**EX:No.8 221501034**

**15/04/25**

**IMPLEMENT PROGRAM FOR DECOMPOSING TIME SERIES DATA INTO TREND AND SEASONALITY.**

**AIM:**

To implement program for decomposing time series data into trend and seasonality.

**ALGORITHM:**

**OBJECTIVE:**

Smooth the electric production data to reduce noise, highlight trends, and prepare for forecasting.

**BACKGROUND:**

1.Time series data has short-term fluctuations.

2.Moving average reduces noise and clarifies trends.

3.Smoothed data improves forecast accuracy and interpretability.

**SCOPE OF THE PROGRAM:**

1.Load and clean dataset

2.Convert date column to datetime

3.Aggregate data monthly and yearly

4.Apply 3-month and 12-month moving averages

5.Plot original vs smoothed data

**ALGORITHM:**

1.Import libraries

2.Load dataset

3.Preprocess and set datetime index

4.Resample data (monthly, yearly)

5.Apply 3-month & 12-month smoothing

6.Visualize results

**PROCESS:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from statsmodels.tsa.arima.model import ARIMA

from sklearn.metrics import mean\_squared\_error

# Load the dataset (ensure the date parsing and indexing is done correctly)

df = pd.read\_csv('AAPL.csv', parse\_dates=['Date'], index\_col='Date')

# Display the first few rows of the dataset

print(df.head())

# Use the 'Close' price for the time series

time\_series = df['Close']

# Optional: Resample to monthly frequency (if needed for smoothing)

# time\_series = time\_series.resample('M').mean()

# Split the data into training and testing sets (80% training, 20% testing)

train\_size = int(len(time\_series) \* 0.8)

train, test = time\_series[:train\_size], time\_series[train\_size:]

# Fit ARIMA model (you can adjust the order as needed)

model = ARIMA(train, order=(5, 1, 0)) # Tune (p, d, q) for best fit

model\_fit = model.fit()

# Forecasting

forecast = model\_fit.forecast(steps=len(test))

# Plot training data

plt.figure(figsize=(10, 4))

plt.plot(train, label='Training Data', color='blue')

plt.title('Training Data (AAPL Close Price)')

plt.xlabel('Date')

plt.ylabel('Close Price (USD)')

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

# Plot actual test data

plt.figure(figsize=(10, 4))

plt.plot(test, label='Actual Test Data', color='green')

plt.title('Actual Test Data (AAPL Close Price)')

plt.xlabel('Date')

plt.ylabel('Close Price (USD)')

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

# Plot forecast vs actual

plt.figure(figsize=(10, 4))

plt.plot(test.index, forecast, label='ARIMA Forecast', color='red')

plt.plot(test, label='Actual Test Data', color='green')

plt.title('ARIMA Forecast vs Actual (AAPL Close Price)')

plt.xlabel('Date')

plt.ylabel('Close Price (USD)')

plt.legend()

plt.grid(True)

plt.tight\_layout()

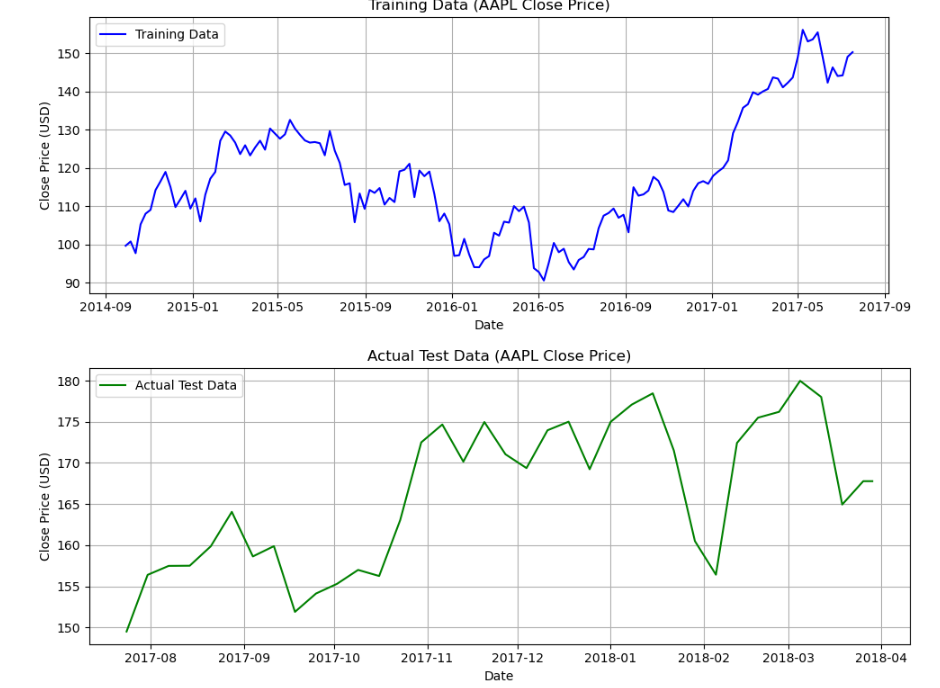
plt.show()

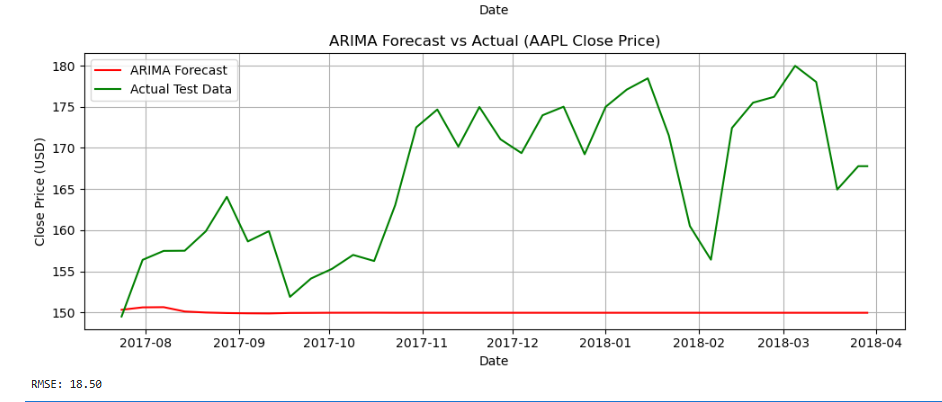
# Calculate RMSE (Root Mean Squared Error)

rmse = np.sqrt(mean\_squared\_error(test, forecast))

print(f'RMSE: {rmse:.2f}')

**OUTPUT:**

******

****

**RESULT:**

The program to implement of program to Create an ARIMA model for time series forecasting created and executed successfully.