Gaayathri AG

24MSD7007

Import Libraries and Load Dataset

```
#import the necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.tsa.api import ExponentialSmoothing
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.datasets import co2
from sklearn.metrics import mean_squared_error ,mean_absolute_error

#load dataset
data = co2.load_pandas().data
data.head()
```

```
               co2
1958-03-29  316.1
1958-04-05  317.3
1958-04-12  317.6
1958-04-19  317.5
1958-04-26  316.4
```

Convert to Monthly Time Series Data

```
'''  here we are grouping the weeks into months using resample
and then using mean to aggregate the values'''
data = data.resample("M").mean()
```

```
C:\Users\gaaya\AppData\Local\Temp\ipykernel_18660\4226359250.py:3:
FutureWarning: 'M' is deprecated and will be removed in a future
version, please use 'ME' instead.
  data = data.resample("M").mean()
```

Check and Fill Missing Values

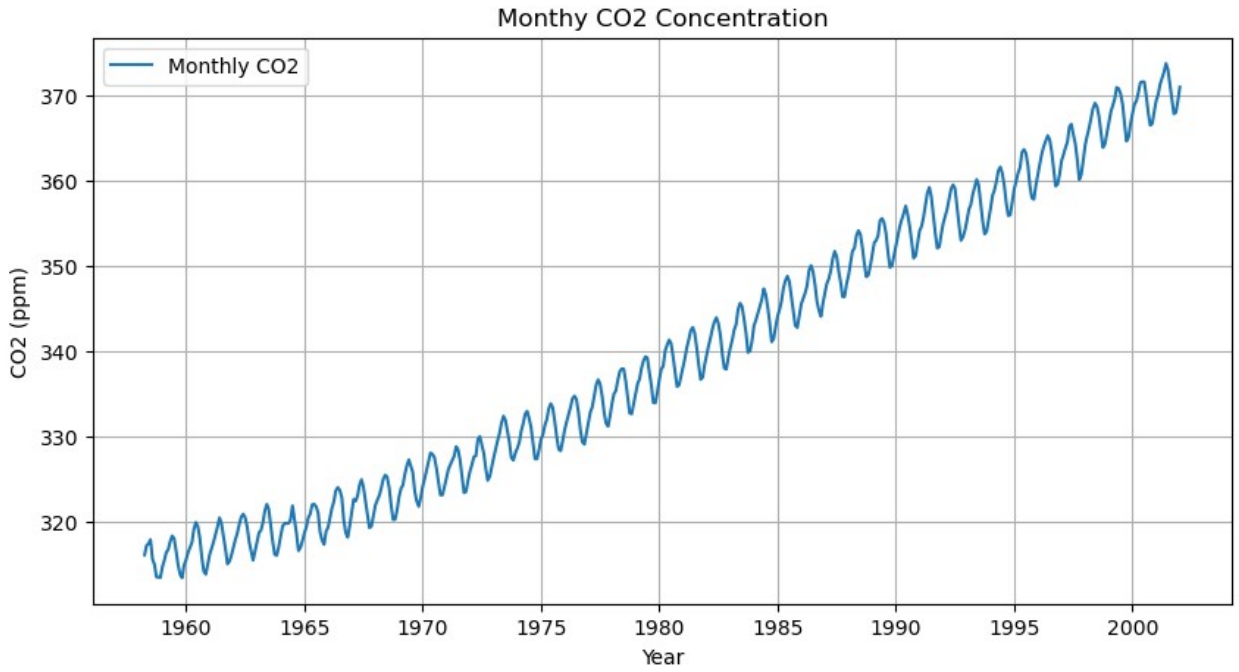
```
print(f"Missing values before filling: {data["co2"].isnull().sum()}")
# to print total no.of missing values
''' filling using forward fill
fills missing values with the last known valid value above it '''

data["co2"].fillna(method= "ffill",inplace=True) # changes saved

''' backfill fills values based on the next
valid value
Backward Fill: acts as a backup for any NaN at the start of the
```

```
series'''

data["co2"].fillna(method="bfill",inplace= True) # changes saved

# print no.of missing values after filling to check

print(f"Missing values after filling: {data["co2"].isnull().sum()}")
```

```
Missing values before filling: 0
Missing values after filling: 0

C:\Users\gaaya\AppData\Local\Temp\ipykernel_18660\90251003.py:5:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  data["co2"].fillna(method= "ffill",inplace=True) # changes saved
C:\Users\gaaya\AppData\Local\Temp\ipykernel_18660\90251003.py:5:
FutureWarning: Series.fillna with 'method' is deprecated and will
raise in a future version. Use obj.ffill() or obj.bfill() instead.
  data["co2"].fillna(method= "ffill",inplace=True) # changes saved
C:\Users\gaaya\AppData\Local\Temp\ipykernel_18660\90251003.py:11:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  data["co2"].fillna(method="bfill",inplace= True) # changes saved
C:\Users\gaaya\AppData\Local\Temp\ipykernel_18660\90251003.py:11:
FutureWarning: Series.fillna with 'method' is deprecated and will
raise in a future version. Use obj.ffill() or obj.bfill() instead.
  data["co2"].fillna(method="bfill",inplace= True) # changes saved
```

Plot the Monthly Time Series Data

```
plt.figure(figsize =(10,5))
plt.plot(data["co2"],label = "Monthly CO2") # using the co2
concentration for plotting
plt.title("Monthy CO2 Concentration")
plt.xlabel("Year")
plt.ylabel("CO2 (ppm)")
plt.legend()
plt.grid(True)
plt.show()
```

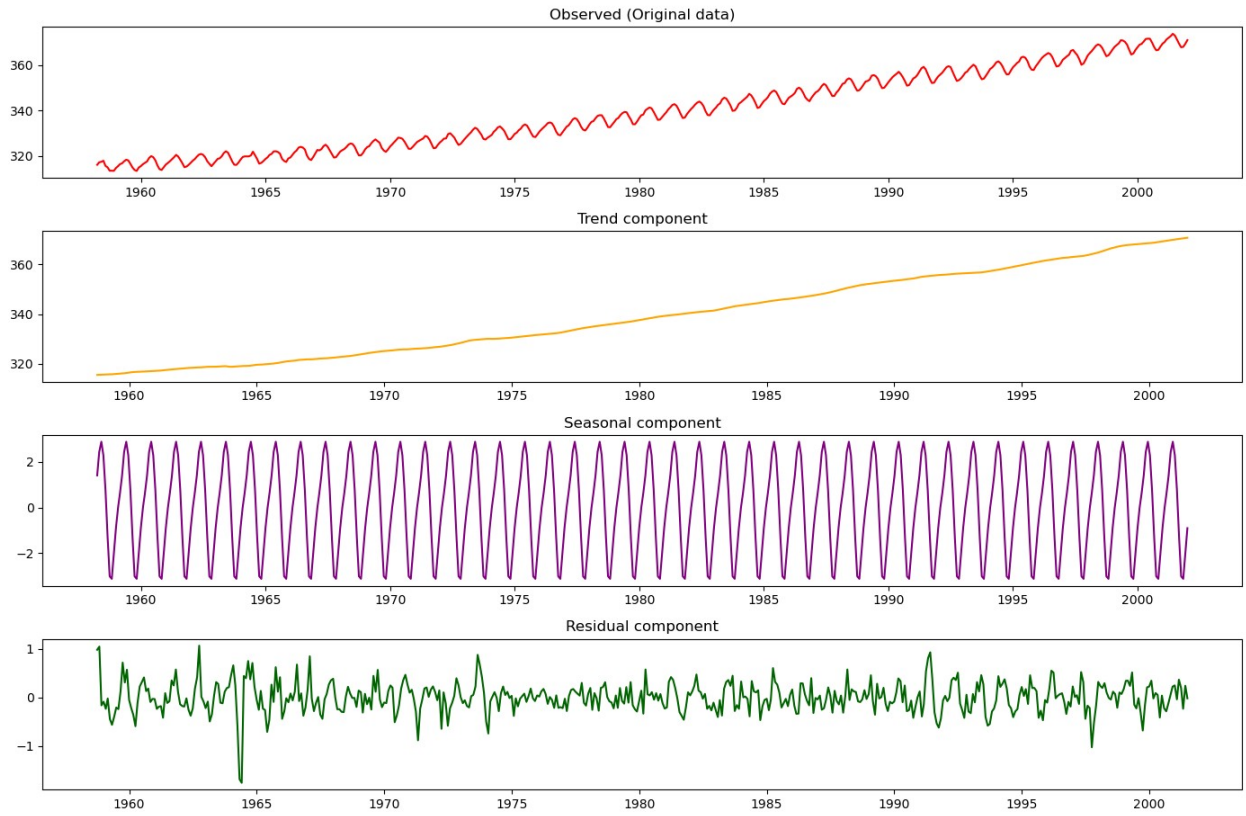


Observation: The data shows a clear upward trend with strong seasonality. This is because during spring and summer seaosons the plants intake co2 for photosynthesis causing dip in co2 levels, usually during the months of may to august. Whereas during winter and autumn seaosns when plants shed leaves, the photosynthesis is very low and causes a rise in co2 levels,usually in the months of september to april.

Decompose the Time Series

```
''' Time series decomposition is the process of breaking a time series
into its three main components:
Trend – the long-term direction of the data (e.g., upward slope)
Seasonality – regular repeating patterns (e.g., yearly CO2 cycles)
Residual – the noise or randomness that can't be explained by trend or
seasonality'''

# we use additive model as the seasonal flutuations are constant
decomposition = seasonal_decompose(data["co2"],model = "additive" )
```

```
# extract the components
observed = decomposition.observed
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

plt.figure(figsize=(14,10))

# observed
plt.subplot(411) # 411 specifies the positon of the plot in the stack
of plots. 4 rows 1 column and 1 is the index position
plt.plot(observed,color= "red")
plt.title("Observed (Original data)",fontsize = 12)

# trend
plt.subplot(412)
plt.plot(trend,color = "orange")
plt.title("Trend component",fontsize = 12)

# seasonal
plt.subplot(413)
plt.plot(seasonal,color = "purple")
plt.title("Seasonal component" , fontsize = 12)

# residual
plt.subplot(414)
plt.plot(residual,color = "darkgreen")
plt.title("Residual component", fontsize = 12)

plt.suptitle("Decomposition of CO2 time series",fontsize = 16 )
plt.tight_layout(rect=[0, 0, 1, 0.96]) #to leave space for the title
plt.show()
```



Train-Test Split (Last 48 Months for Test Set)

```
train = data.iloc[ : -48] # all rows expect the last 48 rows
test = data.iloc[ -48 : ] # the last 48 rows
```

Exponential Smoothing Models and Forecasting

```
def evaluate_model(train,test,model_type,**params): # Flexible keyword arguments (like smoothing_level, etc.) passed to .fit()
    ''' trend='add': Adds a linear trend (used for double and triple ES)
seasonal='add': Adds a seasonal component (only in triple ES)
seasonal_periods=12: Tells the model that seasonality repeats every 12 months (1 year)'''
    model = ExponentialSmoothing(train["co2"],
                                 trend = "add" if model_type in ["double","triple"] else None,
                                 seasonal = "add" if model_type == "triple" else None,
                                 seasonal_periods = 12 if model_type == "triple" else None)
    fit = model.fit(**params) # **params lets you pass smoothing parameters like: smoothing_level, smoothing_trend, smoothing_seasonal, etc.
```

```
    forecast = fit.forecast(48) # Predicts the next 48 time steps
(same length as the test set) This is your model's output for
comparison
    forecast = pd.Series(forecast, index=test.index)  # Convert
forecast to a pandas Series with same index as test

    mae = mean_absolute_error(test["co2"],forecast)
    mse = mean_squared_error(test["co2"],forecast)
    ''' mae: How far off your predictions were, on average
mse: squaring the errors (more sensitive to large errors)
forecast: The predicted values for 48 months
fit: The full fitted model object '''
    return mae, mse, forecast, fit
```

Train Models (Single, Double, Triple)

```
# create an empty dictionary to store the values
results = {}
''' smoothing_level = 0.2 (α, for level)
smoothing_slope = 0.2 (β, for trend)
smoothing_seasonal = 0.2 (γ, for seasonality)'''
results["single"] = evaluate_model(train,test,"single",
smoothing_level= .2)
results["double"] = evaluate_model(train,test,"double" ,
smoothing_level=0.2, smoothing_slope=0.2)
results["triple"] = evaluate_model(train,test, "triple",
smoothing_level = 0.2,smoothing_slope = 0.2,smoothing_seasonal = 0.2)
```

```
C:\Users\gaaya\AppData\Local\Temp\ipykernel_18660\1572493817.py:9:
FutureWarning: the 'smoothing_slope' keyword is deprecated, use
'smoothing_trend' instead.
  fit = model.fit(**params) # **params lets you pass smoothing
parameters like: smoothing_level, smoothing_trend, smoothing_seasonal,
etc.
C:\Users\gaaya\AppData\Local\Temp\ipykernel_18660\1572493817.py:9:
FutureWarning: the 'smoothing_slope' keyword is deprecated, use
'smoothing_trend' instead.
  fit = model.fit(**params) # **params lets you pass smoothing
parameters like: smoothing_level, smoothing_trend, smoothing_seasonal,
etc.
```

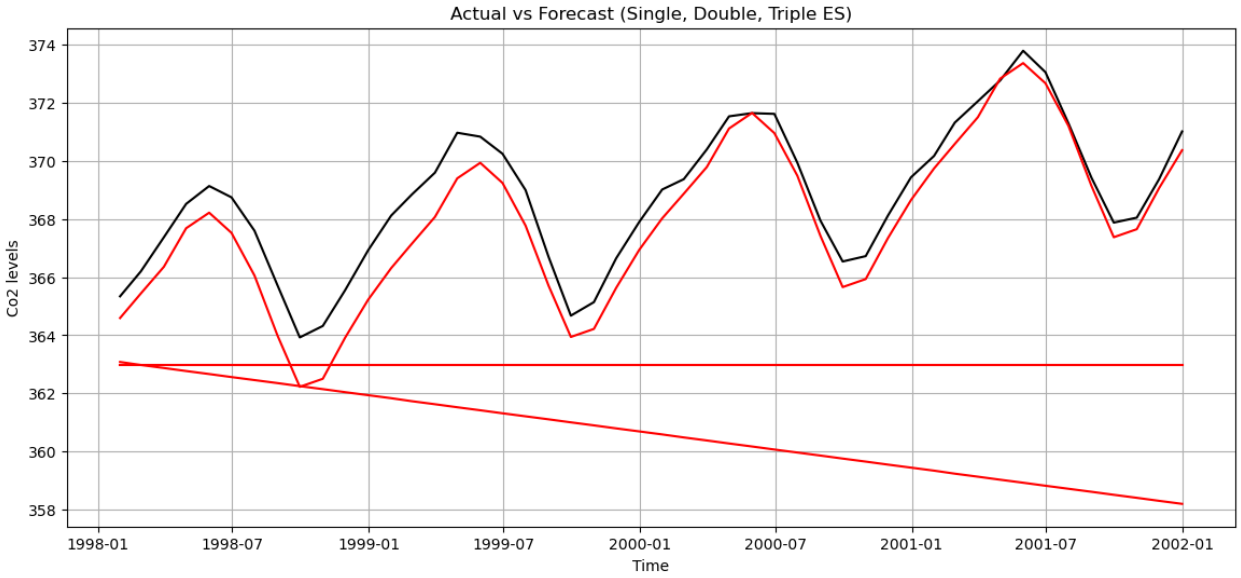
Compare MAE & MSE

```
for key in results:
    mae,mse,_,_ = results[key] # _ placeholder to ignore forecast and
fit for printing
    print(f"{key.capitalize()}: Exponential Smoothing")
    print(f"MAE: {mae:.3f}")
    print(f"MSE: {mse:.3f}\n")
```

```
Single: Exponential Smoothing
MAE: 5.780
MSE: 39.072

Double: Exponential Smoothing
MAE: 8.128
MSE: 77.501

Triple: Exponential Smoothing
MAE: 0.879
MSE: 1.014
```

Plot Actual vs Forecast

```
plt.figure(figsize=(14,6))
plt.plot(test.index,test["co2"],label = "Actual", color = "black")
for key in results:
    _,_,forecast,_ = results[key]
    plt.plot(forecast.index,forecast,label = f"{key.capitalize()}
Forecast" ,color = "red")
plt.legend
plt.title("Actual vs Forecast (Single, Double, Triple ES)")
plt.xlabel("Time")
plt.ylabel("Co2 levels")
plt.grid(True)
plt.show()
```



Final Forecast for Next 6 Months (Using Best Model)

```
best_model_type = min(results,key = lambda x: results[x][0])
#results[x][0] extracts the MAE for each model.
```

```
print(f"Best Model: {best_model_type.capitalize()} Exponential
Smoothing")
```

```
Best Model: Triple Exponential Smoothing
```

```
# refit the best model on the whole dataset
full_model = ExponentialSmoothing(data["co2"],
                                  trend= "add" if best_model_type in
["double" ,"triple"] else None,
                                  seasonal= "add" if best_model_type
== "triple" else None,
                                  seasonal_periods=12 if
best_model_type == "triple" else None)

fit_full = full_model.fit(
    smoothing_level=0.2,
    smoothing_slope=0.2 if best_model_type != 'single' else None,
    smoothing_seasonal=0.2 if best_model_type == 'triple' else None)
future_forecast = fit_full.forecast(6) # this line forecasts the next
6 future time steps (6 months) from the full model.
print("Forecast for next 6 months:\n", future_forecast)
```

```
Forecast for next 6 months:
 2002-01-31    371.809534
2002-02-28    372.604857
2002-03-31    373.488202
2002-04-30    374.637687
2002-05-31    375.145610
2002-06-30    374.609691
Freq: ME, dtype: float64

C:\Users\gaaya\AppData\Local\Temp\ipykernel_18660\1316751825.py:7:
FutureWarning: the 'smoothing_slope' keyword is deprecated, use
'smoothing_trend' instead.
  fit_full = full_model.fit(
```

Conclusion:

Based on the evaluation using MAE and MSE, the best-performing model is likely to be Triple Exponential Smoothing because

the CO2 data has trend + seasonality. This model is ideal for such patterns and provides the most accurate forecasting.