Aim - Implement time series forecasting for Healthcare diagnosis.

Theory -

What is Time Series Forecasting?

Time series forecasting involves analyzing sequential data points collected over time to identify patterns, trends, and future values. It is widely used in various fields such as finance, healthcare, and weather prediction. The goal is to understand past behaviors and use them to make informed predictions.

Dataset Description

The dataset used for analysis contains healthcare-related data, specifically heart-related parameters such as cholesterol levels (chol), blood pressure (trestbps), and other vital indicators. The dataset is structured in a time series format, where each row represents a specific day's observation.

Decomposition Techniques

Time series data often consists of different components that influence the values over time. Decomposition helps separate these components for better analysis:

1. **Trend** – The overall upward or downward movement in the data over time.
2. **Seasonality** – Recurring patterns at regular intervals (e.g., daily, monthly, yearly).
3. **Residual (Noise)** – Irregular, random fluctuations that cannot be explained by trends or seasonality.

Decomposition separates a time series into its components:

$$Y_t = T_t + S_t + R_t \quad \text{(Additive Model)}$$

$$Y_t = T_t \times S_t \times R_t \quad \text{(Multiplicative Model)}$$

Where:

- Y_t = Observed value at time t
- T_t = Trend component
- S_t = Seasonal component
- R_t = Residual (noise)

Smoothing Techniques

Smoothing techniques help reduce noise in time series data to highlight patterns:

1. **Moving Average (MA)** – Calculates the average of the last ‘n’ observations, reducing short-term fluctuations.

$$MA_t = \frac{1}{n}\sum_{i=0}^{n-1} Y_{t-i}$$

Where:

- n = Window size
- Y_t = Observed value at time t

2. **Weighted Moving Average (WMA)** – Assigns different weights to past observations, giving more importance to recent values.

$$WMA_t = \frac{\sum_{i=0}^{n-1} w_i Y_{t-i}}{\sum_{i=0}^{n-1} w_i}$$

Where:

- w_i = Weight assigned to each past value
- Y_{t-i} = Observed value at past time $t - i$

3. **Exponential Moving Average (EMA)** – Similar to WMA but applies exponentially decreasing weights, making it more responsive to recent changes.

$$EMA_t = \alpha Y_t + (1 - \alpha) EMA_{t-1}$$

Where:

- $\alpha = \frac{2}{n+1}$ (Smoothing factor, where n is the window size)
- EMA_t = Exponential moving average at time t
- Y_t = Observed value at time t
- EMA_{t-1} = Previous EMA value

Interpretation of Results

Decomposition Table Interpretation (Original vs. Seasonality)

- The "**Original**" column represents the actual recorded values.
- The "**Seasonality**" column captures recurring patterns in the data.
- For example, on **2023-01-07**, the original value is **294**, and the seasonal component is **31.98**, meaning this day's value was slightly elevated due to seasonal factors.
- On **2023-01-05**, the seasonality value is **-30.92**, indicating a lower-than-usual value for that period.

Smoothing Table Interpretation (Original vs. Exp Moving Average)

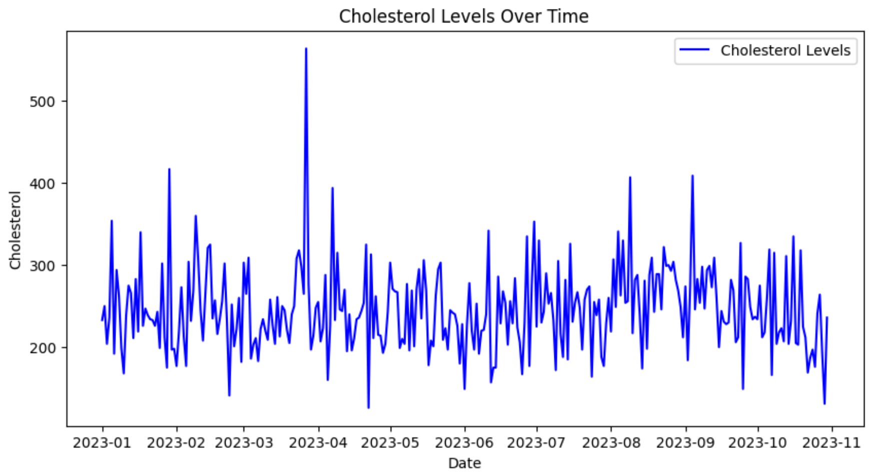
- The "**Exp_Moving_Avg**" column shows a smoothed version of the original data.
- The first value remains the same as the original since no past data exists to smooth it.
- The values gradually adapt, with recent values impacting the smoothed results more than older ones.
- For instance, on **2023-01-06**, the original value drops to **192**, but the EMA remains at **242.40**, indicating a gradual response to changes rather than sudden fluctuations.
- This smooth transition makes EMA useful for identifying long-term trends while filtering out short-term noise.

Implementation -

```
!pip install pandas numpy matplotlib scikit-learn
statsmodels tensorflow keras --quiet
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import
seasonal_decompose
```

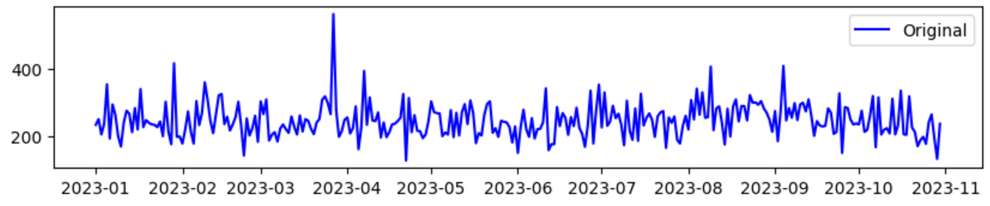
```
# Load dataset
df =
pd.read_csv("/content/Healthcare_Diagnosis_Heart.csv")
# Create a time index (assuming daily patient visits)
df['date'] = pd.date_range(start='2023-01-01',
periods=len(df), freq='D')
df.set_index('date', inplace=True)
```

```
# Select a column for time series analysis (e.g.,
cholesterol 'chol')
series = df['chol']
# Plot original series
plt.figure(figsize=(10, 5))
plt.plot(series, label='Cholesterol Levels', color='blue')
plt.xlabel('Date')
plt.ylabel('Cholesterol')
plt.title('Cholesterol Levels Over Time')
plt.legend()
plt.show()
```



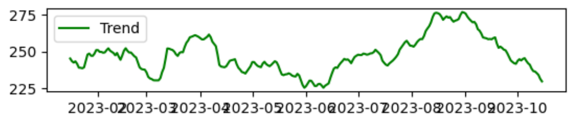
Decomposition

```
# Perform decomposition
decomposition = seasonal_decompose(series,
model='additive', period=30)
# Plot decomposition components
plt.figure(figsize=(10, 8))
plt.subplot(411)
plt.plot(series, label='Original', color='blue')
plt.legend()
```



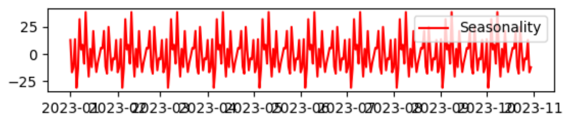
Component 1 - Trend

```
plt.subplot(412)
plt.plot(decomposition.trend, label='Trend', color='green')
plt.legend()
```



Component 2 - Seasonality

```
plt.subplot(413)
plt.plot(decomposition.seasonal, label='Seasonality',
color='red')
plt.legend()
```

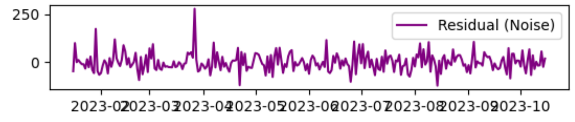


```
# Create a DataFrame to display decomposition results
decomposition_df = pd.DataFrame({
    'Original': series,
    'Seasonality': decomposition.seasonal,
})
# Display first 10 rows of decomposition results
print(decomposition_df.head(10))
```

```
            Original  Seasonality
date
2023-01-01       233    12.913747
2023-01-02       250   -16.782549
2023-01-03       204   -12.097364
2023-01-04       236    13.617451
2023-01-05       354   -30.926994
2023-01-06       192   -13.473290
2023-01-07       294    31.982265
2023-01-08       263     4.160043
2023-01-09       199     8.441525
2023-01-10       168    -8.778846
```

Component 3 - Noise

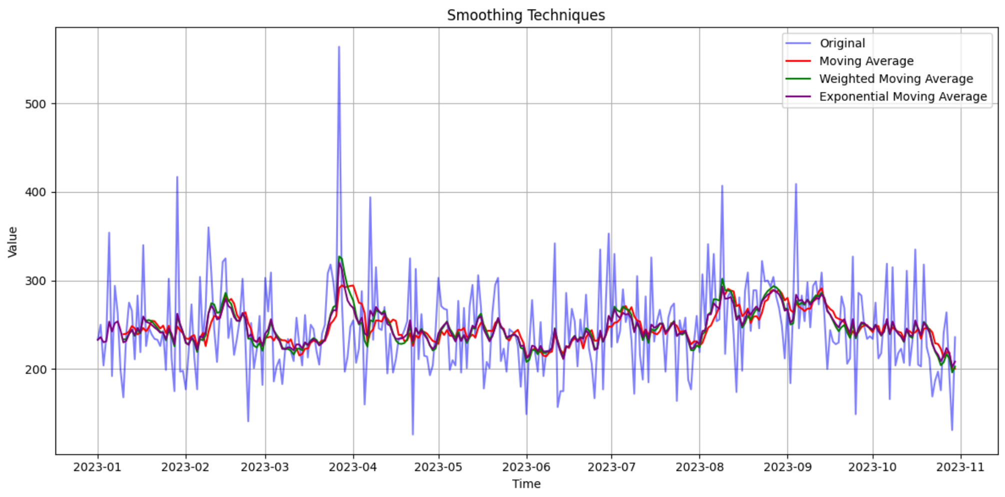
```
plt.subplot(414)
plt.plot(decomposition.resid, label='Residual (Noise)',
color='purple')
plt.legend()
```



Smoothing

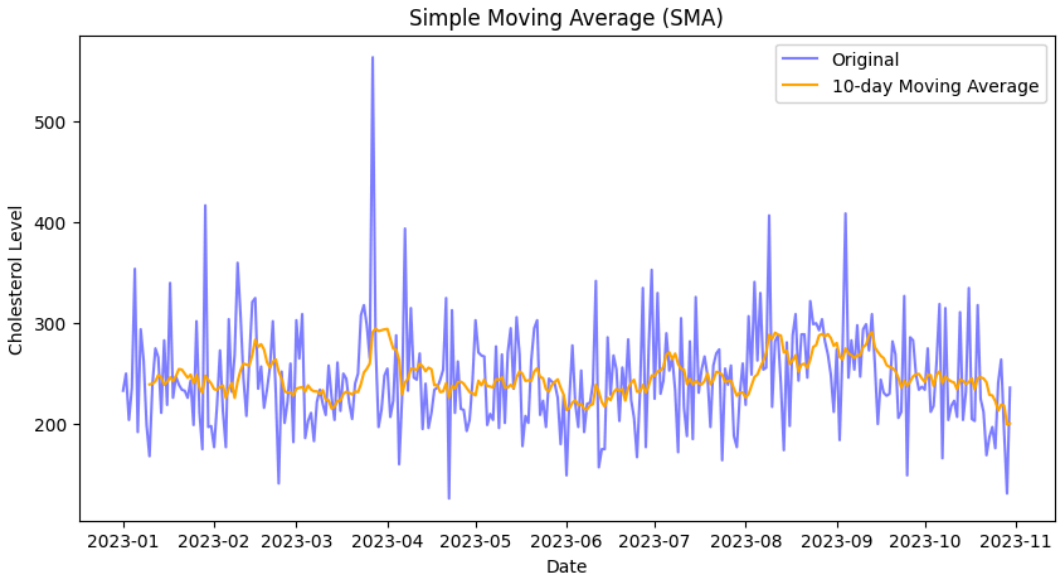
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# Define window size
window_size = 10
# Moving Average (Simple)
moving_avg =
series.rolling(window=window_size).mean()
# Weighted Moving Average
weights = np.arange(1, window_size + 1)
weighted_moving_avg =
series.rolling(window=window_size).apply(lambda x:
np.dot(x, weights) / weights.sum(), raw=True)
# Exponential Moving Average
exp_moving_avg = series.ewm(span=window_size,
adjust=False).mean()
# Create a DataFrame to display smoothing results
smoothing_df = pd.DataFrame({
  'Original': series,
  'Exp_Moving_Avg': exp_moving_avg
})
# Display first 10 rows
print(smoothing_df.head(10))
# Plot smoothing results
plt.figure(figsize=(12, 6))
plt.plot(series, label='Original', color='blue', alpha=0.5)
plt.plot(moving_avg, label='Moving Average', color='red')
plt.plot(weighted_moving_avg, label='Weighted Moving
Average', color='green')
plt.plot(exp_moving_avg, label='Exponential Moving
Average', color='purple')
plt.legend()
plt.title('Smoothing Techniques')
plt.xlabel('Time')
plt.ylabel('Value')
plt.grid()
plt.tight_layout()
plt.show()
```

```
            Original  Exp_Moving_Avg
date
2023-01-01       233      233.000000
2023-01-02       250      236.090909
2023-01-03       204      230.256198
2023-01-04       236      231.300526
2023-01-05       354      253.609521
2023-01-06       192      242.407790
2023-01-07       294      251.788192
2023-01-08       263      253.826702
2023-01-09       199      243.858211
2023-01-10       168      230.065809
```



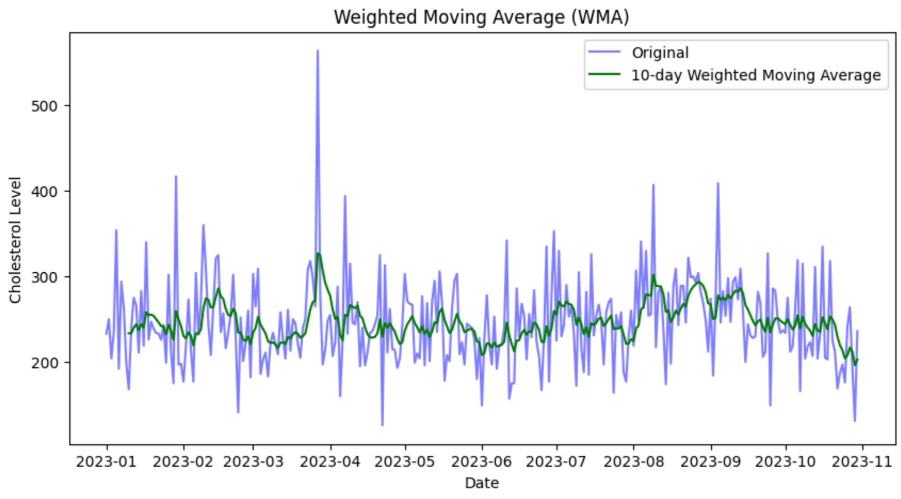
Type 1 - Simple Moving Average

```
window_size = 10  # Define window size
df['SMA'] = series.rolling(window=window_size).mean()
# Plot SMA
plt.figure(figsize=(10,5))
plt.plot(series, label="Original", color='blue', alpha=0.5)
plt.plot(df['SMA'], label=f'{window_size}-day Moving Average', color='orange')
plt.xlabel("Date")
plt.ylabel("Cholesterol Level")
plt.title("Simple Moving Average (SMA)")
plt.legend()
plt.show()
```



Type 2 - Weighted Moving Average

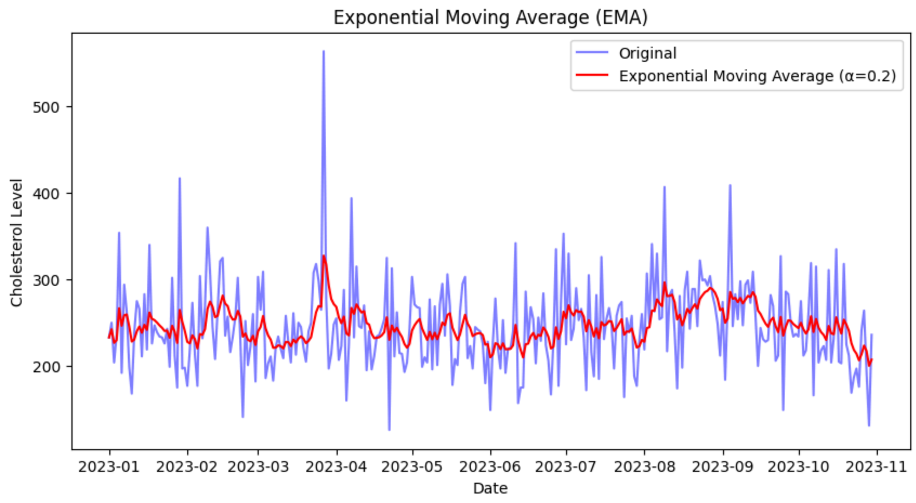
```
weights = np.arange(1, window_size + 1)  # Increasing weights
df['WMA'] = series.rolling(window_size).apply(lambda x: np.dot(x, weights) / weights.sum(), raw=True)
# Plot WMA
plt.figure(figsize=(10,5))
plt.plot(series, label="Original", color='blue', alpha=0.5)
plt.plot(df['WMA'], label=f'{window_size}-day Weighted Moving Average', color='green')
plt.xlabel("Date")
plt.ylabel("Cholesterol Level")
plt.title("Weighted Moving Average (WMA)")
plt.legend()
plt.show()
```



Type 3 - Exponential Moving Average

```
alpha = 0.2  # Smoothing factor
```

```
df['EMA'] = series.ewm(alpha=alpha).mean()
# Plot EMA
plt.figure(figsize=(10,5))
plt.plot(series, label="Original", color='blue', alpha=0.5)
plt.plot(df['EMA'], label=f'Exponential Moving Average
(α={alpha})', color='red')
plt.xlabel("Date")
plt.ylabel("Cholesterol Level")
plt.title("Exponential Moving Average (EMA)")
plt.legend()
plt.show()
```



Conclusion -

Time series forecasting is crucial for analyzing trends and making future predictions in healthcare. Decomposition techniques help break down time series data into trend, seasonality, and residual components, providing insights into underlying patterns. Smoothing techniques, such as moving averages and exponential smoothing, reduce noise and highlight trends, improving forecasting accuracy. By applying these methods to healthcare datasets, we can enhance diagnosis and decision-making, ultimately leading to better patient outcomes.