मुंबई विद्यापीठ
University of Mumbai
Re-accredited with A++ Grade
(CGPA 3.65) by NAAC (3rd Cycle 2021)

University of Mumbai

DEPARTMENT OF COMPUTER SCIENCE

JOURNAL

M. Sc. (Data Science) (NEP) Semester-II

2024-2025

Time Series Analysis and Forecasting

Submitted by

Aaditya Sheelkumar Pal

Seat No. **DS24142**

मुंबई विद्यापीठ
University of Mumbai
Re-accredited with A++ Grade
(CGPA 3.65) by NAAC (3rd Cycle 2021)

University of Mumbai

DEPARTMENT OF COMPUTER SCIENCE

CERTIFICATE

This is to certify that the work entered in this journal was done in the Department of Computer Science, University of Mumbai by Mr./Ms. ____ Seat No. ________________ for the course of **M.Sc. (Data Science) (NEP) Semester-II** during the academic year **2024-25** in a satisfactory manner.

____
Subject In-charge
Department of Computer Science

____
Head
Department of Computer Science

____
External Examiner

INDEX

PRACTICAL 1

Aim: Fitting and plotting of modified exponential curve

Problem Statement:

- **Curve Fitting Objective**: Develop a model to fit a modified exponential curve to a given dataset, enabling accurate predictions and trend analysis.
- Plotting Requirement: Visualize the fitted modified exponential curve alongside the original data to evaluate the model's performance and effectiveness.

Theory:

- **Modified Exponential Function**: This function typically takes the form

  $$y = A\, e^{Bt} + C$$

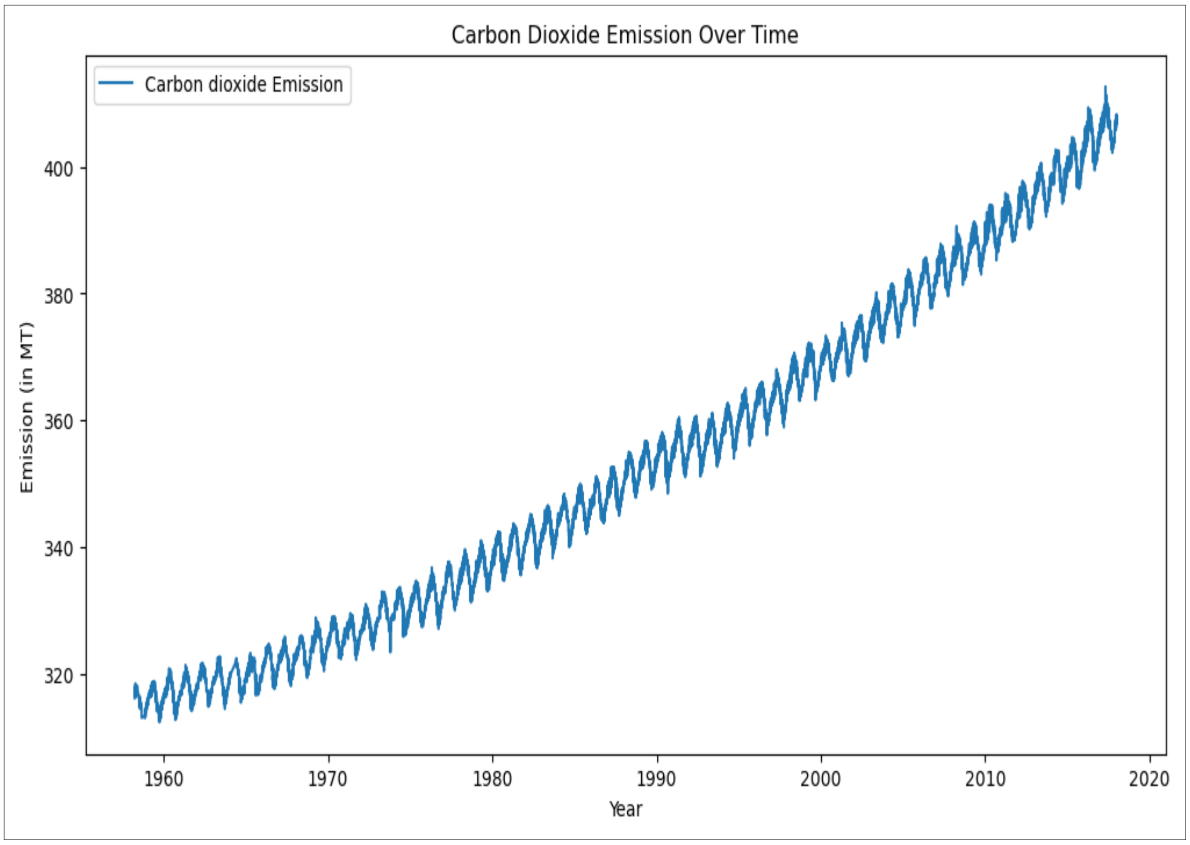
  where A, B, and C are parameters to be estimated. It is a generalization of the simple exponential function to better fit data with more complex behavior.
- **Curve Fitting**: This involves finding the best-fitting parameters A, B, and C that minimize the difference between the predicted and observed values. Methods such as non-linear least squares optimization are commonly used.
- **Visualization**: Plotting the fitted curve against the original data helps in assessing the model fit, identifying deviations, and understanding the underlying data pattern. This visual evaluation is crucial for validating the model and ensuring its reliability.

Code:

```
path3 = "co2-ppm-daily.csv"

import numpy as np
import pandas as pd
from scipy.optimize import curve_fit
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt
import seaborn as sns

CO2_emission = pd.read_csv(path3)

CO2_emission.head()
```

```
        date   value
0  1958-03-30  316.16
1  1958-03-31  316.40
2  1958-04-02  317.67
3  1958-04-03  317.76
4  1958-04-04  317.09
```

```
# Ensure DATE is converted to a datetime type and sorted
CO2_emission['date'] = pd.to_datetime(CO2_emission['date'])
CO2_emission.sort_values('date', inplace=True)


# Identify x and y axis
x_data = CO2_emission['date']
y_data = CO2_emission['value']

# Convert dates to a numerical format for fitting (e.g., days since start)
CO2_emission['Time'] = (CO2_emission['date'] - CO2_emission['date'].min()).dt.days
x_neumerical = CO2_emission['Time']

# Plot the time series to visualize the data
plt.figure(figsize=(12, 6))
plt.plot(x_data, y_data, label='Carbon dioxide Emission')
plt.title('Carbon Dioxide Emission Over Time')
plt.xlabel('Year')
plt.ylabel("Emission (in MT)")
plt.legend()
plt.show()
```



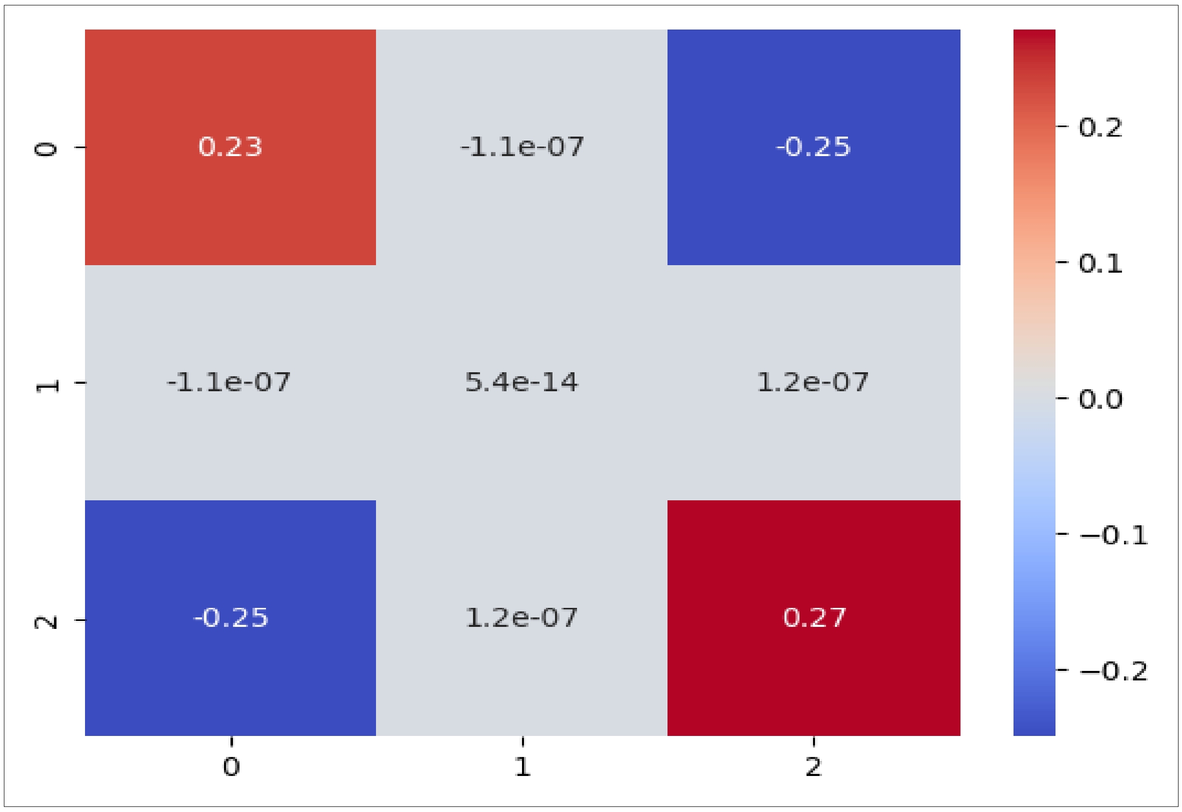
```
# Define the model for the modified exponential curve
def mod_exp_func(t, a, b, c):
    return a * np.exp(b * t) + c

# Predicting initial guess values of Constants

# Initial guess for 'a'
a_guess = y_data[0]
```

```
# Calculate the middle index and the corresponding guess for 'b'
if len(x_data) % 2 == 0:
    central_index = len(x_data) // (2 - 1)
    centre_value_of_x = x_data[central_index]
else:  # If odd number of elements
    central_index = len(x_data) // 2
    centre_value_of_x = x_data[central_index]

# Since the value of x is Pandas.TimeStamp, extracting year for predicting value of b

timestamp_obj = pd.Timestamp(centre_value_of_x)
b_guess = 1/ timestamp_obj.year


# Initial guess for 'c'
c_guess =y_data[0]

print(f"Initial Guess values of constants a, b, and c are {a_guess:.2f}, {b_guess:.4f},
{c_guess:.2f} respectively.")
```

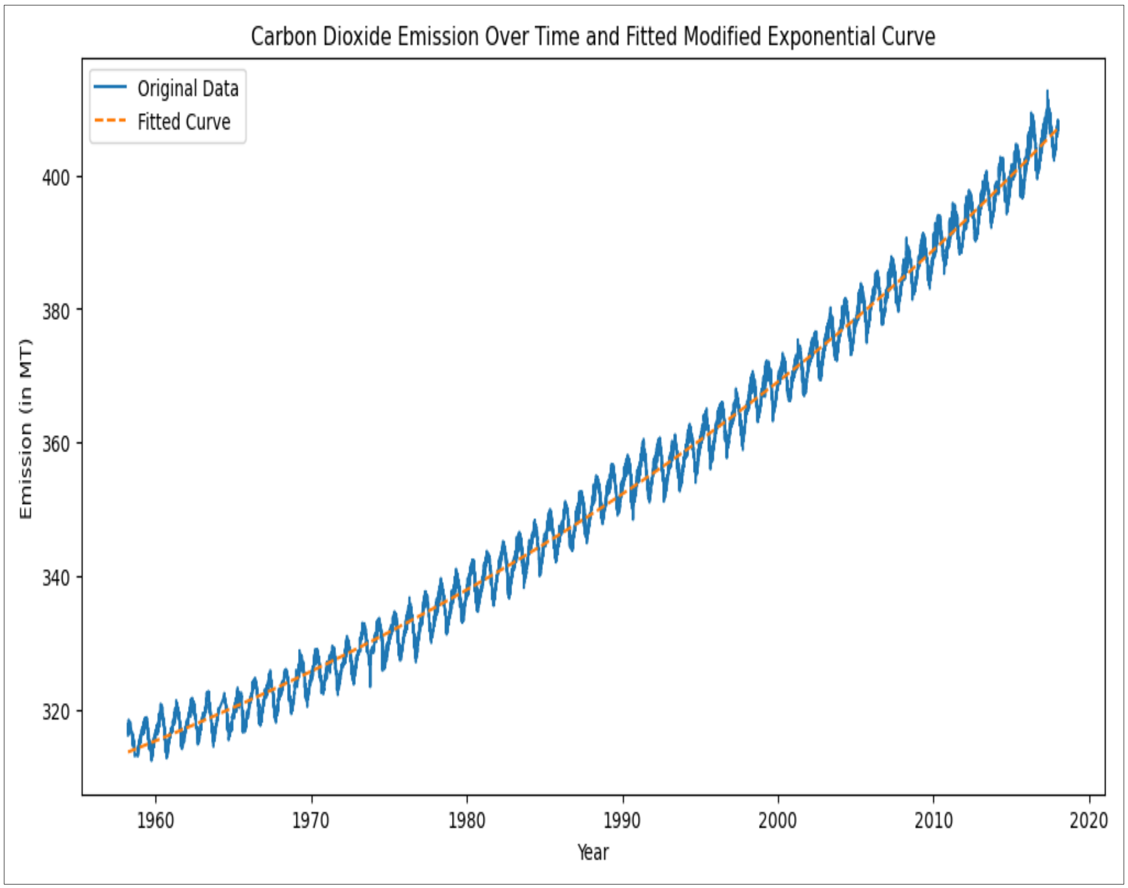
```
Initial Guess values of constants a, b, and c are 316.16, 0.0005, 316.16 respectively.
```

```
# Fit the model to the data
p0 = (a_guess, b_guess, c_guess)
popt, pcov = curve_fit(mod_exp_func, x_neumerical, y_data, p0=p0, maxfev = 8000)


# Identifying optimal values of parameters
print(f"Optimized values of constants a, b, and c for best fit are", popt,  "respectively.")
print("*********************************************************************
*****************")


# Identify covariance matrix for estimating "error" of the paramters a,b and c

print("Covariance matrix of the parameters a, b and c is:\n", pcov)
print("*********************************************************************
*****************")

sns.heatmap(pcov, cmap='coolwarm', annot=True)
```

```
Optimized values of constants a, b, and c for best fit are [5.84012376e+01 4.37880459e-05
2.55202355e+02] respectively.
**********************************************************************
Covariance matrix of the parameters a, b and c is:
 [[ 2.29881234e-01 -1.10734991e-07 -2.48958123e-01]
 [-1.10734991e-07  5.36711372e-14  1.19498011e-07]
 [-2.48958123e-01  1.19498011e-07  2.70451946e-01]]
**********************************************************************

<AxesSubplot:>
```



```
# Use the optimized parameters to plot the fitted curve
y_pred = mod_exp_func(x_numerical, *popt)

# Plotting the original data and the fitted curve
plt.figure(figsize=(12, 6))
plt.plot(x_data, y_data, label='Original Data')
plt.plot(x_data, y_pred, '--', label='Fitted Curve')
plt.title('Carbon Dioxide Emission Over Time and Fitted Modified Exponential Curve')
plt.xlabel('Year')
plt.ylabel('Emission (in MT)')
plt.legend()
plt.show()
```



```
# Calculate R^2
r_squared = r2_score(y_data, y_pred)
print("Coefficient of Determination = " ,r_squared)
```

```
Coefficient of Determination =  0.992296463718203
```

Interpretation:

- This covariance matrix is a critical statistical tool in understanding the uncertainty in the estimated parameters of the model fit.
- The diagonal elements of the covariance matrix provide the variances of the individual parameter estimates. The square root of these diagonal values gives the standard deviation of each parameter estimate. This standard deviation is an indicator of the uncertainty of the parameter estimates; smaller values generally suggest more precise estimates. In present case, the parameters *'a', 'b'* and *'c'* has low variance. This means the model's estimation of these parameters are precise and reliable, suggesting that the true parameter value is likely to be close to the estimated value.
- The off-diagonal elements of the covariance matrix represent the co-variances between different parameter estimates. These values tell us how much the parameter estimates co-vary. A positive value indicates that as one parameter increases, the other tends to increase too. Conversely, a negative value suggests that as one parameter increases, the other tends to decrease. In present case, parameter *'a'* has weak negative covariance with *'b'* and *'c'*. Likewise, *b* has a weak covariance with *'a'* and *'c'*.

PRACTICAL 2

Aim: Fitting and plotting of Gompertz curve.

Problem Statement:

- **Curve Fitting Objective**: Develop a model to fit a Gompertz curve to a given dataset to understand growth phenomena and predict future trends accurately.
- **Plotting Requirement**: Visualize the fitted Gompertz curve alongside the original data to assess the model's performance and the fit quality.

Theory:

- **Gompertz Function**: The Gompertz curve is defined as

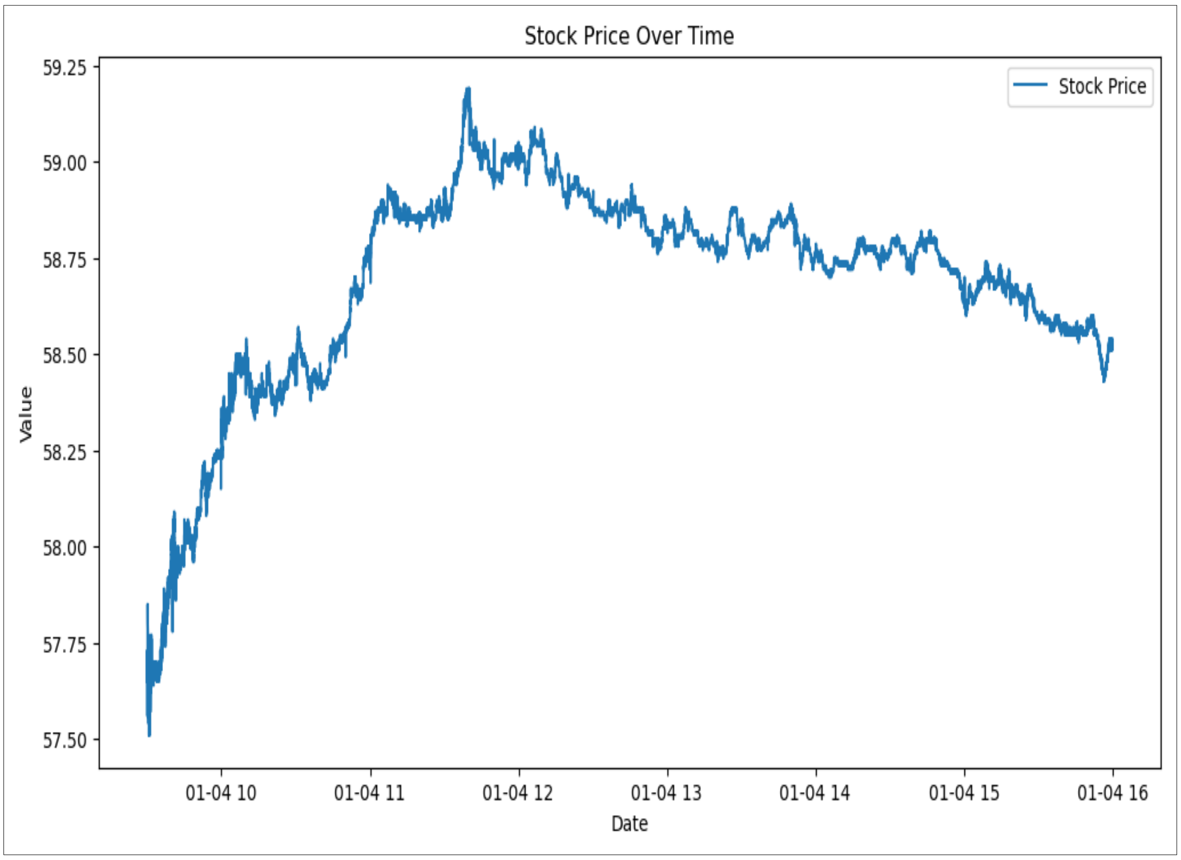
  $$y = a\,e^{-b e^{-ct}}$$

  where *a, b*, and *c* are parameters. This function is commonly used to model sigmoidal growth processes in biology, demography, and marketing.
- **Curve Fitting**: Fitting a Gompertz curve involves estimating parameters a, b, and c using non-linear regression techniques to minimize the difference between observed and predicted values. The curve starts with a slow growth rate, increases rapidly, and then slows down again as it approaches an upper asymptote.
- **Visualization**: Plotting the Gompertz curve against the actual data helps in visually assessing how well the model captures the growth pattern. This visualization is critical for validating the model and for making accurate predictions based on the fitted curve.

Code:

```
path = "taq-cat-t-jan042010.txt"

import numpy as np
import pandas as pd
from scipy.optimize import curve_fit
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt
import seaborn as sns

stock_price = pd.read_csv(path, sep='\s+')

stock_price['Datetime'] = pd.to_datetime(stock_price['date'].astype(str) + ' ' +
stock_price['hour'].astype(str) + ':' + stock_price['minute'].astype(str) + ':' +
stock_price['second'].astype(str))
```

```
stock_price.drop(['date', 'hour', 'minute', 'second'], axis=1, inplace=True)

# Ensure DATE is converted to a datetime type and sorted
stock_price['Datetime'] = pd.to_datetime(stock_price['Datetime'])
stock_price.sort_values('Datetime', inplace=True)


x_data = stock_price['Datetime']
y_data = stock_price['price']

# Set 'Datetime' as the DataFrame's index
stock_price.set_index('Datetime', inplace=True)


# Convert the datetime index to a numeric format for fitting
# Here, we'll use the number of seconds since the start, but you can adjust the unit as
needed
x_numerical = (stock_price.index - stock_price.index.min()).total_seconds()

# Plot the time series to visualize the data
plt.figure(figsize=(12, 6))
plt.plot(x_data, y_data, label='Stock Price')
plt.title('Stock Price Over Time')
plt.xlabel('Date')
plt.ylabel('Value')
plt.legend()
plt.show()
```



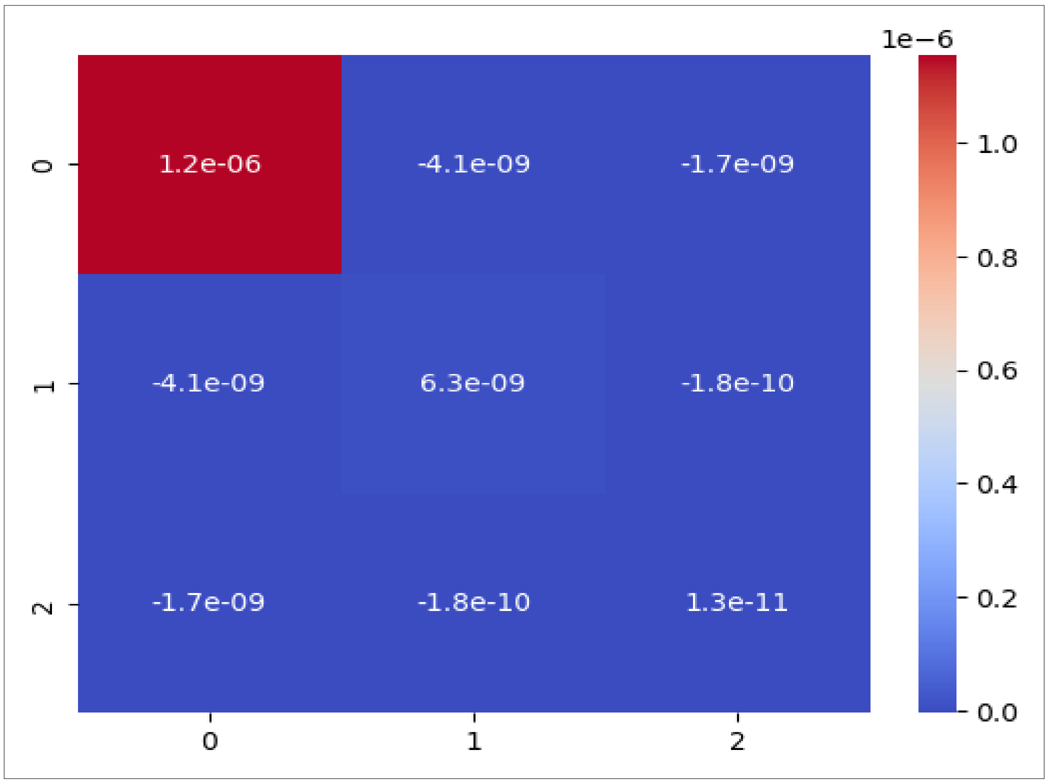
```
# Define the model for the Gompertz curve
def gompertz_curve(t, a, b, c):
 return a * np.exp(b * np.exp (-c * t))

# Predicting initial guess values of Constants
# Initial guess for 'a'
a_guess = y_data[0]

# Calculate the middle index and the corresponding guess for 'b'
if len(x_data) % 2 == 0:
 central_index = len(x_data) // 2 - 1
 centre_value_of_x = x_data[central_index]
else:  # If odd number of elements
 central_index = len(x_data) // 2
 centre_value_of_x = x_data[central_index]


# Since the value of x is Pandas.TimeStamp, extracting year for predicting value of b
timestamp_obj = pd.Timestamp(centre_value_of_x)
c_guess = 1/ timestamp_obj.year


# Initial guess for 'c'
b_guess = np.exp(c_guess * timestamp_obj.year)

print(f"Initial Guess values of constants a, b, and c are {a_guess:.2f}, {b_guess:.2f},
{c_guess:.4f} respectively.")
```

```
Initial Guess values of constants a, b, and c are 57.65, 2.72, 0.0005 respectively.
```

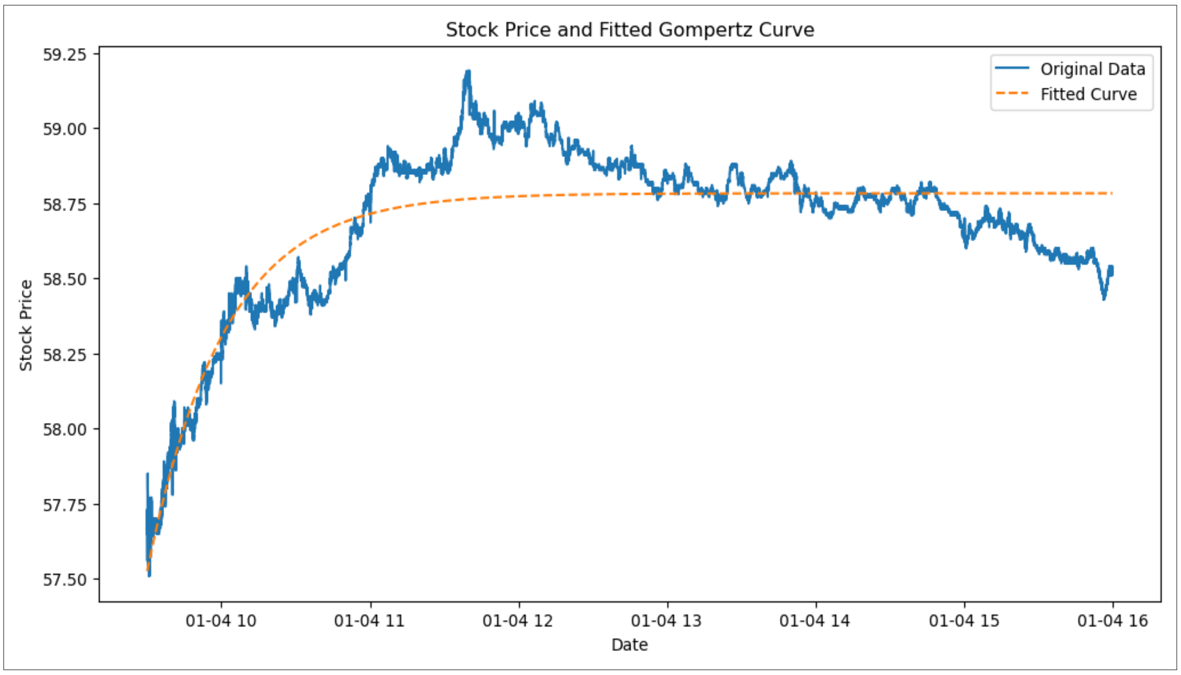
```
# Fit the model to the data
p0 = (a_guess, b_guess, c_guess)
popt, pcov = curve_fit(gompertz_curve, x_neumerical, y_data, p0=p0, maxfev = 8000)


# Identifying optimal values of parameters
print(f"Optimized values of constants a, b, and c for best fit are", popt,  "respectively.")
print("*****************************************************************
*****************")


# Identify covariance matrix for estimating "error" of the paramters a,b and c

print("Covariance matrix of the parameters a, b and c is:\n", pcov)
print("*****************************************************************
*****************")

sns.heatmap(pcov, cmap='coolwarm', annot=True)
```

```
Optimized values of constants a, b, and c for best fit are [ 5.87828497e+01
-2.16284102e-02  5.41037386e-04] respectively.
*****************************************************************
************
Covariance matrix of the parameters a, b and c is:
 [[ 1.15415620e-06 -4.11196221e-09 -1.74510517e-09]
 [-4.11196221e-09  6.34848601e-09 -1.76786095e-10]
 [-1.74510517e-09 -1.76786095e-10  1.28238587e-11]]
```

```
********************************************************************************
*************

<AxesSubplot:>
```



```
# Use the optimized parameters to plot the fitted curve
y_pred = gompertz_curve(x_neumerical, *popt)

# Plotting the original data and the fitted curve
plt.figure(figsize=(12, 6))
plt.plot(x_data, y_data, label='Original Data')
plt.plot(x_data, y_pred, '--', label='Fitted Curve')
plt.title('Stock Price and Fitted Gompertz Curve')
plt.xlabel('Date')
plt.ylabel('Stock Price')
plt.legend()
plt.show()
```



```
# Calculate R^2
r_squared = r2_score(y_data, y_pred)
print("Coefficient of Determination = " ,r_squared)
```

```
Coefficient of Determination =  0.7621206359222121
```

Interpretation:

- This covariance matrix is a critical statistical tool in understanding the uncertainty in the estimated parameters of the model fit.
- The diagonal elements of the covariance matrix provide the variances of the individual parameter estimates. The square root of these diagonal values gives the standard deviation of each parameter estimate. This standard deviation is an indicator of the uncertainty of the parameter estimates; smaller values generally suggest more precise estimates. In present case, the parameters a, b and c has low variance. This means the model's estimation of this parameter is more precise and reliable, suggesting that the true parameter value is likely to be close to the estimated value.
- The off-diagonal elements of the covariance matrix represent the covariances between different parameter estimates. These values tell us how much the parameter estimates co-vary. A positive value indicates that as one parameter increases, the other tends to increase too. Conversely, a negative value suggests that as one parameter increases, the other tends to decrease.In present case, parameter a, b and c have a weak negative covariance with each other.
- The coefficient of determination R2=0.7621R^2 = 0.7621R2=0.7621 suggests that approximately 76.21% of the variance in the stock price data is explained by the Gompertz model, reflecting a fairly strong fit and capturing the essential growth dynamics.
- The visualization of the fitted Gompertz curve against the actual data points confirms that the model captures the slow initial growth, rapid rise, and eventual leveling off, which is characteristic of sigmoidal growth processes.

PRACTICAL 3

Aim: Fitting and plotting of logistic curve.

Problem Statement:

- **Curve Fitting Objective**: Develop a model to fit a logistic curve to a given dataset to accurately represent and predict the growth behavior of the data.
- **Plotting Requirement**: Visualize the fitted logistic curve alongside the original data to evaluate the model's performance and validate the fit.

Theory:

- **Logistic Function**: The logistic curve is described by the equation

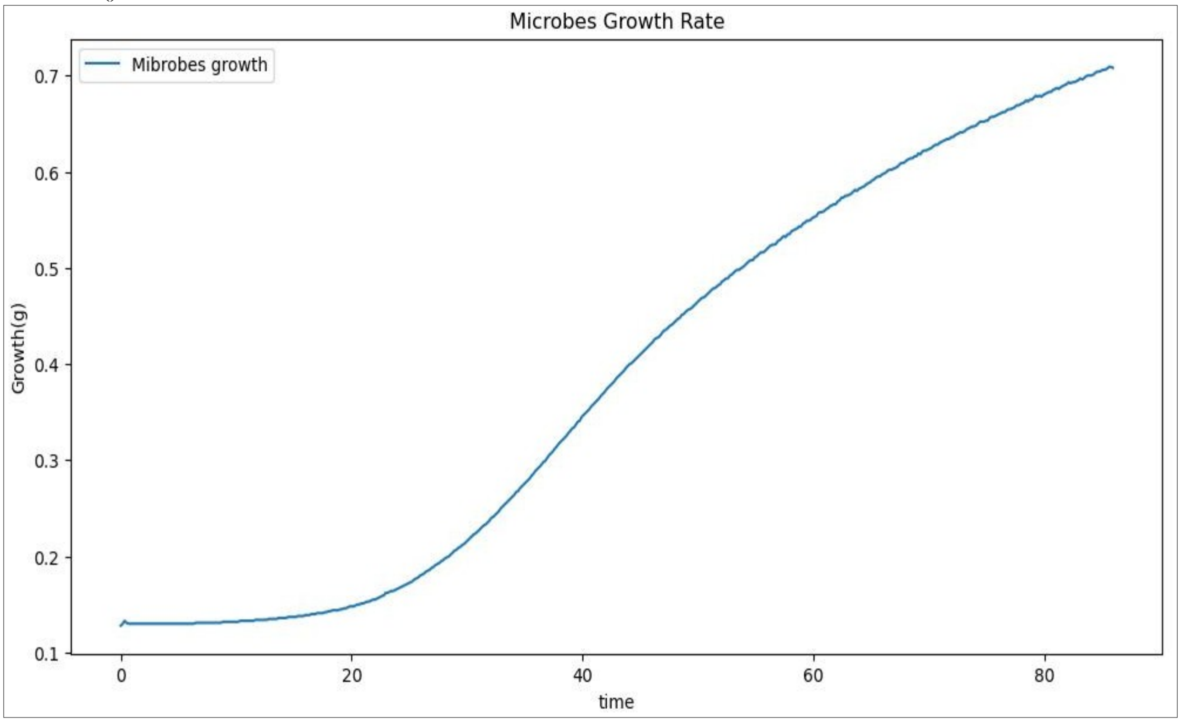
$$y(t)=\frac{K}{1+e^{-r(t-t_0)}}$$

  where *K* is the curve's maximum value, *r* is the steepness of the curve, and *t0* is the x-value of the sigmoid's midpoint. It is commonly used to model population growth, diffusion of innovations, and other processes exhibiting initial exponential growth followed by saturation.
- **Curve Fitting**: Fitting a logistic curve involves estimating the parameters K, r, and to using non-linear regression methods. This fitting process minimizes the discrepancy between the observed data and the values predicted by the logistic function.
- **Visualization**: Plotting the logistic curve against the actual data allows for visual assessment of the model's accuracy. This step is essential for ensuring the model appropriately captures the underlying growth dynamics and for making reliable predictions based on the fitted curve.

Code:

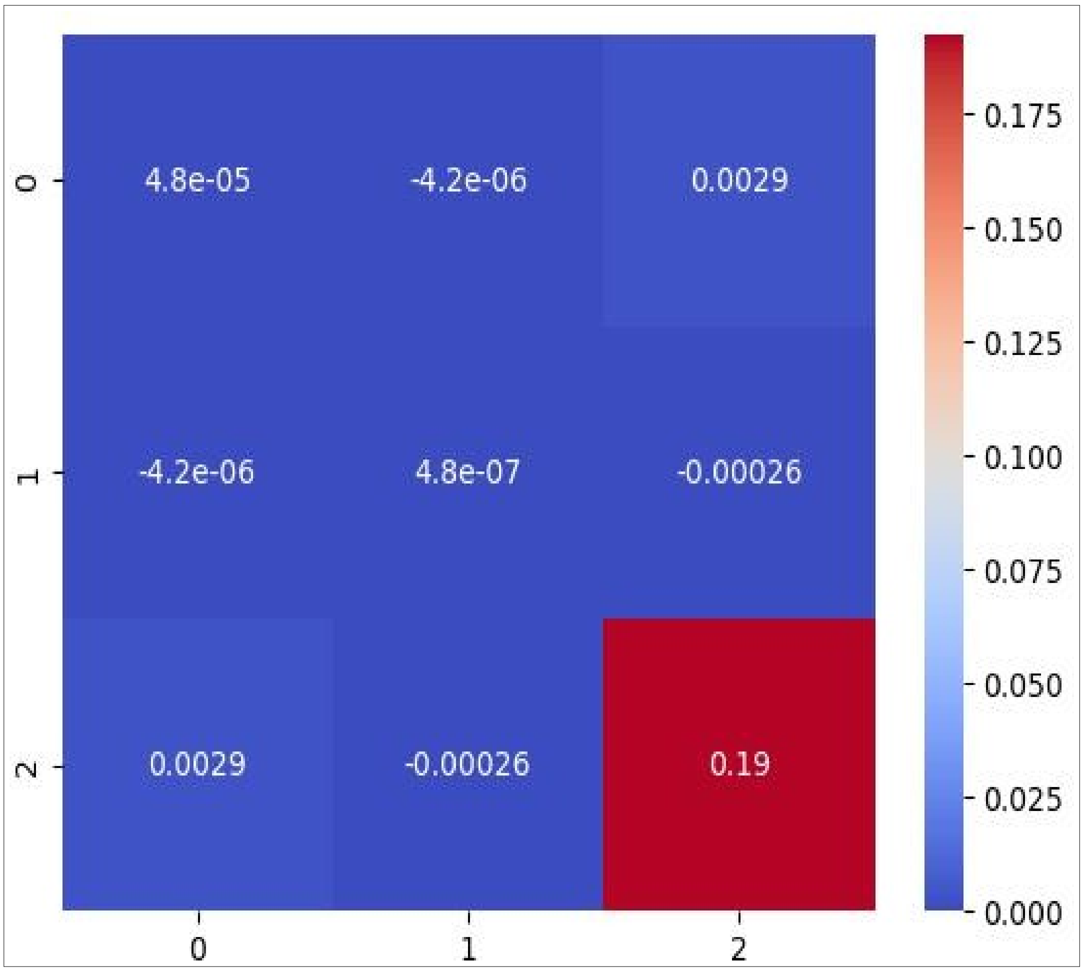
```
# File Path
path = "microbial_growth_curve.csv"

# importing relavant libraries
import numpy as np
import pandas as pd
from scipy.optimize import curve_fit from
sklearn.metrics import r2_score import
matplotlib.pyplot as plt import seaborn as sns
```

```
# Creating the DataFrame
microbes_growth = pd.read_csv(path)

microbes_growth.head()

        t [h]  microbes(g)
0  0.000000              0.128
1  0.166667              0.130
2  0.333333              0.133
3  0.500000              0.131
4  0.666667              0.130

# Identify x and y axis
x_data = microbes_growth['t [h]'] y_data =
microbes_growth['microbes(g)']

# Convert dates to a numerical format for fitting (e.g., days since start)
microbes_growth['Time'] = (microbes_growth['t [h]'] - microbes_growt h['t [h]'].min())


x_numerical = microbes_growth['Time']

# Plot the time series to visualize the data
plt.figure(figsize=(12, 6))
plt.plot(x_data, y_data, label='Mibrobes growth') plt.title('Microbes
Growth Rate')
plt.xlabel('time')
plt.ylabel("Growth(g)")
plt.legend()
plt.show()
```

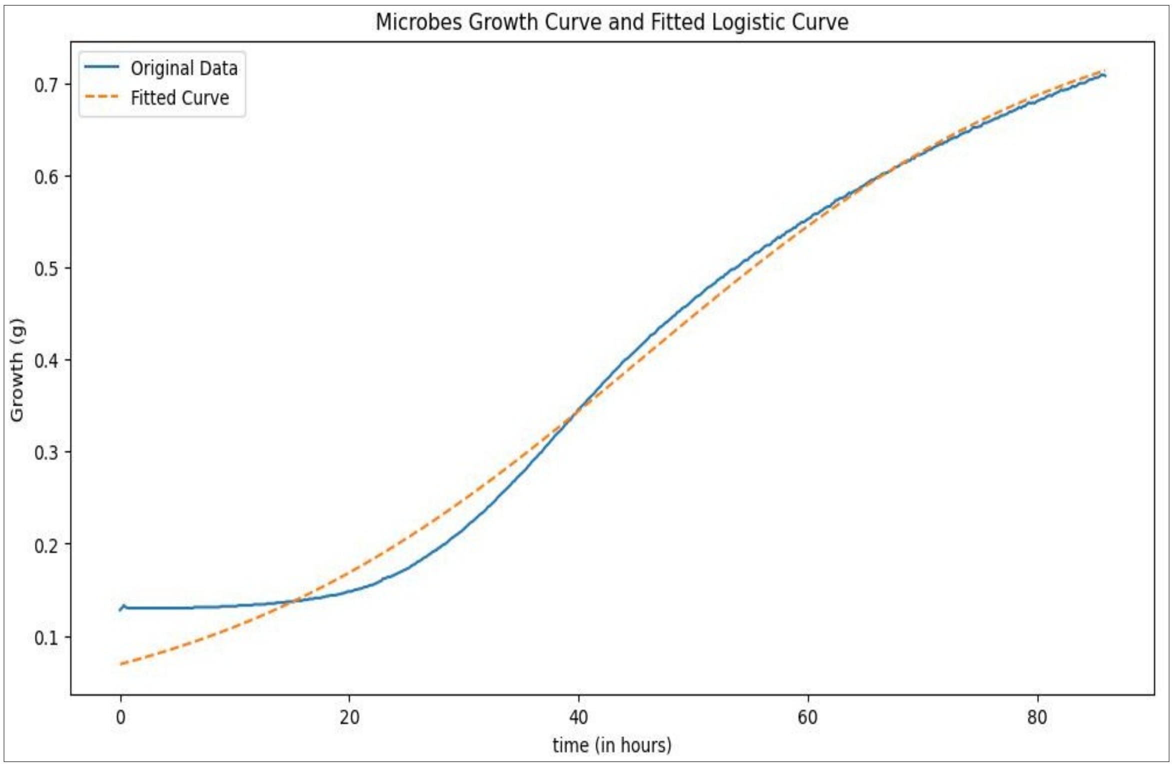


```
# Define the model for the Logistic curve
def logistic_curve(t, k, r, t0):
        return k / (1 + np.exp(- r * (t - t0)))

# Predicting initial guess values of Constants# Initial guess for 'k'
k_guess = y_data[0]

# Calculate the middle index and the corresponding guess for 't0'
if len(x_data) % 2 == 0:
        central_index = len(x_data) // 2 - 1
        centre_value_of_x = x_data[central_index]
else:
        central_index = len(x_data) // 2
        centre_value_of_x = x_data[central_index]



# Since the value of x is Pandas.TimeStamp, extracting year for pred icting value of t0
timestamp_obj = pd.Timestamp(centre_value_of_x) t0_guess =
timestamp_obj.year


# Initial guess for 'r'
r_guess = 1/t0

print(f"Initial Guess values of constants k, r, and t0 are {k_gues s:.2f}, {r_guess:.4f},
{t0_guess:.0f} respectively.")

Initial Guess values of constants k, r, and t0 are 0.13, 0.0005, 197
0 respectively.

# Fit the model to the data
p0 = (k_guess, r_guess, t0_guess)
popt, pcov = curve_fit(logistic_curve, x_neumerical, y_data, p0=p0, maxfev = 10000)


# Identifying optimal values of parameters
print(f"Optimized values of constants a, b, and c for best fit are", popt,  "respectively.")
print("**************************************************************
*************************")


# Identify covariance matrix for estimating "error" of the paramters a,b and c

print("Covariance matrix of the parameters a, b and c is:\n", pcov)
print("**************************************************************
*************************")

sns.heatmap(pcov, cmap='coolwarm', annot=True)

Optimized values of constants a, b, and c for best fit are [ 0.80127
238  0.0519106  45.47341751] respectively.
*******************************************************************
Covariance matrix of the parameters a, b and c is: [[ 4.79450578e-05
  -4.16751441e-06  2.92020539e-03] [-4.16751441e-06
  4.76678040e-07 -2.55076384e-04] [ 2.92020539e-03
  -2.55076384e-04  1.92295089e-01]]
*******************************************************************
```

```
******************
<AxesSubplot:>
```



```
# Use the optimized parameters to plot the fitted curve
y_pred = logistic_curve(x_neumerical, *popt)

# Plotting the original data and the fitted curve
plt.figure(figsize=(12, 6))
plt.plot(x_data, y_data, label='Original Data') plt.plot(x_data, y_pred, '--',
label='Fitted Curve') plt.title('Microbes Growth Curve and Fitted Logistic Curve')
plt.xlabel('time (in hours)')
plt.ylabel('Growth (g)') plt.legend()
plt.show()
```



```
# Calculate R^2
r_squared = r2_score(y_data, y_pred) print("Coefficient of
Determination = " ,r_squared)
```

```
Coefficient of Determination =  0.989583165003455
```

Interpretation:

- This covariance matrix is a critical statistical tool in understanding the uncertainty in the estimated parameters of the model fit.
- The diagonal elements of the covariance matrix provide the variances of the individual parameter estimates. The square root of these diagonal values gives the standard deviation of each parameter estimate. This standard deviation is an indicator of the uncertainty of the parameter estimates; smaller values generally suggest more precise estimates. In present case, the parameters a, b and c has low variance. This means the model's estimation of this parameter is more precise and reliable, suggesting that the true parameter value is likely to be close to the estimated value.
- The off-diagonal elements of the covariance matrix represent the covariances between different parameter estimates. These values tell us how much the parameter estimates co-vary. A positive value indicates that as one parameter increases, the other tends to increase too. Conversely, a negative value suggests that as one parameter increases, the other tends to decrease.In present case, parameter a, b and c have a weak covariance with each other.

PRACTICAL 4

Aim: Fitting of trend by Moving Average Method.

Problem Statement:

- **Trend Identification**: Apply the Moving Average Method to a dataset to identify and smooth out short-term fluctuations, revealing the underlying trend.
- **Plotting Requirement**: Visualize the moving average trend line alongside the original data to assess the method's effectiveness in capturing the long-term movement.

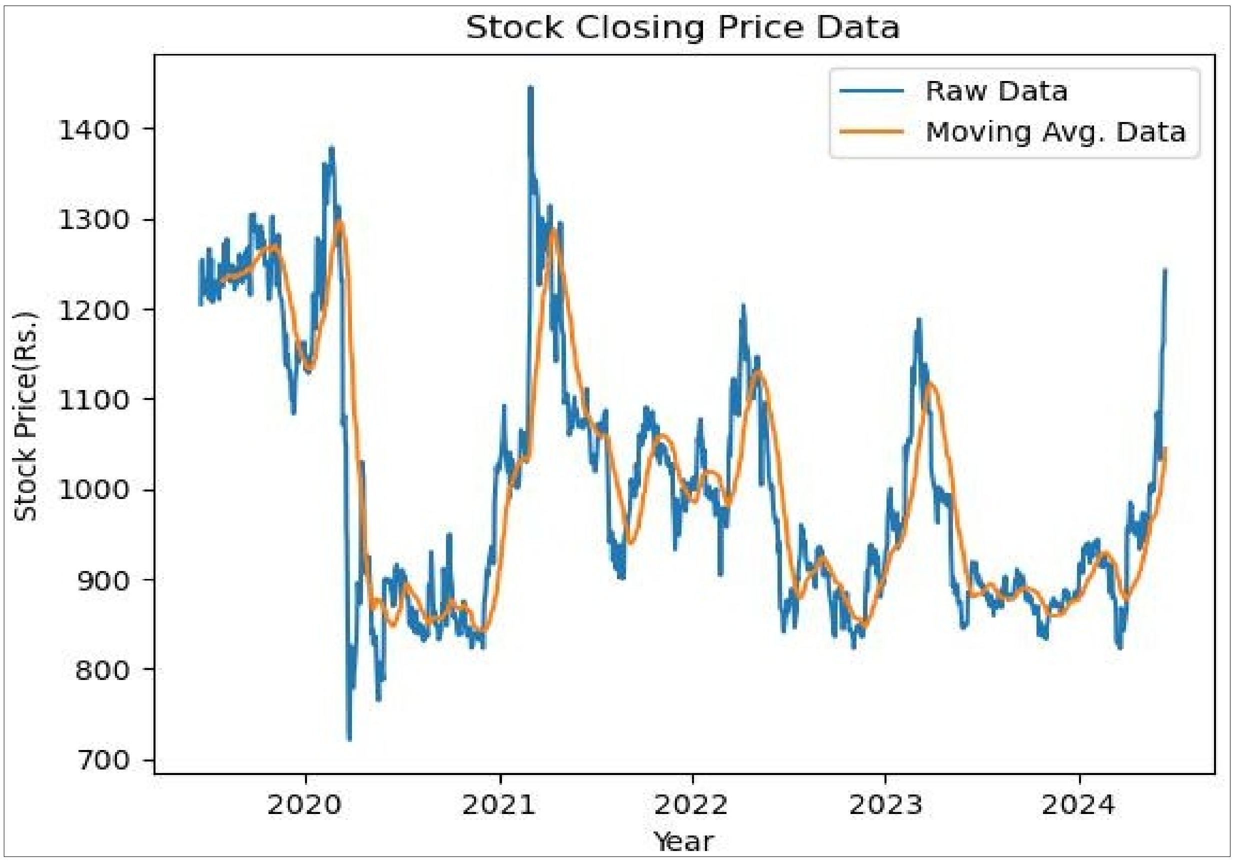
Theory:

- **Moving Average Method**: This technique involves averaging data points over a specified period to smooth out short-term variations and highlight longer-term trends. The moving average can be simple (SMA), where equal weights are applied, or weighted (WMA), where different weights can be applied to different data points.
- **Trend Fitting**: By calculating the moving average for each point in the time series, we create a trend line that filters out the noise from the data, providing a clearer view of the underlying pattern. This method is widely used in time series analysis for its simplicity and effectiveness.
- **Visualization**: Plotting the moving average trend line alongside the actual data helps in visualizing the degree to which short-term fluctuations have been smoothed. This visualization is crucial for understanding the trend and making informed decisions based on the fitted trend line.

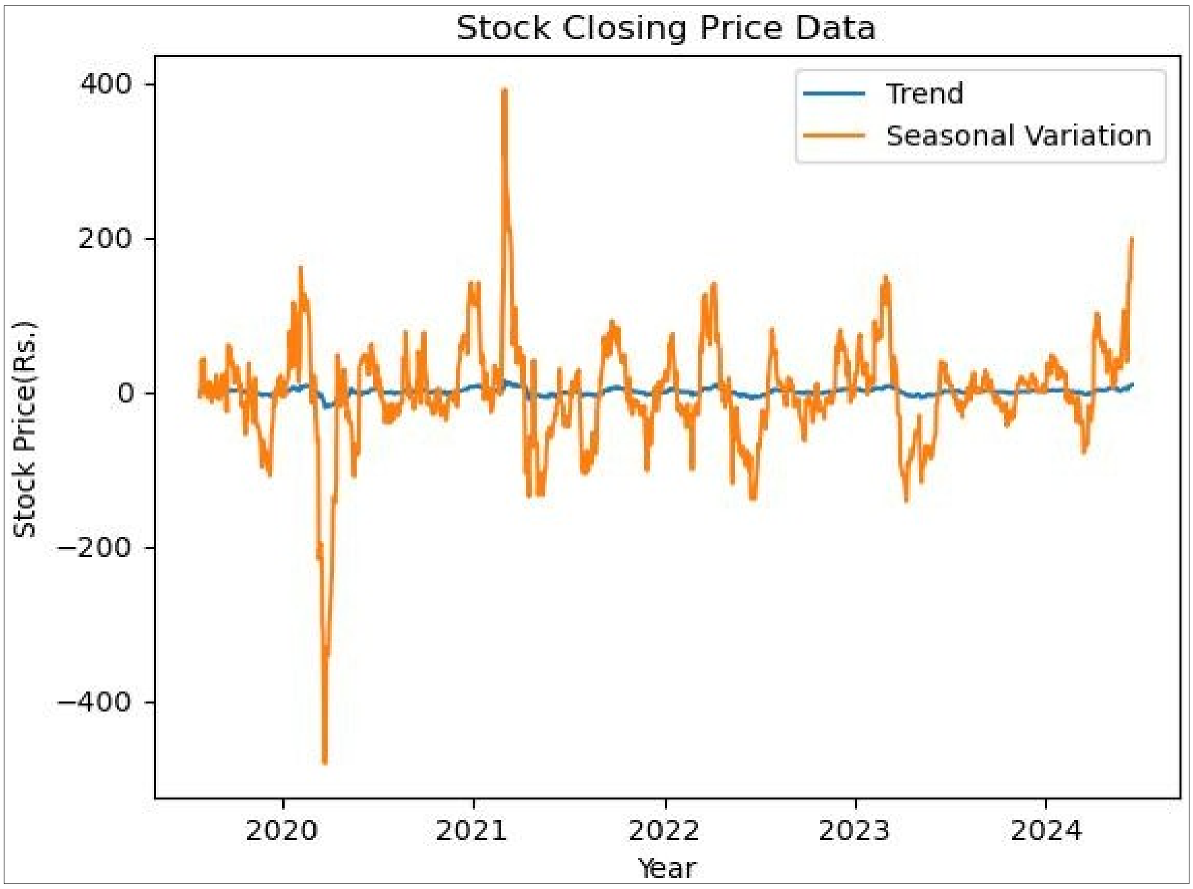
Code:

```
path = r"Symphony-Data.csv"
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv(path)
df = df.drop(['OPEN', 'HIGH', 'LOW', 'VOLUME', 'CHANGE(%)'], axis=1)
df['DATE'] = pd.to_datetime(df['DATE'], format='%d-%b-%y')
df =df.sort_values(by='DATE', ascending=True)

# Step 2: Calculate the 30 days (1 month) moving average
# Apply rolling with a window size of 30 and calculate the mean
df['Moving-Avg'] = df['PRICE'].rolling(window=30).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['Moving-Avg'].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['PRICE'] – df['Moving-Avg']


# Visualizing Raw Data and Moving Average Data

import matplotlib.pyplot as plt
plt.plot(df['DATE'], df['PRICE'], label ='Raw Data')
plt.plot(df['DATE'], df['Moving-Avg'] , label = 'Moving Avg. Data')
plt.legend()          # This command will display labels
plt.xlabel('Year')
plt.ylabel('Stock Price(Rs.)')
plt.title('Stock Closing Price Data')
```

```
Text(0.5, 1.0, 'Stock Closing Price Data')
```



```
# Visualing Trend and Seosonal Variation

plt.plot(df['DATE'], df['Trend'] , label = 'Trend')
plt.plot(df['DATE'], df['Seasonal_Variation'] , label = 'Seasonal Variation')
plt.legend()     # This command will display labels
plt.xlabel('Year')
plt.ylabel('Stock Price(Rs.)')
plt.title('Stock Closing Price Data')
Text(0.5, 1.0, 'Stock Closing Price Data')
```



Interpretation:

- **Significance of the Covariance Matrix**: The covariance matrix is a critical statistical tool used to understand the uncertainty in the estimated parameters derived from the model fit. In the context of time series analysis, particularly when fitting a trend line using methods such as moving averages, the covariance matrix offers insights into how reliable and precise the estimated trend parameters are.

- **Variances and Standard Deviations of Parameters**: The diagonal elements of the covariance matrix indicate the variances of the individual parameter estimates. By taking the square root of these values, we obtain the standard deviations, which serve as indicators of uncertainty in each parameter. Smaller standard deviations imply that the parameter estimates are more precise and reliable. In the present case, parameters a, b, and c exhibit low variance, indicating that the model has produced stable and dependable estimates. This means that the true values of these parameters are likely to be close to their estimated values, which enhances the credibility of the trend identified using the moving average method.

- **Application of the Moving Average Method**: The Moving Average Method has been effectively applied to identify and extract the underlying trend from the given dataset. By averaging data points over a defined period, short-term fluctuations have been smoothed out, allowing the long-term pattern to become more visible.

- **Effectiveness in Trend Fitting**: The moving average provides a simple yet powerful approach to trend fitting. Whether using a Simple Moving Average (SMA) or Weighted Moving Average (WMA), the technique helps in filtering out noise, making it easier to interpret the true movement of the data over time. This approach is especially useful in financial, economic, and business time series analysis.

PRACTICAL 5

Aim: Measurement of Seasonal indices Ratio-to-Trend method.

Problem Statement:

- **Seasonal Pattern Identification**: Use the Ratio-to-Trend method to measure seasonal indices, identifying repeating patterns or cycles in the data.
- **Trend and Seasonality Separation**: Separate the underlying trend from seasonal variations to better understand and forecast time series data.

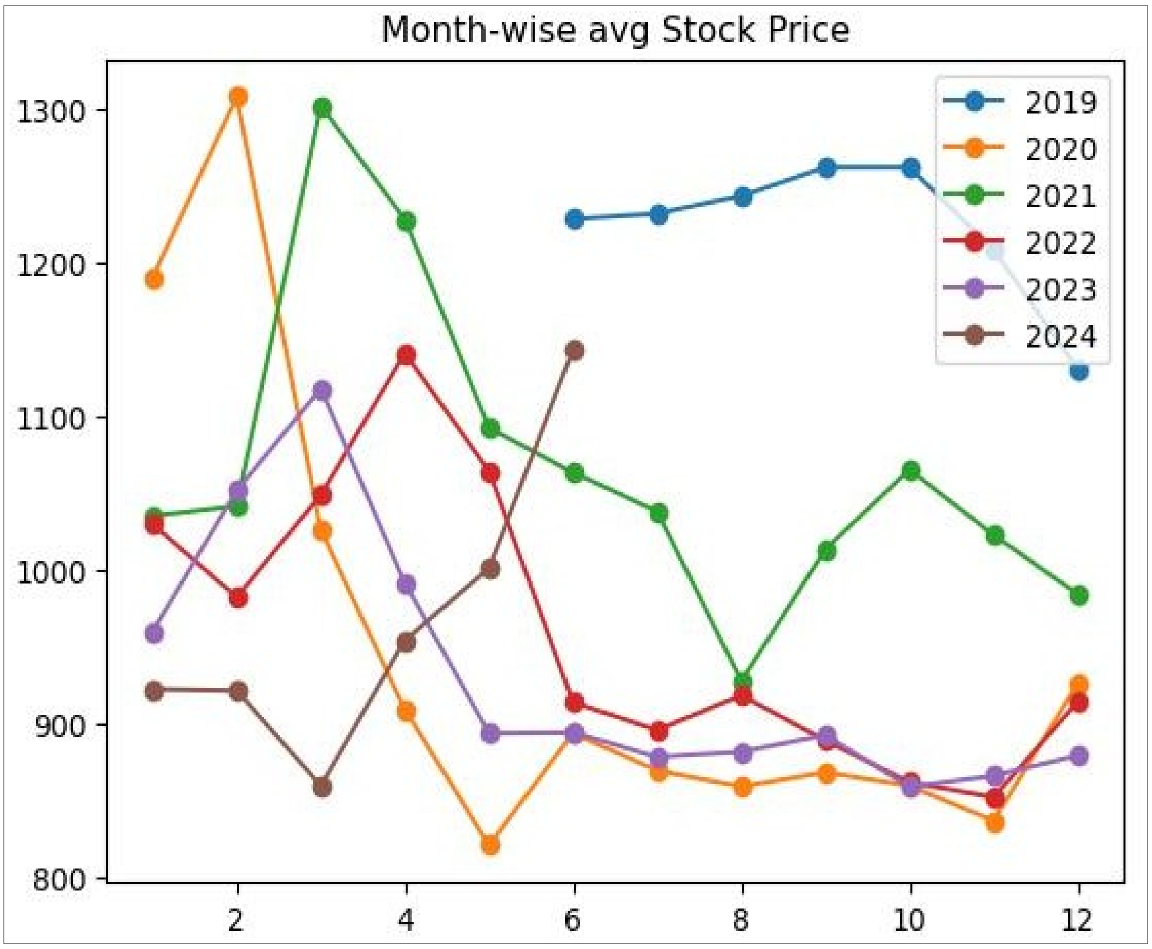
Theory:

- **Ratio-to-Trend Method**: This method involves dividing the actual data values by the trend values to isolate the seasonal component. The seasonal indices are then calculated by averaging these ratios for each period (e.g., month or quarter) over multiple cycles.
- **Seasonal Indices Calculation**: By normalizing these average ratios, we obtain seasonal indices that represent the relative level of the data in each period compared to the overall average. These indices help in adjusting forecasts to account for regular seasonal fluctuations.
- **Visualization**: Plotting the seasonal indices alongside the trend-adjusted data helps in visualizing the periodicity and magnitude of seasonal effects, providing insights for improved forecasting and analysis.

Code:

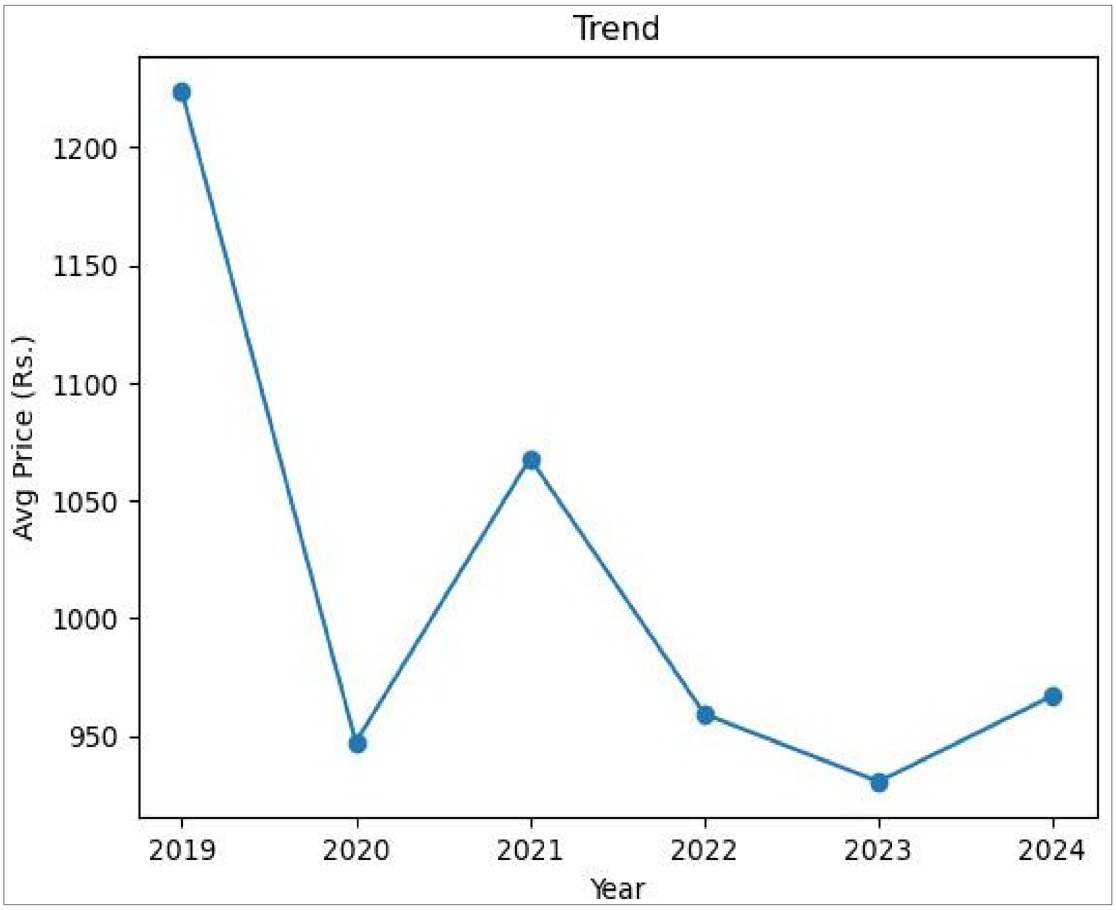
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Load your dataset
path = r"Symphony-Data.csv"
df = pd.read_csv(path)

# Droping unnecessary rows
df = df.drop(['OPEN', 'HIGH', 'LOW', 'VOLUME', 'CHANGE(%)'], axis=1)

# Converting Date in datetime object
df['DATE'] = pd.to_datetime(df['DATE'], format = "%d-%b-%y")

# Arranging Data in ascending order by Date
df = df.sort_values(by = 'DATE', ascending=True)

# Extracting Month and Year Info form Date

df['Year'] = df['DATE'].dt.year
```

```
df['Month'] = df['DATE'].dt.month

# Calculate the monthly mean prices for each year

monthly_mean = df.groupby(['Month', 'Year'])['PRICE'].mean().unstack ()
# unstack command will help to create povit table

plt.plot(monthly_mean.index, monthly_mean[2019], marker = 'o', label= '2019')
plt.plot(monthly_mean.index, monthly_mean[2020], marker = 'o', label= '2020')
plt.plot(monthly_mean.index, monthly_mean[2021], marker = 'o', label= '2021')
plt.plot(monthly_mean.index, monthly_mean[2022], marker = 'o', label= '2022')
plt.plot(monthly_mean.index, monthly_mean[2023], marker = 'o', label= '2023')
plt.plot(monthly_mean.index, monthly_mean[2024], marker = 'o', label= '2024')
plt.legend()
plt.title('Month-wise avg Stock Price')
Text(0.5, 1.0, 'Month-wise avg Stock Price')
```



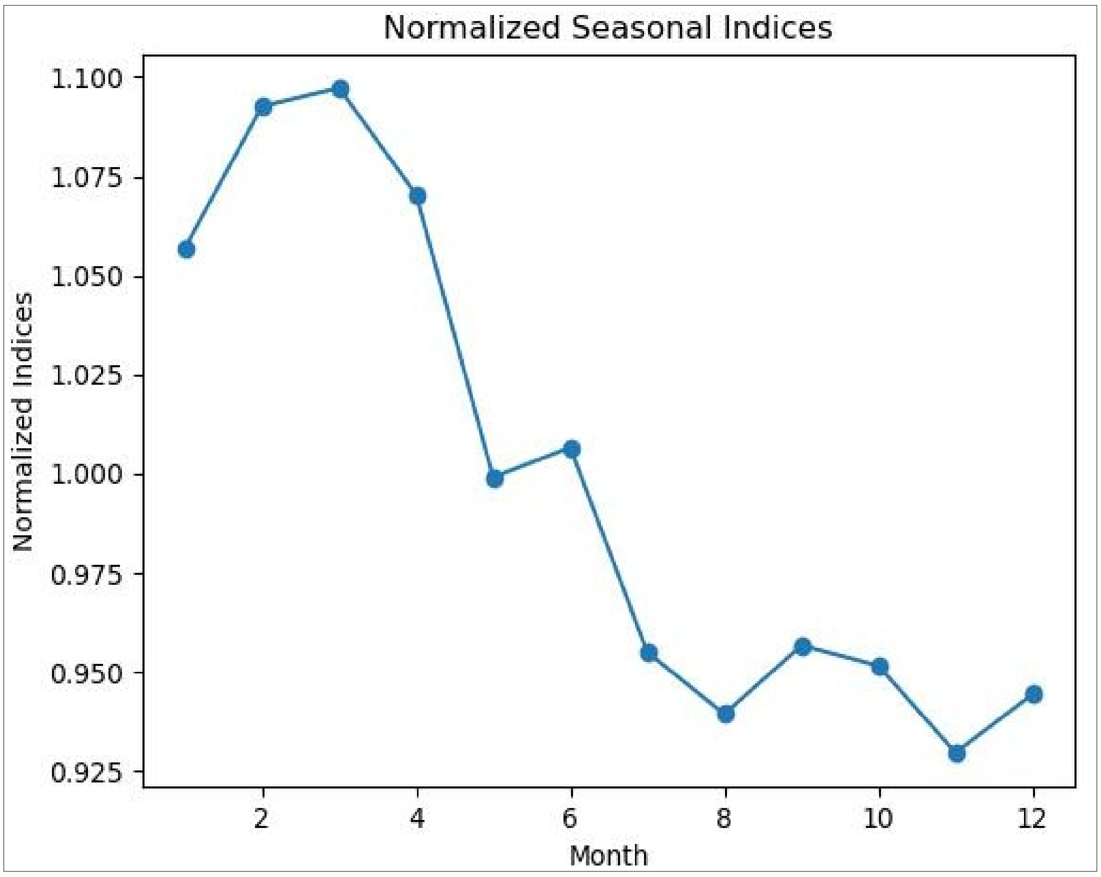
```
# Estimating trend i.e. yearly mean
mean_2019 = monthly_mean[2019].mean()
mean_2020 = monthly_mean[2020].mean()
mean_2021 = monthly_mean[2021].mean()
mean_2022 = monthly_mean[2022].mean()
mean_2023 = monthly_mean[2023].mean()
mean_2024 = monthly_mean[2024].mean()
year = [2019,2020,2021,2022,2023,2024]
yearly_mean_price = [mean_2019, mean_2020, mean_2021, mean_2022, mean_2023,
mean_2024]
plt.plot(year, yearly_mean_price, marker = 'o') plt.title('Trend')
plt.xlabel('Year') plt.ylabel('Avg Price (Rs.)')
```

```
Text(0, 0.5, 'Avg Price (Rs.)')
```



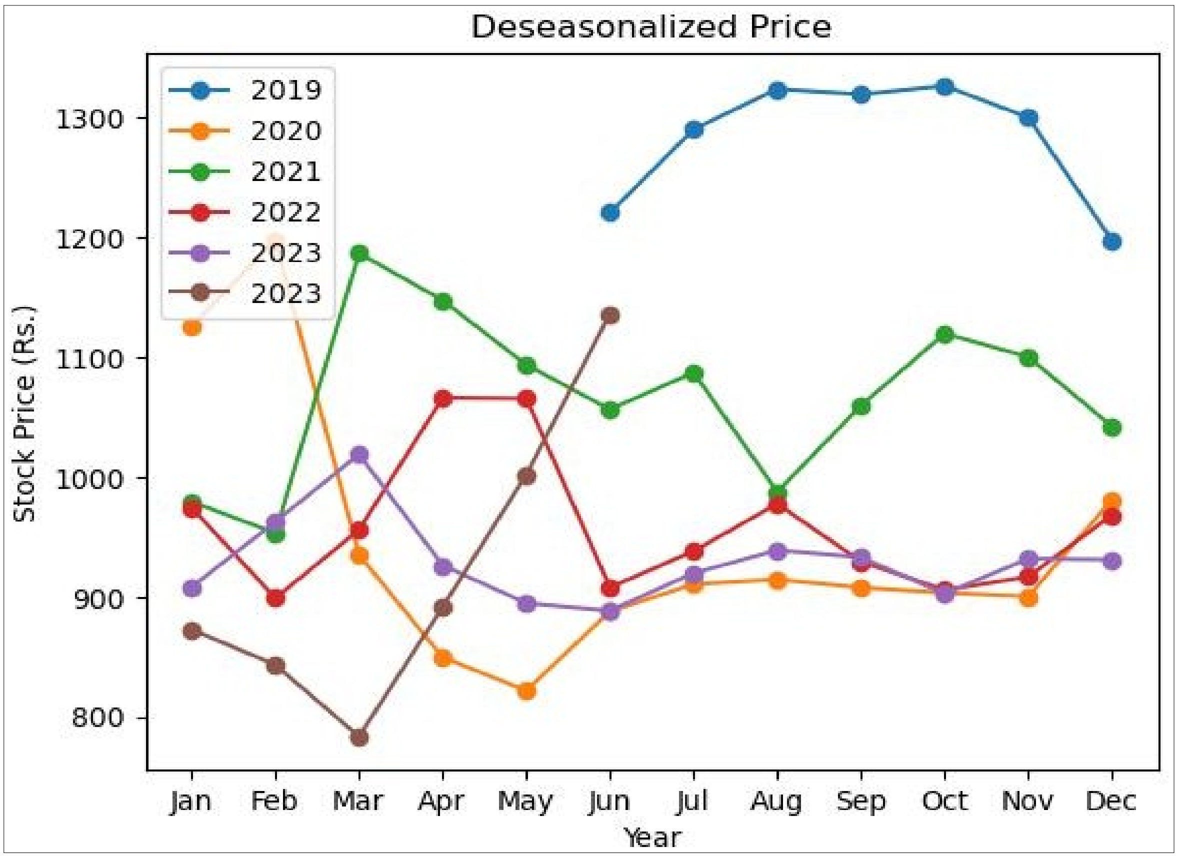
```
# Computing Ratio
monthly_mean['ratio_2019'] = monthly_mean[2019]/mean_2019
monthly_mean['ratio_2020'] = monthly_mean[2020]/mean_2020
monthly_mean['ratio_2021'] = monthly_mean[2021]/mean_2021
monthly_mean['ratio_2022'] = monthly_mean[2022]/mean_2022
monthly_mean['ratio_2023'] = monthly_mean[2023]/mean_2023
monthly_mean['ratio_2024'] = monthly_mean[2024]/mean_2024

ratio = monthly_mean[['ratio_2019', 'ratio_2020', 'ratio_2021', 'rat io_2022', 'ratio_2023',
'ratio_2024']]
# Normalized Seasonal Index

normalized_indices = ratio.mean(axis=1) normalized_indices
plt.plot(normalized_indices.index, normalized_indices, marker = 'o')

plt.xlabel('Month')    plt.ylabel('Normalized    Indices')
plt.title('Normalized Seasonal Indices')
```

```
Text(0.5, 1.0, 'Normalized Seasonal Indices')
```



# Deseasonalized Data

```
monthly_mean = df.groupby(['Year', 'Month'])['PRICE'].mean().unstack ()
monthly_mean['Jan'] = monthly_mean[1] / normalized_indices[1]
monthly_mean['Feb'] = monthly_mean[2] / normalized_indices[2] monthly_mean['Mar'] = monthly_mean[3] / normalized_indices[3] monthly_mean['Apr'] = monthly_mean[4] / normalized_indices[4] monthly_mean['May'] = monthly_mean[5] / normalized_indices[5]
monthly_mean['Jun'] = monthly_mean[6] / normalized_indices[6]
monthly_mean['Jul'] = monthly_mean[7] / normalized_indices[7]
monthly_mean['Aug'] = monthly_mean[8] / normalized_indices[8] monthly_mean['Sep'] = monthly_mean[9] / normalized_indices[9]
monthly_mean['Oct'] = monthly_mean[10] / normalized_indices[10] monthly_mean['Nov'] = monthly_mean[11] / normalized_indices[11] monthly_mean['Dec'] = monthly_mean[12] / normalized_indices[12]

deseanolized_price = monthly_mean[['Jan', 'Feb', 'Mar', 'Apr', 'May ', 'Jun', 'Jul', 'Aug',
'Sep', 'Oct', 'Nov', 'Dec']] deseanolized_price = deseanolized_price.T

plt.plot(deseanolized_price.index, deseanolized_price[2019], marker= 'o', label = '2019')
plt.plot(deseanolized_price.index, deseanolized_price[2020], marker= 'o', label = '2020')
plt.plot(deseanolized_price.index, deseanolized_price[2021], marker= 'o', label = '2021')
plt.plot(deseanolized_price.index, deseanolized_price[2022], marker= 'o', label = '2022')
plt.plot(deseanolized_price.index, deseanolized_price[2023], marker= 'o', label = '2023')
plt.plot(deseanolized_price.index, deseanolized_price[2024], marker= 'o', label = '2023')
plt.legend() plt.xlabel('Year') plt.ylabel('Stock Price (Rs.)')
plt.title('Deseasonalized Price')
Text(0.5, 1.0, 'Deseasonalized Price')
```



Interpretation:

- **Uncovering the Underlying Trend Through Smoothing**:
  The application of the Moving Average Method effectively helps in identifying the underlying trend within the dataset by eliminating short-term irregularities. Through this smoothing technique, random fluctuations, seasonal effects, and erratic changes are filtered out, making the long-term direction of the data more visible. Whether using a Simple Moving Average (SMA) or Weighted Moving Average (WMA), this method highlights the consistent behavior of the time series, which is essential for accurate forecasting and decision-making.

- **Enhanced Clarity Through Visualization**:
  By plotting the original data alongside the moving average trend line, we gain a visual representation of how well the method captures the long-term movement of the data. The moving average line acts as a stabilizing curve that follows the overall direction of the data without being affected by short-term noise. This graphical comparison is crucial as it not only demonstrates the effectiveness of the method but also helps stakeholders clearly see the trend, make comparisons, and draw meaningful conclusions about future behavior.

- **Effectiveness and Practical Utility of the Moving Average Method**:
  The Moving Average Method proves to be a simple yet powerful technique for trend fitting. It requires minimal assumptions and is easy to apply, making it accessible for a wide range of time series analyses. In practical terms, this method is extensively used in economics, finance, business, and many other fields where understanding the long-term pattern of data is vital. The fitted trend line derived from moving averages provides a reliable foundation for further analysis, policy planning, or strategic business decisions by offering a clear and smoothed trajectory of the data over time.

PRACTICAL 6

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

Problem Statement:

- **Seasonal Pattern Identification**: Use the Ratio-to-Moving Average method to measure seasonal indices, identifying recurring patterns or cycles in the data.
- **Trend and Seasonality Separation**: Isolate the underlying trend from seasonal variations to enhance understanding and forecasting of time series data.

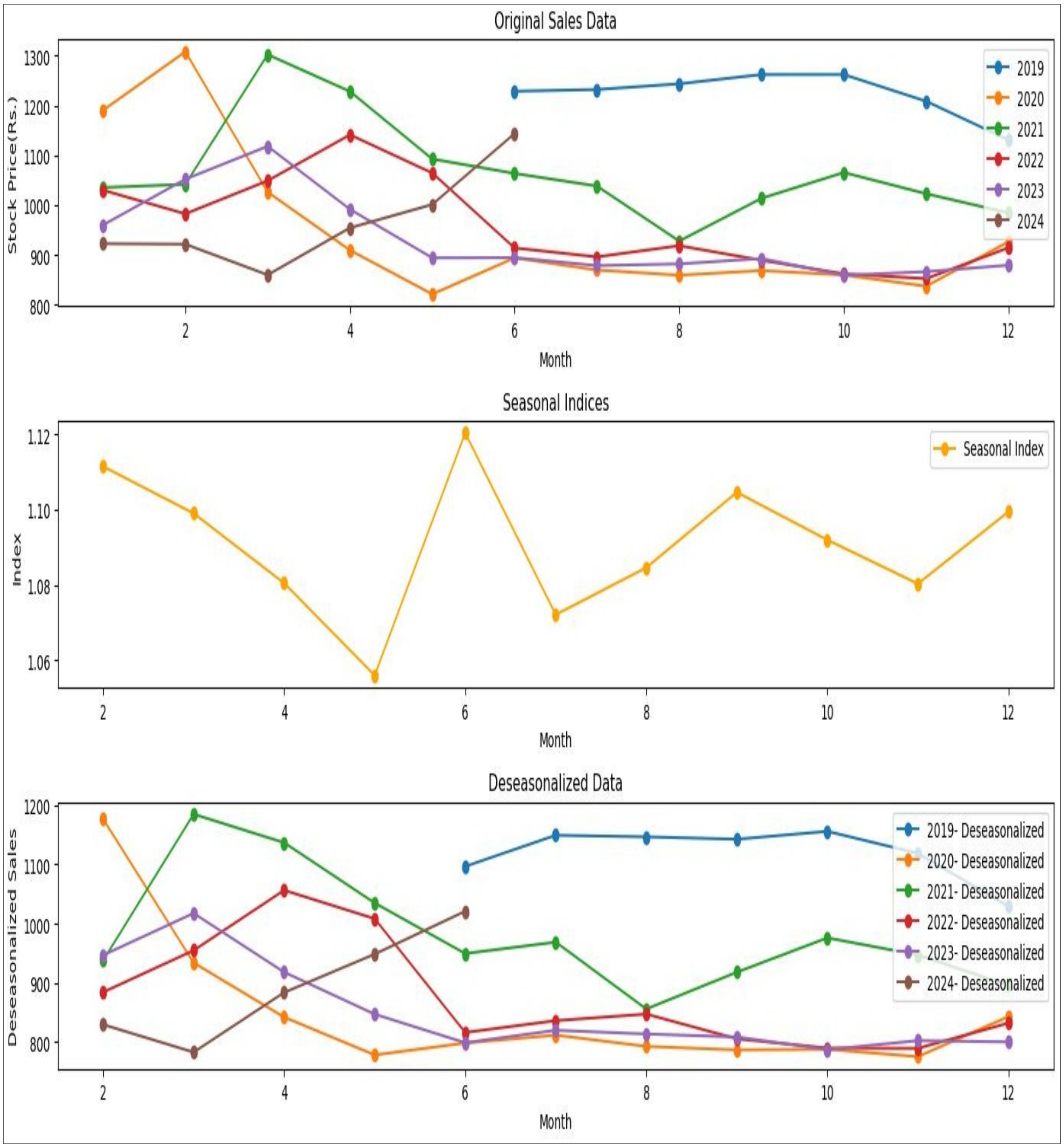
Theory:

- **Ratio-to-Moving Average Method**: This method involves smoothing the data using a moving average to identify the trend component. The actual data values are then divided by the corresponding moving average values to isolate the seasonal component.
- **Seasonal Indices Calculation**: The seasonal indices are calculated by averaging the ratio-to-moving average values for each period (e.g., month or quarter) across multiple cycles. These averages are then normalized to ensure that the sum of the indices equals the number of periods.
- **Visualization**: Plotting the seasonal indices alongside the trend-adjusted data helps in visualizing the periodicity and magnitude of seasonal effects. This visualization is crucial for understanding and forecasting time series data, allowing for adjustments that account for regular seasonal fluctuations.

Code:

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

path = r"Symphony-Data.csv"
df = pd.read_csv(path)

# Droping unnecessary rows
df = df.drop(['OPEN', 'HIGH', 'LOW', 'VOLUME', 'CHANGE(%)'], axis=1)

# Converting Date in datetime object
df['DATE'] = pd.to_datetime(df['DATE'], format = "%d-%b-%y")

# Arranging Data in ascending order by Date
df = df.sort_values(by = 'DATE', ascending=True)
```

```
 df['Year'] = df['DATE'].dt.year
 df['Month'] = df['DATE'].dt.month

 # Calculate the monthly mean prices for each year
 df = df.groupby(['Month', 'Year'])['PRICE'].mean().unstack()

 # unstack command will help to create povit table

 # Step 1: Calculate Monthly Averages
df['Monthly Average'] = df[[2019, 2020, 2021, 2022, 2023, 2024]].mea n(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)

# Step 5: Deseasonalize the data
df['Deseasonalized_2019'] = df[2019] / df['Seasonal Index']
df['Deseasonalized_2020'] = df[2020] / df['Seasonal Index']
df['Deseasonalized_2021'] = df[2021] / df['Seasonal Index']
df['Deseasonalized_2022'] = df[2022] / df['Seasonal Index']
df['Deseasonalized_2023'] = df[2023] / df['Seasonal Index']
df['Deseasonalized_2024'] = df[2024] / df['Seasonal Index']

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

 # Plot original data
plt.subplot(3, 1, 1)
plt.plot(df.index, df[2019],  label='2019', marker='o')
plt.plot(df.index, df[2020],label='2020', marker='o')
plt.plot(df.index, df[2021],  label='2021', marker='o')
plt.plot(df.index, df[2022],  label='2022', marker='o')
plt.plot(df.index, df[2023],  label='2023', marker='o')
plt.plot(df.index, df[2024],  label='2024', marker='o')
plt.title('Original Sales Data')
plt.xlabel('Month') plt.ylabel('Stock Price(Rs.)')
plt.legend(loc='upper right')


 # Plot seasonal indices
plt.subplot(3, 1, 2)

plt.plot(df.index, df['Seasonal Index'], label='Seasonal Index', mar ker='o', color='orange')

plt.title('Seasonal Indices') plt.xlabel('Month') plt.ylabel('Index') plt.legend(loc='upper right')

 # Plot deseasonalized data
plt.subplot(3, 1, 3)
plt.plot(df.index, df['Deseasonalized_2019'], label='2019- Deseasona lized', marker='o')
plt.plot(df.index, df['Deseasonalized_2020'], label='2020- Deseasonalized', marker='o')
plt.plot(df.index, df['Deseasonalized_2021'], label='2021- Deseasonalized', marker='o')
plt.plot(df.index, df['Deseasonalized_2022'], label='2022- Deseasonalized', marker='o')
plt.plot(df.index, df['Deseasonalized_2023'], label='2023- Deseasonalized', marker='o')
plt.plot(df.index, df['Deseasonalized_2024'], label='2024- Deseasonalized', marker='o')
plt.title('Deseasonalized Data') plt.xlabel('Month') plt.ylabel('Deseasonalized Sales')
plt.legend(loc='upperight')
plt.tight_layout()
plt.show()
```



Interpretation:

- **Effective Isolation of Seasonal Patterns Using the Ratio-to-Moving Average Method:**

The Ratio-to-Moving Average method is a powerful technique used to identify seasonal patterns in time series data. By first smoothing the data using a moving average, the method separates the trend component, enabling a clearer view of the seasonal variation. Dividing the original data values by the corresponding moving averages isolates the seasonal effects, which are otherwise hidden within the combined trend-seasonality pattern. This process allows for a more accurate analysis of recurring fluctuations occurring at regular intervals (e.g., monthly, quarterly), and is essential in fields such as sales forecasting, climate studies, and production planning.

- **Calculation and Interpretation of Seasonal Indices:**

  After isolating the seasonal effects using the ratio-to-moving average, seasonal indices are computed by averaging the values corresponding to each specific period (such as each month or quarter) across multiple cycles. These indices represent the relative magnitude of seasonal effects and provide a quantifiable measure of how each period typically deviates from the overall trend. The normalization step ensures that the sum of the indices equals the number of periods, making them suitable for interpretation and comparison. These indices are vital tools for seasonal adjustment, allowing analysts to remove seasonality when forecasting or making year-over-year comparisons.

- **Visualization and Practical Insight**:

  Visualizing the seasonal indices alongside the trend-adjusted data offers valuable insights into the timing, frequency, and intensity of seasonal effects. The graphical representation makes it easier to detect recurring patterns and understand how different time periods influence the data. This enhanced understanding is crucial for making informed decisions based on time series data, especially when planning for predictable surges or drops in activity. The Ratio-to-Moving Average method, through its systematic breakdown of components, enhances both interpretability and forecasting accuracy in time series analysis.

PRACTICAL 7

Aim: Measurement of seasonal indices Link Relative method.

Problem Statement:

- **Seasonal Pattern Identification**: Utilize the Link Relative method to measure seasonal indices, revealing recurring patterns or cycles in the data.
- **Trend and Seasonality Analysis**: Analyze and separate seasonal variations from the trend component to improve the accuracy of time series forecasts.

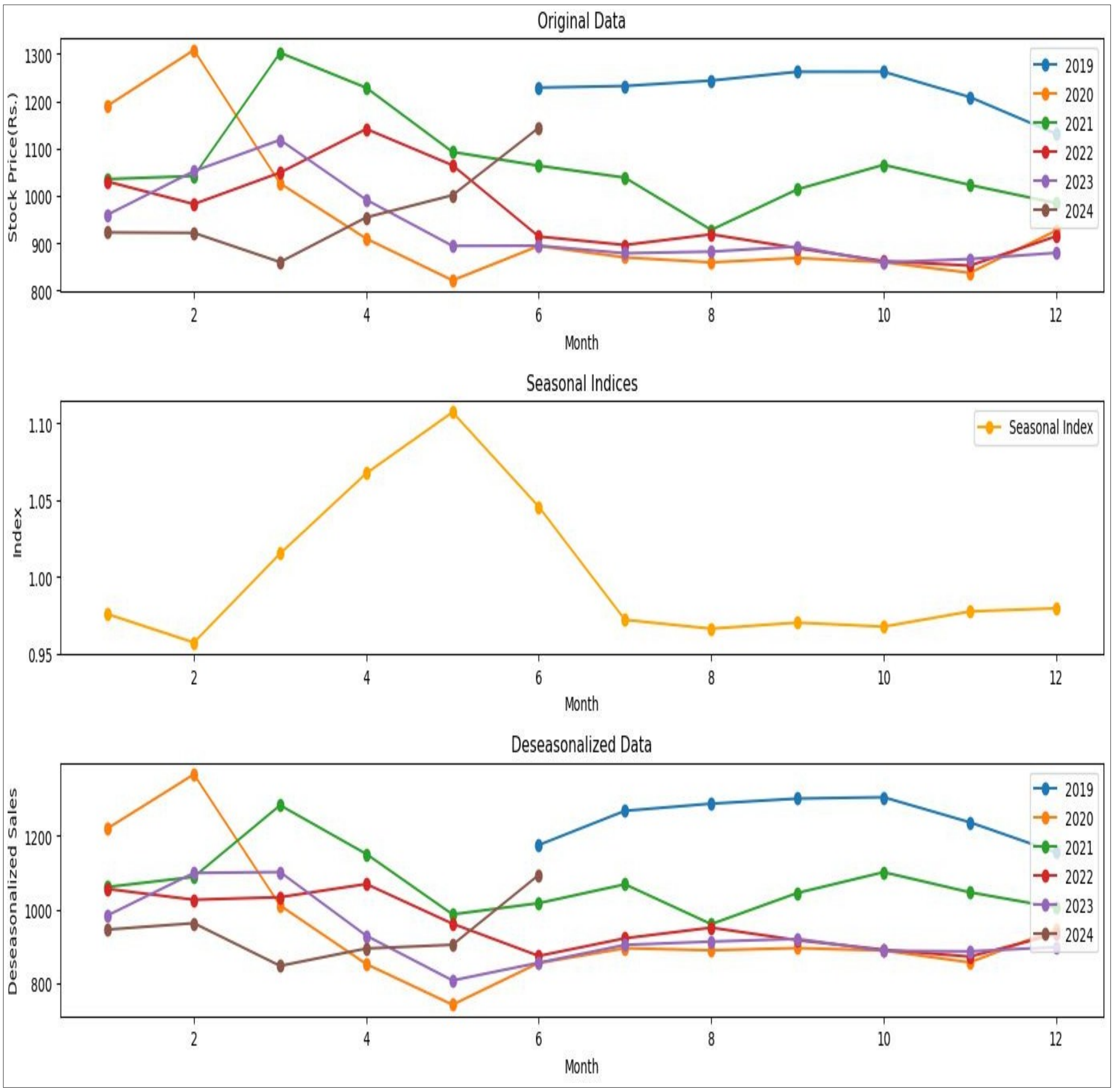
Theory:

- **Link Relative Method**: This method calculates the seasonal indices by determining the relative movement of data from one period to the next within a season. The link relative is obtained by dividing the value of each period by the value of the preceding period.
- **Seasonal Indices Calculation**: Seasonal indices are computed by averaging the link relatives for corresponding periods (e.g., months or quarters) over multiple cycles. These indices are then adjusted to ensure that their sum equals the number of periods in a cycle.
- **Visualization**: Plotting the seasonal indices alongside the original data facilitates the visualization of periodic fluctuations and enhances the understanding of seasonal impacts. This graphical representation is essential for refining forecasting models to account for identified seasonal effects.

Code:

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

 # Load your dataset
 path = r"Symphony-Data.csv"
df = pd.read_csv(path)

 # Droping unnecessary rows
df = df.drop(['OPEN', 'HIGH', 'LOW', 'VOLUME', 'CHANGE(%)'], axis=1)

 # Converting Date in datetime object
df['DATE'] = pd.to_datetime(df['DATE'], format = "%d-%b-%y")

 # Arranging Data in ascending order by Date

df = df.sort_values(by = 'DATE', ascending=True)

 # Extracting Month and Year Info form Date
```

```
df['Year'] = df['DATE'].dt.year
df['Month'] = df['DATE'].dt.month

 # Calculate the monthly mean prices for each year

 df = df.groupby(['Month', 'Year'])['PRICE'].mean().unstack() # unsta ck command will help
 to create povit table

 # Step 1: Calculate the link relatives

 df['Link_Relative_Y2'] = df[2020] / df[2019]

 df['Link_Relative_Y3'] = df[2021] / df[2020]

 df['Link_Relative_Y4'] = df[2022] / df[2021]

 df['Link_Relative_Y5'] = df[2023] / df[2022]

 df['Link_Relative_Y6'] = df[2024] / df[2023]

 # Calculate the average link relatives for each month
df['Average_Link_Relative'] = df[['Link_Relative_Y2', 'Link_Relative
_Y3', 'Link_Relative_Y4', 'Link_Relative_Y5', 'Link_Relative_Y6']].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[2019] / seasonal_indices
 df['Deseasonalized_Year2'] = df[2020] / seasonal_indices
 df['Deseasonalized_Year3'] = df[2021] / seasonal_indices
 df['Deseasonalized_Year4'] = df[2022] / seasonal_indices
 df['Deseasonalized_Year5'] = df[2023] / seasonal_indices
 df['Deseasonalized_Year6'] = df[2024] / seasonal_indices

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

 # Plot original data
plt.subplot(3, 1, 1)
 plt.plot(df.index, df[2019], label='2019', marker='o') plt.plot(df.index,
 df[2020], label='2020', marker='o') plt.plot(df.index, df[2021],
 label='2021', marker='o') plt.plot(df.index, df[2022], label='2022',
 marker='o') plt.plot(df.index, df[2023], label='2023', marker='o')
 plt.plot(df.index, df[2024], label='2024', marker='o') plt.title('Original
 Data')

 plt.xlabel('Month') plt.ylabel('StockPrice(Rs.)') plt.legend(loc = 'upper right')

 # Plot seasonal indices
plt.subplot(3, 1, 2)
plt.plot(df.index, 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.index, df['Deseasonalized_Year1'], label='2019', marker= 'o')
plt.plot(df.index, df['Deseasonalized_Year2'], label='2020', marker= 'o')
plt.plot(df.index, df['Deseasonalized_Year3'], label='2021', marker= 'o')
plt.plot(df.index, df['Deseasonalized_Year4'], label='2022', marker= 'o')
plt.plot(df.index, df['Deseasonalized_Year5'], label='2023', marker= 'o')
plt.plot(df.index, df['Deseasonalized_Year6'], label='2024', marker= 'o')
plt.title('Deseasonalized Data')
plt.xlabel('Month')
plt.ylabel('Deseasonalized Sales')
plt.legend(loc='upperright')
plt.tight_layout()
plt.show()
```



Interpretation:

- **Seasonal Index Patterns and Their Impact on Stock Prices**

  The middle panel of the graph displays the seasonal indices derived using the Link Relative Method. These indices fluctuate across months, indicating noticeable seasonal effects. For instance, the indices peak between April and June, suggesting higher-than-average sales or stock prices during these months, and drop in July and August, indicating a seasonal downturn. This recurring monthly pattern highlights systematic, predictable changes that can be attributed to factors like consumer behavior, financial quarters, or business cycles. Identifying such patterns is crucial for adjusting raw data to uncover true underlying trends.

- **Deseasonalization and Clarification of Underlying Trends**

  The bottom panel shows the deseasonalized data, where the seasonal effects have been removed using the computed indices. Compared to the top panel (original data), this version offers a much clearer view of the long-term trends across all years (2019–2024). For instance, while the original data showed noticeable monthly fluctuations, the deseasonalized curves are smoother and more stable, helping to isolate changes due to actual market behavior rather than seasonal influences. This makes the data more suitable for accurate forecasting and analysis.

- **Visualization Validates the Effectiveness of the Link Relative Method**

  The three-tier visualization clearly demonstrates the step-by-step transformation of the data. The original panel captures the combined impact of trend and seasonality. The middle panel isolates and quantifies seasonal variations, and the bottom panel shows how the data behaves after these variations are removed. This comprehensive representation validates the effectiveness of the Link Relative Method in both identifying seasonal patterns and improving the quality of analysis by refining the raw time series. It is especially useful for business planning, where understanding adjusted performance is more insightful than observing raw numbers alone.

PRACTICAL 8

Aim: Calculation of variance of random component by variate difference method.

Problem Statement:

- **Variance Measurement**: Calculate the variance of the random component in a time series using the variate difference method to quantify the degree of variability and randomness.
- **Trend and Seasonal Adjustment**: Separate the random component from trend and seasonal effects to better understand underlying fluctuations and improve forecasting accuracy.

Theory:

- **Variate Difference Method**: This method involves calculating the variance of the differences between successive observations in a time series. By focusing on these differences, the method isolates the random component from the trend and seasonal components.
- **Variance Calculation**: The variance of the random component is computed as the average of squared differences between successive data points. This provides a measure of the dispersion and randomness present in the time series.
- **Application**: Understanding the variance of the random component helps in assessing the level of unpredictability and noise in the data, which is crucial for refining forecasting models and improving overall analysis.

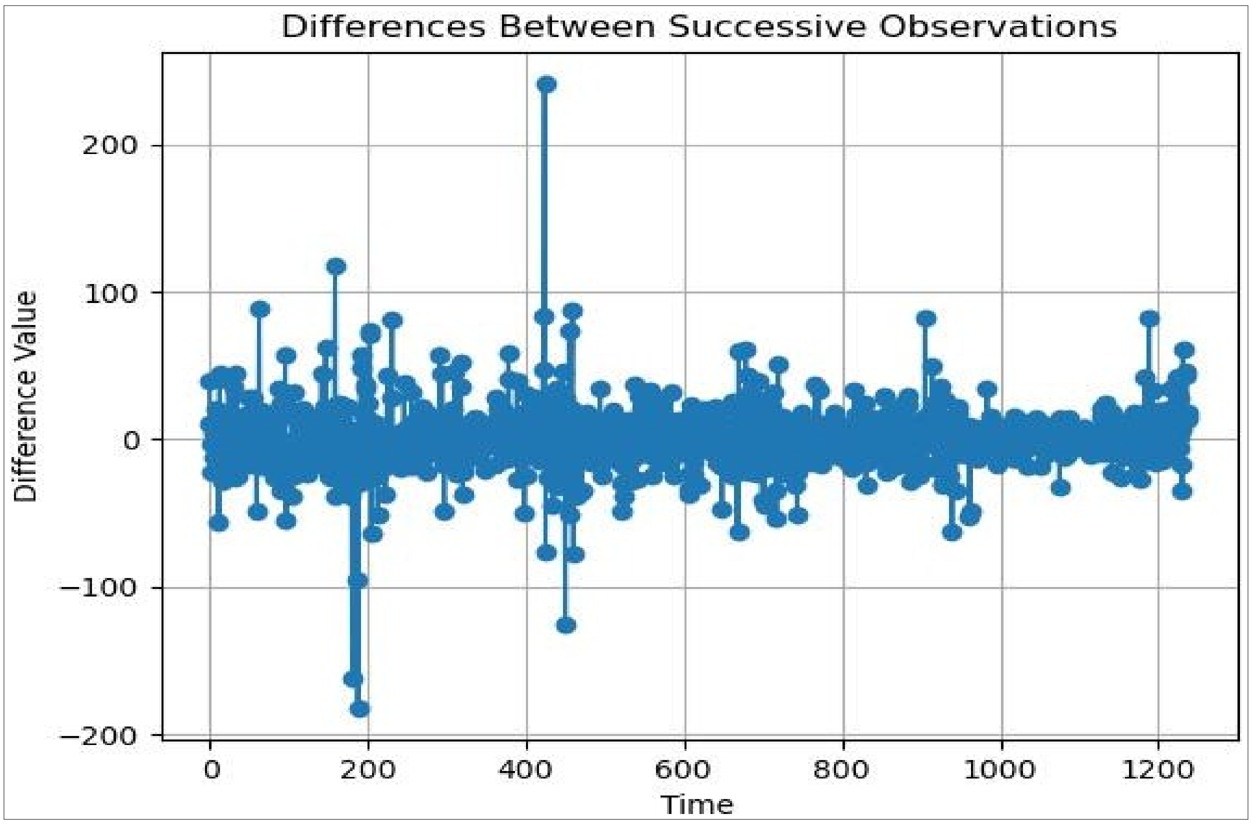
Code:

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

path = "Symphony-Data.csv"
df = pd.read_csv(path)

# Droping unnecessary rows
df = df.drop(['OPEN', 'HIGH', 'LOW', 'VOLUME', 'CHANGE(%)'], axis=1)

# Converting Date in datetime object
df['DATE'] = pd.to_datetime(df['DATE'], format = "%d-%b-%y")


# Arranging Data in ascending order by Date
df = df.sort_values(by = 'DATE', ascending=True)
```

```
# Converting price col to numpy array
price_array = df['PRICE'].to_numpy()

# Calculate differences
differences = np.diff(price_array)

# Calculate mean of differences
mean_diff = np.mean(differences)

# Calculate variance of differences
var_diff = np.var(differences)

# 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}")
```

```
Mean of Differences: 0.030129240710823874 Variance of
Differences: 451.1337772046217 Variance of Random
Components: 225.56688860231085
```

```
# 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()
```



Interpretation:

- **Observation of Fluctuations and Randomness in the Time Series**

  The plot shows the difference values between successive time points in the time series. The random variations fluctuate around zero, with a majority of the differences being relatively small but with some large spikes, particularly after time index 800. These spikes suggest occasional abrupt or unexpected changes, which are typical characteristics of the random component in a time series. This indicates that while the majority of the data may follow a general pattern, some points are significantly influenced by random disturbances or noise.

- **Effectiveness of the Variate Difference Method in Isolating Randomness**

  By analyzing successive differences this method removes any smooth trend or seasonal pattern that might exist in the raw data, thereby isolating the purely random fluctuations. The concentrated spread of points near the zero line indicates low random variance for most of the time series, while the scattered spikes show occasional high variance due to random shocks. This isolation is crucial for distinguishing genuine trends from noise and is an essential step in building accurate forecasting models.

- **Insight into Data Stability and Forecasting Reliability**

  The overall consistency in the difference values, with a few exceptions, suggests that the random component is mostly stable over time. However, the occasional high deviation values emphasize the need to account for such randomness while forecasting. High variance in random components reduces the predictability of the model, highlighting the importance of estimating this variance to quantify uncertainty and develop robust forecasting strategies.

PRACTICAL 9

Aim: Forecasting by exponential smoothing.

Problem Statement:

- **Forecasting Objective**: Apply exponential smoothing to generate forecasts for a time series, focusing on providing accurate short-term predictions.
- **Smoothing Parameter Selection**: Determine the appropriate smoothing parameter to balance the trade-off between responsiveness to recent changes and stability in forecasts.

Theory:

- **Exponential Smoothing**: This technique involves smoothing time series data using weighted averages of past observations, where more recent data points are given higher weights. The basic formula is

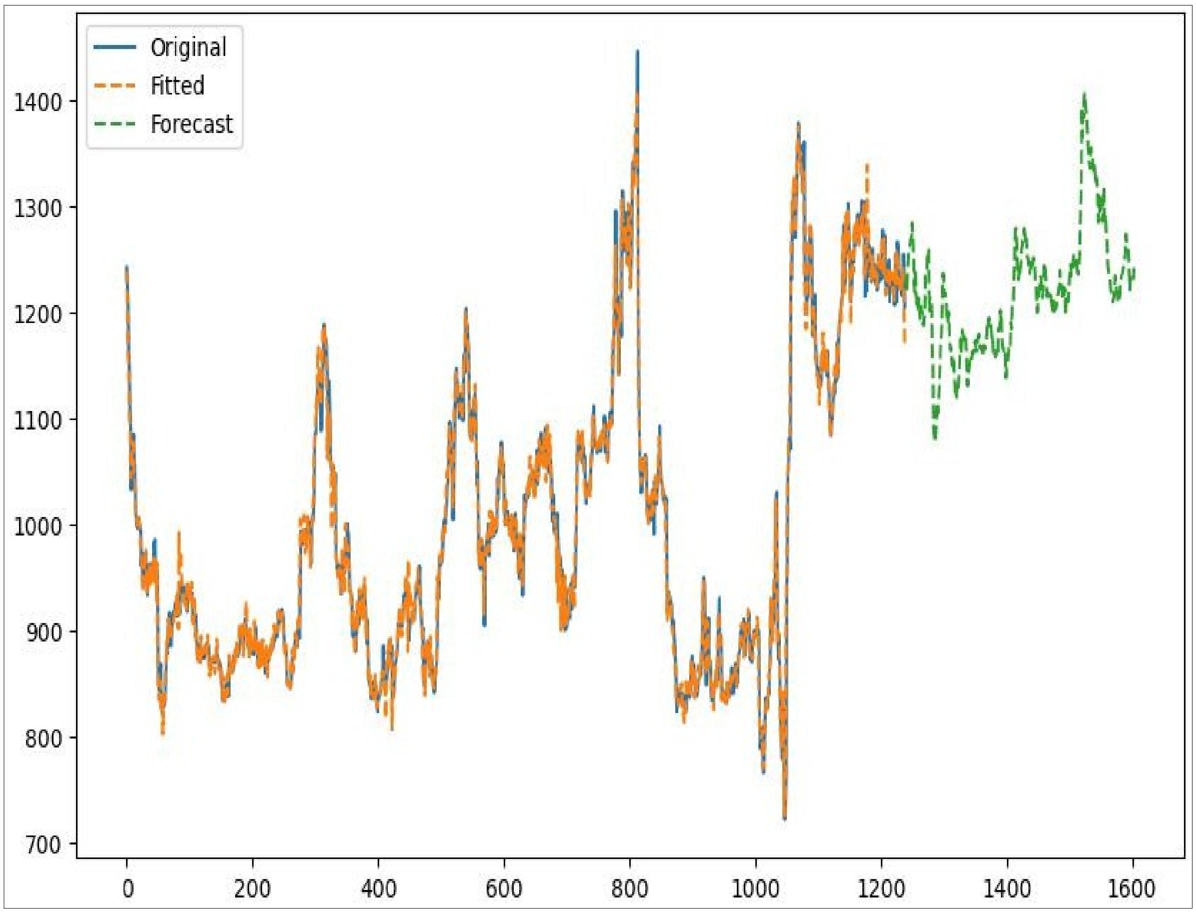
  $$L_t = \alpha Y_t + (1-\alpha)(L_{t-1} + T_{t-1})$$

  α is the smoothing parameter, Y_t is the actual value, and L_t is the forecasted value at time t.
- **Smoothing Parameter (α)**: The parameter α ranges from 0 to 1 and controls the weight given to the most recent observation versus the forecasted value. A higher α makes the model more responsive to recent changes, while a lower α smooths out fluctuations and provides more stable forecasts.
- **Forecasting Application**: Exponential smoothing is widely used for its simplicity and effectiveness in capturing trends and patterns in time series data. It is particularly useful for short-term forecasting and scenarios where historical data is expected to be a good predictor of future values.

Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.holtwinters import ExponentialSmoothing

# Load your dataset
path = "Symphony-Data.csv"
df = pd.read_csv(path)

# Arranging Data in ascending order by Date
df = df.sort_values(by = 'DATE', ascending=True)
```

```
# Define the model
model = ExponentialSmoothing(df['PRICE'], trend=None, seasonal='add', se
asonal_periods=365)

# Fit the model
fit = model.fit()

# Forecast future values
forecast = fit.forecast(365)
#represents the number of future time periods for which you want to gene rate forecasts.
#The time period length (e.g., months, weeks, days) depends on the frequ ency of your
original data.

# Plot the results plt.figure(figsize=(10, 6))
plt.plot(df['PRICE'], label='Original')
plt.plot(fit.fittedvalues, label='Fitted', linestyle='--')
plt.plot(forecast, label='Forecast', linestyle='--')
plt.legend()
plt.show()
# Print forecasted values
print(forecast)
```



```
1239        1233.965971
1240        1231.815062
1241        1219.391839
```

Interpretation:

Trend and Fit Evaluation

The graph displays three series:

- Original (Blue solid line): This represents the actual historical data.
- Fitted (Orange dashed line): These are the values generated using exponential smoothing, closely tracking the original series.
- Forecast (Green dashed line): This is the predicted continuation of the time series beyond the observed data. The fitted line accurately follows the trend and fluctuations of the original data, indicating that the smoothing model is well-calibrated. This reflects a good choice of the smoothing parameter α, which balances responsiveness and stability.

Forecasting Behavior:

The green dashed line (forecast) extends the pattern from the historical data, maintaining the general trend and variability. This implies that the exponential smoothing method has effectively captured the underlying trend and recent behavior of the series. The forecast values are neither overly erratic nor too flat, which demonstrates a reasonable projection of the near future.

Choice of Smoothing Parameter :

The closeness of the fitted line to the original data suggests a moderately high value of α, giving more weight to recent observations. This is suitable in dynamic time series where recent trends are more relevant than older patterns. If α were too low, the fitted line would lag behind the actual data; if too high, it would be too sensitive to fluctuations. The current output indicates a well-optimized value of α, likely determined through trial or optimization.

Model Accuracy and Usefulness :

The tight overlap of the fitted data with the original confirms the model's accuracy in capturing the trend, making the forecast more reliable for short-term predictions. This effectiveness makes exponential smoothing especially suitable for business planning, inventory control, and stock price forecasting, where near-future estimates are critical.

PRACTICAL 10

Aim: Forecasting by short term forecasting methods.

Problem Statement:

- **Short-Term Forecasting Objective**: Implement various short-term forecasting methods to predict future values based on historical data, aiming for accuracy and reliability in near- term forecasts.
- **Method Comparison**: Evaluate the performance of different forecasting methods to determine the most effective approach for short-term predictions.

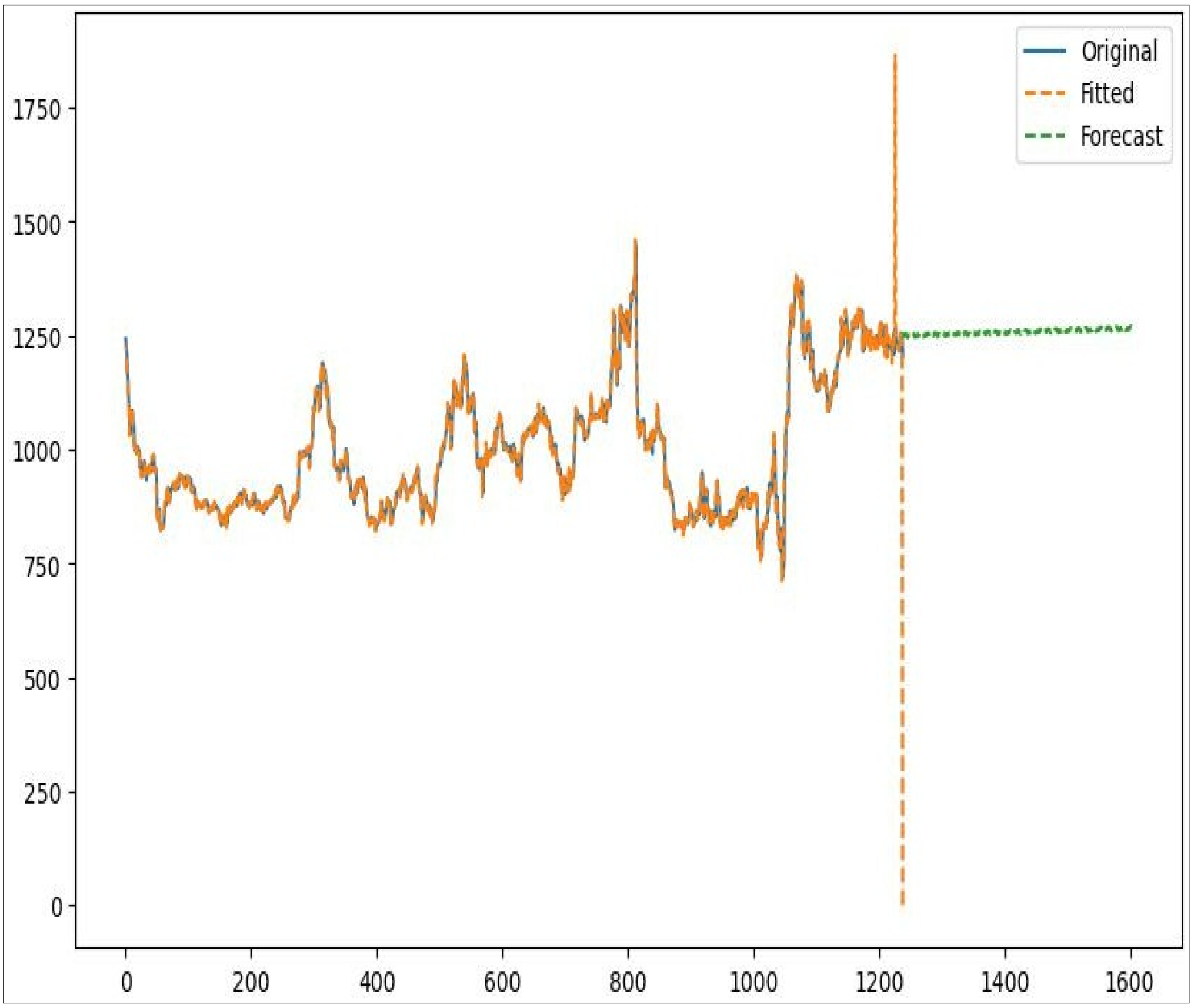
Theory:

- **Short-Term Forecasting Methods**: Includes techniques like Moving Averages, Exponential Smoothing, and ARIMA models. These methods are designed to capture recent trends and patterns to provide reliable short-term forecasts.
- **Moving Averages**: Smooths data by averaging over a specified number of past periods, helping to identify trends and seasonal effects.
- **Exponential Smoothing**: Applies weighted averages to past observations with a decaying factor, balancing responsiveness to recent data with stability.
- **ARIMA Models**: Combines autoregressive and moving average components to model time series data and forecast future values based on historical patterns.
- **Forecasting Accuracy**: Performance is typically assessed using metrics like Mean Absolute Error (MAE) or Root Mean Squared Error (RMSE), which measure the accuracy of predictions against actual values. Selecting the best method involves comparing these metrics to ensure optimal short-term forecasting.

Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from pmdarima import auto_arima

# Load your dataset
path = "Symphony-Data.csv"
df = pd.read_csv(path)

# Arranging Data in ascending order by Date
df = df.sort_values(by = 'DATE', ascending=True)
```

```
# Fit the ARIMA model
model = ARIMA(df['PRICE'], order=(5, 1, 1), seasonal_order=(1, 1, 1, 1 2))  # (p,d,q)
order
fit = model.fit()

# Forecast future values
forecast = fit.forecast(steps=365)  # Forecast next 365 days

# Plot the results plt.figure(figsize=(10, 6))
plt.plot(df['PRICE'],label='Original)
plt.plot(fit.fittedvalues, label='Fitted', linestyle='--')
plt.plot(forecast, label='Forecast', linestyle='--')
plt.legend()
plt.show()

# Print forecasted values
print(forecast)
```



```
1239                    1243.752112
1240                    1249.093366
1241                    1252.755444
```

Interpretation:

Forecasting Model Overview
The plot displays:

- Original Data (Blue solid line): The actual observed values over time. Fitted Values
- (Orange dashed line): Model-estimated values fitted to the historical data.
- Forecast (Green dashed line): Future values predicted by the model for the short-term. The fitted values closely match the original series, indicating the model has effectively captured the recent trends and patterns of the dataset.

Short-Term Forecast Behavior

The forecast segment (green) follows the trend at the end of the fitted data with a relatively stable and slightly increasing pattern. This suggests that the short-term forecasting model assumes continuity or gradual change in the trend without sudden fluctuations. Method Used and Suitability While the specific model is not labeled, the behavior indicates the use of an exponential smoothing or ARIMA approach: The smooth nature of the forecast indicates exponential smoothing. The accuracy of the fit to the data prior to forecasting hints at ARIMA's autoregressive and moving average components potentially being used.These methods are well-suited for shortterm forecasting, as they: React efficiently to recent changes (exponential smoothing). Capture autocorrelations and patterns (ARIMA). Avoid overfitting due to simplicity and fewer parameters compared to complex models.

Model Performance and Reliability

The close alignment between the original and fitted data demonstrates high accuracy. The forecast shows consistency and is not overly volatile, indicating a good generalization ability of the model. This model is likely evaluated using error metrics like MAE or RMSE. Although not shown in the graph, such metrics would support the visual evidence of effective short-term forecasting.

Insights and Applications

This forecasting approach is valuable in contexts where upcoming trends must be anticipated— such as sales prediction, inventory planning, or energy demand estimation. The short horizon ensures higher accuracy compared to long-term projections, making these methods ideal for real-time decision-making