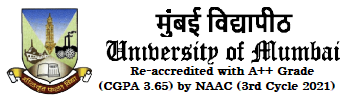
UNIVERSITY OF MUMBAI

**DEPARTMENT OF DATA SCIENCE**



M.Sc. Data Science – Semester II

TIME SERIES ANALYSIS & FORECASTING

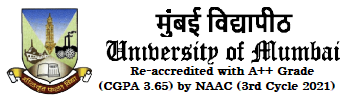
JOURNAL

**2023-2024**

SUBMITTED BY

SATISH MAHENDRA BHANUSHALI

Seat No. 1302071



UNIVERSITY OF MUMBAI

**DEPARTMENT OF DATA SCIENCE**

**CERTIFICATE**

This is to certify that the work entered in this journal was done in the University Department of Data Science laboratory by Mr. SATISH M BHANUSHALI Seat No.1302071for the course of M.Sc. (Data Science) - Semester II (NEP 2020) during the academic year 2023- 2024 in a satisfactory manner.

**\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_**

**Subject In-charge Head of Department**

**\_\_\_\_\_\_\_\_\_\_\_\_**

**External Examiner**

**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SR No. | **PRACTICALS** | **Page No.** | **Date** | **SIGN** |
| **1.** | Fitting and plotting of modified exponential curve. | 04-06 |  |  |
| **2.** | Fitting and plotting of Gompertz curve. | 07-09 |  |  |
| **3.** | Fitting and plotting of logistic curve. | 10-12 |  |  |
| **4.** | Fitting of trend by Moving Average Method | 13-14 |  |  |
| **5.** | Measurement of Seasonal indices Ratio-to-Trend method. | 15-17 |  |  |
| **6.** | Measurement of Seasonal indices Ratio-to-Moving Average method. | 18-20 |  |  |
| **7.** | Measurement of seasonal indices Link Relative method. | 21-23 |  |  |
| **8.** | Calculation of variance of random component by variate difference method. | 24-26 |  |  |
| **9.** | Forecasting by exponential smoothing. | 27-28 |  |  |
| **10.** | Forecasting by short term forecasting methods. | 29-30 |  |  |

**PRACTICAL NO 1**

**AIM:** Fitting and plotting of modified exponential curve.

**INPUT:**

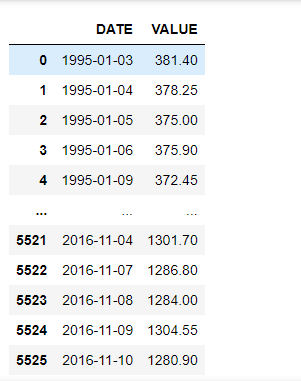
import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

gold\_rates = pd.read\_csv("C:/Users/Satish/Downloads/Gold.csv")

gold\_rates



# Plot the time series to visualize the data

plt.figure(figsize=(12, 6))

plt.plot(gold\_rates['VALUE'], label='Gold Rates')

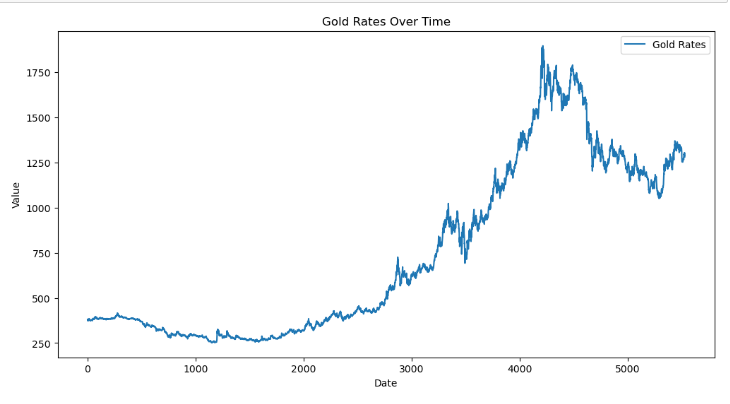
plt.title('Gold Rates Over Time')

plt.xlabel('Date')

plt.ylabel('Value')

plt.legend()

plt.show()



For next part we must convert datatype of date element to date n time datatype

# Ensure DATE is converted to a datetime type and sorted

gold\_rates['DATE'] = pd.to\_datetime(gold\_rates['DATE'])

gold\_rates.sort\_values('DATE', inplace=True)

# Convert dates to a numerical format for fitting (e.g., days since start)

gold\_rates['Time'] = (gold\_rates['DATE'] - gold\_rates['DATE'].min()).dt.days

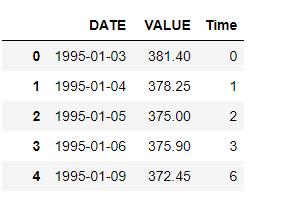
#Define the model for the modified exponential curve

def mod\_exp\_func(t, a, b, c):

return a \* np.exp(b \* t) + c

#Check the converted data

gold\_rates



from scipy.optimize import curve\_fit

# Fit the model to the data

initial\_guess = [1, 0.001, gold\_rates['VALUE'].min()] # Adjust these values as needed

params, covariance = curve\_fit(mod\_exp\_func, gold\_rates['Time'], gold\_rates['VALUE'], p0=initial\_guess)

# Use the optimized parameters to plot the fitted curve

t\_fit = np.linspace(gold\_rates['Time'].min(), gold\_rates['Time'].max(), 1000)

y\_fit = mod\_exp\_func(t\_fit, \*params)

# Plotting the original data and the fitted curve

plt.figure(figsize=(12, 6))

plt.plot(gold\_rates['DATE'], gold\_rates['VALUE'], 'o', label='Original Data')

plt.plot(gold\_rates['DATE'].min() + pd.to\_timedelta(t\_fit, unit='D'), y\_fit, '-', label='Fitted Curve')

plt.title('Gold Rates and Fitted Modified Exponential Curve')

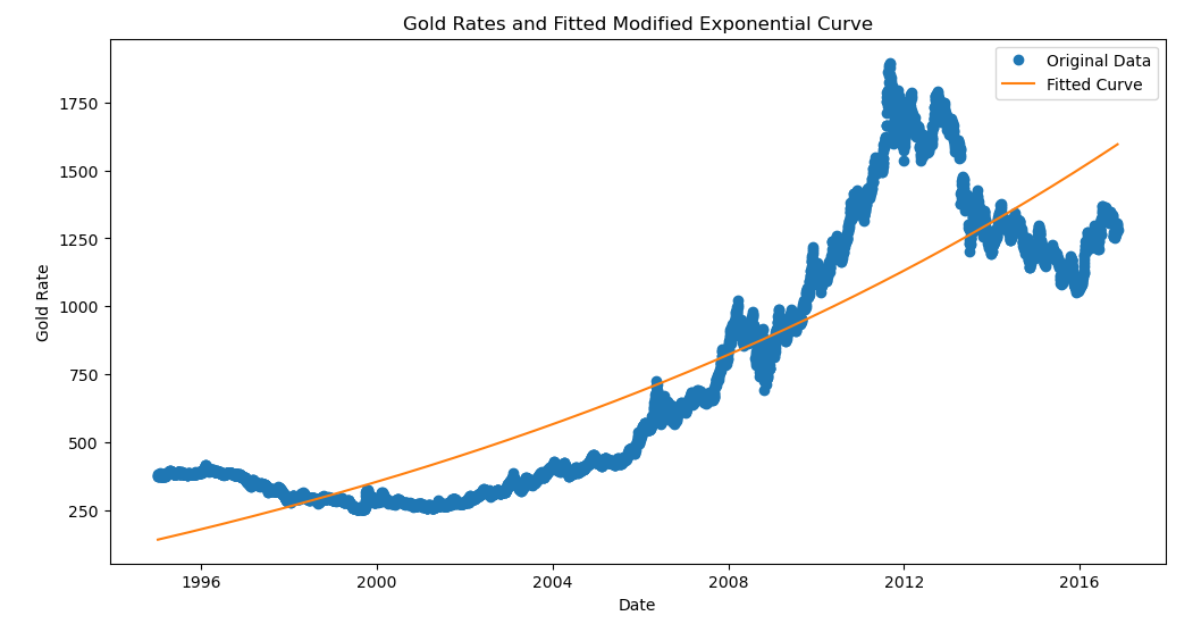
plt.xlabel('Date')

plt.ylabel('Gold Rate')

plt.legend()

plt.show()

**OUTPUT:**



**PRACTICAL NO 2**

**AIM:** Fitting and plotting of Gompertz curve.

**INPUT:**

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

gold\_rates = pd.read\_csv("C:/Users/Satish/Downloads/Gold.csv")

# Plot the time series to visualize the data

plt.figure(figsize=(12, 6))

plt.plot(gold\_rates['VALUE'], label='Gold Rates')

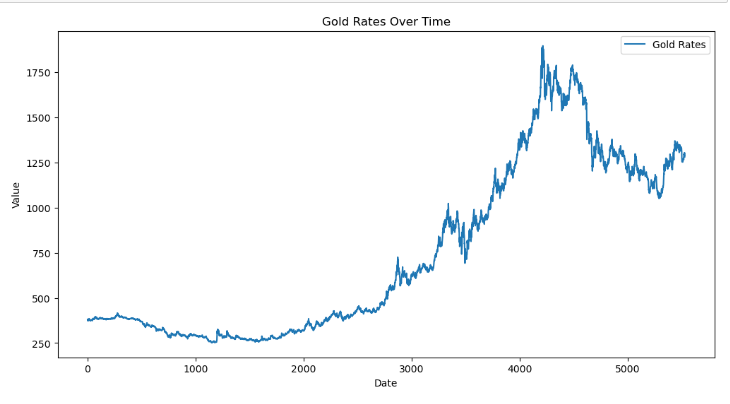
plt.title('Gold Rates Over Time')

plt.xlabel('Date')

plt.ylabel('Value')

plt.legend()

plt.show()



For next part we must convert datatype of date element to date n time datatype

# Ensure DATE is converted to a datetime type and sorted

gold\_rates['DATE'] = pd.to\_datetime(gold\_rates['DATE'])

gold\_rates.sort\_values('DATE', inplace=True)

# Convert dates to a numerical format for fitting (e.g., days since start)

gold\_rates['Time'] = (gold\_rates['DATE'] - gold\_rates['DATE'].min()).dt.days

# Define the Gompertz function

def gompertz\_function(x, A, B, C):

return A \* np.exp(-B \* np.exp(-C \* x))

# Fit the Gompertz curve to the 'Price' data

x\_data = np.arange(len(gold\_rates))

y\_data = gold\_rates['VALUE'].values

# Initial guesses for parameters

p0 = [max(y\_data), 0.1, 0.1]

# Bounds for parameters

bounds = ([0, 0, 0], [2 \* max(y\_data), 1, 1])

# Perform curve fitting with initial guesses and bounds

popt, pcov = curve\_fit(gompertz\_function, x\_data, y\_data, p0=p0, bounds=bounds)

# Plot the original data and the fitted curve

plt.figure(figsize=(10, 6))

plt.scatter(x\_data, y\_data, label='Original data')

plt.plot(x\_data, gompertz\_function(x\_data, \*popt), 'r-', label='Fitted curve')

plt.xlabel('Index')

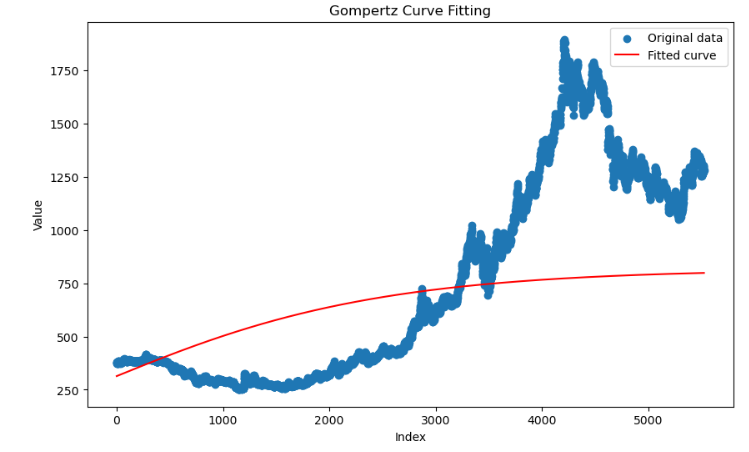
plt.ylabel('Value')

plt.title('Gompertz Curve Fitting')

plt.legend()

plt.show()

**OUTPUT:**



**PRACTICAL NO 3**

**AIM:** fitting and plot of logistic curve.

**INPUT:**

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

gold\_rates = pd.read\_csv("C:/Users/Satish/Downloads/Gold.csv")

# Plot the time series to visualize the data

plt.figure(figsize=(12, 6))

plt.plot(gold\_rates['VALUE'], label='Gold Rates')

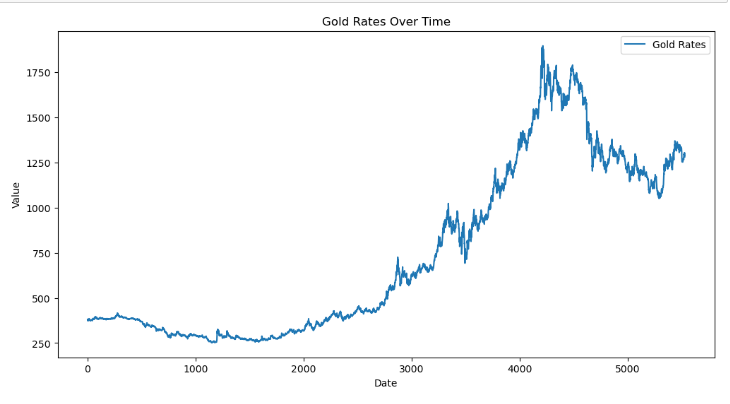
plt.title('Gold Rates Over Time')

plt.xlabel('Date')

plt.ylabel('Value')

plt.legend()

plt.show()



For next part we must convert datatype of date element to date n time datatype

# Ensure DATE is converted to a datetime type and sorted

gold\_rates['DATE'] = pd.to\_datetime(gold\_rates['DATE'])

gold\_rates.sort\_values('DATE', inplace=True)

# Convert dates to a numerical format for fitting (e.g., days since start)

gold\_rates['Time'] = (gold\_rates['DATE'] - gold\_rates['DATE'].min()).dt.days

# Define the logistic function

def logistic\_function(x, L, k, x0):

return L / (1 + np.exp(-k \* (x - x0)))

# Fit the logistic curve to the 'Price' data

x\_data = np.arange(len(gold\_rates))

y\_data = gold\_rates['VALUE'].values

# Initial guesses for parameters

p0 = [max(y\_data), 0.1, np.median(x\_data)]

# Bounds for parameters

bounds = ([0, 0, 0], [2 \* max(y\_data), 1, len(gold\_rates)])

# Perform curve fitting with initial guesses and bounds

popt, pcov = curve\_fit(logistic\_function, x\_data, y\_data, p0=p0, bounds=bounds)

# Plot the original data and the fitted curve

plt.figure(figsize=(10, 6))

plt.scatter(x\_data, y\_data, label='Original data')

plt.plot(x\_data, logistic\_function(x\_data, \*popt), 'r-', label='Fitted curve')

plt.xlabel('Index')

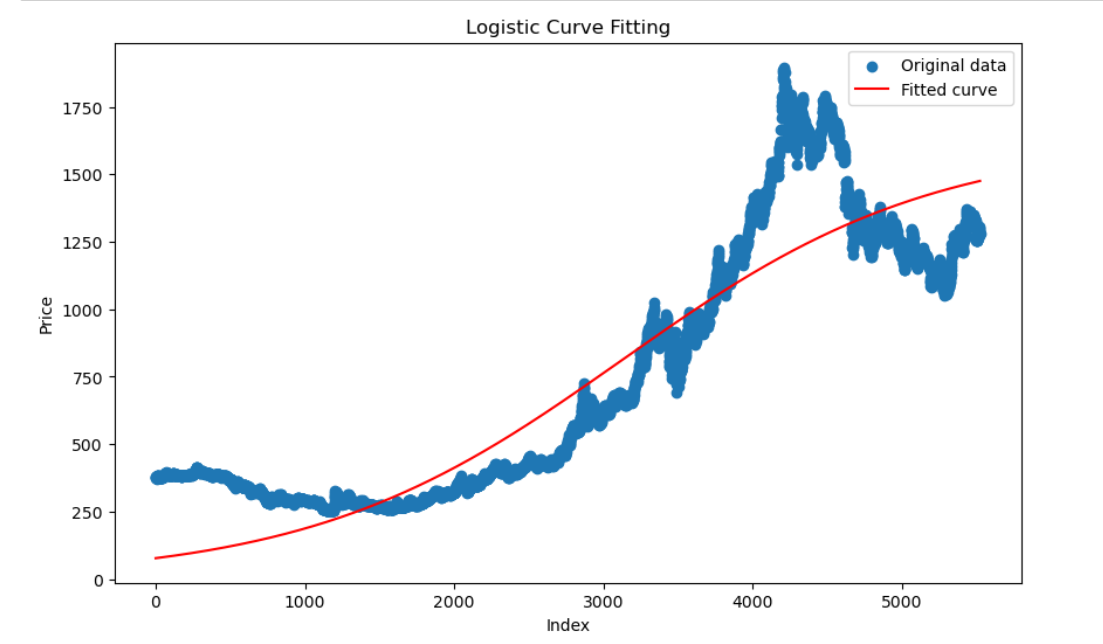
plt.ylabel('Price')

plt.title('Logistic Curve Fitting')

plt.legend()

plt.show()

**OUTPUT:**



**PRACTICAL NO 4**

**AIM:** Fitting of trend by Moving Average Method

**INPUT:**

import pandas as pd

import matplotlib.pyplot as plt

# Step 1: Create a DataFrame with given monthly sales revenue data

data = {

    'Month': ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'],

    'Sales\_000': [125, 145, 186, 131, 151, 192, 137, 157, 198, 143, 163, 204]

}

# Create DataFrame

df = pd.DataFrame(data)

# Step 2: Calculate the three-month moving average

# Apply rolling with a window size of 3 and calculate the mean

df['3\_Month\_MA'] = df['Sales\_000'].rolling(window=3).mean()

# Step 3: Calculate the trend (rate of change)

# The trend can be observed from the three-month moving average

# Calculate the difference between consecutive moving averages

df['Trend'] = df['3\_Month\_MA'].diff()

# Step 4: Calculate the seasonal variation

# Using the additive model, calculate the seasonal variation as the difference between actual sales and moving average

df['Seasonal\_Variation'] = df['Sales\_000'] - df['3\_Month\_MA']

# Plot the data to visualize the original data, moving average, and seasonal variation

plt.figure(figsize=(12, 6))

plt.plot(df['Month'], df['Sales\_000'], label='Original Sales')

plt.plot(df['Month'], df['3\_Month\_MA'], label='3-Month Moving Average', linewidth=2, color='orange')

plt.title('Sales with 3-Month Moving Average')

plt.xlabel('Month')

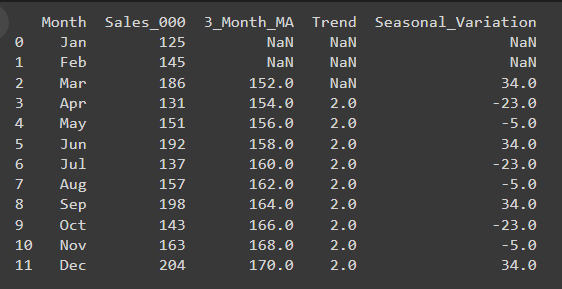
plt.ylabel('Sales ($000)')

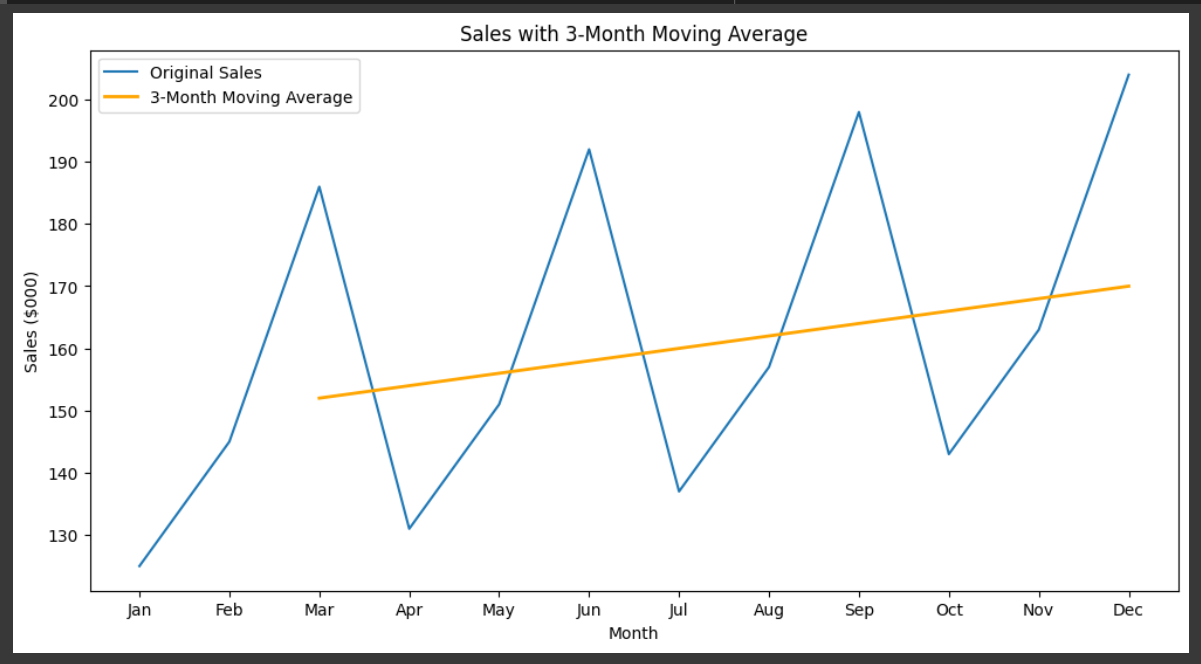
plt.legend()

plt.show()

**OUTPUT:**

Print(df)





**PRACTICAL NO 5**

**AIM:** Measurement of Seasonal indices Ratio-to-Trend method.

**INPUT:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Sample quarterly sales data

data = {

'Quarter': ['Q1', 'Q2', 'Q3', 'Q4'],

'Year1': [150, 200, 250, 300],

'Year2': [160, 210, 260, 310],

'Year3': [170, 220, 270, 320]

}

# Create a DataFrame

df = pd.DataFrame(data)

# Calculate the trend using the overall average for each year

df['Trend\_Year1'] = df[['Year1', 'Year2', 'Year3']].mean(axis=1)

df['Trend\_Year2'] = df[['Year1', 'Year2', 'Year3']].mean(axis=1)

df['Trend\_Year3'] = df[['Year1', 'Year2', 'Year3']].mean(axis=1)

# Calculate the ratio to trend

df['Ratio\_Year1'] = df['Year1'] / df['Trend\_Year1']

df['Ratio\_Year2'] = df['Year2'] / df['Trend\_Year2']

df['Ratio\_Year3'] = df['Year3'] / df['Trend\_Year3']

# Calculate the seasonal indices

seasonal\_indices = {

'Q1': np.mean([df['Ratio\_Year1'][0], df['Ratio\_Year2'][0], df['Ratio\_Year3'][0]]),

'Q2': np.mean([df['Ratio\_Year1'][1], df['Ratio\_Year2'][1], df['Ratio\_Year3'][1]]),

'Q3': np.mean([df['Ratio\_Year1'][2], df['Ratio\_Year2'][2], df['Ratio\_Year3'][2]]),

'Q4': np.mean([df['Ratio\_Year1'][3], df['Ratio\_Year2'][3], df['Ratio\_Year3'][3]])

}

# Normalize the seasonal indices so they sum to 4 (the number of quarters)

total\_indices = sum(seasonal\_indices.values())

normalized\_indices = {k: v \* 4 / total\_indices for k, v in seasonal\_indices.items()}

# Deseasonalize the data

df['Deseasonalized\_Year1'] = df['Year1'] / df['Quarter'].map(normalized\_indices)

df['Deseasonalized\_Year2'] = df['Year2'] / df['Quarter'].map(normalized\_indices)

df['Deseasonalized\_Year3'] = df['Year3'] / df['Quarter'].map(normalized\_indices)

# Plotting

plt.figure(figsize=(14, 8))

# Plot original data

plt.subplot(3, 1, 1)

plt.plot(df['Quarter'], df['Year1'], label='Year 1', marker='o')

plt.plot(df['Quarter'], df['Year2'], label='Year 2', marker='o')

plt.plot(df['Quarter'], df['Year3'], label='Year 3', marker='o')

plt.title('Original Data')

plt.xlabel('Quarter')

plt.ylabel('Sales')

plt.legend()

# Plot seasonal indices

plt.subplot(3, 1, 2)

plt.plot(df['Quarter'], [normalized\_indices[q] for q in df['Quarter']], label='Seasonal Index', marker='o', color='orange')

plt.title('Seasonal Indices')

plt.xlabel('Quarter')

plt.ylabel('Index')

plt.legend()

# Plot deseasonalized data

plt.subplot(3, 1, 3)

plt.plot(df['Quarter'], df['Deseasonalized\_Year1'], label='Year 1 Deseasonalized', marker='o')

plt.plot(df['Quarter'], df['Deseasonalized\_Year2'], label='Year 2 Deseasonalized', marker='o')

plt.plot(df['Quarter'], df['Deseasonalized\_Year3'], label='Year 3 Deseasonalized', marker='o')

plt.title('Deseasonalized Data')

plt.xlabel('Quarter')

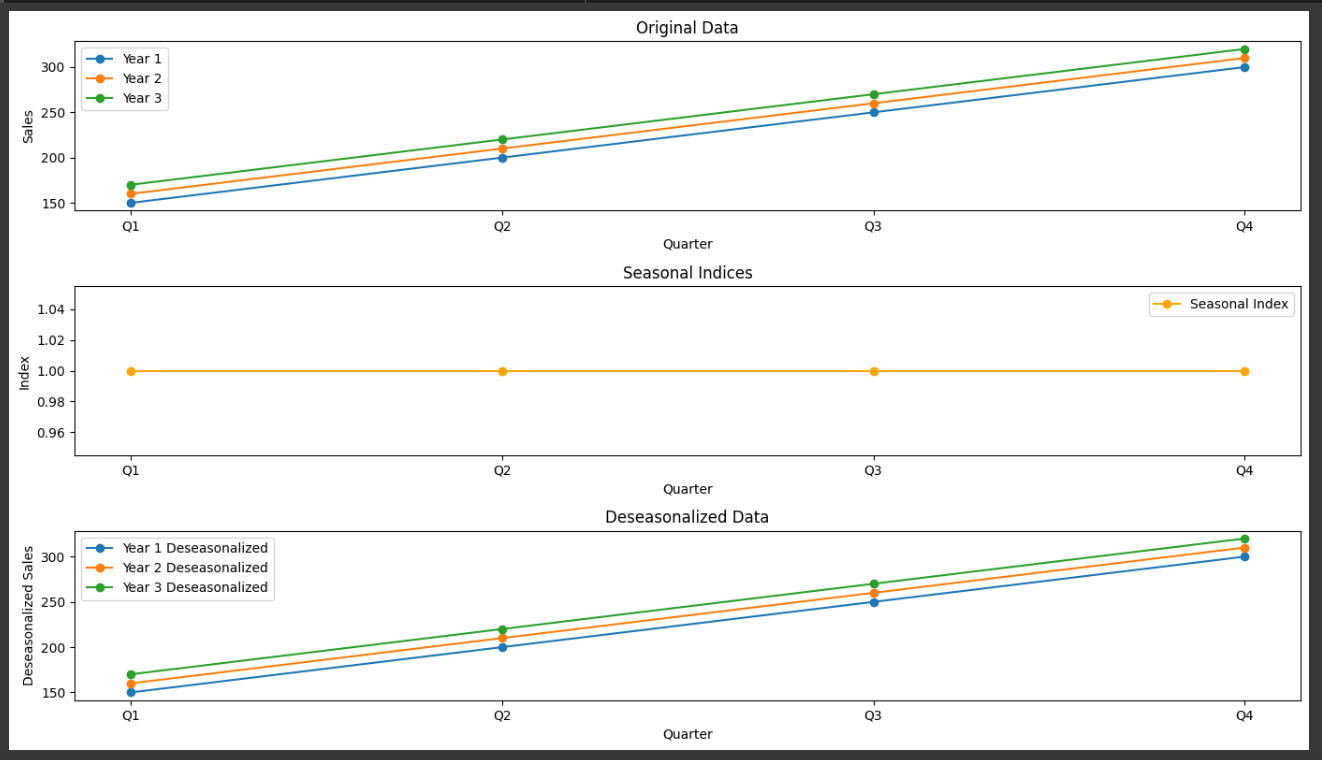
plt.ylabel('Deseasonalized Sales')

plt.legend()

plt.tight\_layout()

plt.show()

**OUTPUT:**



**PRACTICAL NO 6**

**AIM:** Measurement of Seasonal indices Ratio-to-Moving Average method.

**INPUT:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Sample monthly sales data

data = {

'Month': ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'],

'Year1': [150, 180, 200, 220, 250, 270, 300, 320, 350, 370, 400, 420],

'Year2': [160, 190, 210, 230, 260, 280, 310, 330, 360, 380, 410, 430]

}

# Create a DataFrame

df = pd.DataFrame(data)

# Step 1: Calculate Monthly Averages

df['Monthly Average'] = df[['Year1', 'Year2']].mean(axis=1)

# Step 2: Compute Centered Moving Averages

df['Centered Moving Average'] = df['Monthly Average'].rolling(window=2, center=True).mean()

# Step 3: Calculate the Ratio of Actual to Moving Average

df['Ratio'] = df['Monthly Average'] / df['Centered Moving Average']

# Step 4: Estimate Seasonal Indexes

# Normalize the ratios so they sum to the number of months

sum\_ratios = df['Ratio'].sum()

df['Seasonal Index'] = df['Ratio'] \* (len(df) / sum\_ratios)

# Deseasonalize the data

df['Deseasonalized\_Year1'] = df['Year1'] / df['Seasonal Index']

df['Deseasonalized\_Year2'] = df['Year2'] / df['Seasonal Index']

plt.figure(figsize=(14, 8))

# Plot original data

plt.subplot(3, 1, 1)

plt.plot(df['Month'], df['Year1'], label='Year 1', marker='o')

plt.plot(df['Month'], df['Year2'], label='Year 2', marker='o')

plt.title('Original Sales Data')

plt.xlabel('Month')

plt.ylabel('Sales')

plt.legend()

# Plot seasonal indices

plt.subplot(3, 1, 2)

plt.plot(df['Month'], df['Seasonal Index'], label='Seasonal Index', marker='o', color='orange')

plt.title('Seasonal Indices')

plt.xlabel('Month')

plt.ylabel('Index')

plt.legend()

# Plot deseasonalized data

plt.subplot(3, 1, 3)

plt.plot(df['Month'], df['Deseasonalized\_Year1'], label='Year 1 Deseasonalized', marker='o')

plt.plot(df['Month'], df['Deseasonalized\_Year2'], label='Year 2 Deseasonalized', marker='o')

plt.title('Deseasonalized Data')

plt.xlabel('Month')

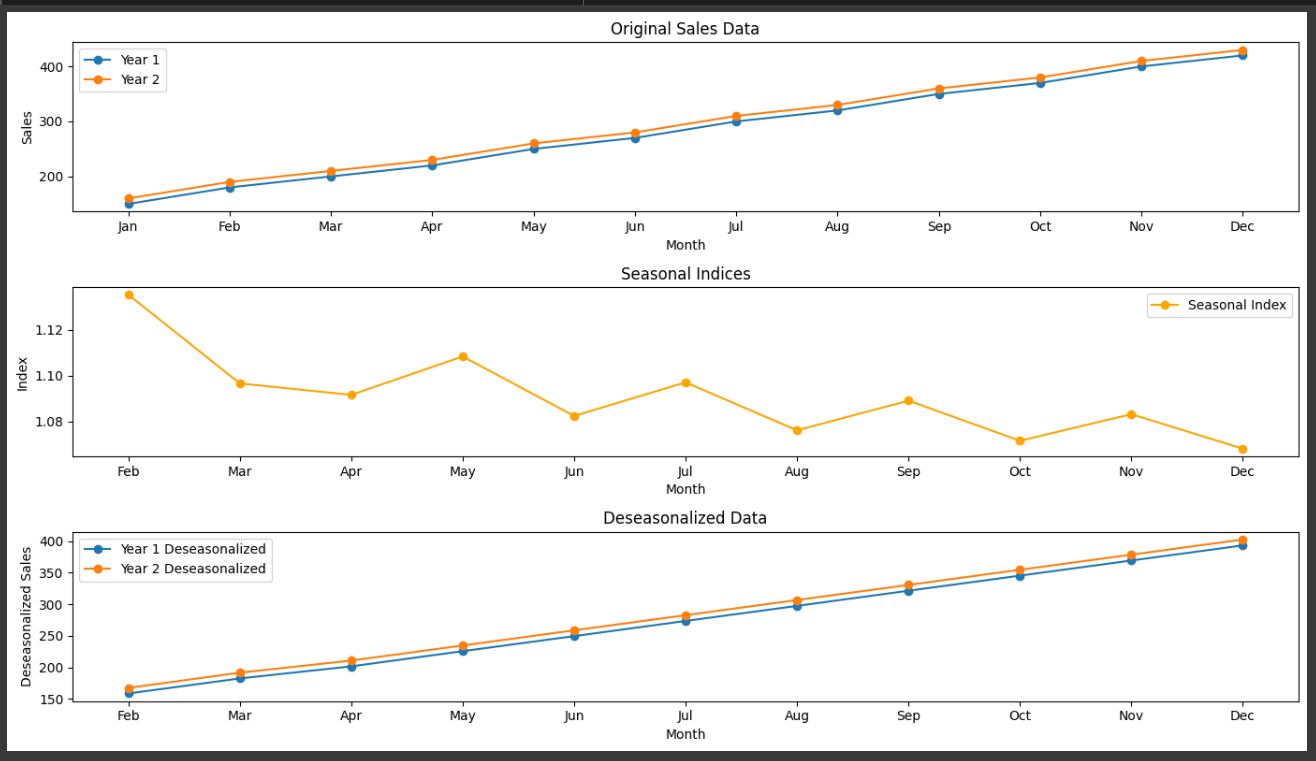
plt.ylabel('Deseasonalized Sales')

plt.legend()

plt.tight\_layout()

plt.show()

**OUTPUT:**



**PRACTICAL NO 7**

**AIM:** Measurement of seasonal indices Link Relative method.

**INPUT:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

# Sample monthly sales data for three years

data = {

'Month': ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'],

'Year1': [150, 180, 200, 220, 250, 270, 300, 320, 350, 370, 400, 420],

'Year2': [160, 190, 210, 230, 260, 280, 310, 330, 360, 380, 410, 430],

'Year3': [170, 200, 220, 240, 270, 290, 320, 340, 370, 390, 420, 440]

}

# Create a DataFrame

df = pd.DataFrame(data)

# Calculate the link relatives

df['Link\_Relative\_Y2'] = df['Year2'] / df['Year1']

df['Link\_Relative\_Y3'] = df['Year3'] / df['Year2']

# Calculate the average link relatives for each month

df['Average\_Link\_Relative'] = df[['Link\_Relative\_Y2', 'Link\_Relative\_Y3']].mean(axis=1)

# Normalize the seasonal indices

seasonal\_indices = df['Average\_Link\_Relative']

seasonal\_indices /= seasonal\_indices.sum()

seasonal\_indices \*= 12 # Because we have 12 months

# Deseasonalize the data

df['Deseasonalized\_Year1'] = df['Year1'] / seasonal\_indices

df['Deseasonalized\_Year2'] = df['Year2'] / seasonal\_indices

df['Deseasonalized\_Year3'] = df['Year3'] / seasonal\_indices

# Plotting

plt.figure(figsize=(14, 8))

# Plot original data

plt.subplot(3, 1, 1)

plt.plot(df['Month'], df['Year1'], label='Year 1', marker='o')

plt.plot(df['Month'], df['Year2'], label='Year 2', marker='o')

plt.plot(df['Month'], df['Year3'], label='Year 3', marker='o')

plt.title('Original Data')

plt.xlabel('Month')

plt.ylabel('Sales')

plt.legend()

# Plot seasonal indices

plt.subplot(3, 1, 2)

plt.plot(df['Month'], seasonal\_indices, label='Seasonal Index', marker='o', color='orange')

plt.title('Seasonal Indices')

plt.xlabel('Month')

plt.ylabel('Index')

plt.legend()

# Plot deseasonalized data

plt.subplot(3, 1, 3)

plt.plot(df['Month'], df['Deseasonalized\_Year1'], label='Year 1 Deseasonalized', marker='o')

plt.plot(df['Month'], df['Deseasonalized\_Year2'], label='Year 2 Deseasonalized', marker='o')

plt.plot(df['Month'], df['Deseasonalized\_Year3'], label='Year 3 Deseasonalized', marker='o')

plt.title('Deseasonalized Data')

plt.xlabel('Month')

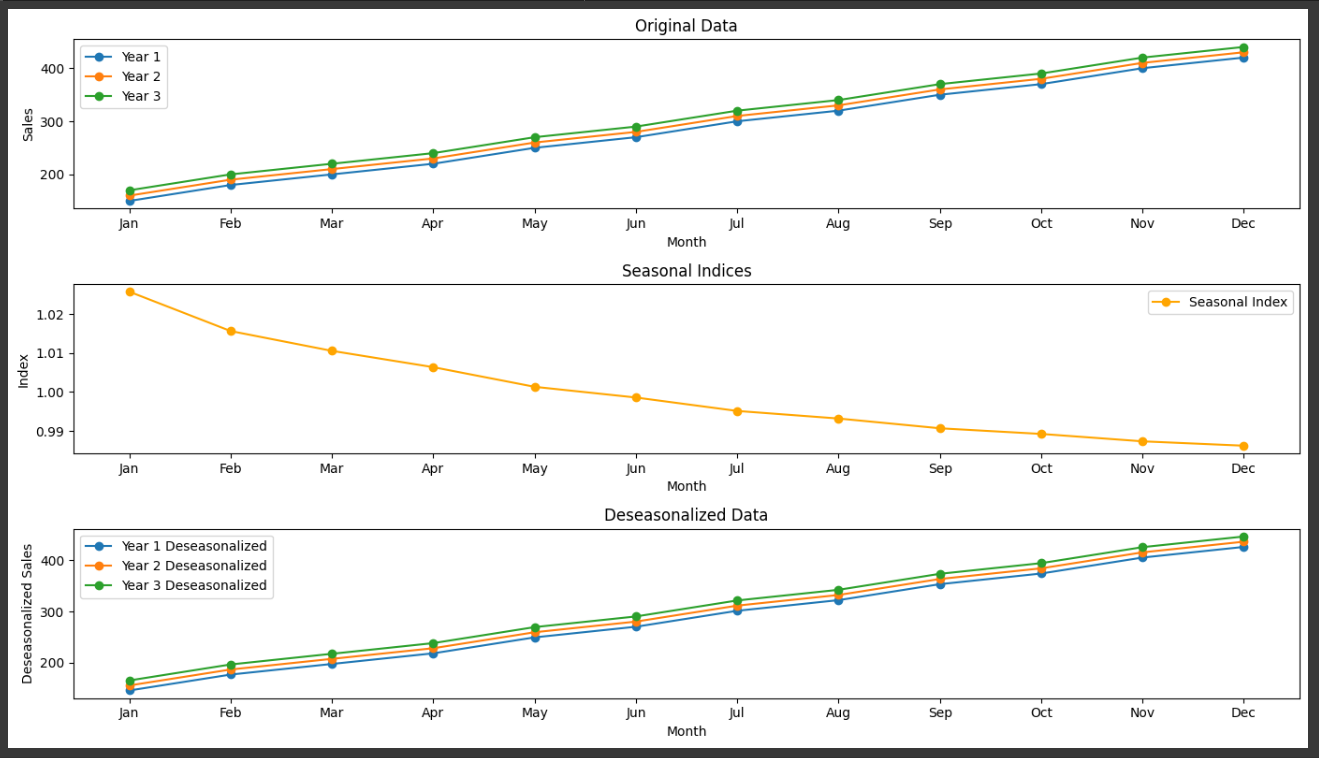
plt.ylabel('Deseasonalized Sales')

plt.legend()

plt.tight\_layout()

plt.show()

**OUTPUT:**



**PRACTICAL NO 8**

**AIM:** Calculation of variance of random component by variate difference method.

**INPUT:**

import numpy as np

import matplotlib.pyplot as plt

# Given time series data

time\_series = [47, 64, 23, 71, 38, 64, 55, 41, 59, 48]

# Calculate differences

differences = np.diff(time\_series)

# Calculate mean of differences

mean\_diff = np.mean(differences)

# Calculate variance of differences

var\_diff = np.var(differences, ddof=1)

# Calculate variance of random components

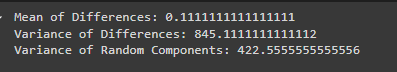
var\_random = var\_diff / 2

# Print results

print(f"Mean of Differences: {mean\_diff}")

print(f"Variance of Differences: {var\_diff}")

print(f"Variance of Random Components: {var\_random}")



# Plot the differences

plt.plot(differences, marker='o')

plt.title('Differences Between Successive Observations')

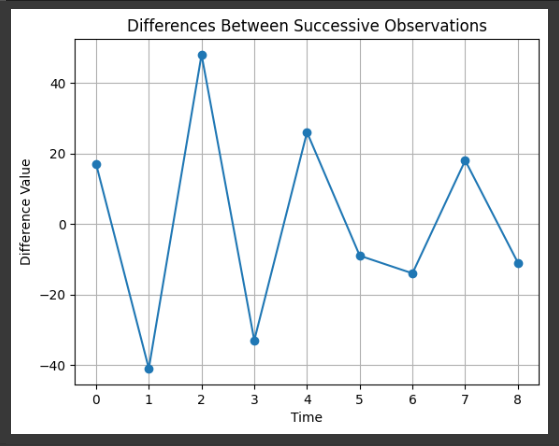
plt.xlabel('Time')

plt.ylabel('Difference Value')

plt.grid(True)

plt.show()

**OUTPUT:**



# Create a figure and axis

fig, ax1 = plt.subplots()

# Plot the time series on the primary y-axis

color = 'tab:blue'

ax1.set\_xlabel('Time')

ax1.set\_ylabel('Time Series', color=color)

ax1.plot(time\_series, marker='o', color=color, label='Time Series')

ax1.tick\_params(axis='y', labelcolor=color)

# Create a secondary y-axis for the differences

ax2 = ax1.twinx()

color = 'tab:red'

ax2.set\_ylabel('Differences', color=color)

ax2.plot(range(1, len(time\_series)), differences, marker='x', linestyle='--', color=color, label='Differences')

ax2.tick\_params(axis='y', labelcolor=color)

# Add title and grid

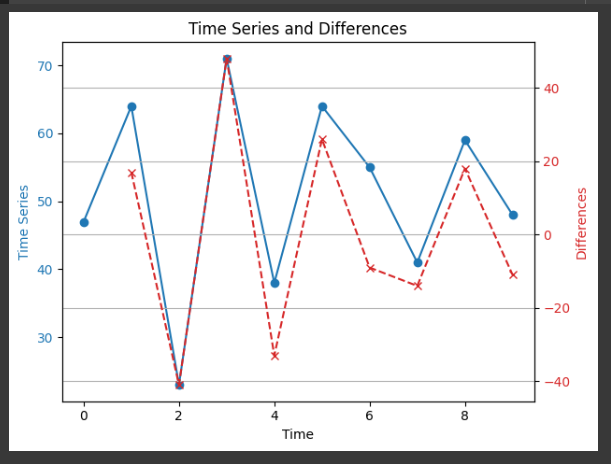
plt.title('Time Series and Differences')

fig.tight\_layout() # Adjust layout to make room for both y-axes

plt.grid(True)

plt.show()

**OUTPUT:**



**PRACTICAL NO 9**

**AIM:** Forecasting by exponential smoothing.

**INPUT:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from statsmodels.tsa.holtwinters import ExponentialSmoothing

# Sample data

data = [112, 118, 132, 129, 121, 135, 148, 148, 136, 119, 104, 118, 115, 126, 141, 135, 125, 149, 170, 170,158, 133, 114, 140, 145, 150, 178, 163, 172, 178, 199, 199, 184, 162, 146, 166, 171, 180, 193, 181,183, 218, 230, 242, 209, 191, 172, 194, 196, 196, 236, 235, 229, 243, 264, 272, 237, 211, 180, 201,204, 188, 235, 227, 234, 264, 302, 293, 259, 229, 203, 242, 233, 267, 269, 270, 315, 364, 347, 312,274, 237, 278, 284, 277, 317, 313, 318, 374, 413, 405, 355, 306, 271, 306, 315, 301, 356, 348, 355,422, 465, 467, 404, 347, 305, 336, 340, 318, 362, 348, 363, 435, 491, 505, 404, 359, 310, 337, 360,342, 406, 396, 420, 472, 548, 559, 463, 407, 362, 405, 417, 391, 419, 461, 472, 535, 622, 606, 508,461, 390, 432]

# Convert data to pandas DataFrame

df = pd.DataFrame(data, columns=['value'])

# Define the model

model = ExponentialSmoothing(df['value'], trend='add', seasonal='add', seasonal\_periods=12)

# Fit the model

fit = model.fit()

# Forecast future values

forecast = fit.forecast(12)

#represents the number of future time periods for which you want to generate forecasts.

#The time period length (e.g., months, weeks, days) depends on the frequency of your original data.

# Plot the results

plt.figure(figsize=(10, 6))

plt.plot(df['value'], label='Original')

plt.plot(fit.fittedvalues, label='Fitted', linestyle='--')

plt.plot(forecast, label='Forecast', linestyle='--')

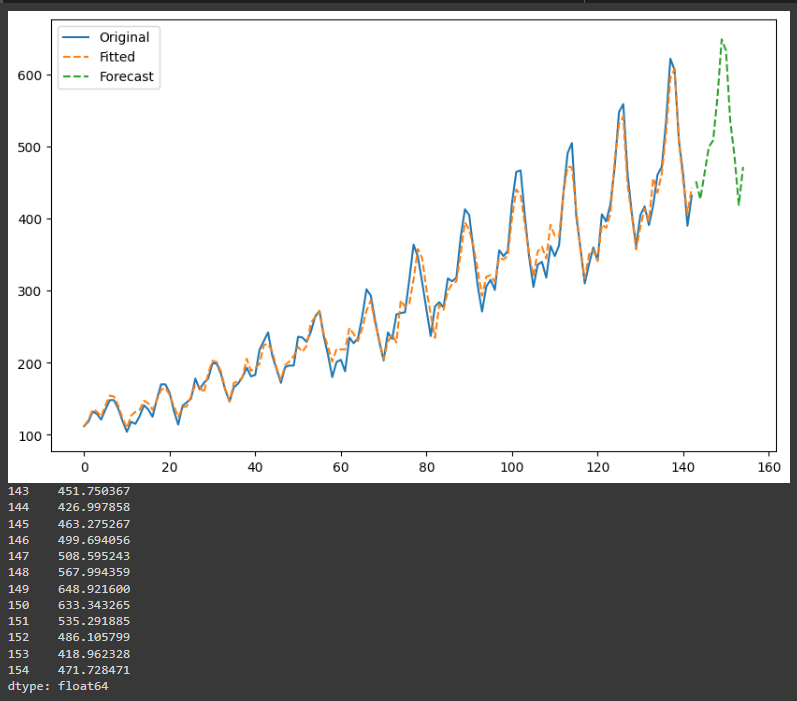
plt.legend()

plt.show()

# Print forecasted values

print(forecast)

**OUTPUT:**



**PRACTICAL NO 10**

**AIM:** Forecasting by short term forecasting methods.

**INPUT:**

import pandas as pd

import matplotlib.pyplot as plt

from statsmodels.tsa.arima.model import ARIMA

# Sample data

data = [112, 118, 132, 129, 121, 135, 148, 148, 136, 119, 104, 118, 115, 126, 141, 135, 125, 149, 170, 170,158, 133, 114, 140, 145, 150, 178, 163, 172, 178, 199, 199, 184, 162, 146, 166, 171, 180, 193, 181]

# Convert data to pandas DataFrame

df = pd.DataFrame(data, columns=['value'])

# Fit the ARIMA model

model = ARIMA(df['value'], order=(5, 1, 1))  # (p,d,q) order

fit = model.fit()

# Forecast future values

forecast = fit.forecast(steps=10)  # Forecast next 10 periods

# Plot the results

plt.figure(figsize=(10, 6))

plt.plot(df['value'], label='Original')

plt.plot(fit.fittedvalues, label='Fitted', linestyle='--')

plt.plot(range(len(df), len(df) + 10), forecast, label='Forecast', linestyle='--')

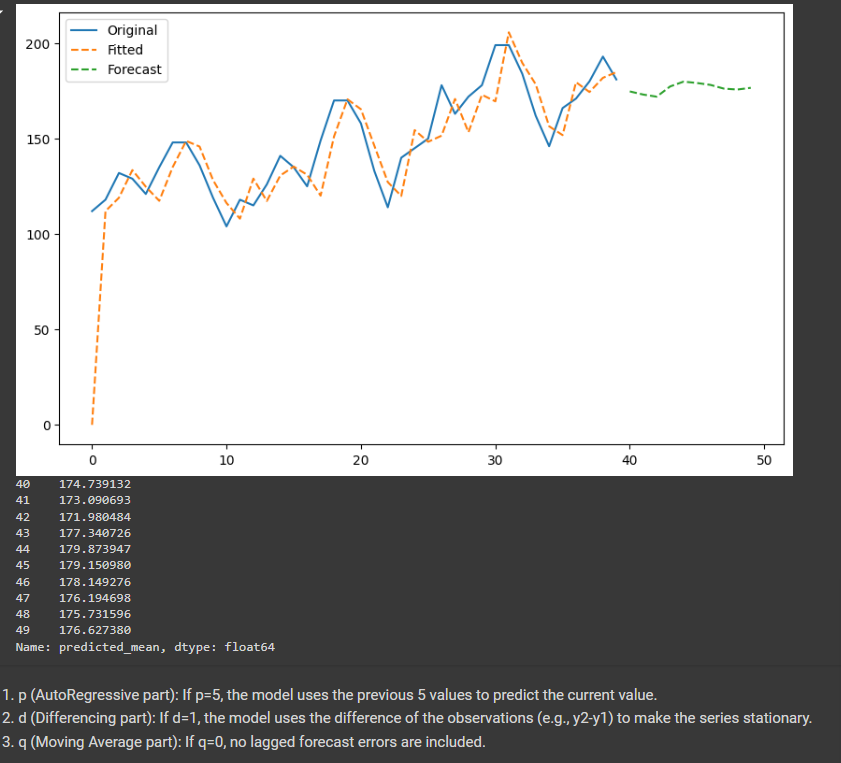
plt.legend()

plt.show()

# Print forecasted values

print(forecast)

**OUTPUT:**



.

1. p (AutoRegressive part): If p=5, the model uses the previous 5 values to predict the current value.
2. d (Differencing part): If d=1, the model uses the difference of the observations (e.g., y2-y1) to make the series stationary.
3. q (Moving Average part): If q=0, no lagged forecast errors are included.