LAMPIRAN

Lampiran 1 *Input* Data

```
import gdown
url =
'https://drive.google.com/file/d/1DKddtrLESeQdpLmAg9ZDBNoLfNM5e52d/vie
w?usp=sharing'
output_path = 'dataset.csv'
gdown.download(url, output_path, quiet=False,fuzzy=True)
```

Lampiran 2 *Display* Data

```
import pandas as pd
import matplotlib.pyplot as plt
try:
    data = pd.read_csv('dataset.csv', encoding='latin1', sep=';', on_bad_lines='skip',
engine='python')
except UnicodeDecodeError:
    data = pd.read_csv('dataset.csv', encoding='ISO-8859-1', sep=';',
on_bad_lines='skip', engine='python')
data.columns = data.columns.str.replace('ï»¿', '')
display(data)
```

Lampiran 3 *Null Check*

```
print(df.isnull().sum())
```

Lampiran 4 *Conversion Data*

```
df = pd.DataFrame(data)
df["TANGGAL"]s = pd.to_datetime(df["TANGGAL"], format="%m/%d/%Y")

print(df.dtypes)
```

Lampiran 5 *Sorted Datetime Index*

```
if not isinstance(df_daily.index, pd.DatetimeIndex):
    print("Error: Index is not a DatetimeIndex.")
elif not df_daily.index.is_monotonic_increasing:
    print("Error: Index is not sorted chronologically.")
else:
    print("Index is a sorted DatetimeIndex.")
    expected_index = pd.date_range(start=df_daily.index.min(),
end=df_daily.index.max(), freq='D')
    missing_dates = expected_index.difference(df_daily.index)
    if missing_dates.empty:
        print("No dates are skipped in the time series data.")
    else:
        print(f"Found {len(missing_dates)} skipped dates.")
        print("Skipped dates:")
        display(missing_dates)
```

Lampiran 6 *Data Encoding*

```
data = data[['TANGGAL', 'HARI', 'KETERANGAN ABSEN']]
data['TANGGAL'] = pd.to_datetime(data['TANGGAL'], dayfirst=True,
errors='coerce')
def classify_attendance_by_keterangan(ket):
    if ket == 'H':
        return 1  # Hadir penuh
    elif ket == 'PL':
        return 1  # Hadir, walau pulang lebih awal
    elif ket == 'A':
        return 0  # Tidak hadir
    elif ket == 'L':
        return None  # Libur, tidak dihitung
    else:
        return None  # Kode tidak dikenali
data['KEHADIRAN'] = data['KETERANGAN
ABSEN'].apply(classify_attendance_by_keterangan)
data = data.dropna(subset=['KEHADIRAN'])
print(data[['TANGGAL', 'HARI', 'KETERANGAN ABSEN',
'KEHADIRAN']].head())
```

Lampiran 7 Data *Split*

```
from sklearn.model_selection import TimeSeriesSplit
test_size = int(len(sample_data) * 0.2)
train_val_data = sample_data[:-test_size]
test_data = sample_data[-test_size:]
tscv = TimeSeriesSplit(n_splits=5)
for train_index, val_index in tscv.split(train_val_data):
    train_fold = train_val_data.iloc[train_index]
    val_fold = train_val_data.iloc[val_index]
print(f"Train Size: {len(train_fold)}, Validation Size: {len(val_fold)}")
```

Lampiran 8 *Display Data Split*

```
display(train_fold)
display(val_fold)
display(test_data)
```

Lampiran 9 Visualisasi Kehadiran Dosen Perhari

```
kehadiran_per_hari = data.groupby('HARI')['KEHADIRAN'].sum()
kehadiran_per_hari.plot(kind='bar', color='skyblue', edgecolor='black')
plt.xlabel('Hari')
plt.ylabel('Total Kehadiran')
plt.title('Analisis Kehadiran Dosen Berdasarkan Hari')
plt.xticks(rotation=45)
plt.show()
```

Lampiran 10 Visualisasi Kehadiran Dosen Perbulan

```
data['TANGGAL'] = pd.to_datetime(data['TANGGAL'], format="%d/%m/%Y")
data['BULAN'] = data['TANGGAL'].dt.strftime('%B')

kehadiran_per_bulan =
data.groupby('BULAN')['KEHADIRAN'].sum().sort_values()

plt.figure(figsize=(10,6))
kehadiran_per_bulan.plot(kind='bar', color='lightcoral', edgecolor='black')
plt.xticks(rotation=45)
plt.xlabel('Bulan')
plt.ylabel('Total Kehadiran')
plt.title('Total Kehadiran Dosen Berdasarkan Bulan')
plt.show()
```

Lampiran 11 Stasioneritas

```
import pandas as pd
from statsmodels.tsa.stattools import adfuller
def check_stationarity(timeseries):
    print('Results of Dickey-Fuller Test:')
    dftest = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags
Used','Number of Observations Used'])
    for key,value in dftest[4].items():
       dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)
    if dftest[1] <= 0.05:
        print("Kesimpulan: Data stasioner dalam rataan (p-value <= 0.05).")
        return True
    else:
        print("Kesimpulan: Data tidak stasioner dalam rataan (p-value > 0.05).")
        return False
print("Melakukan uji ADF pada data harian:")
is_stationary = check_stationarity(train_fold.dropna())
if not is_stationary:
    print("\nMelakukan differencing orde 1...")
    df_daily_diff1 = train_fold.diff().dropna()
    print("Melakukan uji ADF pada data harian setelah differencing orde 1:")
    is_stationary_diff1 = check_stationarity(df_daily_diff1)
    if not is_stationary_diff1:
        print("\nData masih belum stasioner setelah differencing orde 1.
Pertimbangkan differencing orde 2 atau musiman.")
else:
    print("\nData sudah stasioner dalam rataan, tidak diperlukan differencing orde
1.")
```

Lampiran 12 Tren Kehadiran Harian

```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 6))
sns.lineplot(x=df_daily.index, y=df_daily)
plt.xlabel("Tanggal")
plt.ylabel("Jumlah Kehadiran Harian")
plt.title("Tren Kehadiran Dosen Harian")
plt.grid(True)
plt.show()
```

Lampiran 13 Tren Kehadiran Perminggu

```
df_resampled = df_daily.resample('W').mean()
plt.figure(figsize=(12, 5))
sns.lineplot(x=df_resampled.index, y=df_resampled, marker="o", linestyle="-")
plt.xlabel("Tanggal")
plt.ylabel("Rata-rata Kehadiran")
plt.title("Tren Kehadiran Mingguan")
plt.xticks(rotation=45)
plt.grid()
plt.show()
```

Lampiran 14 Dekomposisi

```
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(sample_data, model="additive",
period=30)
fig = decomposition.plot()
fig.set_size_inches(12, 8)
plt.show()
```

Lampiran 15 Plot ACF dan PACF Non-Musiman

```
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import pandas as pd
data_for_acf_pacf = train_fold.dropna()
print("\nMelakukan plot ACF dan PACF pada data harian yang stasioner.")
# Plot ACF
plt.figure(figsize=(12, 6))
ax1 = plt.subplot(211)
plot_acf(data_for_acf_pacf, lags=40, ax=ax1)
plt.title('Autocorrelation Function (ACF)')
# Plot PACF
ax2 = plt.subplot(212)
plot_pacf(data_for_acf_pacf, lags=40, ax=ax2)
plt.title('Partial Autocorrelation Function (PACF)')
plt.tight_layout()
plt.show()
```

Lampiran 16 Plot ACF dan PACF Musiman

```
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.seasonal import seasonal_decompose
try:
    decomposition_weekly = seasonal_decompose(train_fold.dropna(),
model='additive', period=7)
    seasonal_component = decomposition_weekly.seasonal
    print("\nPlotting ACF and PACF of the Seasonal Component (Period=7):")
    plt.figure(figsize=(12, 6))
    ax1 = plt.subplot(211)
    plot_acf(seasonal_component, lags=2*7, ax=ax1)
    plt.title('ACF of Weekly Seasonal Component')
    ax2 = plt.subplot(212)
    plot_pacf(seasonal_component, lags=2*7, ax=ax2)
    plt.title('PACF of Weekly Seasonal Component')
    plt.tight_layout()
    plt.show()
    print("\nInterpreting the plots for seasonal parameters (P, Q) with seasonal
period (s) = 7:")
    print("- Seasonal Autoregressive (P): Look at the PACF plot. A significant spike
at lag s (7) followed by a cutoff suggests a seasonal AR(1) term (P=1). Significant
spikes at lags s, 2s (14), etc., followed by a cutoff, might suggest a seasonal AR(P)
term.")
    print("- Seasonal Moving Average (Q): Look at the ACF plot. A significant
spike at lag s (7) followed by a cutoff suggests a seasonal MA(1) term (Q=1).
Significant spikes at lags s, 2s (14), etc., followed by a cutoff, might suggest a
seasonal MA(Q) term.")
    print("- Seasonal Differencing (D): This is usually determined by looking at the
seasonal pattern in the decomposition. If the seasonal component doesn't have a
stable mean over time, seasonal differencing (D=1) might be needed. Based on the
previous decomposition plot (Cell 1AEN_jm5IBdq), the seasonal component
seems relatively stable, suggesting D=0.")
except ValueError as e:
    print(f"Could not perform seasonal decomposition: {e}")
    print("Please ensure the data length is sufficient for the specified period (at least
2 full cycles).")
```

Lampiran 17 *Seasonal*

```
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_acf
import pandas as pd
train_fold_seasonal_diff = train_fold.diff(periods=7).dropna()
print("\nPlotting ACF of the Seasonally Differenced Data (Period=7):")
plt.figure(figsize=(12, 6))
plot_acf(train_fold_seasonal_diff, lags=2*7)
plt.title('ACF of Seasonally Differenced Data (Period=7)')
plt.xlabel('Lags')
plt.ylabel('Autocorrelation')
plt.show()
print("\nInterpreting the ACF of Seasonally Differenced Data:")
print("If the ACF of the seasonally differenced data quickly drops to zero or within the confidence intervals, it suggests that seasonal differencing of order 1 (D=1) was sufficient to make the data seasonally stationary.")
print("If there are still significant spikes at seasonal lags (multiples of 7), further seasonal differencing might be needed (though this is less common).")
print("Based on the previous seasonal decomposition plot (Cell 1AEN_jm5IBdq), the seasonal component looked relatively stable, which initially suggested D=0. This plot will help confirm if D=1 is necessary or if D=0 is sufficient.")
```

Lampiran 18 SARIMA *Resault*

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
order = (1, 0, 2)
seasonal_order = (1, 1, 1, 7)
model_2 = SARIMAX(train_fold, order=order, seasonal_order=seasonal_order, enforce_stationarity=False, enforce_invertibility=False)
model_2_fit = model_2.fit()
print(model_2_fit.summary())
```

Lampiran 19 Estimasi Parameter

```
import pandas as pd
import numpy as np
parameter_estimates = {}
def extract_params_from_summary(model_fit_summary):
    summary_str = model_fit_summary.tables[1].as_html()
    params_df = pd.read_html(summary_str, header=0, index_col=0)[0]
    return params_df
try:
    params_121 = extract_params_from_summary(model_2_fit.summary())
    parameter_estimates['SARIMA(1,0,2) x (1,1,1,7)'] = params_121
except Exception as e:
    print(f"Could not extract parameters for SARIMA(1,0,2) x (1,1,1,7): {e}")
    parameter_estimates['SARIMA(1,0,2) x (1,1,1,7)'] = pd.DataFrame()
combined_params_list = []
for model_name, params_df in parameter_estimates.items():
    if not params_df.empty:
        params_df['Model'] = model_name
        combined_params_list.append(params_df.reset_index().rename(columns={'in
dex': 'Parameter'}))
if combined_params_list:
    final_params_df = pd.concat(combined_params_list, ignore_index=True)
    column_mapping = {
        'index': 'Parameter',
        'coef': 'Koefisien',
        'std err': 'Std. Error',
        'z': 'Z',
        'P>|z|': 'P>|z|',
        '[0.025': '[0.025',
        '0.975]': '0.975]' }
    final_params_df = final_params_df.rename(columns=column_mapping)
    if 'P>|z|' in final_params_df.columns and not
final_params_df['P>|z|'].isnull().all():
        significance_level = 0.05
        final_params_df['Hasil Uji Signifikan'] = final_params_df['P>|z|'].apply(
            lambda p_value: 'Signifikan' if not pd.isna(p_value) and p_value <
significance_level else 'Tidak Signifikan' )
    else:
        final_params_df['Hasil Uji Signifikan'] = 'N/A (No P-value)'
    desired_order = ['Model', 'Parameter', 'Koefisien', 'Std. Error', 'Z', 'P>|z|', 'Hasil
Uji Signifikan', '[0.025', '0.975]']
    existing_ordered_columns = [col for col in desired_order if col in
final_params_df.columns]
```

```
final_params_df = final_params_df[existing_ordered_columns]
   print(" 📋 Tabel Hasil Pendugaan Parameter Model ARIMA:")
   display(final_params_df)
else:
   print("Tidak dapat membuat tabel parameter. Pastikan model sudah di-fit dan
ringkasan model dapat diakses.")
```

Lampiran 20 *Diagnostik Checking*

```
import pandas as pd
from statsmodels.stats.diagnostic import acorr_ljungbox
models = {
   'ARIMA(1,0,2) x (1, 1, 1, 7) (model_2)': model_2_fit,}
ljung_box_table_data = []
test_lags = [1,2,3,4,5,6,7,8,9,10]
print(" 📊 Tabel Hasil Uji White Noise (Ljung-Box Test) pada Residual Model:")
print("-" * 80)
for model_name, model_fit in models.items():
   try:
      residuals = model_fit.resid
      ljung_box_results = acorr_ljungbox(residuals, lags=test_lags,
return_df=True)
      for index, row in ljung_box_results.iterrows():
         lag = index
         p_value = row['lb_pvalue']
         test_result = 'White Noise' if p_value > 0.05 else 'Not White Noise'
         if test_result == 'White Noise':
            ljung_box_table_data.append({
               'Model': model_name,
               'Lag': lag,
               'P.Value': p_value,
               'Hasil Uji': test_result})
   except Exception as e:
      print(f"Error performing Ljung-Box test for {model_name}: {e}")
      for lag in test_lags:
          ljung_box_table_data.append({
            'Model': model_name,
            'Lag': lag,
            'P.Value': np.nan,
            'Hasil Uji': f'Error: {e}'})
ljung_box_df = pd.DataFrame(ljung_box_table_data)
if ljung_box_df.empty:
   print("Tidak ada lag yang menunjukkan residual White Noise (p-value > 0.05)
untuk model-model yang diuji pada lag yang ditentukan.")
else:
   display(ljung_box_df)
```

Lampiran 21 Plot Residual

```
import pandas as pd
from statsmodels.stats.diagnostic import acorr_ljungbox
import matplotlib.pyplot as plt
residuals = model_2_fit.resid
plt.figure(figsize=(12, 6))
residuals.plot(title="Residuals of SARIMA Model")
plt.xlabel("Tanggal")
plt.ylabel("Residual")
plt.grid(True)
plt.show()
test_lags = [1, 7, 14, 20]
print("\n📊 Hasil Uji White Noise (Ljung-Box Test) pada Residual Model SARIMA:")
print("-" * 80)
ljung_box_results = acorr_ljungbox(residuals, lags=test_lags, return_df=True)
display(ljung_box_results)
```

Lampiran 22 Evaluasi *Error*

```
from sklearn.metrics import mean_absolute_error, mean_squared_error,
mean_absolute_percentage_error
import numpy as np
import pandas as pd
model_evaluations = {}
def evaluate_model(model_fit, train_data, test_data):
    start_index = len(train_data)
    end_index = len(train_data) + len(test_data) - 1
    forecast = model_fit.predict(start=start_index, end=end_index)
    forecast = pd.Series(forecast.values, index=test_data.index)
    forecast = np.maximum(0, forecast)
    mae = mean_absolute_error(test_data, forecast)
    mse = mean_squared_error(test_data, forecast)
    rmse = np.sqrt(mse)
    y_true, y_pred = test_data.values, forecast.values
    non_zero_mask = y_true != 0
```

```
    if np.sum(non_zero_mask) > 0:
        mape = np.mean(np.abs((y_true[non_zero_mask] - y_pred[non_zero_mask])
/ y_true[non_zero_mask])) * 100
    else:
    mape = np.nan
    return mae, mse, rmse, mape
mae_1, mse_1, rmse_1, mape_1 = evaluate_model(model_2_fit, train_val_data,
test_data)
model_evaluations['SARIMA(1,0,2) x (1, 1, 1, 7)'] = {
    'MAE': mae_1, 'MSE': mse_1, 'RMSE': rmse_1,'MAPE (%)': mape_1,}
evaluations_df = pd.DataFrame.from_dict(model_evaluations, orient='index')
evaluations_df.index.name = 'Model ARIMA'
print("\n📊 Tabel Hasil Evaluasi Model ARIMA pada Data Test:")
display(evaluations_df)
```

Lampiran 23 Hasil Peramalan

```
import pandas as pd
import datetime as dt
from IPython.display import display
import matplotlib.pyplot as plt
def rekomendasi_kegiatan_tahunan(model_fit, train_data):
    today = pd.to_datetime(dt.datetime.today().date())
    current_year = today.year
    days_ahead = (5 - today.weekday() + 7) % 7
    next_monday = today + pd.Timedelta(days=days_ahead)
    last_date = train_data.index[-1]
    start_date = max(last_date + pd.Timedelta(days=1), next_monday)
    #start_date = pd.Timestamp(year=today.year, month=today.month, day=1)
    end_date = pd.to_datetime(f"{current_year}-12-31")
    n_days = (end_date - start_date).days + 1
    forecast_index = pd.date_range(start=start_date, periods=n_days, freq='D')
    forecast = model_fit.predict(start=len(train_data), end=len(train_data) + n_days
- 1)
```

```
print(f"🗓️ Prediksi Kehadiran Dosen per Bulan untuk Tahun {current_year}\n")
    for bulan in range(1, 13):
        mask = (forecast_series.index.month == bulan) & (forecast_series.index.year
== current_year)
        forecast_bulan = forecast_series[mask]
        if forecast_bulan.empty:
            continue
        df_bulan = pd.DataFrame({
            'Tanggal': forecast_bulan.index,
            'Prediksi Hadir': forecast_bulan.round(0).astype(int),
        })
        df_bulan['Hari'] = df_bulan['Tanggal'].dt.weekday
        df_bulan['Hari Efektif'] = df_bulan['Hari'].apply(lambda x: 'v' if x < 5 else
'x')
        df_bulan.set_index('Tanggal', inplace=True)
        df_bulan['Hari Efektif Khusus'] = ''
        for minggu, grup in df_bulan[df_bulan['Hari'] <
5].groupby(pd.Grouper(freq='W-MON')):
            if not grup.empty:
                best_day = grup['Prediksi Hadir'].idxmax()
                df_bulan.loc[best_day, 'Hari Efektif Khusus'] = '⭐'
best_day_bulan = forecast_bulan.idxmax()
        best_value = forecast_bulan.max()
        print(f"📌 Bulan: {forecast_bulan.index[0].strftime('%B %Y')}")
        print(f"✅ Rekomendasi Hari Terbaik Bulanan:
{best_day_bulan.strftime('%A, %d %B %Y')} — Prediksi: {int(best_value)}
dosen")
        styled_df = df_bulan.copy()
        styled_df['Tanggal'] = styled_df.index.strftime('%d/%m/%Y')
        styled_df = styled_df[['Tanggal', 'Prediksi Hadir', 'Hari Efektif', 'Hari Efektif
Khusus']]
        styled = styled_df.style.applymap(
            lambda val: 'background-color: #d4edda' if val == 'v' else 'background-
color: #f8d7da',
            subset=['Hari Efektif']
        ).applymap(
            lambda val: 'background-color: #fff3cd' if val == '⭐' else '',
            subset=['Hari Efektif Khusus']
)
```

```
display(styled)
        plt.figure(figsize=(10, 4))
        plt.plot(forecast_bulan, marker='o', label='Prediksi Kehadiran')
        plt.axvline(best_day_bulan, color='red', linestyle='--', label='Hari Terbaik
Bulan')
        plt.title(f'Prediksi Kehadiran - {forecast_bulan.index[0].strftime("%B
%Y")}')
        plt.xlabel('Tanggal')
        plt.ylabel('Jumlah Dosen Hadir')
        plt.grid(True)
        plt.legend()
        plt.tight_layout()
        plt.show()
```

Lampiran 24 Frekuensi Keterangan Absen

```
import matplotlib.pyplot as plt
import seaborn as sns
keterangan_absen_filter = ['A', 'H', 'L', 'PC', 'PL']
data_filtered = data_asli[data_asli['KETERANGAN
ABSEN'].isin(keterangan_absen_filter)].copy()
attendance_counts = data_filtered.groupby(['HARI', 'KETERANGAN
ABSEN']).size().reset_index(name='Count')
hari_order = ['Senin', 'Selasa', 'Rabu', 'Kamis', 'Jumat', 'Sabtu', 'Minggu']
attendance_counts['HARI'] = pd.Categorical(attendance_counts['HARI'],
categories=hari_order, ordered=True)
attendance_counts = attendance_counts.sort_values('HARI')
plt.figure(figsize=(12, 7))
sns.barplot(data=attendance_counts, x='HARI', y='Count', hue='KETERANGAN
ABSEN', palette='tab10')
plt.xlabel('Hari')
plt.ylabel('Jumlah Kejadian')
plt.title('Frekuensi Keterangan Absen (A, H, L, PC, PL) per Hari')
plt.xticks(rotation=45)
plt.legend(title='Keterangan Absen', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```

Lampiran 25 Perbandingan Jumlah Kehadiran

```
comparison_df_daily = pd.DataFrame({
    'Tanggal': test_data.index,
    'Kehadiran Sebenarnya': test_data.values,
    'Forecast SARIMA': forecast_series.values.round(0).astype(int)})
comparison_df_daily['Selisih (Sebenarnya - Forecast)'] =
comparison_df_daily['Kehadiran Sebenarnya'] - comparison_df_daily['Forecast
SARIMA']
comparison_df_daily.set_index('Tanggal', inplace=True)
print(" 📊 Tabel Perbandingan Kehadiran Harian (Data Test vs Forecast
SARIMA):")
display(comparison_df_daily)
```

RIWAYAT HIDUP

Saya, Putu Tiara Widiastini, lahir di Banyuning pada tanggal 22 Oktober 2001. Saya adalah anak dari pasangan Bapak Made Widiarta, S.E. dan Ibu Luh Ayu Sri Gustini. Saya merupakan anak pertama dari lima bersudara. Saya menempuh pendidikan di SD Negeri 7 Banyuning dan lulus pada tahun 2014, kemudian melanjutkan pendidikan ke jenjang SMP di SMP Negeri 2 Singaraja dan lulus pada tahun 2017. Pendidikan menengah saya selesaikan di SMK Negeri 3 Singaraja pada tahun 2020, lalu melanjutkan studi ke Universitas Pendidikan Ganesha Jurusan Teknik Informatika Program S1 Studi Ilmu Komputer.