BAB IV

PEMBAHASAN DAN HASIL

4.1 Implementasi

Semua tahapan perancangan penelitian yang telah dijabarkan dalam bentuk *flowchart* pada gambar 3.1, diimplementasikan pada kode program menggunakan bahasa pemrograman Python 3.7. Proses pada kode program akan dijabarkan pada bab ini:

4.1.1 Import liberary

Gambar 4.1 mendefinisikan *liberary* yang digunakan. Terlebih dahulu melakuakan *liberary* dan *package* yang dibutuhan yang nantinya akan digunakan pada pemrosesan algoritma XGBoost.

```
import os
import pandas as pd
import numpy as np
import math
import datetime as dt

from    sklearn.metrics    import    mean_squared_error,
mean_absolute_error, explained_variance_score, r2_score
from   sklearn.metrics   import   mean_poisson_deviance,
mean_gamma_deviance, accuracy_score
from sklearn.preprocessing import MinMaxScaler

from itertools import cycle
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots

import seaborn as sns
import matplotlib.pyplot as plt
from colorama import Fore
```

Gambar 4.1 *Import library*

4.1.2 Import dataset

Gambar 4.2 mendefinisakan fungsi untuk mendapatkan dataset memanfaatkan *library*

pandas data *reader* pada baris kode pertama, memanggil fungsi *Datareader* untuk mengambil file dataset.csv. pada baris kode ke 2, memanggil fungsi df.loc untuk mengambil hanya data bitcoin dari dataset. Pada baris ke 3 kode berfingsi untuk menapilkan data *bitcoin* yang terdapat pada dataset.

```
df = pd.read_csv('/content/dataset.csv')

btc = df.loc[df['crypto_name'] == 'Bitcoin'].copy()
btc
```

Gambar 4.2 *Import dataset*

Gambar 4.3 adalah visualisasi dataset *Bitcoin*, dapat dilihat bahwa data *Bitcoin* memiliki 3.248 data dan 10 label.

	Unnamed: 0	open	high	low	close	volume	marketCap	timestamp	crypto_name	date
0	0	112.900002	118.800003	107.142998	115.910004	0.000000e+00	1.288693e+09	2013-05-05T23:59:59.999Z	Bitcoin	2013-05-05
2	2	115.980003	124.663002	106.639999	112.300003	0.000000e+00	1.249023e+09	2013-05-06T23:59:59.999Z	Bitcoin	2013-05-06
4	4	112.250000	113.444000	97.699997	111.500000	0.000000e+00	1.240594e+09	2013-05-07T23:59:59.999Z	Bitcoin	2013-05-07
7	7	109.599998	115.779999	109.599998	113.566002	0.000000e+00	1.264049e+09	2013-05-08T23:59:59.999Z	Bitcoin	2013-05-08
9	9	113.199997	113.459999	109.260002	112.669998	0.000000e+00	1.254535e+09	2013-05-09T23:59:59.999Z	Bitcoin	2013-05-09
...	...	...	...	...	...	...	...	...	...	...
72692	72692	18936.311515	19134.733194	18696.468304	18802.097976	2.335997e+10	3.602593e+11	2022-09-25T23:59:59.999Z	Bitcoin	2022-09-25
72752	72752	19311.848708	19370.309281	18970.620583	19044.107272	2.076596e+10	3.650248e+11	2022-10-02T23:59:59.999Z	Bitcoin	2022-10-02
72815	72815	19417.479411	19542.538654	19349.258953	19446.426194	1.683726e+10	3.728768e+11	2022-10-09T23:59:59.999Z	Bitcoin	2022-10-09
72845	72845	19068.913560	19389.603520	19068.913560	19268.092801	1.798892e+10	3.695843e+11	2022-10-16T23:59:59.999Z	Bitcoin	2022-10-16
72912	72912	19207.734651	19646.651542	19124.196965	19567.007398	2.212879e+10	3.754443e+11	2022-10-23T23:59:59.999Z	Bitcoin	2022-10-23

3248 rows × 10 columns

Gambar 4.3 Data *Bitcoin*

4.1.3 Preprosesing data

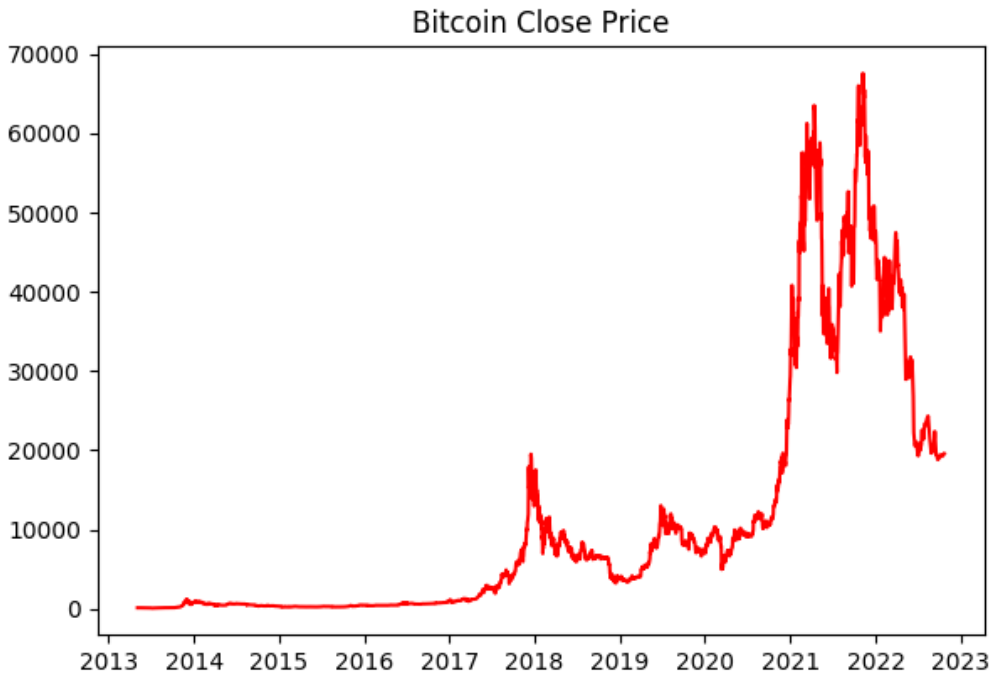
Gambar 4.4 adalah kode python untuk preprosesing data, pada baris kode pertama, memanggil fungsi fillna yaitu menghilangkan data yang memiliki nilai none atau null. Pada baris kode ke 2 sampai ke 4, yaitu mengubah format date ke datetime pada kolom date. Pada baris kode ke 5 sampai ke 8, untuk memvisualisakan data pada kolom date dan close setelah dilakukan preprosesing dan formatting dapat dilihat pada gambar 4.5.

```
btc = btc.fillna(method = 'ffill')

btc['date'] = pd.to_datetime(btc.date)
btc.head().style.set_properties(subset=['date','close'],
**{'background-color': 'skyblue'})

fig = plt.figure(figsize = (15,10))
plt.subplot(2, 2, 1)
plt.plot(btc['date'], btc['close'], color="red")
plt.title('Bitcoin Close Price')
```

Gambar 4.4 *fillna* dan *datetime* preprosesing



Gambar 4.5 Visualisasi *Bitcoin* setelah preprosesing dan formatting

4.1.4 Data Analisis

Gambar 4.6 adalah kode python untuk analisis data dengan cara memplotting data bitcoin dari rentang tahun 2020-2022 hal ini dilakukan karena jika merujuk pada gambar 4.5 pergerakan harga *bitcoin* yang sangat signifikan berada pada rentang tahun 2020-2022, oleh sebab itu perlu dianalasis lebih detail. Pada baris kode pertama, berfungsi untuk memfilter atau hanya mengambil data *bitcoin* dengan periode tanggal lebih besar dari 01-

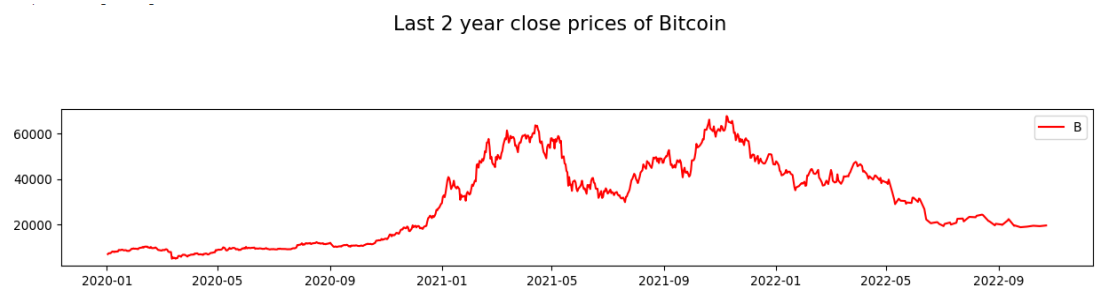
2020. Pada baris kode ke 2 sampai ke 6 berfungsi untuk memvisualisasikan data *bitcoin* setelah dilakuakan *plotting* data.

```
last2year_btc = btc[btc['date'] > '01-2020']

fig = plt.figure(figsize = (15,10))
fig.suptitle("Last 1 year close prices of Bitcoin",
fontsize=16)
plt.subplot(4, 1, 1)
plt.plot(last2year_btc['date'], last2year_btc['close'],
color="red")
plt.legend("B")
```

Gambar 4.6 Plotting data *Bitcoin* tahun 2020-2022

Gambar 4.7 adalah visualisasi data *bitcoin* setelah dilakuakan ploting dengan hanya mengambil data *bitcoin* dengan periode tanggal lebih besar dari 01-2020.



Gambar 4.7 Visualisasi plotting data *Bitcoin* tahun 2020-2022

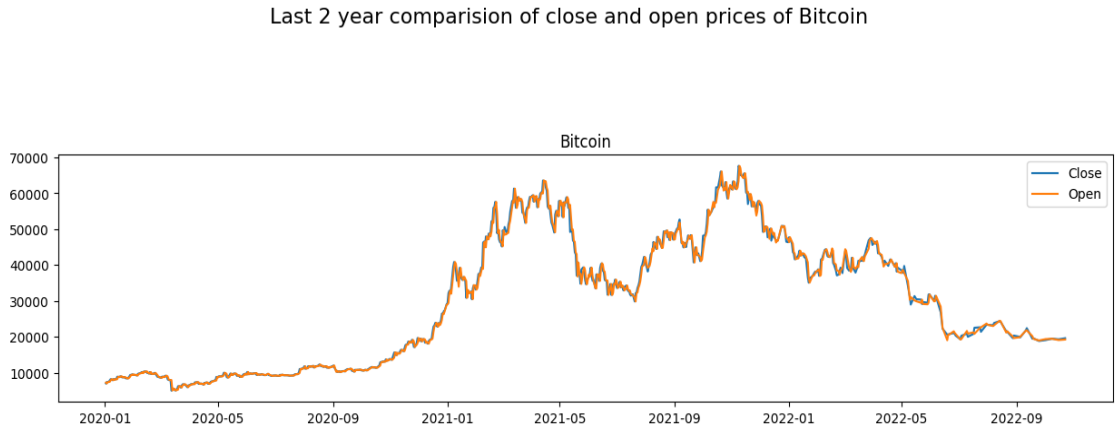
Gambar 4.8 adalah kode python untuk memvisualisasikan data *close* dan *open* pada data *bitcoin*, hal ini dilakukan untuk melihat seberapa besar perbedaan harga *close* dan *open* yang akan berpengaruh terhadap data yang akan digunakan sebagai training dan prediksi. Pada baris kolom pertama, berfungsi untuk hanya mengambil data lebih besar dari 01-2020. Pada baris kolom ke 2 sampai 10 berfungsi untuk mengambil data dan memvisualisasikan data dalam betuk plot diagram.

```
last2year_btc = btc[btc['date'] > '01-2020']
fig = plt.figure(figsize = (15,15))
fig.suptitle("Last month comparision of close and open
prices of Bitcoin", fontsize=16)
fig.tight_layout()
plt.subplot(4, 1, 1)
plt.plot(last2year_btc['date'], last2year_btc['close'])
plt.plot(last2year_btc['date'], last2year_btc['open'])
plt.legend(["Close", "Open"])
plt.title("Bitcoin")

plt.show()
```

Gambar 4.8 Perbandingan harga *close* dan *open*

Gambar 4.9 adalah visualisasi perbandingan harga *close* dan *open* dari rentang tahun 2020-2022, jika melihat plot diagram pada gambar terlihat perbedaan antara harga *close* dan *open* tidak signifikan. Artinya, dalam melakukan training dan prediksi dapat mengacu pada salah satu data tersebut.



Gambar 4.9 Visualisasi perbandingan harga *close* dan *open*

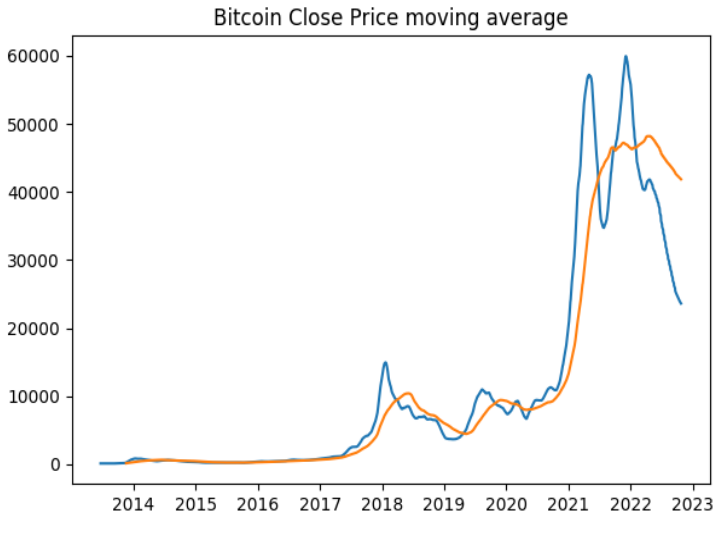
Gambar 4.10 adalah kode python untuk analisis menggunakan teknik *moving everages.* Pada baris kode pertama sampai ke 5 berfungsi untuk mengambil dan memvisualisasikan data *date* dan *close* kemudian dilakukan rolling 50 hari dan 200 hari.

```
fig = plt.figure(figsize = (15,10))

plt.subplot(2, 2, 1)
plt.plot(btc['date'], btc['close'].rolling(50).mean())
plt.plot(btc['date'], btc['close'].rolling(200).mean())
plt.title('Bitcoin Close Price moving average')
```

Gambar 4.10 *Moving everages*

Gambar 4.11 adalah visualisasi *Moving everages*, dapat dilihat pola antara rolling 50 dan rolling 200 cendrung memiliki pergerakan naik dan turun yang sama, hanya jika kita melihat pada rolling 200 (warna orange) memiliki pola pergerakan naik dan turun lebih halus disbanding rolling 50 (warna biru).



Gambar 4.11 Visualisasi *Moving everages*

Gambar 4.12 adalah kode python untuk memvisualisasikan rata-rata harga *close Bitcoin.* Pada baris kode pertama sampai ke 6 berfungsi untuk memvisualisasikan data rata rata harga *close Bitcoin.*

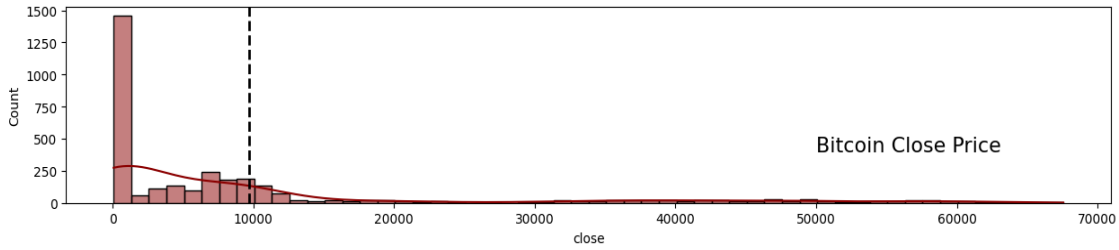
```
fig = plt.figure(figsize = (15,12))

fig.tight_layout()

plt.subplot(4, 1, 1)
sns.histplot(btc['close'],color='darkred', kde=True)
plt.axvline(btc['close'].mean(),                color='k',
linestyle='dashed', linewidth=2)
plt.text(50000,400,'Bitcoin Close Price', fontsize=16)
```

Gambar 4.12 Indikator rata-rata harga *close Bitcoin*

Gambar 4.13 adalah Visualisasi indikator rata-rata harga *close Bitcoin,* dimana dapat dilihat bahwa harga 0-100000 memiliki jumlah data yang paling banyak berada pada rentang 250 sampai 1500 data



Gambar 4.13 Visualisasi indikator rata-rata harga *close Bitcoin*

4.1.5 Normalisai Data

Gambar 4.14 berfungsi untuk mengambil data yang akan di normalisasi, pada tahap ini untuk menentukan data yang akan diambil berdasarkan hasil analisis pada tahapan sebelumnya dan hasil analisis visual menunjukan pergerakan harga *bitcoin* yang sangat signifikan berada pada rentang tahun 2020 sampai 2022. Pada baris kode pertama berfungsi untuk mengambil data hanya dalam rentang lebih besar dari 2020-01-01, pada baris kode ke 2 berfungsi untuk mengcopy data closedf ke close_stock, pada baris kode ke 3 berfungsi untuk melihat jumlah data yang berada pada closed.

```
closedf = closedf[closedf['date'] > '2020-01-01']
close_stock = closedf.copy()
print("Total data for prediction: ",closedf.shape[0])
```

Gambar 4.14 Filtering data

Gambar 4.15 adalah kode python yang berfungsi untuk normalisasi data dengan metode *minmax,* Pada baris kode pertama berfungsi untuk menghapus label *date* pada data closedf. Pada baris kode ke 2 berfungsi untuk memanggil *MinMaxScaler* dengan feature_range = 0, 1 dan kemudian akan digunakan untuk menormalisasi data closedf. Pada baris ke 3 berfungsi untuk merubah kolom closed. Pada baris ke 4 berfungsi untuk melihat jumlah data closed.

```
del closedf['date']
scaler=MinMaxScaler(feature_range=(0,1))
closedf=scaler.fit_transform(np.array(closedf).reshape(-
1,1))
print(closedf.shape)
```

Gambar 4.15 Normalisasi menggunakan Minmax

4.1.6 Training dan testing data

Gambar 4.16 adalah implementasi kode python untuk membagi antara data training dan testing. Pada baris kode pertama adalah menentukan data training_size 70% dari data closed. Pada baris kode ke 2 adalah menentukan data test_size sejumlah 30% dari data closedf. Pada baris kode ke 3 dan ke 4 adalah membuat data train_data dan test_data berdasrkan training_size dan test_size pada data closed. Pada baris ke 5 dan ke 6 berfungsi untuk menampilkan jumlah data pada training_data dan test_data.

```
training_size=int(len(closedf)*0.70)
test_size=len(closedf)-training_size
train_data,test_data=closedf[0:training_size,:],closedf[
training_size:len(closedf),:1]
print("train_data: ", train_data.shape)
print("test_data: ", test_data.shape)
```

Gambar 4.16 Training dan testing data

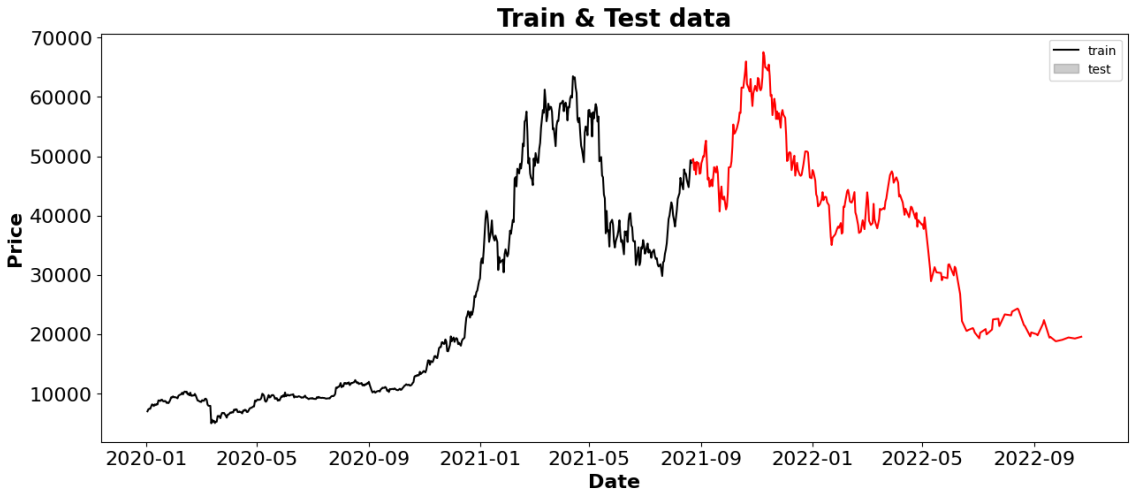
Gambar 4.17 adalah kode python untuk mevisualisasikan pembagian training dan testing data.

```
#Train & test data view
fig, ax = plt.subplots(figsize=(15, 6))
sns.lineplot(x = close_stock['date'][:571], y =
close_stock['close'][:571], color = 'black')
sns.lineplot(x = close_stock['date'][571:], y =
close_stock['close'][571:], color = 'red')

# Formatting
ax.set_title('Train & Test data', fontsize = 20,
loc='center', fontdict=dict(weight='bold'))
ax.set_xlabel('Date', fontsize = 16,
fontdict=dict(weight='bold'))
ax.set_ylabel('Weekly Sales', fontsize = 16,
fontdict=dict(weight='bold'))
plt.tick_params(axis='y', which='major', labelsize=16)
plt.tick_params(axis='x', which='major', labelsize=16)
plt.legend(loc='upper right' ,labels = ('train', 'test'))
```

Gambar 4.17 Kode python visaualisasi training dan testing data

Gambar 4.18 adalah visualisasi dari data train dan test. Dimana warna hitam adalah data training dan warna merah adalah data testing



Gambar 4.18 visaualisasi training dan testing data

4.1.7 Time Series data training

Gambar 4.19 adalah kode python untuk membuat time series data training dan mengkonversi data tersebut kedalam nilai matriks kumpulan data. Pada baris kode 12-15 berfungsi untuk melihat jumlah data pada masing masing data frame.

```
def create_dataset(dataset, time_step=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-time_step-1):
           a  =  dataset[i:(i+time_step),  0]     ###i=0,
0,1,2,3-----99   100
        dataX.append(a)
        dataY.append(dataset[i + time_step, 0])
    return np.array(dataX), np.array(dataY)

time_step = 15
X_train, y_train = create_dataset(train_data, time_step)
X_test, y_test = create_dataset(test_data, time_step)

print("X_train: ", X_train.shape)
print("y_train: ", y_train.shape)
print("X_test: ", X_test.shape)
print("y_test", y_test.shape)
```

Gambar 4.19 Time Series data training

Gambar 4.20 adalah output dari baris kode 12-15 pada gambar 4.19 yang berisi jumlah data pada data frame X_train, y_train, X_test, dan y_test. Dimana, masing masing data berjumlah X-train : (555, 15), y_train: (555,), X-test: (229, 15) dan y_test (229,)

```
X_train:  (555, 15)
y_train:  (555,)
X_test:  (229, 15)
y_test (229,)
```

Gambar 4.20 Jumlah data train dan test

4.1.8 Implementasi algoritma XGBoost

Gambar 4.21 adalah kode python untuk mengimplementasikan algoritma XGBoost. pada baris kode pertama yaitu mengimport XGBregressor untuk mengimplementasikan algoritma XGBoost. Pada baris kode kedua yaitu implementasi nisao n_estimators dan gamma dimana pada penelitian ini mengguanakan n_estimators=100 dan gamma=0.01.

pada baris kode ketiga mengambil data yang akan diimplementasikan kedalam algoritma XGBoost yaitu X_train dan y_train.

```
from xgboost import XGBRegressor
my_model = XGBRegressor(n_estimators=100, gamma=0.01)
my_model.fit(X_train, y_train, verbose=False)
```

Gambar 4.21 Implementasi algoritma XGBoost

Gambar 4.22 adalah tampilan algoritma XGBoost setelah kode program pada Gambar 4.21 dijalankan

```
XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=0.01, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=100, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)
```

Gambar 4.22 Algoritma XGBoost

4.1.9 Testing dan evaluasi model

Gambar 4.23 pada baris kode pertama yaitu fungsi program untuk memprediksi harga *bitcoin* menggunakan data X_test. Pada baris kode kedua yaitu fungsi program untuk mengevaluasi hasil prediksi menggunakan *Mean Absolute Error (MAE)*, Pada baris kode ketiga yaitu fungsi program untuk mengevaluasi hasil prediksi menggunakan *Root Mean Square Error (RMSE)*

```
predictions = my_model.predict(X_test)
print("Mean Absolute Error - MAE : " +
str(mean_absolute_error(y_test, predictions)))
print("Root Mean squared Error - RMSE : " +
str(math.sqrt(mean_squared_error(y_test, predictions))))
```

Gambar 4.23 Testing dan evaluasi model

Gambar 4.24 yaitu nilai evaluasi dari prediksi harga *bitcoin*, Dimana nilai *Mean Absolute Error (MAE)* = 0,0416 sedangkan nilai *Root Mean Square Error (RMSE) = 0,056*

```
Mean Absolute Error - MAE : 0.041678597926819 16
Root Mean squared Error - RMSE : 0.05694288857417103
```

Gambar 4.24 Nilai MAE dan RMSE

Gambar 4.25 pada baris kode pertama dan kedua yaitu memprediksi data train (X_train) dan data test (X_test) menggunakan model algoritma XGBoost. Pada baris kode ketiga dan keempat yaitu mendefinisikan fungsi reshape pada data train_predict dan test_predict, adapun fungsi reshape digunakan untuk membuat atau merubah baris kolom. Pada baris kode kelima dan keenam yaitu menapilkan jumlah data pada train_predict dan test predict

```
train_predict=my_model.predict(X_train)
test_predict=my_model.predict(X_test)

train_predict = train_predict.reshape(-1,1)
test_predict = test_predict.reshape(-1,1)

print("Train data prediction:", train_predict.shape)
print("Test data prediction:", test_predict.shape)
```

Gambar 4.25 Train predict dan test predict

Gambar 4.26 adalah output dari baris kode kelima dan keenam pada Gambar 4.25. dimana, train data prediction memiliki 255 data dan test data predictin memiliki 229 data.

```
Train data prediction: (555, 1)
Test data prediction: (229, 1)
```

Gambar 4.26 Jumlah data train predict dan test predict

Gambar 4.27 pada yaitu mengubah data train_predict dan test_predict kedalam bentuk aslinya dengan fungsi scaler.inverse_transform, kemudian merubah baris kolom menggunakan reshape.

```
train_predict = scaler.inverse_transform(train_predict)
test_predict = scaler.inverse_transform(test_predict)
original_ytrain                                          =
scaler.inverse_transform(y_train.reshape(-1,1))
original_ytest                                           =
scaler.inverse_transform(y_test.reshape(-1,1))
```

Gambar 4.27 Transform data train_predict dan test_predict

Gambar 4.28 pada baris kode pertama sampai keenam adalah fungsi program untuk memplotting data train predict. Pada baris kode ketujuh sampai sebelas adalah fungsi program untuk memplotting data test predict. Pada baris ke sebelas sampai ke dua puluh satu adalah fungsi program untuk memvisualisasikan data original close price, traine predicted close price dan test predicted close price dalam bentuk grafik plotting perbandingan antara ketiga data tersebut.

```
# shift train predictions for plotting
look_back=time_step
trainPredictPlot = np.empty_like(closedf)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[look_back:len(train_predict)+look_back,
:] = train_predict
print("Train predicted data: ", trainPredictPlot.shape)

# shift test predictions for plotting
testPredictPlot = np.empty_like(closedf)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(train_predict)+(look_back*2)+1:len(c
losedf)-1, :] = test_predict
print("Test predicted data: ", testPredictPlot.shape)

names = cycle(['Original close price','Train predicted
close price','Test predicted close price'])

plotdf = pd.DataFrame({'date': close_stock['date'],
                                    'original_close':
close_stock['close'],
                                'train_predicted_close':
trainPredictPlot.reshape(1,-1)[0].tolist(),
                                 'test_predicted_close':
testPredictPlot.reshape(1,-1)[0].tolist()})

fig          =          px.line(plotdf,x=plotdf['date'],
y=[plotdf['original_close'],plotdf['train_predicted_clos
```

```
e'],
                                                plotdf['test_p
redicted_close']],
                  labels={'value':'Close  price','date':
'Date'})
fig.update_layout(title_text='Comparision        between
original close price vs predicted close price',
                      plot_bgcolor='white',  font_size=15,
font_color='black',legend_title_text='Close Price')
fig.for_each_trace(lambda     t:      t.update(name     =
next(names)))

fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.show()
```

Gambar 4.28 Plotting hasil prediksi

Gambar 4.29 adalah hasil visualisasi perbandingan original close price dan predicted close price dari implementasi algoritma *XGBoost*. Dimana dapat dilihat pada gambar grafik dibagi menjadi tiga data yaitu original close price, train predicted close price dan test predicted close price.

Gambar 4.29 Visualisasi Hasil prediksi

4.2 Pembahasan

Dalam tahap ini penulis akan mengevaluasi hasil implementasi algoritma *XGBoost*

dengan cara memebandingakan nilai *MAE* dan *RMSE* terhadap teknik dan model yang dilakuakan dalam implementasi algoritma *XGBoost*. Dilakuakannya tahapan ini bertujuan untuk mendapkan hasil terbaik dari prediksi harga *bitcoin* menggunakan algoritma *XGBoost*.

4.2.1 Validasi Normalisasi

Tabel 4.1 adalah validasi perbandingan implementasi algoritma *XGBoost* menggunakan normaliasai *Minmax* dan implementasi algoritma *XGBoost* tanpa menggunakan normalisasi *Minmax*. Pada tabel 4.1 validasi normalisasi, nilai *RMSE* dan *MAE* dalam implementasi algoritma *XGBoost* untuk prediksi harga *bitcoin* mendapat nilai terkecil ketika menggunakan normalisasi *Minmax*, dengan nilai *RMSE* 0,057 dan *MAE* 0,042.

Algoritma	Normalisasi	*RMSE*	*MAE*
XGBoost	*MinMax*	0,057	0,042
XGBoost	-	3.038,68	2.225,66

Tabel 4.1 Validasi Normalisasi

4.2.2 Validasi *Hyperparameter tuning*

Tabel 4.2 adalah konfigurasi *hyperparameter tuning*, dimana pada implementasi algoritma *XGBoost* untuk prediksi harga *bitcoin* menggunkan proses *hyperparameter tuning* yang berfungsi untuk optimasi parameter yang berguna untuk meningkatkan kinerja model dalam memprediksi harga *bitcoin.* Terdapat 2 parameter yang digunakan pada penelitian ini yaitu *n_estimator* dan *gamma.* Konfigurasi *hyperparameter* dari *gird search* dipilih berdasarkan nilai *RMSE* dan *MAE* terkecil. Adapun *gird search* yang menghasilkan nilai *RMSE* dan *MAE* terkecil yaitu *n_estimator*= 100 dan *gamma*= 0,01.

Hyperparameter	*Gird Search Value*	Nilai *Hyperparameter* Terbaik
n_estimator	400, 300, 200, 100	100
gamma	0,01, 0,1, 0,2, 0,3, 0,4, 1, 1,5, 2	0,01

Tabel 4.2 Validasi *hyperparameter tuning*