**Custom colormap for a Matplotlib plot**

**Aim:** Implement a custom colormap for a Matplotlib plot.

### **Steps to Implement a Custom Colormap**

1. **Using** LinearSegmentedColormap: This method creates a smooth gradient between defined colors.
2. **Using** ListedColormap: This method specifies a list of exact colors without interpolation.

### Example 1: Custom Colormap using LinearSegmentedColormap

In this example, we create a smooth gradient from red, to green, to blue.

import matplotlib.pyplot as plt

import numpy as np

from matplotlib.colors import LinearSegmentedColormap

# Define the custom colormap

colors = [(1, 0, 0), (0, 1, 0), (0, 0, 1)] # Red, Green, Blue

n\_bins = 100 # Number of color bins

cmap\_name = 'my\_custom\_colormap'

# Create the colormap

custom\_cmap = LinearSegmentedColormap.from\_list(cmap\_name, colors, N=n\_bins)

# Generate some data for the plot

x = np.linspace(0, 10, 100)

y = np.sin(x)

# Create a scatter plot with the custom colormap

plt.figure(figsize=(8, 6))

sc = plt.scatter(x, y, c=y, cmap=custom\_cmap, s=100)

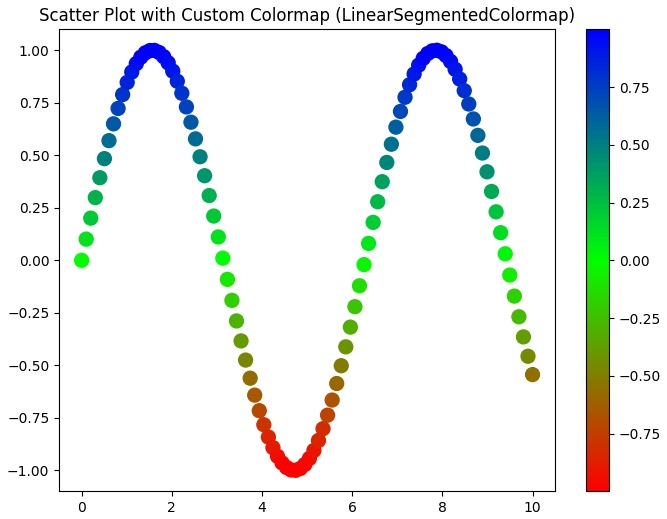
# Add a colorbar to show the mapping of values to colors

plt.colorbar(sc)

# Show the plot

plt.title('Scatter Plot with Custom Colormap (LinearSegmentedColormap)')

plt.show()



### Explanation:

1. **Colors**: We define a list of tuples representing RGB values: red (1, 0, 0), green (0, 1, 0), and blue (0, 0, 1). These colors will be interpolated to create the colormap.
2. LinearSegmentedColormap.from\_list: Creates a colormap that smoothly transitions between the specified colors. The parameter N=n\_bins defines how many color bins to use.
3. scatter **plot**: We use the colormap in the cmap parameter of the scatter plot, and the c parameter is set to the y-values to color the points accordingly.

### Example 2: Custom Colormap using ListedColormap

This example uses discrete colors without blending, creating distinct steps in the colormap.

from matplotlib.colors import ListedColormap

# Define a list of discrete colors

discrete\_colors = ['#FF0000', '#00FF00', '#0000FF', '#FFFF00'] # Red, Green, Blue, Yellow

# Create a ListedColormap

custom\_cmap = ListedColormap(discrete\_colors)

# Generate some data for the plot

x = np.linspace(0, 10, 100)

y = np.sin(x)

# Create a scatter plot with the custom colormap

plt.figure(figsize=(8, 6))

sc = plt.scatter(x, y, c=y, cmap=custom\_cmap, s=100)

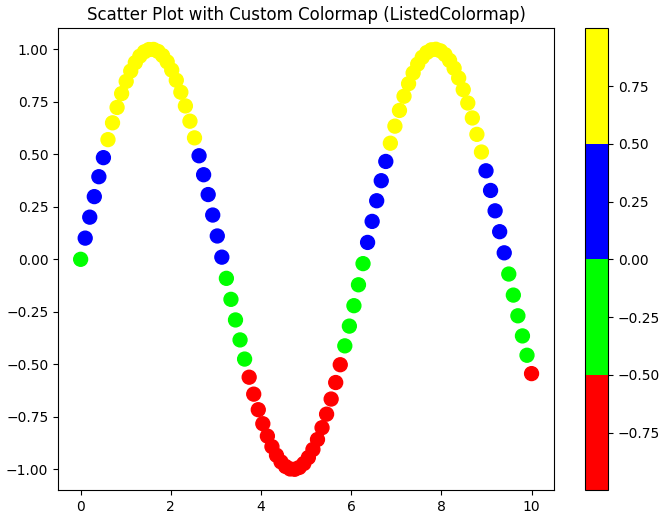
# Add a colorbar

plt.colorbar(sc)

# Show the plot

plt.title('Scatter Plot with Custom Colormap (ListedColormap)')

plt.show()



### Explanation:

1. ListedColormap: We define a list of hex colors to use for the colormap. This method applies these colors directly without blending.
2. **Color Mapping**: The colors are mapped to the y-values in the scatter plot, and you’ll see sharp transitions between the discrete colors.

### Key Differences:

* LinearSegmentedColormap: Used for smooth gradients and continuous data.
* ListedColormap: Best for discrete data or when you want distinct, sharp color transitions.