In [ ]:

```python
from typing import Callable
from IPython.display import display, HTML
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors

def _color_series(
    values: pd.Series,
    colormap: plt.cm,
    norm: mcolors.Normalize | Callable | None = None,
    na_color: str | None = None,
) -> pd.Series:
    """
    Color a Series of numeric values using a Matplotlib colormap and norm

    Parameters:
    values : pd.Series
        The Series to color.
    colormap : plt.cm
        The Matplotlib colormap to use.
    norm : mcolors.Normalize | Callable | None
        A matplotlib normalization object (e.g., Normalize, LogNorm) or a
        normalized valeus within the range [0, 1]. If None, defaults to l
        between the minimum and maximum values.
    na_color : str, optional
        The color to use for NaN values.

    Returns:
    pd.Series
        A new Series with hex color strings.
    """
    # Normalize the data
    if norm is None:  # Defaults to linear normalization between min and
        norm = mcolors.Normalize(values.min(), values.max())
    normalized = norm(values.values)
    colors = colormap(normalized)

    # Convert RGBA colors to hex, handling NaN values
    hex_colors = [mcolors.to_hex(color) if not np.isnan(color[0]) else na

    return pd.Series(hex_colors, index=values.index)


def display_colored_series(
    values: pd.Series,
    colormap: plt.cm,
    norm: mcolors.Normalize | Callable | None = None,
    na_color: str | None = None,
) -> None:
    """
    For demonstration purposes using _color_series.
    """

    colored = _color_series(values, colormap, norm, na_color)
    display_df = pd.DataFrame({'values': values, 'colors': colored})
    return display_df.style.map(lambda x: f'background-color: {x}', subse
```

```
def display_two_colorings_side_by_side(
    values: pd.Series,
    colormap: plt.cm,
    norm: mcolors.Normalize | Callable | None = None,
    alternative_norm: mcolors.Normalize | Callable | None = None,
    na_color: str | None = None,
    norm_caption: str = 'Non-Standard Normalization',
    alternative_caption: str = 'Standard Min-Max scaling',
) -> None:
    """
    For demonstration purposes using _color_series.
    """
    non_standard_norm_stlyed = display_colored_series(values, colormap, n
    standard_norm_stlyed = display_colored_series(values, colormap, norm=

    # Convert styled DataFrames to HTML
    html1 = non_standard_norm_stlyed.set_caption(norm_caption).to_html()
    html2 = standard_norm_stlyed.set_caption(alternative_caption).to_html

    # Combine the HTML strings with div and display inline
    combined_html = f'<div style="display: inline-block;">{html1}</div><d

    # Display side by side
    display(HTML(combined_html))
```

Example 1: Linear normalization between min and max

Example 1.1

Simply use the min-max values to scale to the interval [0,1]

In [ ]:
```
values = pd.Series([np.nan, 1,2,3,4])
display_two_colorings_side_by_side(values, plt.cm.viridis)
```

Non-Standard Normalization

	values	colors
0	nan	
1	1.000000	#440154
2	2.000000	#31688e
3	3.000000	#35b779
4	4.000000	#fde725

Standard Min-Max scaling

	values	colors
0	nan	
1	1.000000	#440154
2	2.000000	#31688e
3	3.000000	#35b779
4	4.000000	#fde725

Example 1.2

Set vmin and vmax

In [ ]:
```
values = pd.Series([np.nan, 1,2,3,4])
display_two_colorings_side_by_side(values, plt.cm.viridis, norm=mcolors.N
```

Normalize with vmin=2, vmax=3.5

	values	colors
0	nan	
1	1.000000	#440154
2	2.000000	#440154
3	3.000000	#35b779
4	4.000000	#fde725

Standard Min-Max scaling

	values	colors
0	nan	
1	1.000000	#440154
2	2.000000	#31688e
3	3.000000	#35b779
4	4.000000	#fde725

Example 2: Centered normalization with a diverging colormap

When using a diverging color map, it's ideal to be able to control which value maps to the center point. E.g. 0 if +ve and -ve values have difference significance.

In [ ]:
```
values = pd.Series(range(-10, 25, 5))
display_two_colorings_side_by_side(values, plt.cm.bwr, mcolors.CenteredNo
```

Centered Normalization with center=0

	values	colors
0	-10	#8080ff
1	-5	#c0c0ff
2	0	#fffefe
3	5	#ffbebe
4	10	#ff7e7e
5	15	#ff3e3e
6	20	#ff0000

Standard Min-Max scaling

	values	colors
0	-10	#0000ff
1	-5	#5454ff
2	0	#aaaaff
3	5	#fffefe
4	10	#ffaaaa
5	15	#ff5454
6	20	#ff0000

Example 3: TwoSlopeNorm

These are useful when the scale should be different either side of a central value. E.g. -ve values range from 0 to -100 but positive values have much larger magnitude

ranging from 0 - 1000. The example below shows how this can improve over a simpler `CenteredNorm`

In [ ]:
```
values = pd.Series(list(range(-100, 0, 20)) + list(range(0, 1000, 200)))
display_two_colorings_side_by_side(values, plt.cm.bwr, mcolors.TwoSlopeNo
                                   norm_caption='Centered Normalization w
```

Centered Normalization with center=0

	values	colors
0	-100	#0000ff
1	-80	#3232ff
2	-60	#6666ff
3	-40	#9898ff
4	-20	#ccccff
5	0	#fffefe
6	200	#ffbebe
7	400	#ff7e7e
8	600	#ff3e3e
9	800	#ff0000

Centered Normalization with center=0

	values	colors
0	-100	#e0e0ff
1	-80	#e6e6ff
2	-60	#ececff
3	-40	#f2f2ff
4	-20	#f8f8ff
5	0	#fffefe
6	200	#ffbebe
7	400	#ff7e7e
8	600	#ff3e3e
9	800	#ff0000

Example 4

Log scaling

In [ ]:
```
values = pd.Series([10 ** i for i in range(-2, 6)])
display_two_colorings_side_by_side(values, plt.cm.bwr, mcolors.LogNorm(),
```

Centered Normalization with center=0

	values	colors
0	0.010000	#0000ff
1	0.100000	#4848ff
2	1.000000	#9292ff
3	10.000000	#dadaff
4	100.000000	#ffdada
5	1000.000000	#ff9292
6	10000.000000	#ff4848
7	100000.000000	#ff0000

Standard Min-Max scaling

	values	colors
0	0.010000	#0000ff
1	0.100000	#0000ff
2	1.000000	#0000ff
3	10.000000	#0000ff
4	100.000000	#0000ff
5	1000.000000	#0404ff
6	10000.000000	#3232ff
7	100000.000000	#ff0000