)
```

In line 128-133 are codes to perform ANN algorithm.

```
134 ann_predictions = ann.predict(X_test)
135 ann_predictions
```

Line 134-135 are codes to show the result from the ANN model.

```
136 X_train
```

Line 136 shows the training part of the first sequence (x).

```
137 X_test
```

Line 137 shows the test part of the first sequence (x).

```
138 print(Y_test,"   ",ann_predictions)
```

Line 138 shows the test part of the sequence (y)

#Root Mean Square Error

```
139 trainScore = ann.evaluate(X_train, Y_train, verbose=0)
140
   print('Train Score: %.2f MSE (%.2f RMSE)' % (trainScore, math.sqrt(trainSc
   ore)))
141 testScore = ann.evaluate(X_test, Y_test, verbose=0)
142
   print('Test Score: %.2f MSE (%.2f RMSE)' % (testScore, math.sqrt(testScore
   )))
```

Line 139-142 display the train and test score for MSE and RMSE

```
143 import tensorflow as tf
144 import keras
```

Line 143-144 are codes to import tensorflow and keras library

```
145 ann_predictions= ann_predictions * (closeMax-closeMin) + closeMin
146 Y_test= Y_test * (closeMax-closeMin) + closeMin
```

Line 145-146 is to normalize the data with Min-Max normalization.

```
147 ann_mae = tf.keras.losses.MeanAbsoluteError()
148 ann_mape = tf.keras.losses.MeanAbsolutePercentageError()
149 ann_mse = tf.keras.losses.MeanSquaredError()
150 ann_rmse=math.sqrt(ann_mse(Y_test, ann_predictions).numpy())
151 print("ANN MSE : ",ann_mse(Y_test, ann_predictions).numpy())
152 print("ANN RMSE : ",ann_rmse)
153 print("ANN MAE : ",ann_mae(Y_test, ann_predictions).numpy())
154 print("ANN MAPE : ",ann_mape(Y_test, ann_predictions).numpy())
```

Line 147-154 are code to perform the metric performance such as MSE,RMSE,MAE,and MAPE on the ANN machine learning model.

```
155 lab = preprocessing.LabelEncoder()
```

```
156 ann_transformed = np.asarray(ann_predictions)
157 ytest_transformed = np.asarray(Y_test)
158 plt.plot(ann_predictions,color='green',label='model')
159 plt.show()
160 plt.plot(Y_test,color='blue',label='value')
161 plt.show()
```

Line 155-161 show the code to display actual graph and graph from the ANN prediction model.

4.2.4. Random Forest Part

```
162 lab = preprocessing.LabelEncoder()
163 y_transformed = lab.fit_transform(Y_train)
164 y_test_transformed = lab.fit_transform(Y_test)
165
    RF = RandomForestClassifier(n_estimators=800, max_depth=40, random_state=0
    )
166 RF.fit(X_train, y_transformed)
```

Line 162-166 are code to perform Random Forest machine learning model.The author split data into 80% for training and 20% for test.

```
167 rf_predictions = RF.predict(X_test) # predict our file test data
168 rf_predictions= rf_predictions * (closeMax-closeMin) + closeMin
169 Y_test= Y_test * (closeMax-closeMin) + closeMin
```

Line 167-169 are code to perform Min-Max Normalization.

```
170 rf_mse = mean_squared_error(y_test_transformed, rf_predictions)
171 rf_mae = mean_absolute_error(y_test_transformed, rf_predictions)
172
    rf_mape = mean_absolute_percentage_error(y_test_transformed, rf_prediction
    s)
173 rf_rmse = rf_mse**.5
174 print("Random Forest MSE : ",rf_mse)
175 print("Random Forest RMSE : ",rf_rmse)
176 print("Random Forest MAE : ",rf_mae)
177 print("Random Forest MAPE : ",rf_mape)
```

Line 167-177 are code to perform the metric performance such as MSE,RMSE,MAE,and MAPE on the Random Forest machine learning model.

```
178 rf_transformed = np.asarray(rf_predictions)
179 ytest_transformed = np.asarray(Y_test * (closeMax*closeMin) + closeMin)
180 plt.title("Random Forest")
181 plt.plot(rf_predictions,color='green',label='predicted')
182 plt.plot(y_test_transformed,color='blue',label='actual')
```

```
plt.legend()
183 plt.show()
184 Print(np.asarray(rf_predictions))
185 print(ytest_transformed)
```

Line 178-185 are code to show the actual graph and the graph of Random Forest.

4.2.5. ARIMA Part

```
186 ADRO = pd.read_csv('ADRO.JK.csv', parse_dates=['Date'], index_col="Date")
187 Close = ADRO.Close.isnull()
188 for i in range(0, len(Close)):
    189 if Close[i] == 1:
        190 nan_value_index.append(i)
        191 ADRO['Close'][i] = 0
192 ADRO_monthly = ADRO.resample('M').mean()
193 closeMin= ADRO_monthly.Close.min()
194 closeMax= ADRO_monthly.Close.max()
195 ADRO_ts = (ADRO_monthly.Close - closeMin) / (closeMax - closeMin)
196 plt.plot(ADRO_ts)
```

Line 186-196 are code to clean and normalize the data.

```
197 import pandas as pd
198 import numpy as np
199 import matplotlib.pyplot as plt
200 %matplotlib inline
201 from datetime import timedelta
202 from statsmodels.tsa.arima_model import ARIMA as ARIMA
203 from keras.models import Sequential
204 from keras.layers import Dense
205 from keras.layers import LSTM
206 from sklearn.preprocessing import MinMaxScaler
207 from sklearn.metrics import mean_squared_error
208 from scipy.ndimage.filters import gaussian_filter
209 import os
210 os.environ['TF_CPP_MIN_LOG_LEVEL']='2'
211 import warnings
212 warnings.filterwarnings('ignore')
```

Line 197-212 is used to import libraries needed to construct ARIMA

```
# order consist of p,d,q
213 def evaluate_arima_model(series, order):
  214 size = int(len(series) * data_split)
  215 train, test = series[0:size], series[size:]
  216 history = [val for val in train]
  217 predictions= list()
  218 for t in range(len(test)):
    219 model = ARIMA(history, order=order)
    220 model_fit = model.fit()
    221 yhat= model_fit.forecast()[0]
```

```
    222 predictions.append(yhat)
    223 history.append(test[t])
  224 error= mean_squared_error(test,predictions)
  225 return error

226 def evaluate_models(dataset, p_values, d_values, q_values):
  227 dataset = dataset.astype('float32')
  228 best_score, best_cfg = float("inf"), None
  229 for p in p_values:
    230 for d in d_values:
      231 for q in q_values:
        232 order = (p,d,q)
        233 try:
          234 mse = evaluate_arima_model(dataset, order)
          235 if mse < best_score:
            236 best_score, best_cfg = mse, order
          237 print('ARIMA%s MSE=%.3f' % (order,mse))
        238 except:
          239 continue
  240 print('Best ARIMA%s MSE=%.3f' % (best_cfg, best_score))

    241 p_values = [0, 1, 2, 4, 6,]

    242 d_values = range(0, 3)

    243 q_values = range(0, 3)

    244 warnings.filterwarnings("ignore")

    245 evaluate_models(ADRO_ts, p_values, d_values, q_values)
```

In Line 213-245 are code to evaluate hyperparameter in ARIMA model with by doing Grid Search based on MSE values.

```
246 import matplotlib.dates as mdates
247
   from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_scor
   e, roc_curve

248 def arima_model(series, data_split, params, future_periods, log):

      # log transformation of data if user selects log as true
    249 if log == True:
        250 series_dates = series.index
        251 series = pd.Series(np.log(series), index=series.index)

    252 # create training and testing data sets based on user split fraction
    253 size = int(len(series) * data_split)
    254 train, test = series[0:size], series[size:len(series)]
    255 history = [val for val in train]
    256 predictions = []


    # creates a rolling forecast by testing one value from the test set, and
   then add that test value

    # to the model training, followed by testing the next test value in the
  series
```

```
    for t in range(len(test)):
        model = ARIMA(history, order=(params[0], params[1], params[2]))
        model_fit = model.fit(disp=0)
        output = model_fit.forecast()
        yhat = output[0]
        predictions.append(yhat[0])
        obs = test[t]
        history.append(obs)


# forecasts future periods past the input testing series based on user inp
ut
    future_forecast = model_fit.forecast(future_periods)[0]
    future_dates = [test.index[-
1]+timedelta(i*365/12) for i in range(1, future_periods+1)]
    test_dates = test.index


# if the data was originally log transformed, the inverse transformation i
s performed
    if log == True:
        predictions = np.exp(predictions)
        test = pd.Series(np.exp(test), index=test_dates)
        future_forecast = np.exp(future_forecast)


# creates pandas series with datetime index for the predictions and foreca
st values
    forecast = pd.Series(future_forecast, index=future_dates)
    predictions = pd.Series(predictions, index=test_dates)
predictions= predictions * (closeMax-closeMin) + closeMin
    test= test * (closeMax-closeMin) + closeMin

    # generates plots to compare the predictions for out-of-
sample data to the actual test values
    fig = plt.figure()
    ax = fig.add_subplot(111)
    myFmt = mdates.DateFormatter('%m%/%y')
    ax.xaxis.set_major_formatter(myFmt)
plt.title('ARIMA')
    plt.plot(predictions, c='red')
    plt.plot(test)
    plt.show()

    # calculates root mean squared errors (RMSEs) for the out-of-
sample predictions
    rmse = np.sqrt(mean_squared_error(predictions, test))
    mse=mean_squared_error(predictions, test)
    mape=mean_absolute_percentage_error(predictions, test)
    mae=mean_absolute_error(predictions, test)
    print('ARIMA MAE: %.3f' % mae)
    print('ARIMA RMSE: %.3f' % rmse)
    print('ARIMA MAPE: %.3f' % mape)
    print('ARIMA MSE: %.3f' % mse)

    return predictions, test, future_forecast
```

Line 246-293 to use library and define arima model and forecast the values

```
294 data_split = 0.8
295 p = 6
296 d = 0
297 q = 1
298 params = [p, d, q]
299 future_periods = 12
300 log = True

301 predictions, test, forecast = arima_model(ADRO_ts, data_split, params,
   future_periods, log)
```

Line 294-301 are used to construct the training model chart and the model test.

4.3. Tuning Hyperparameter

After the dataset is normalized, the trainer run some numbers to get the best number for hyperparameter in each machine learning Model

Table 4.1 Learning Rate for ANN

Learning Rate	MSE Value
0.0001	8.059676
0.001	8.062363
0.01	**8.049343**
0.1	8.060739
0.2	8.066856
0.5	8.050134

The author uses KerasRegressor to get the best learning rate for the model. By evaluating with MSE value, the author attained the best learning rate which is 0.01 from the KerasRegressor.

Table 4.2 Finding Hyperparameter on ANN

	Epochs					
Batch Size		100	200	300	400	500
	10	0.0178	0.0265	0.0457	0.0337	0.0347
	50	0.0071	0.0209	0.0185	0.0245	0.0263
	100	**0.0036**	0.0069	0.0187	0.0205	0.0218
	150	0.0037	0.0074	0.0194	0.0212	0.0208
	200	0.0039	0.0065	0.0184	0.0202	0.0219

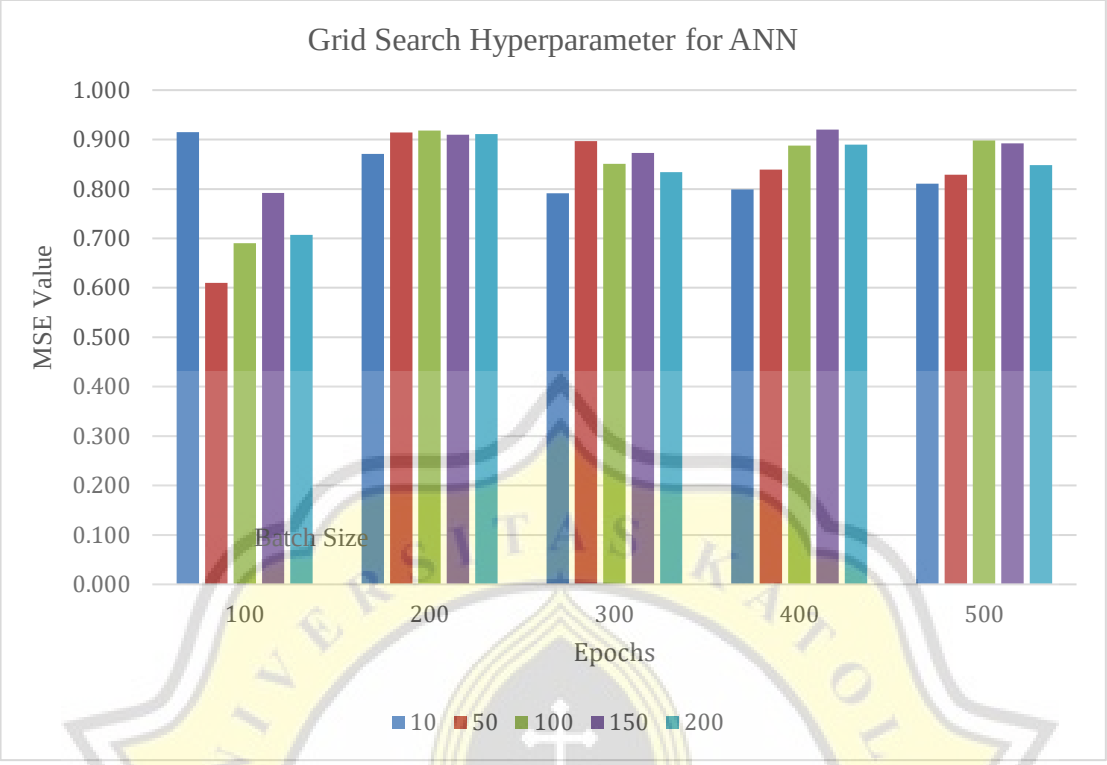

Figure 4.1 Bar Chart of Grid Search Hyperparameter on ANN

The author uses Grid Search to get the best batch and epoch for the model on ANN. The result shows that with batch size of 150 and epochs of 400 produce with 0.920 MSE value which is the best scenario for the model with learning rate of 0.0001. Therefore, the author will use this parameter on this project.

Table 4.3 Grid Search Hyperparameter on Random Forest

	max_depth					
n_estimators		10	20	30	40	50
	200	2283724.45	2283753.825	2283753.825	2283753.825	2283753.825
	400	2277760.27	2277768.751	2277768.751	2277768.751	2277768.751
	600	2278939.331	2278931.948	2278931.948	2278931.948	2278931.948
	800	2279902.725	2279886.628	2279886.628	2279886.628	2279886.628
	1000	2276943.832	**2276934.973**	2276934.973	2276934.974	2276934.974
	1200	2279159.863	2279155.227	2279155.227	2279155.227	2279155.227
	1400	2282254.919	2282251.463	2282251.564	2282251.463	2282251.463

After running Random Forest Regressor, the author get the best hyperparameter which is n_estimator = 1000, and max_depth = 20 which produces 2276934.973 MSE value. Therefore, the author will use 1000 number of trees in the forest and 20 for maximum number of levels in each decision tree.

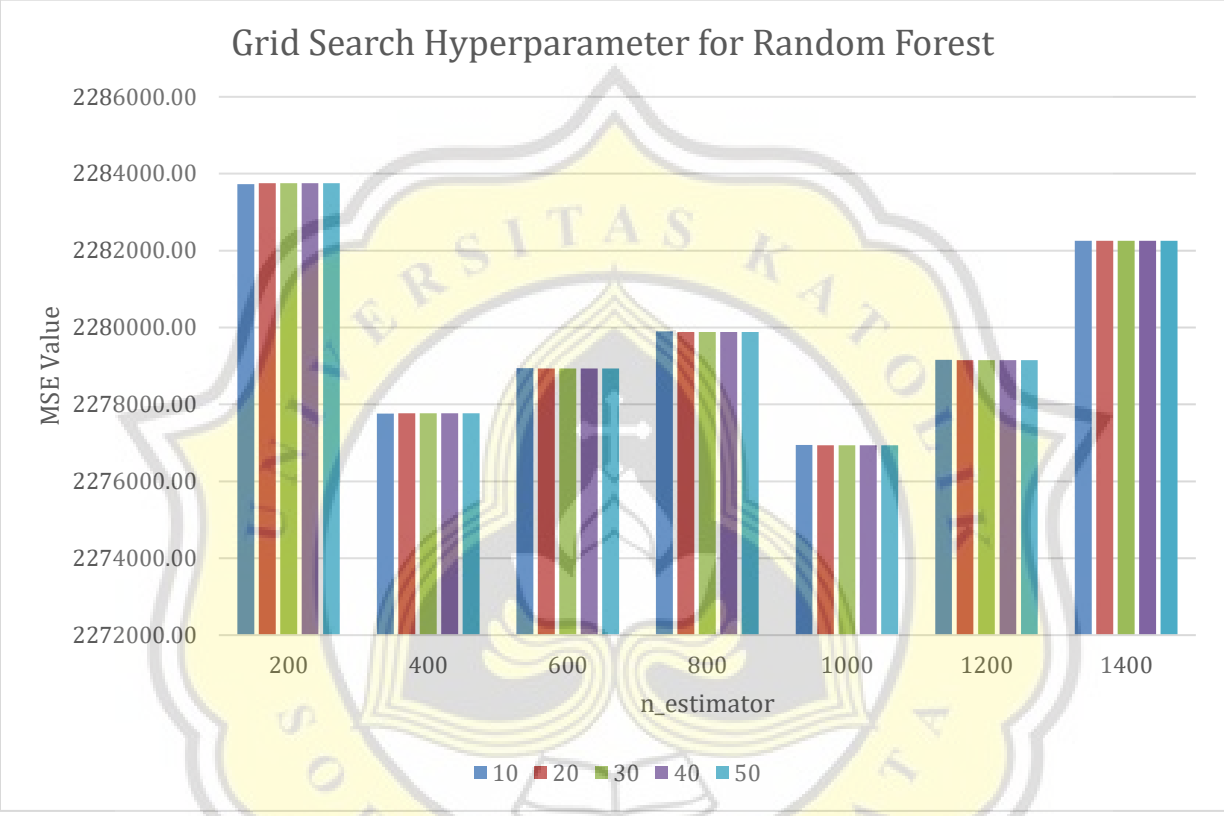

Figure 4.2 Bar Chart of Grid Search Hyperparameter on Random Forest

The data in the Table 4.3 and Figure 4.2 are the experiment to get the based hyperparameter number on Random Forest. The result shows that with n_estimator = 1000, and max_depth = 20 produce the least MSE value therefore the author will use this parameter on this project.

Table 4.4 Grid Search Hyperparameter on ARIMA

	p					
d,q		0	1	2	4	6
	0,0	0.017	0.005	0.005	0.005	0.005
	0,1	0.008	0.005	0.005	0.005	**0.005**
	0,2	0.009	0.005	0.005	0.005	0.005
	1,0	0.005	0.005	0.005	0.005	0.005
	1,1	0.005	0.006	0.005	0.005	0.005
	1,2	0.005	0.005	0.005	0.006	0.005
	2,0	0.012	0.008	0.007	0.006	0.006
	2,1	0.005	0.005	0.005	0.006	0.005
	2,2	0.005	0.005	0.005	0.005	0.006

These are the values to get the best optimized hyperparameter on ARIMA Model by running Grid Search. The model is evaluated and produces value ARIMA(6, 0, 1) with MSE value of 0.005 as the best hyperparameter .

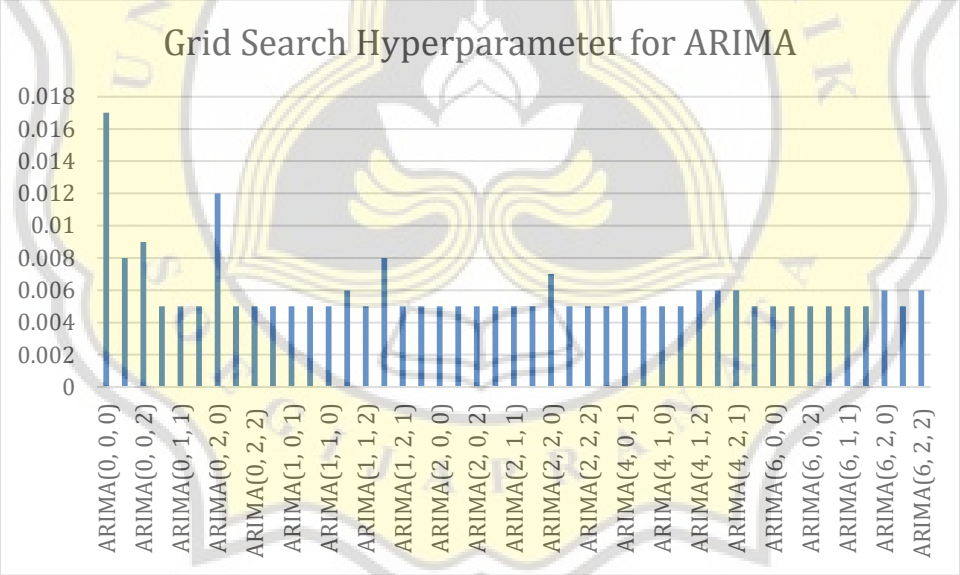

Figure 4.3 Bar Chart of Hyperparameter on ARIMA Model

The data in the Table 4.4 and Figure 4.2 are the experiment to get the based hyperparameter number on ARIMA Model. The result shows that with p = 6 (lag observations in the model), d=0 (difference of the number of times the raw observation) and q=1 (the number moving average being used) produce the least error value therefore the author will use this parameter on this project.

4.4. Results

This section will provide the results from the execution of three machine learning model which are ANN, Random Forest, and ARIMA Model. The following results will show the comparison between actual data and prediction data which consists of 80% of trained data and 20% of data validation test. The results will consist of graphs and numbers for performance metric.

4.4.1. Machine Learning model Table for ADRO

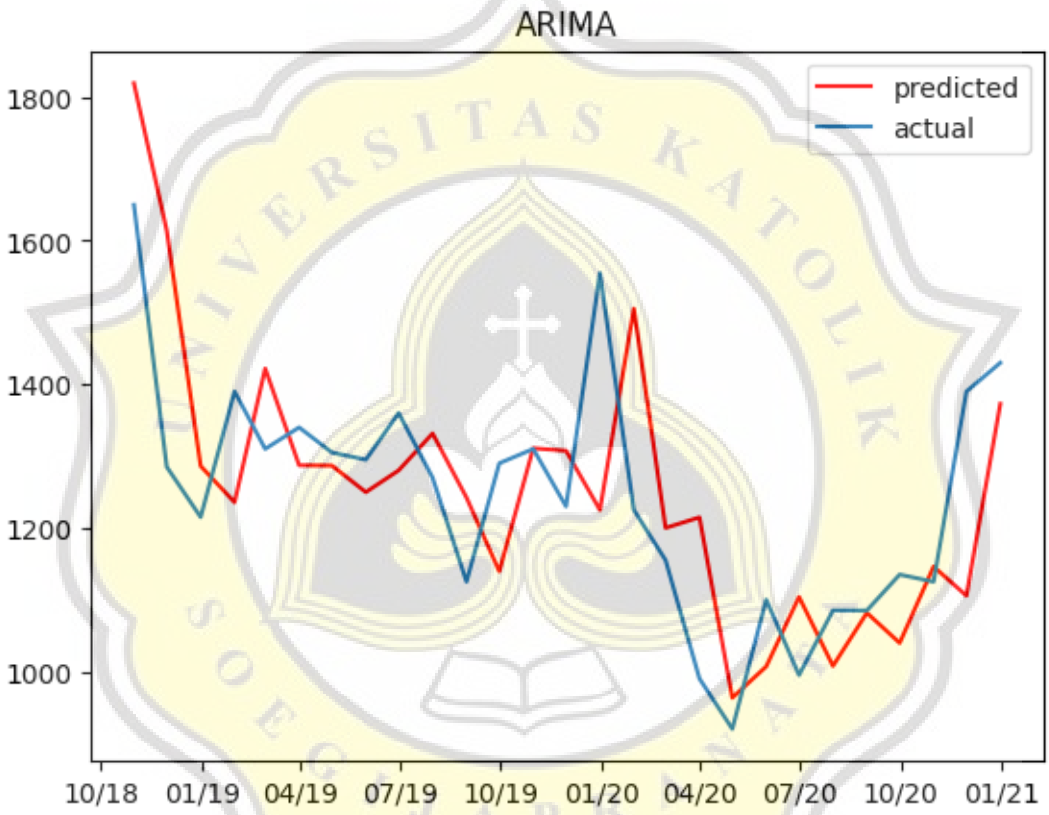

Figure 4.4 Testing Result on ARIMA Model

The result of ARIMA working by predicting the next value based on the previous day's close value.The lag in the period causes the delay therefore causes a big error.Because ARIMA relies on one variable,it belongs to univariate forecasting model.

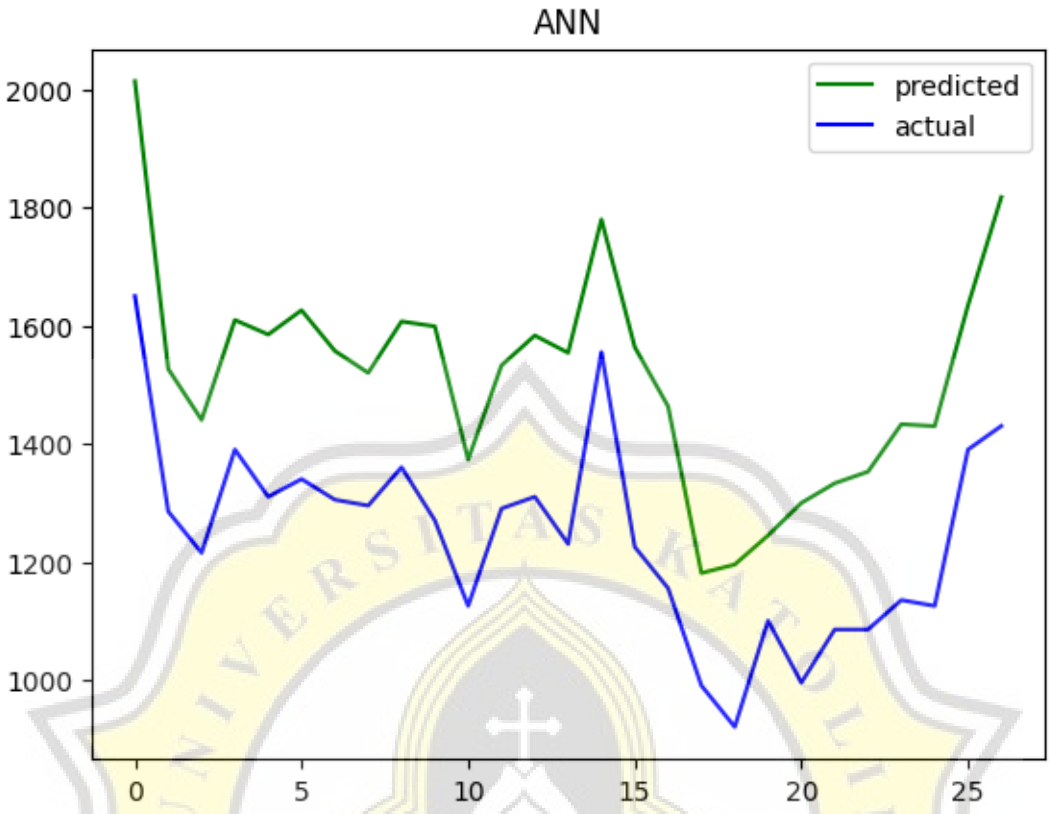

Figure 4.5 Testing Result on ANN Model

This ANN model is constructed by LHOV from a dataset that comes from test and training.L stands for Low stands for High, O represents open score and V represents volume score in that particular day.This graph is display from walk-forward-validation whereby the result comes from applying validation set through optimizing the score from both training and test.This method belongs to multivariate forecasting which uses more than one variable to perform analysis.

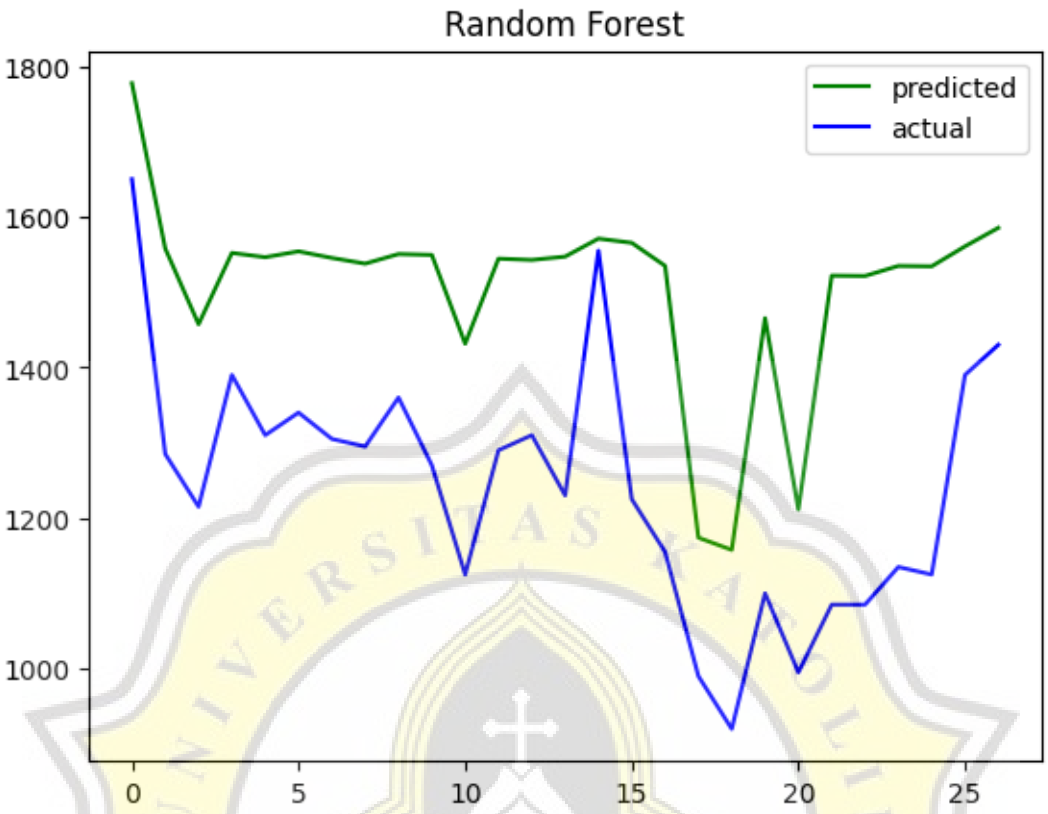

Figure 4.6 Testing Result on Random Forest Model

Random Forest model is build based on classification and regression tasks.These method also use walk-forward-validation which divide dataset into training and validation test and optimize those data into validation set.This method also categorized as multivariate forecasting because it process more than one variable.

4.4.2. Table for Performance Metric

These are the results from performance metrics for each model.

Table 4.5 Performance Metrics Table

Machine Learning Model	MSE	RMSE	MAE	MAPE
ANN	23286.096	152.598	141.660	11.607
Random Forest	2276934.973	1508.952	1503.574	4570.636
ARIMA	22167.497	148.888	114.974	0.091

Figure 4.7 Bar Chart of MSE Performance Metrics

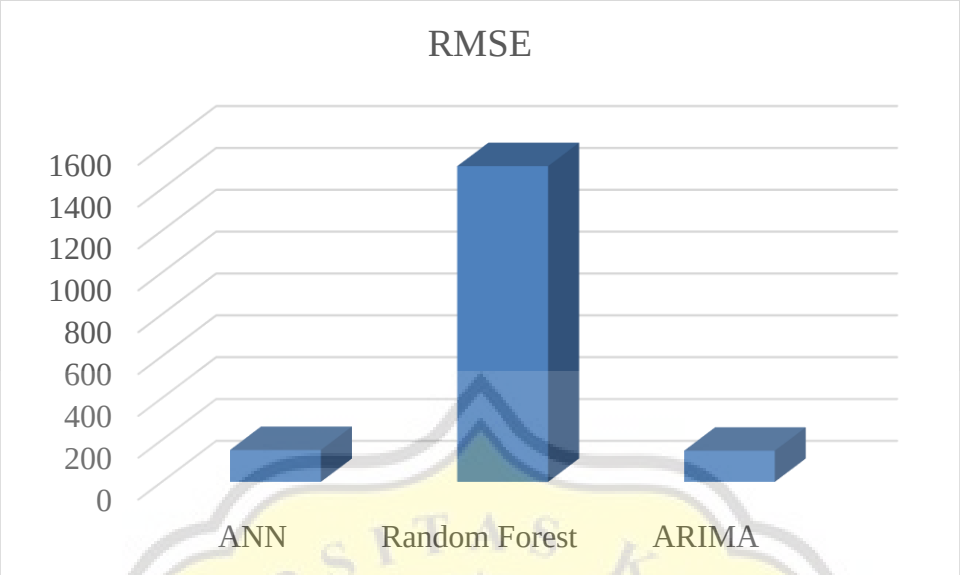

Figure 4.8 Bar Chart of RMSE Performance Metrics

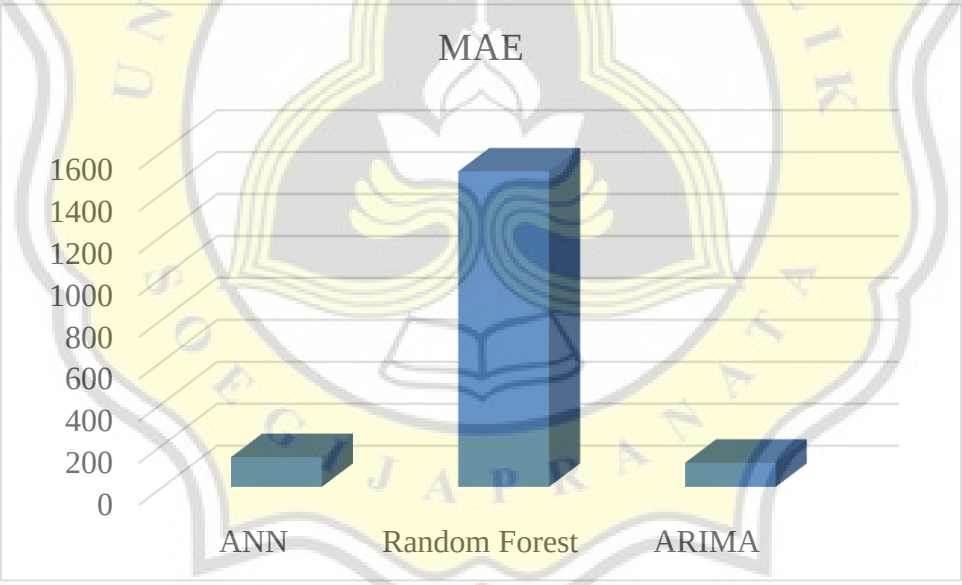

Figure 4.9 Bar Chart of MAE Performance Metrics

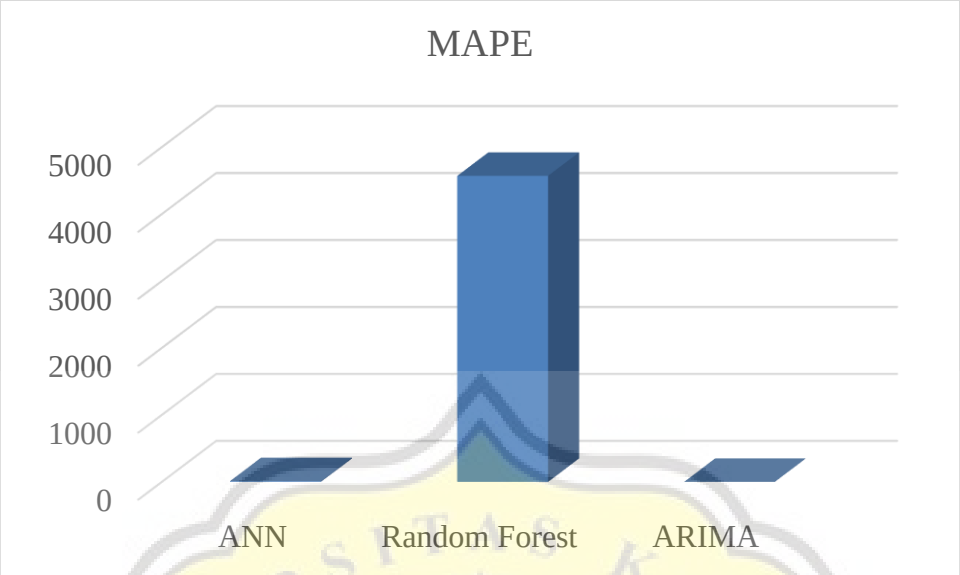

Figure 4.10 Bar Chart of MAPE Performance Metrics

Based on Table 4.5 and Figure 4.7, in MSE performance metric, the result shows that Random Forest has the highest MSE value as much as 2276934.973, followed by ANN with 23286.096 MSE values, and lastly ARIMA which has the lowest MSE value equal to 22167.497. In RMSE performance metric, the result shows that Random Forest has the highest RMSE value equal to 1508.95, followed by ANN which has RMSE value as much as 152.598. On the other hand, ARIMA has the lowest RMSE value equal to 148.888. In MAE performance metric, Random Forest has the highest MAE value as much as 1503.574, followed by ANN which has 141.66 MAE values. Meanwhile, ARIMA has the lowest MAE value equal to 114.974. In MAPE performance metric, Random Forest has the highest MAPE value equal to 4570.636, followed by ANN in the second place with MAPE value of 11.607. On the other hand, ARIMA has the lowest MAPE value equal to 0.091.