```
# PRACTICAL 01: Modified Exponential Curve
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.optimize import curve_fit
from sklearn.metrics import r2_score

# Step 1: Load the dataset
path = "co2-ppm-daily.csv"  # Update with your actual path
df = pd.read_csv(path)

# Step 2: Preprocess and transform
df['date'] = pd.to_datetime(df['date'])
df.sort_values('date', inplace=True)
df['Time'] = (df['date'] - df['date'].min()).dt.days
x = df['Time']
y = df['value']

# Step 3: Define the modified exponential model
def mod_exp(t, A, B, C):
    return A * np.exp(B * t) + C

# Step 4: Estimate initial values
A_init = y.iloc[0]
B_init = 1 / df['date'].iloc[len(df)//2].year
C_init = y.iloc[0]
p0 = [A_init, B_init, C_init]

# Step 5: Fit the model
params, cov = curve_fit(mod_exp, x, y, p0=p0, maxfev=10000)

# Step 6: Generate predictions
y_pred = mod_exp(x, *params)

# Step 7: Visualize
plt.figure(figsize=(12, 6))
plt.plot(df['date'], y, label="Actual Data")
plt.plot(df['date'], y_pred, '--', label="Fitted Modified Exponential Curve")
plt.title("CO2 Emission - Modified Exponential Fit")
plt.xlabel("Date")
plt.ylabel("CO2 Emission (ppm)")
plt.legend()
plt.grid()
plt.tight_layout()
plt.show()

# Step 8: Evaluate model
r2 = r2_score(y, y_pred)
print("R² Score:", r2)
print("Optimized Parameters: A = {:.3f}, B = {:.6f}, C = {:.3f}".format(*params))
print("Covariance Matrix:\n", cov)

# Heatmap of parameter covariance
sns.heatmap(cov, annot=True, cmap='coolwarm')
plt.title("Covariance Matrix Heatmap")
plt.show()



# PRACTICAL 02: Gompertz Curve Fitting
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.optimize import curve_fit
from sklearn.metrics import r2_score

# Step 1: Load the dataset
path = "taq-cat-t-jan042010.txt"  # Update with your actual path
df = pd.read_csv(path, sep='\s+')

# Step 2: Combine datetime fields
df['Datetime'] = pd.to_datetime(df['date'].astype(str) + ' ' +
                                df['hour'].astype(str) + ':' +
                                df['minute'].astype(str) + ':' +
                                df['second'].astype(str))

# Drop original time columns
df.drop(['date', 'hour', 'minute', 'second'], axis=1, inplace=True)

# Step 3: Sort and set time index
df.sort_values('Datetime', inplace=True)
df.set_index('Datetime', inplace=True)

# Step 4: Prepare data
x_time = (df.index - df.index.min()).total_seconds()
y = df['price']

# Step 5: Define Gompertz model
def gompertz_curve(t, a, b, c):
    return a * np.exp(b * np.exp(-c * t))

# Step 6: Estimate initial guesses
a_init = y.iloc[0]
b_init = 2.7
c_init = 1 / df.index[int(len(df)/2)].year
p0 = [a_init, b_init, c_init]

# Step 7: Fit the model
params, cov = curve_fit(gompertz_curve, x_time, y, p0=p0, maxfev=10000)

# Step 8: Predict
y_pred = gompertz_curve(x_time, *params)

# Step 9: Visualize
plt.figure(figsize=(12, 6))
plt.plot(df.index, y, label="Original Stock Price")
plt.plot(df.index, y_pred, '--', label="Fitted Gompertz Curve")
plt.title("Stock Price Over Time - Gompertz Curve Fit")
plt.xlabel("Datetime")
plt.ylabel("Price")
plt.legend()
plt.grid()
plt.tight_layout()
plt.show()

# Step 10: Model evaluation
r2 = r2_score(y, y_pred)
print("R² Score:", r2)
print("Optimized Parameters: a = {:.4f}, b = {:.4f}, c = {:.6f}".format(*params))
print("Covariance Matrix:\n", cov)

# Heatmap for covariance matrix
sns.heatmap(cov, annot=True, cmap="coolwarm")
plt.title("Covariance Matrix of Parameters")
plt.show()



# PRACTICAL 03: Logistic Curve Fitting
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.optimize import curve_fit
from sklearn.metrics import r2_score

# Step 1: Load the dataset
path = "microbial_growth_curve.csv"  # Update with your actual path
df = pd.read_csv(path)

# Step 2: Prepare data
x = df['t [h]']
y = df['microbes(g)']
x_time = x - x.min()  # Normalize time to start at 0

# Step 3: Define logistic model
def logistic_curve(t, K, r, t0):
    return K / (1 + np.exp(-r * (t - t0)))

# Step 4: Initial guesses
K_init = y.max()
```

```
r_init = 1
t0_init = x.median()
p0 = [K_init, r_init, t0_init]

# Step 5: Fit model
params, cov = curve_fit(logistic_curve, x_time, y, p0=p0, maxfev=10000)

# Step 6: Predict
y_pred = logistic_curve(x_time, *params)

# Step 7: Plotting
plt.figure(figsize=(12, 6))
plt.plot(x, y, label="Original Microbes Growth")
plt.plot(x, y_pred, '--', label="Fitted Logistic Curve")
plt.title("Microbial Growth - Logistic Curve Fit")
plt.xlabel("Time (hours)")
plt.ylabel("Microbial Mass (g)")
plt.legend()
plt.grid()
plt.tight_layout()
plt.show()

# Step 8: Evaluation
r2 = r2_score(y, y_pred)
print("R² Score:", r2)
print("Optimized Parameters: K = {:.4f}, r = {:.4f}, t0 = {:.4f}".format(*params))
print("Covariance Matrix:\n", cov)

# Step 9: Covariance Matrix Heatmap
sns.heatmap(cov, annot=True, cmap="coolwarm")
plt.title("Covariance Matrix of Logistic Curve Parameters")
plt.show()



# PRACTICAL 04: Trend Fitting using Moving Average Method
import pandas as pd
import matplotlib.pyplot as plt

# Step 1: Load dataset
path = "Symphony-Data.csv"  # Update with your actual path
df = pd.read_csv(path)

# Step 2: Preprocessing
df = df.drop(columns=['OPEN', 'HIGH', 'LOW', 'VOLUME', 'CHANGE(%)'])  # Drop irrelevant columns
df['DATE'] = pd.to_datetime(df['DATE'], format='%d-%b-%y')
df = df.sort_values('DATE')

# Step 3: Calculate 30-day (1 month) moving average
df['Moving_Avg'] = df['PRICE'].rolling(window=30).mean()

# Step 4: Calculate trend (rate of change in moving average)
df['Trend'] = df['Moving_Avg'].diff()

# Step 5: Calculate seasonal variation (actual - moving average)
df['Seasonal_Variation'] = df['PRICE'] - df['Moving_Avg']

# Step 6: Plot original data with moving average
plt.figure(figsize=(12, 6))
plt.plot(df['DATE'], df['PRICE'], label='Original Data')
plt.plot(df['DATE'], df['Moving_Avg'], label='Moving Average', color='red')
plt.xlabel("Date")
plt.ylabel("Stock Price (Rs.)")
plt.title("Stock Price with Moving Average Trend")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Step 7: Plot trend and seasonal variation
plt.figure(figsize=(12, 6))
plt.plot(df['DATE'], df['Trend'], label='Trend (Δ Moving Avg)', color='green')
plt.plot(df['DATE'], df['Seasonal_Variation'], label='Seasonal Variation', color='orange')
plt.xlabel("Date")
plt.ylabel("Value")
plt.title("Trend and Seasonal Variation")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()



# PRACTICAL 05: Seasonal Indices using Ratio-to-Trend Method
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Load dataset
path = "Symphony-Data.csv"  # Update with your actual path
df = pd.read_csv(path)

# Step 2: Preprocess
df = df.drop(columns=['OPEN', 'HIGH', 'LOW', 'VOLUME', 'CHANGE(%)'])
df['DATE'] = pd.to_datetime(df['DATE'], format="%d-%b-%y")
df.sort_values('DATE', inplace=True)

# Step 3: Extract year and month
df['Year'] = df['DATE'].dt.year
df['Month'] = df['DATE'].dt.month

# Step 4: Calculate monthly means for each year
monthly_avg = df.groupby(['Month', 'Year'])['PRICE'].mean().unstack()

# Step 5: Plot monthly averages
monthly_avg.plot(marker='o', figsize=(12, 6), title="Month-wise Average Stock Prices")
plt.xlabel("Month")
plt.ylabel("Average Price")
plt.grid(True)
plt.legend(title="Year")
plt.tight_layout()
plt.show()

# Step 6: Calculate yearly trend (average price)
yearly_trend = monthly_avg.mean()

# Step 7: Compute ratio of each month to yearly trend
ratios = monthly_avg.copy()
for year in yearly_trend.index:
    ratios[f'ratio_{year}'] = monthly_avg[year] / yearly_trend[year]

# Step 8: Compute normalized seasonal indices
ratio_cols = [col for col in ratios.columns if 'ratio_' in str(col)]
normalized_indices = ratios[ratio_cols].mean(axis=1)

# Step 9: Plot normalized seasonal indices
plt.figure(figsize=(10, 5))
plt.plot(normalized_indices.index, normalized_indices, marker='o')
plt.title("Normalized Seasonal Indices")
plt.xlabel("Month")
plt.ylabel("Index")
plt.grid(True)
plt.tight_layout()
plt.show()

# Step 10: Deseasonalized data calculation (for all years)
monthly_mean = df.groupby(['Year', 'Month'])['PRICE'].mean().unstack()
deseasonalized = pd.DataFrame()
for month in range(1, 13):
    deseasonalized[month] = monthly_mean[month] / normalized_indices[month]

# Step 11: Plot deseasonalized prices
plt.figure(figsize=(12, 6))
for year in deseasonalized.index:
    plt.plot(deseasonalized.columns, deseasonalized.loc[year], marker='o', label=str(year))

plt.title("Deseasonalized Monthly Prices")
plt.xlabel("Month")
plt.ylabel("Price (Deseasonalized)")
plt.legend(title="Year")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
# PRACTICAL 06: Ratio-to-Moving Average Seasonal Indices
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Load dataset
path = "Symphony-Data.csv"  # Update with your actual path
df = pd.read_csv(path)

# Step 2: Preprocessing
df = df.drop(columns=['OPEN', 'HIGH', 'LOW', 'VOLUME', 'CHANGE(%)'])
df['DATE'] = pd.to_datetime(df['DATE'], format="%d-%b-%y")
df.sort_values('DATE', inplace=True)

# Step 3: Extract Month and Year
df['Month'] = df['DATE'].dt.month
df['Year'] = df['DATE'].dt.year

# Step 4: Calculate monthly average for each year
monthly_avg = df.groupby(['Month', 'Year'])['PRICE'].mean().unstack()

# Step 5: Calculate overall monthly average
monthly_avg['Monthly_Avg'] = monthly_avg.mean(axis=1)

# Step 6: Calculate centered moving average
monthly_avg['Centered_MA'] = monthly_avg['Monthly_Avg'].rolling(window=2, center=True).mean()

# Step 7: Calculate ratio of actual to centered MA
monthly_avg['Ratio'] = monthly_avg['Monthly_Avg'] / monthly_avg['Centered_MA']

# Step 8: Normalize ratios to sum = 12 (months)
sum_ratios = monthly_avg['Ratio'].sum()
monthly_avg['Seasonal_Index'] = monthly_avg['Ratio'] * (12 / sum_ratios)

# Step 9: Deseasonalize yearly data using seasonal index
for year in [2019, 2020, 2021, 2022, 2023, 2024]:
    monthly_avg[f'Deseasonalized_{year}'] = monthly_avg[year] / monthly_avg['Seasonal_Index']

# Step 10: Plotting
plt.figure(figsize=(14, 10))

# Original Data
plt.subplot(3, 1, 1)
for year in [2019, 2020, 2021, 2022, 2023, 2024]:
    plt.plot(monthly_avg.index, monthly_avg[year], marker='o', label=str(year))
plt.title("Original Monthly Stock Prices")
plt.xlabel("Month")
plt.ylabel("Price")
plt.legend(loc='upper right')
plt.grid()

# Seasonal Indices
plt.subplot(3, 1, 2)
plt.plot(monthly_avg.index, monthly_avg['Seasonal_Index'], marker='o', color='orange', label="Seasonal Index")
plt.title("Seasonal Indices (Ratio-to-MA Method)")
plt.xlabel("Month")
plt.ylabel("Index")
plt.legend(loc='upper right')
plt.grid()

# Deseasonalized Data
plt.subplot(3, 1, 3)
for year in [2019, 2020, 2021, 2022, 2023, 2024]:
    plt.plot(monthly_avg.index, monthly_avg[f'Deseasonalized_{year}'], marker='o', label=f"{year} Deseasonalized")
plt.title("Deseasonalized Monthly Prices")
plt.xlabel("Month")
plt.ylabel("Price")
plt.legend(loc='upper right')
plt.grid()

plt.tight_layout()
plt.show()



# PRACTICAL 07: Seasonal Indices using Link Relative Method
import pandas as pd
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Load dataset
path = "Symphony-Data.csv"  # Update with your actual path
df = pd.read_csv(path)

# Step 2: Preprocessing
df = df.drop(columns=['OPEN', 'HIGH', 'LOW', 'VOLUME', 'CHANGE(%)'])
df['DATE'] = pd.to_datetime(df['DATE'], format="%d-%b-%y")
df.sort_values('DATE', inplace=True)

# Step 3: Extract Month and Year
df['Month'] = df['DATE'].dt.month
df['Year'] = df['DATE'].dt.year

# Step 4: Monthly average per year
monthly_avg = df.groupby(['Month', 'Year'])['PRICE'].mean().unstack()

# Step 5: Calculate link relatives (e.g., 2020/2019, 2021/2020, ...)
for i in range(1, len(monthly_avg.columns)):
    prev_year = monthly_avg.columns[i-1]
    curr_year = monthly_avg.columns[i]
    monthly_avg[f'LinkRel_{curr_year}'] = monthly_avg[curr_year] / monthly_avg[prev_year]

# Step 6: Compute average link relatives per month
link_rel_cols = [col for col in monthly_avg.columns if 'LinkRel_' in str(col)]
monthly_avg['Avg_Link_Rel'] = monthly_avg[link_rel_cols].mean(axis=1)

# Step 7: Normalize to make sum = 12 (months)
seasonal_index = monthly_avg['Avg_Link_Rel']
seasonal_index = seasonal_index / seasonal_index.sum() * 12

# Step 8: Deseasonalize original data using seasonal index
for year in monthly_avg.columns[:6]:  # Assuming years 2019 to 2024
    monthly_avg[f'Deseasonalized_{year}'] = monthly_avg[year] / seasonal_index

# Step 9: Plotting
plt.figure(figsize=(14, 10))

# Subplot 1: Original
plt.subplot(3, 1, 1)
for year in monthly_avg.columns[:6]:
    plt.plot(monthly_avg.index, monthly_avg[year], label=str(year), marker='o')
plt.title("Original Monthly Prices")
plt.xlabel("Month")
plt.ylabel("Price")
plt.legend(loc="upper right")
plt.grid()

# Subplot 2: Seasonal Indices
plt.subplot(3, 1, 2)
plt.plot(seasonal_index.index, seasonal_index, marker='o', color='orange', label="Seasonal Index")
plt.title("Seasonal Indices (Link Relative Method)")
plt.xlabel("Month")
plt.ylabel("Index")
plt.legend()
plt.grid()

# Subplot 3: Deseasonalized
plt.subplot(3, 1, 3)
for year in monthly_avg.columns[:6]:
    plt.plot(monthly_avg.index, monthly_avg[f'Deseasonalized_{year}'], label=f"{year} Deseasonalized", marker='o')
plt.title("Deseasonalized Monthly Prices")
plt.xlabel("Month")
plt.ylabel("Price")
plt.legend(loc="upper right")
plt.grid()

plt.tight_layout()
plt.show()



# PRACTICAL 08: Variance of Random Component (Variate Difference Method)
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Load dataset
path = "Symphony-Data.csv"  # Update with your actual path
df = pd.read_csv(path)

# Step 2: Preprocessing
df = df.drop(columns=['OPEN', 'HIGH', 'LOW', 'VOLUME', 'CHANGE(%)'])
df['DATE'] = pd.to_datetime(df['DATE'], format="%d-%b-%y")
df.sort_values('DATE', inplace=True)

# Step 3: Convert price column to numpy array
price_array = df['PRICE'].to_numpy()

# Step 4: Calculate first-order differences
differences = np.diff(price_array)

# Step 5: Compute statistics
mean_diff = np.mean(differences)
var_diff = np.var(differences)
var_random = var_diff / 2

# Step 6: Print results
print("Mean of Differences       :", mean_diff)
print("Variance of Differences   :", var_diff)
print("Variance of Random Component :", var_random)

# Step 7: Plot the differences
plt.figure(figsize=(12, 5))
plt.plot(differences, marker='o', linestyle='-', color='purple')
plt.title("First-order Differences of Stock Prices")
plt.xlabel("Time Index")
plt.ylabel("Difference Value")
plt.grid(True)
plt.tight_layout()
plt.show()



# PRACTICAL 09: Forecasting using Exponential Smoothing
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.holtwinters import ExponentialSmoothing

# Step 1: Load dataset
path = "Symphony-Data.csv"  # Update with your actual path
df = pd.read_csv(path)

# Step 2: Preprocess
df = df.drop(columns=['OPEN', 'HIGH', 'LOW', 'VOLUME', 'CHANGE(%)'])
df['DATE'] = pd.to_datetime(df['DATE'], format="%d-%b-%y")
df.sort_values('DATE', inplace=True)
df.set_index('DATE', inplace=True)

# Step 3: Fit Exponential Smoothing model (with additive seasonality)
# seasonal_periods = 30 (assuming ~monthly seasonality in daily data)
model = ExponentialSmoothing(
    df['PRICE'],
    trend=None,
    seasonal='add',
    seasonal_periods=30  # adjust based on your data granularity
)
fit = model.fit()

# Step 4: Forecast future values
forecast_period = 60  # e.g., forecast next 60 days
forecast = fit.forecast(forecast_period)

# Step 5: Plotting results
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['PRICE'], label='Original Data')
plt.plot(fit.fittedvalues.index, fit.fittedvalues, label='Fitted', linestyle='--')
plt.plot(forecast.index, forecast, label='Forecast', linestyle='--')
plt.title("Exponential Smoothing Forecast")
plt.xlabel("Date")
plt.ylabel("Stock Price")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Step 6: Print forecasted values
print("Forecasted Values:\n")
print(forecast)



# PRACTICAL 10: Short-Term Forecasting using ARIMA
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA

# Step 1: Load dataset
path = "Symphony-Data.csv"  # Update with your actual path
df = pd.read_csv(path)

# Step 2: Preprocessing
df = df.drop(columns=['OPEN', 'HIGH', 'LOW', 'VOLUME', 'CHANGE(%)'])
df['DATE'] = pd.to_datetime(df['DATE'], format="%d-%b-%y")
df.sort_values('DATE', inplace=True)
df.set_index('DATE', inplace=True)

# Step 3: Fit ARIMA model
# Example: ARIMA(5,1,1) — feel free to tune based on auto_arima or ACF/PACF plots
model = ARIMA(df['PRICE'], order=(5, 1, 1))  # (p,d,q)
fit = model.fit()

# Step 4: Forecast next N steps
forecast_steps = 60  # forecast next 60 days
forecast = fit.forecast(steps=forecast_steps)

# Step 5: Plot results
plt.figure(figsize=(12, 6))
plt.plot(df.index, df['PRICE'], label='Original Data')
plt.plot(fit.fittedvalues.index, fit.fittedvalues, label='Fitted Values', linestyle='--')
forecast_index = pd.date_range(start=df.index[-1], periods=forecast_steps + 1, freq='D')[1:]
plt.plot(forecast_index, forecast, label='Forecast', linestyle='--', color='red')
plt.title("ARIMA Short-Term Forecasting")
plt.xlabel("Date")
plt.ylabel("Stock Price")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Step 6: Output forecasted values
print("Forecasted Prices:")
print(forecast)
```