LAMPIRAN

Lampiran 1. Dataset Penelitian

Data penelitian dapat diakses melalui *link* berikut :

https://bit.ly/DatasetHargaCabaiRawitJaTim

Lampiran 2. Data *Understanding*

```
# Import library yang diperlukan
import pandas as pd
import matplotlib.pyplot as plt
from plotly.offline import init_notebook_mode, iplot
from plotly import graph_objs as go

# Import Data
df = pd.read_excel("JAN 2020 - APRIL 2024.xlsx")

# Menampilkan banyak baris dan kolom data
df.shape

# Menampilkan banyak null dalam data dan tipe data
df.info()

# Menampilkan statistika deskriptif
df.describe()

# Mengubah nama kolom
df.rename(columns={"TANGGAL": "Tanggal",
                   "HARGA CABAI": "Harga Cabai"}, inplace=True)

# Checking dan handling missing value
df.isnull().sum()

# Menyalin Dataset dengan Nama Variabel Baru
data = df.copy()

# Menentukan Indeks
data = df.set_index('Tanggal')
```

```
data.sort_index(inplace=True)
# Ubah indeks menjadi datetime
data.index = pd.to_datetime(data.index)

# Melakukan iterasi
for i, row in data.iterrows():
    if i.weekday() == 5:
        previous_friday = i - pd.Timedelta(days=1)
        next_friday = i + pd.Timedelta(days=7 - i.weekday())

        # Menghitung rata-rata hari Jumat sebelumnya dan berikutnya
        average_value = (data.loc[previous_friday]['Harga Cabai'] +
data.loc[next_friday]['Harga Cabai']) / 2

        # Mengisi nilai hari Sabtu dan Minggu dengan rata-rata
        data.at[i, 'Harga Cabai'] = average_value
        data.at[i + pd.Timedelta(days=1), 'Harga Cabai'] = average_value

print(data)

# Menampilkan boxplot
fig, ax = plt.subplots()
ax.boxplot(data['Harga Cabai'], vert=False, showmeans=True, meanline=True,
           labels=['Harga Cabai'], patch_artist=True,
           medianprops={'linewidth': 2, 'color': 'blue'},
           meanprops={'linewidth': 2, 'color': 'red'})
plt.show()

# Menampilkan histogram
tanggal = df['Tanggal']
harga_cabai = data['Harga Cabai']

plt.hist(harga_cabai, bins=10, edgecolor='black', alpha=0.7)
plt.xlabel('Harga Cabai')
plt.ylabel('Frekuensi')
plt.title('Histogram Harga Cabai')
plt.show()

# Menampilkan line chart
plt.figure(figsize=(10, 6))
```

```
data['Harga Cabai'].plot()
plt.ylabel('Harga Cabai (Rp)')
plt.xlabel('Tahun')
plt.show()

# Menampilkan line chart interaktif
def plotly_df(data, title=''):
    """Visualize all the dataframe columns as line plots."""
    common_kw = dict(x=data.index, mode='lines')
    data = [go.Scatter(y=data[c], name=c, **common_kw) for c in data.columns]
    layout = dict(title=title)
    fig = dict(data=data, layout=layout)
    iplot(fig, show_link=False)

plotly_df(data, title='Harga Cabai')
```

Lampiran 3. *Code* ARIMA

```
# Import library yang diperlukan
!pip install pmdarima
from IPython.core.debugger import set_trace

import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import time
plt.style.use(style='seaborn')
%matplotlib inline

from sklearn.metrics import mean_absolute_percentage_error, mean_squared_error
from math import sqrt
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from scipy.stats import boxcox
import statsmodels.api as sm
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.stats.diagnostic import acorr_ljungbox
from scipy.stats import kstest, norm
```

```
# Import Data
data = pd.read_excel("JAN 2020 - APRIL 2024 (Sesudah Imputasi Data).xlsx")

# Ubah Nama Kolom
data.rename(columns={"TANGGAL": "Tanggal",
                     "HARGA CABAI": "Harga"}, inplace=True)

# Menentukan Indeks
df = data.set_index('Tanggal')

# Melihat Stasioneritas Dari Plot Rolling Mean dan Rolling Standar Deviasi
rolLmean = df.rolling(12).mean()
rolLstd = df.rolling(12).std()

# Plot Rolling Statistics
plt.figure(figsize=(16,7))
fig = plt.figure(1)
orig = plt.plot(df, color='blue',label='Original')
mean = plt.plot(rolLmean, color='red', label='Rolling Mean')
std = plt.plot(rolLstd, color='black', label = 'Rolling Std')
plt.legend(loc='best')
plt.title('Rolling Mean & Standard Deviation')
plt.show(block=False)

# Melihat Stasioneritas dari ADF Test
result = adfuller(df.Harga.dropna())
print(f"ADF Statistic: {result[0]}")
print(f"p-value: {result[1]}")

# Plot ACF Data Pengamatan
plot_acf(df['Harga'])
plt.show()

# Plot PACF Data Pengamatan
plot_pacf(df['Harga'])
plt.show()

# Melihat Nilai Box-Cox
# Ambil kolom 'harga' dari DataFrame sebagai data
data = df['Harga']
```

```
# Terapkan transformasi Box-Cox
transformed_data, lambda_value = stats.boxcox(data)
print("Nilai Lambda untuk transformasi Box-Cox:", lambda_value)

# Melakukan Transformasi Logaritma
plt.figure(figsize=(16,7))
fig = plt.figure(1)
ts_log = np.log(df)
plt.plot(ts_log)
plt.show()

# Melihat Rolling Statistics Setelah Trasnformasi Logaritma
rolLmean = ts_log.rolling(12).mean()
rolLstd = ts_log.rolling(12).std()

plt.figure(figsize=(16,7))
fig = plt.figure(1)
orig = plt.plot(ts_log, color='blue',label='Original')
mean = plt.plot(rolLmean, color='red', label='Rolling Mean')
std = plt.plot(rolLstd, color='black', label = 'Rolling Std')
plt.legend(loc='best')
plt.title('Rolling Mean & Standard Deviation')
plt.show(block=False)

# Melakukan Differencing dan Melihat Hasil Rolling Statistic
plt.figure(figsize=(16,7))
fig = plt.figure(1)
ts_log_diff = ts_log - ts_log.shift()
plt.plot(ts_log_diff)

#Determing rolling statistics
rolLmean = ts_log_diff.rolling(12).mean()
rolLstd = ts_log_diff.rolling(12).std()

#Plot rolling statistics:
orig = plt.plot(ts_log_diff, color='blue',label='Original')
mean = plt.plot(rolLmean, color='red', label='Rolling Mean')
std = plt.plot(rolLstd, color='black', label = 'Rolling Std')
plt.legend(loc='best')
plt.title('Rolling Mean & Standard Deviation')
```

```
plt.show(block=False)

# Plot ACF dan PACF Setelah Transformasi Logaritma dan Differencing
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(ts_log_diff.dropna(), ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(ts_log_diff.dropna(), ax=ax2)

# Melihat Summary Model ARIMA (7, 1, 7)
model = ARIMA(df, order=(7,1,7))
results_ARIMA = model.fit()
results_ARIMA.summary()

# Fitting Model ARIMA
model = ARIMA(train, order=(7,1,7))
model_fit = model.fit()

# Plot Residual
residuals_train = model_fit.resid
plt.figure(figsize=(12, 6))
plt.plot(residuals_train)
plt.title('Residuals Plot')
plt.show()

# Plot ACF dan PACF Residual
fig, ax = plt.subplots(2, 1, figsize=(16, 8))

plot_acf(residuals_train, ax=ax[0])
plot_pacf(residuals_train, ax=ax[1])
plt.show()

# Diagnostic Checking White Noise
lb_test = acorr_ljungbox(residuals_train, lags=[20], return_df=True)
print("Ljung-Box Test Results:")
print(lb_test)

# Diagnostic Checking Uji Normalitas dengan Kolmogorov Smirnov
ks_statistic, p_value = kstest(residuals_train, 'norm',
args=(residuals_train.mean(), residuals_train.std()))
```

```
# Tentukan tingkat signifikansi
alpha = 0.05

# Evaluasi hasil uji Kolmogorov Smirnov
if p_value < alpha:
    print("Tolak hipotesis nol. Residual tidak memiliki distribusi normal.")
else:
    print("Tidak cukup bukti untuk menolak hipotesis nol. Residual memiliki
distribusi normal.")

# Splitting Data
train_size = int(len(df) * 0.8)
train, test = df[0:train_size], df[train_size:len(df)]
print("Jumlah data latih: ", len(train))
print("Jumlah data uji: ", len(test))

# Evaluasi Model
def calculate_error_value(data_actual, data_forecast):
    MAPE = mean_absolute_percentage_error(data_actual, data_forecast)
    print('MAPE: %.2f%%' % (MAPE*100))
    mse = mean_squared_error(data_actual, data_forecast)
    rmse = np.sqrt(mse)
    print("RMSE: %.2f" % rmse)

    return MAPE, rmse

# Membuat Variabel Prediksi
forecast = model_fit.forecast(steps=len(test))

# Melihat Hasil MAPE dan RMSE
MAPE, rmse = calculate_error_value(test, forecast)
```

Lampiran 4. *Code Prophet*

```
# Import library yang diperlukan
!pip install prophet
from prophet import Prophet
from sklearn.metrics import mean_absolute_percentage_error,
mean_squared_error
```

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')

# Import Data
df = pd.read_excel("JAN 2020 - APRIL 2024 (Sesudah Imputasi Data).xlsx")

# Ubah Nama Kolom
data.rename(columns={"TANGGAL": "Tanggal",
                     "HARGA CABAI": "Harga"}, inplace=True)

# Membuat Fungsi Evaluasi Model
def calculate_error_value(data_actual, data_forecast):
    MAPE = mean_absolute_percentage_error(data_actual, data_forecast)
    print('MAPE: %.2f%%' % (MAPE*100))

    mse = mean_squared_error(data_actual, data_forecast)
    rmse = np.sqrt(mse)
    print("RMSE: %.2f" % rmse)

    return MAPE, rmse

# Menyalin Data
data = df.copy()

# Mengubah Tipe Data
data['Tanggal'] = pd.to_datetime(data['Tanggal'])

# Splitting Data
df_train = int(len(data) * 0.8)
df_train, df_test = data[0:df_train], data[df_train:len(data)]

print("Amount of train test data")
print("Training data: ", len(df_train))
print("Test data: ", len(df_test))
```

```
# Ubah Nama Kolom Train Test
df_train = df_train.rename(columns={'Tanggal':'ds', 'Harga':'y'})
df_test = df_test.rename(columns={'Tanggal':'ds',  'Harga':'y'})


# Model 1 : Prophet Dasar
model = Prophet()
model.fit(df_train)

# Membuat Prediksi Model 1
future = df_test[['ds']]
forecast = model.predict(future)
forecast['yhat_rounded'] = forecast['yhat'].round()

# Membuat Variabel Data Aktual dan Prediksi Model 1
actual = df_test['y']
predicted = forecast['yhat']

# Melihat Hasil MAPE dan RMSE Model 1
MAPE,rmse = calculate_error_value(actual, predicted)

# Model 2 : Prophet Tambahan Hari Libur
model_2 = Prophet()
model_2.add_country_holidays(country_name='ID')
model_2.fit(df_train)

# Membuat Prediksi Model 2
future_2 = df_test[['ds']]
forecast_2 = model_2.predict(future_2)
forecast_2['yhat_rounded'] = forecast_2['yhat'].round()

# Melihat Daftar Hari Libur
print("Include holidays:", end=" ")
print(sorted(model_2.train_holiday_names.tolist()))

# Membuat Variabel Data Aktual dan Prediksi Model 2
actual_2 = df_test['y']
predicted_2 = forecast_2['yhat']

# Melihat Hasil MAPE dan RMSE Model 2
MAPE,rmse = calculate_error_value(actual_2, predicted_2)
```

```
# Model 3 : Prophet Tambahan Musiman Periode Bulanan
model_3 = Prophet()
model_3.add_seasonality(name='monthly', period=30.5, fourier_order=5)
model_3.fit(df_train)

# Membuat Prediksi Model 3
future_3 = df_test[['ds']]
forecast_3 = model_3.predict(future_3)
forecast_3['yhat_rounded'] = forecast_3['yhat'].round()

# Membuat Variabel Data Aktual dan Prediksi Model 3
actual_3 = df_test['y']
predicted_3 = forecast_3['yhat']

# Melihat Hasil MAPE dan RMSE Model 3
MAPE,rmse = calculate_error_value(actual_3, predicted_3)

# Model 4 : Prophet Tambahan Hari Libur dan Musiman Periode Bulanan
model_4 = Prophet()
model_4.add_country_holidays(country_name='ID')
model_4.add_seasonality(name='monthly', period=30.5, fourier_order=5)
model_4.fit(df_train)

# Membuat Prediksi Model 4
future_4 = df_test[['ds']]
forecast_4 = model_4.predict(future_4)
forecast_4['yhat_rounded'] = forecast_4['yhat'].round()

# Melihat Daftar Hari Libur
print("Include holidays:", end=" ")
print(sorted(model_4.train_holiday_names.tolist()))

# Membuat Variabel Data Aktual dan Prediksi Model 4
actual_4 = df_test['y']
predicted_4 = forecast_4['yhat']

# Melihat Hasil MAPE dan RMSE Model 4
MAPE,rmse = calculate_error_value(actual_4, predicted_4)
```

Lampiran 5. *Code* Prediksi dengan Metode ARIMA

```
# Import library yang diperlukan
import pandas as pd
import numpy as np
import warnings
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from datetime import datetime
warnings.filterwarnings("ignore")

from statsmodels.tsa.arima.model import ARIMA
from matplotlib.dates import MonthLocator, DateFormatter

# Import Data
df = pd.read_excel("JAN 2020 - APRIL 2024 (Sesudah Imputasi Data).xlsx")

# Menentukan Indeks
df.set_index('TANGGAL', inplace=True)

# Melakukan prediksi 3 bulan (Mei – Juli 2024)
model_arima = ARIMA(df['HARGA CABAI'], order=(7,1,7))
model_arima_fit = model_arima.fit()

# Mengatur rentang tanggal prediksi
start_date = datetime(2024, 5, 1)
end_date = datetime(2024, 7, 31)

pred_arima = model_arima_fit.predict(start=start_date, end=end_date,
typ='levels')

# Menyimpan hasil prediksi kedalam csv
pred_arima.to_csv("hasil_prediksi_Mei_Jul24.csv", index=True)

# Menampilkan plot data prediksi
pred_arima_values = pred_arima.values

fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(pred_arima.index.values.astype(np.datetime64), pred_arima_values,
label='Prediksi ARIMA', color='red')
ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
```

```
plt.xlabel('Tanggal')
plt.ylabel('Harga')
plt.title('Prediksi Harga Cabai untuk Rentang Mei - Juli 2024')
plt.legend()
plt.show()

# Menampilkan data prediksi
pd.DataFrame(pred_arima)
```