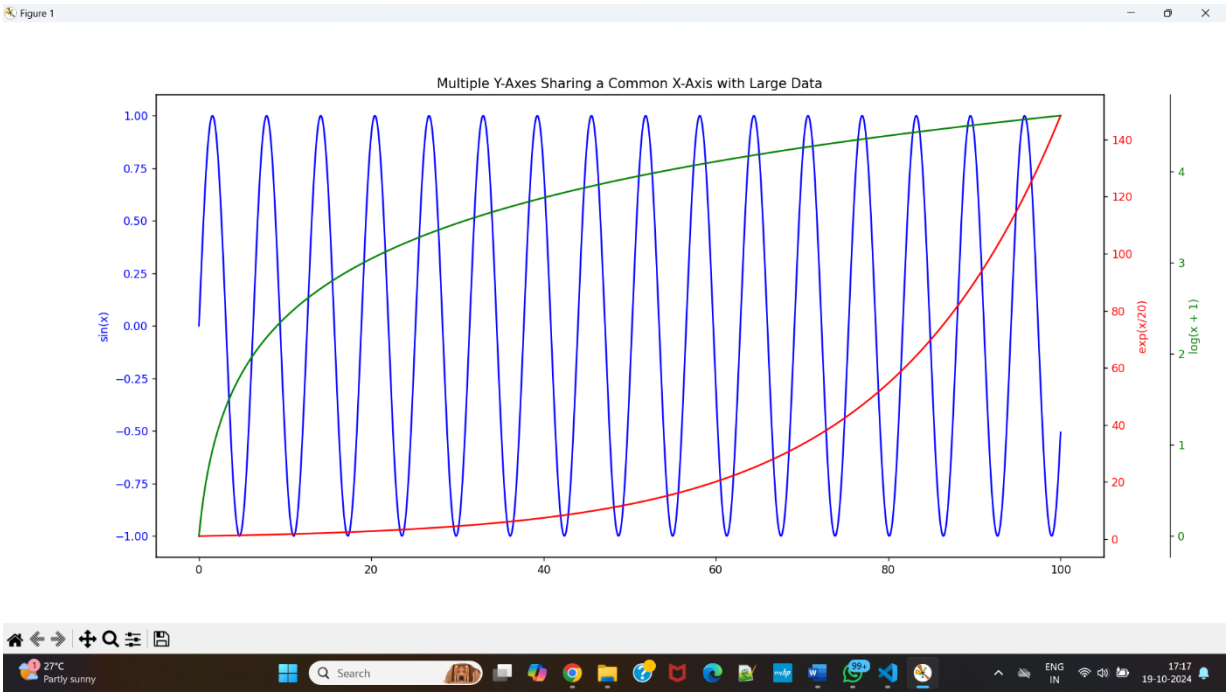
1. Task Description

Task : Multiple Y-Axes Sharing a Common X-Axis Plo

Create a plot with multiple y-axes sharing a common x-axis.

2. Task Output Screenshot



3. Algorithms Used In Task :

1. Data Generation Algorithms

- **Sampling**: Generating sample data points (e.g., using numpy.linspace() to create evenly spaced values). This can be seen as a simple algorithm to sample data over a specific range.
- **Mathematical Functions**: Using mathematical algorithms to compute values. For example:
    - **Sine Function**: $y=\sin(x)$
    - **Exponential Function**: $y=e^{(x/20)}$
    - **Logarithm Function**: $y=\log(x+1)$

2. Plotting Algorithms

- **Line Plotting**: The algorithm used to connect data points with lines. This involves:
    - **Linear Interpolation**: Drawing straight lines between calculated data points.
    - **Color Mapping**: Assigning different colors to each line for differentiation.
- **Axis Scaling**: Adjusting the y-axis scales for each function so that all data can be visualized appropriately on a common x-axis.
- **Legend and Labels**: Algorithms to place and format legends and axis labels properly to make the plot readable.

3. Visualization Techniques

- **Overlaying Plots**: The algorithm that allows multiple plots to share the same x-axis but have separate y-axes, which involves:
    - **Creating Twin Axes**: Using functions like twinx() in Matplotlib to create secondary axes.
- **Customizing Axes**: Algorithms for adjusting tick marks, colors, and positions of axes, such as:
    - **Setting Axis Limits**: Adjusting the limits of each y-axis based on the data range.
    - **Adjusting Position**: Moving axes outward to avoid overlapping, as done with spines['right'].set_position(('outward', 60)).

4. Interactivity and Rendering

- **Event Handling**: Managing user interactions with the plot (e.g., zooming or panning) using built-in Matplotlib features.
- **Rendering Algorithms**: Efficient algorithms to render plots quickly on the screen.

5. Performance Optimization

- **Data Aggregation**: If dealing with larger datasets, techniques to aggregate or downsample data can be used to improve performance while plotting.
- **Vectorization**: Using NumPy for vectorized operations rather than Python loops, which speeds up calculations significantly.