In [1]:
```python
import numpy as np
import pandas as pd
print(pd.__version__)
```
```
2.0.2
```

Style Overview

Pandas' Style functionality allows for highlighting the logic and features of the data, making it more visually striking and clear.

For example, representing a currency value of 25.06 as "$25.06" provides additional information about the currency, and using styles can enhance the presentation of the data. Similarly, representing 0.12 as "12%" makes the data more intuitive, clear, and simple to understand.

The fundamental concept of styling in Pandas is that it allows users to modify the display format of data without affecting the underlying data itself. In other words, the data type remains unchanged, allowing for the use of various mathematical, date, and string functions provided by Pandas for data processing.

Pandas' Style functionality also includes advanced styling options, such as adding colors and bar charts as visual elements.

Main features of Pandas' Style functionality

- Value formatting: Formatting numerical values with features such as thousands separators, decimal places, currency symbols, date formats, percentages, etc.
- Highlighting data: Applying styles to specific rows, columns, or values (e.g., minimum and maximum values) using features like font size, colors, backgrounds, etc.
- Visualizing data relationships: Using color shades or gradients to represent variations in data magnitude.
- Miniature bar charts: Representing proportions or percentages within a small grid cell using color scales.
- Trend representation: Similar to sparklines in Excel, representing trends or changes in data over time with miniature trend charts.

The key difference between styles and visualizations is that visualizations generally focus on presenting data without specific details, while styles decorate the data while retaining its specific content to enhance readability. However, there can be cases where these two approaches overlap and are used together.

Style Object

The Style object in Pandas DataFrame is used to declare the styling of data. Styling is achieved using CSS (Cascading Style Sheets).

All the styling functionalities are available under the `df.style object`. It allows us to display all the data without any specific styling in the notebook.

Style Exporting

The DataFrame with styling can be exported using `df.style.to_excel()` method. The exported Excel file will retain the styling applied to the DataFrame.

In [2]:
```python
df = pd.DataFrame({'Name':['James', 'Messi', 'Durant', 'Irvine', 'Harden'],
                  'Team' : list('ABCDE'),
                  'Q1': np.random.randint(60, 80, 5),
                'Q2': np.random.randint(70, 90, 5),
                   'Q3': np.random.randint(60, 70, 5),
                   'Q4': np.random.randint(30, 80, 5)
})
df
```

Out[2]:

	Name	Team	Q1	Q2	Q3	Q4
0	James	A	79	89	68	62
1	Messi	B	64	76	69	66
2	Durant	C	76	76	63	68
3	Irvine	D	74	76	60	63
4	Harden	E	66	78	60	46

Build-in style function

- `highlight_max()` : Highlight the maximum value in each column.
- `highlight_min()` : Highlight the minimum value in each column.
- `highlight_greater()` : Highlight cells greater than a specified value.
- `highlight_less()` : Highlight cells less than a specified value.
- `highlight_between()` : Highlight cells within a specified range.
- `background_gradient()` : Apply a gradient background color to cells based on their values.
- `bar()` : Draw horizontal bar charts in cells.
- `color_negative()` : Apply a specific color to cells with negative values.
- `set_precision()` : Set the display precision for floating-point numbers

`highlight_null`

In [3]:
```
df.iloc[1,1] = np.NaN # Revise a null value
df.head().style.highlight_null() # Default color red
```

Out[3]:

	Name	Team	Q1	Q2	Q3	Q4
0	James	A	79	89	68	62
1	Messi	🟥 nan	64	76	69	66
2	Durant	C	76	76	63	68
3	Irvine	D	74	76	60	63
4	Harden	E	66	78	60	46

In [4]:
```
# specify the new color
df.style.highlight_null(color = 'blue')
```

Out[4]:

	Name	Team	Q1	Q2	Q3	Q4
0	James	A	79	89	68	62
1	Messi	nan	64	76	69	66
2	Durant	C	76	76	63	68
3	Irvine	D	74	76	60	63
4	Harden	E	66	78	60	46

In [5]:
```
df.head().style.highlight_null(color = '#ccc')
```

Out[5]:

	Name	Team	Q1	Q2	Q3	Q4
0	James	A	79	89	68	62
1	Messi	nan	64	76	69	66
2	Durant	C	76	76	63	68
3	Irvine	D	74	76	60	63
4	Harden	E	66	78	60	46

Highlight `max` and `min` values

In [6]:
```
df.iloc[:, 2:6].style.highlight_max()
```

Out[6]:

	Q1	Q2	Q3	Q4
0	79	89	68	62
1	64	76	69	66
2	76	76	63	68
3	74	76	60	63
4	66	78	60	46

In [7]:
```
df.iloc[:, 2:6].style.highlight_min()
```

Out[7]:

	Q1	Q2	Q3	Q4
0	79	89	68	62
1	64	76	69	66
2	76	76	63	68
3	74	76	60	63
4	66	78	60	46

In [8]:
```
# Max and min simultaneously
df.iloc[:, 2:6].style.highlight_min().highlight_max(color = 'lime')
```

Out[8]:

	Q1	Q2	Q3	Q4
0	79	89	68	62
1	64	76	69	66
2	76	76	63	68
3	74	76	60	63
4	66	78	60	46

In [9]:
```
df.iloc[:, 2:6].style.highlight_min(axis = 1).highlight_max(color = 'lime', axis = 1)
```

Out[9]:

	Q1	Q2	Q3	Q4
0	79	89	68	62
1	64	76	69	66
2	76	76	63	68
3	74	76	60	63
4	66	78	60	46

In [10]:
```
# subset
df.style.highlight_min(subset = ['Q1', 'Q2'])
```

Out[10]:

	Name	Team	Q1	Q2	Q3	Q4
0	James	A	79	89	68	62
1	Messi	nan	64	76	69	66
2	Durant	C	76	76	63	68
3	Irvine	D	74	76	60	63
4	Harden	E	66	78	60	46

`background_gradient`

To apply a gradient color scheme based on the magnitude of the values in a DataFrame, we can use the `background_gradient()` method of the Styler object in Pandas. This method allows you to specify the color map from the `Matplotlib colormap` for the gradient.

In [11]:
```
df.style.background_gradient(axis = 0)
```

Out[11]:

	Name	Team	Q1	Q2	Q3	Q4
0	James	A	79	89	68	62
1	Messi	nan	64	76	69	66
2	Durant	C	76	76	63	68
3	Irvine	D	74	76	60	63
4	Harden	E	66	78	60	46

In [12]:
```
df.style.background_gradient(low = 0.6, high = 0)
```

Out[12]:

	Name	Team	Q1	Q2	Q3	Q4
0	James	A	79	89	68	62
1	Messi	nan	64	76	69	66
2	Durant	C	76	76	63	68
3	Irvine	D	74	76	60	63
4	Harden	E	66	78	60	46

The `text_color_threshold` parameter in the `Styler.background_gradient()` method allows you to specify a threshold value for the text color, When the background color of a cell crosses this threshold, the text color will automatically switch between black and white to ensure optimal visibility.

In below example, if the background color of a cell is darker than 0.5, the text color will be switched to white for better visibility. If the background color is lighter than 0.5, the text color will be black.

In [13]:
```
df.style.background_gradient(text_color_threshold = 0.5)
```

Out[13]:

	Name	Team	Q1	Q2	Q3	Q4
0	James	A	79	89	68	62
1	Messi	nan	64	76	69	66
2	Durant	C	76	76	63	68
3	Irvine	D	74	76	60	63
4	Harden	E	66	78	60	46

The `vmin` and `vmax` parameters in the `Styler.background_gradient()` method allow you to specify the minimum and maximum values for the color mapping. The values in the DataFrame will be mapped to the corresponding colors based on this range.

In [14]:
```
df.style.background_gradient(vmin = 60, vmax = 100)
```

Out[14]:

	Name	Team	Q1	Q2	Q3	Q4
0	James	A	79	89	68	62
1	Messi	nan	64	76	69	66
2	Durant	C	76	76	63	68
3	Irvine	D	74	76	60	63
4	Harden	E	66	78	60	46

In [15]:
```
# A general case
# Chained method using styles
(df.style.background_gradient(subset=['Q1'], cmap='spring')  # Specify the colormap
.background_gradient(subset=['Q2'], vmin=60, vmax=100)  # Specify the applied value range
.background_gradient(subset=['Q3'], low=0.6, high=0) # Gradient percentage range
.background_gradient(subset=['Q4'], text_color_threshold=0.9)  # Text color threshold
)
```

Out[15]:

	Name	Team	Q1	Q2	Q3	Q4
0	James	A	79	89	68	62
1	Messi	nan	64	76	69	66
2	Durant	C	76	76	63	68
3	Irvine	D	74	76	60	63
4	Harden	E	66	78	60	46

Bar Chart

A bar chart represents the magnitude of a value in a table using horizontal bars.

In [16]:
```
df.style.bar()
```

Out[16]:

	Name	Team	Q1	Q2	Q3	Q4
0	James	A	79	89	68	62
1	Messi	nan	64	76	69	66
2	Durant	C	76	76	63	68
3	Irvine	D	74	76	60	63
4	Harden	E	66	78	60	46

In [17]:

```python
# specify the subset
df.style.bar(subset = ['Q1', 'Q2'])
```

Out[17]:

	Name	Team	Q1	Q2	Q3	Q4
0	James	A	79	89	68	62
1	Messi	nan	64	76	69	66
2	Durant	C	76	76	63	68
3	Irvine	D	74	76	60	63
4	Harden	E	66	78	60	46

In [18]:

```python
# specify the color
df.style.bar(color = 'green')
```

Out[18]:

	Name	Team	Q1	Q2	Q3	Q4
0	James	A	79	89	68	62
1	Messi	nan	64	76	69	66
2	Durant	C	76	76	63	68
3	Irvine	D	74	76	60	63
4	Harden	E	66	78	60	46

In [19]:

```python
# Horizongtal calculation
df.style.bar(axis = 1)
```

Out[19]:

	Name	Team	Q1	Q2	Q3	Q4
0	James	A	79	89	68	62
1	Messi	nan	64	76	69	66
2	Durant	C	76	76	63	68
3	Irvine	D	74	76	60	63
4	Harden	E	66	78	60	46

In [20]:

```
# Set the width of the bars to 80 pixels, you can use:
df.style.bar(width = 80)
```

Out[20]:

	Name	Team	Q1	Q2	Q3	Q4
0	James	A	79	89	68	62
1	Messi	nan	64	76	69	66
2	Durant	C	76	76	63	68
3	Irvine	D	74	76	60	63
4	Harden	E	66	78	60	46

In [21]:

```
df.style.bar(align = 'mid')
```

Out[21]:

	Name	Team	Q1	Q2	Q3	Q4
0	James	A	79	89	68	62
1	Messi	nan	64	76	69	66
2	Durant	C	76	76	63	68
3	Irvine	D	74	76	60	63
4	Harden	E	66	78	60	46

In [22]:

```
# 'left': The bars start from the minimum value.
#'zero': The bars are centered around zero (0) value.
# 'mid': The bars are centered around the midpoint
#  between the minimum and maximum values. Negative (positive) values will have zero (0) on the right (left) side.
```

```
df.style.bar(align = 'zero')
```

Out[22]:

	Name	Team	Q1	Q2	Q3	Q4
0	James	A	79	89	68	62
1	Messi	nan	64	76	69	66
2	Durant	C	76	76	63	68
3	Irvine	D	74	76	60	63
4	Harden	E	66	78	60	46

In [23]:
```
df.style.bar(vmin = 50, vmax = 100)
```

Out[23]:

	Name	Team	Q1	Q2	Q3	Q4
0	James	A	79	89	68	62
1	Messi	nan	64	76	69	66
2	Durant	C	76	76	63	68
3	Irvine	D	74	76	60	63
4	Harden	E	66	78	60	46

In [24]:
```
(df.assign(avg = df.mean(axis = 1, numeric_only = True))
 .assign(diff = lambda x: x.avg.diff()) # diff() calculate the first order difference
 .style
 .bar(color = 'yellow', subset = ['Q1'], vmin = 60, vmax = 80)
 .bar(subset = ['avg'], width=90,
      align = 'mid',
      vmin = 60, vmax = 80,
      color = '#5CADAD')
 .bar(subset = ['diff'],
      color = ['#ffe4e4','#bbf9ce'],
      vmin = 0, vmax = 10,
      align = 'zero')
)
```

Out[24]:

	Name	Team	Q1	Q2	Q3	Q4	avg	diff
0	James	A	79	89	68	62	74.500000	nan
1	Messi	nan	64	76	69	66	68.750000	-5.750000
2	Durant	C	76	76	63	68	70.750000	2.000000
3	Irvine	D	74	76	60	63	68.250000	-2.500000
4	Harden	E	66	78	60	46	62.500000	-5.750000

Formatting

Common formatting techniques include adding currency symbols, percentages, and thousands separators. Pandas provides the `.format` method within the Styler class to facilitate data formatting.

The `.format` method allows you to apply formatting options to the displayed values in a DataFrame or Series. It supports various formatting options, such as specifying the number of decimal places, adding currency symbols, controlling alignment, and applying custom formatting patterns.

Syntax

```
Styler.format(self, formatter,
              subset=None,
              na_rep: Union[str, NoneType] = None)
```

```
formatter(str, callable, dict or None)
```

`formatter` : This can be a string, callable, dictionary, or None.

- If formatter is a string, it specifies a format string that will be applied to the selected columns or subsets. You can use formatting placeholders like {} or {:.2f} to define the desired format. For example, '{:.2%}' would format a column as a percentage with 2 decimal places.
- If formatter is a callable, it should be a function that takes a value as input and returns a formatted string. The function will be applied to each element in the selected columns or subsets.

- If formatter is a dictionary, it maps column names or subsets to format strings or callables. Each column or subset will be formatted according to its corresponding format specified in the dictionary.
- If formatter is None, no formatting will be applied to the selected columns or subsets.

`subset` : This parameter specifies the columns or subsets to which the formatting should be applied. It can be a single column name, a list of column names, or a callable that returns a boolean Series indicating the desired subset.

`na_rep` : This parameter specifies the string representation to use for missing values (NaN). It defaults to None, which uses the default representation.

Example

In [25]:
```
data = {
    'Country': ['USA', 'Canada', 'Germany', 'Japan', 'Australia'],
    'Population': [328.2, 37.6, 83.0, 126.5, 25.4],
    'GDP': [21.43, 1.64, 4.42, 5.15, 1.39],
    'Area (sq km)': [9629091, 9976140, 357022, 377915, 7692024]
}
df = pd.DataFrame(data)
df
```

Out[25]:

	Country	Population	GDP	Area (sq km)
0	USA	328.2	21.43	9629091
1	Canada	37.6	1.64	9976140
2	Germany	83.0	4.42	357022
3	Japan	126.5	5.15	377915
4	Australia	25.4	1.39	7692024

In [26]:
```
df.style.format("{:.1f}", subset = ['GDP', 'Population'])
```

Out[26]:

	Country	Population	GDP	Area (sq km)
0	USA	328.2	21.4	9629091
1	Canada	37.6	1.6	9976140
2	Germany	83.0	4.4	357022
3	Japan	126.5	5.2	377915
4	Australia	25.4	1.4	7692024

In [27]:
```python
# Specify the columns to be all upper case
df.style.format({'Country': str.upper})
```

Out[27]:

	Country	Population	GDP	Area (sq km)
0	USA	328.200000	21.430000	9629091
1	CANADA	37.600000	1.640000	9976140
2	GERMANY	83.000000	4.420000	357022
3	JAPAN	126.500000	5.150000	377915
4	AUSTRALIA	25.400000	1.390000	7692024

In [28]:
```python
df.style.format({"GDP": lambda x: "+{:.2f}Trillion".format(abs(x))})
```

Out[28]:

	Country	Population	GDP	Area (sq km)
0	USA	328.200000	+21.43Trillion	9629091
1	Canada	37.600000	+1.64Trillion	9976140
2	Germany	83.000000	+4.42Trillion	357022
3	Japan	126.500000	+5.15Trillion	377915
4	Australia	25.400000	+1.39Trillion	7692024

In [29]:
```python
data = {
    'Name': ['John', 'Emma', 'Michael', 'Sophia', 'William', 'Olivia', 'James'],
    'Grade_Semester1': [80, 85, 90, 92, 78, 88, 90],
    'Grade_Semester2': [88, 92, 87, 85, 90, 95, 82],
    'Grade_Semester3': [95, 90, 88, 92, 85, 90, 88],
    'Tuition_Fees': [5000, 5500, 6000, 6500, 7000, 5500, 6000]
}

df = pd.DataFrame(data)
df
```

Out[29]:

	Name	Grade_Semester1	Grade_Semester2	Grade_Semester3	Tuition_Fees
0	John	80	88	95	5000
1	Emma	85	92	90	5500
2	Michael	90	87	88	6000
3	Sophia	92	85	92	6500
4	William	78	90	85	7000
5	Olivia	88	95	90	5500
6	James	90	82	88	6000

In [30]:

```
# A genral case use the chain rule
(df.assign(avg = df.mean(axis = 1, numeric_only = True)/100)
 .assign(diff = lambda x: x.avg.diff())
 .style
 .format({'Name': str.upper, 'avg': "{:.2f}", 'diff': "{:.2f}"}, na_rep = "-")
)  # Put all the format in the dictionary
```

Out[30]:

	Name	Grade_Semester1	Grade_Semester2	Grade_Semester3	Tuition_Fees	avg	diff
0	JOHN	80	88	95	5000	13.16	-
1	EMMA	85	92	90	5500	14.42	1.26
2	MICHAEL	90	87	88	6000	15.66	1.24
3	SOPHIA	92	85	92	6500	16.92	1.26
4	WILLIAM	78	90	85	7000	18.13	1.21
5	OLIVIA	88	95	90	5500	14.43	-3.70
6	JAMES	90	82	88	6000	15.65	1.22

Formatting Configuration

Pandas provides some options for configuring styles in a more systematic way, eliminating the need for individual settings. It also offers additional methods to enrich the output of styled content.

Add a caption to the table

```
In [31]: df.style.set_caption("Student Grades and Tuition Fees")
```

Out[31]:

Student Grades and Tuition Fees

	Name	Grade_Semester1	Grade_Semester2	Grade_Semester3	Tuition_Fees
0	John	80	88	95	5000
1	Emma	85	92	90	5500
2	Michael	90	87	88	6000
3	Sophia	92	85	92	6500
4	William	78	90	85	7000
5	Olivia	88	95	90	5500
6	James	90	82	88	6000

```
In [32]: # To align the caption in the middle, use CCS styling within the set_caption()
         df.style.set_caption("<div style='text-align: center;'>Student Grades and Tuition Fees</div>")
```

Out[32]:

Student Grades and Tuition Fees

	Name	Grade_Semester1	Grade_Semester2	Grade_Semester3	Tuition_Fees
0	John	80	88	95	5000
1	Emma	85	92	90	5500
2	Michael	90	87	88	6000
3	Sophia	92	85	92	6500
4	William	78	90	85	7000
5	Olivia	88	95	90	5500
6	James	90	82	88	6000

Precision

Set the global data precision.

In [33]:
```
df1 = df.assign(avg = df.mean(axis = 1, numeric_only = True)/100) .assign(diff = lambda x: x.avg.diff())

df1
```

Out[33]:

	Name	Grade_Semester1	Grade_Semester2	Grade_Semester3	Tuition_Fees	avg	diff
0	John	80	88	95	5000	13.1575	NaN
1	Emma	85	92	90	5500	14.4175	1.2600
2	Michael	90	87	88	6000	15.6625	1.2450
3	Sophia	92	85	92	6500	16.9225	1.2600
4	William	78	90	85	7000	18.1325	1.2100
5	Olivia	88	95	90	5500	14.4325	-3.7000
6	James	90	82	88	6000	15.6500	1.2175

In [34]:
```
df1.style.format(precision = 2)
```

Out[34]:

	Name	Grade_Semester1	Grade_Semester2	Grade_Semester3	Tuition_Fees	avg	diff
0	John	80	88	95	5000	13.16	nan
1	Emma	85	92	90	5500	14.42	1.26
2	Michael	90	87	88	6000	15.66	1.24
3	Sophia	92	85	92	6500	16.92	1.26
4	William	78	90	85	7000	18.13	1.21
5	Olivia	88	95	90	5500	14.43	-3.70
6	James	90	82	88	6000	15.65	1.22

Missing values

In [35]:
```
df1.style.format(na_rep = 'Missing')
```

Out[35]:

	Name	Grade_Semester1	Grade_Semester2	Grade_Semester3	Tuition_Fees	avg	diff
0	John	80	88	95	5000	13.157500	Missing
1	Emma	85	92	90	5500	14.417500	1.260000
2	Michael	90	87	88	6000	15.662500	1.245000
3	Sophia	92	85	92	6500	16.922500	1.260000
4	William	78	90	85	7000	18.132500	1.210000
5	Olivia	88	95	90	5500	14.432500	-3.700000
6	James	90	82	88	6000	15.650000	1.217500

Hide index and column

In [36]:
```
df.head().style.hide(axis = 'index')
```

Out[36]:

Name	Grade_Semester1	Grade_Semester2	Grade_Semester3	Tuition_Fees
John	80	88	95	5000
Emma	85	92	90	5500
Michael	90	87	88	6000
Sophia	92	85	92	6500
William	78	90	85	7000

In [37]:
```
df.style.hide(subset=['Grade_Semester1', 'Grade_Semester2'], axis = 'columns')
```

Out[37]:

	Name	Grade_Semester3	Tuition_Fees
0	John	95	5000
1	Emma	90	5500
2	Michael	88	6000
3	Sophia	92	6500
4	William	85	7000
5	Olivia	90	5500
6	James	88	6000

Table styles

To set inline styles using set_properties in Pandas, you can pass a dictionary of CSS properties and values to the `set_properties` method. Each key-value pair in the dictionary represents a CSS property and its corresponding value.

In [38]:
```
# Specify the column to set the font color to red
df.style.set_properties(subset = ['Grade_Semester1'], **{'background-color': 'lightblue','color': 'red'})
```

Out[38]:

	Name	Grade_Semester1	Grade_Semester2	Grade_Semester3	Tuition_Fees
0	John	80	88	95	5000
1	Emma	85	92	90	5500
2	Michael	90	87	88	6000
3	Sophia	92	85	92	6500
4	William	78	90	85	7000
5	Olivia	88	95	90	5500
6	James	90	82	88	6000

In [39]:
```
df.head().style.set_properties(color ="blue", align = "right")
```

Out[39]:

	Name	Grade_Semester1	Grade_Semester2	Grade_Semester3	Tuition_Fees
0	John	80	88	95	5000
1	Emma	85	92	90	5500
2	Michael	90	87	88	6000
3	Sophia	92	85	92	6500
4	William	78	90	85	7000

In [40]:
```
df.head().style.set_properties(**{'width': '100px', 'font-size': '18px'})
```

Out[40]:

	Name	Grade_Semester1	Grade_Semester2	Grade_Semester3	Tuition_Fees
0	John	80	88	95	5000
1	Emma	85	92	90	5500
2	Michael	90	87	88	6000
3	Sophia	92	85	92	6500
4	William	78	90	85	7000

```
In [41]: df.head().style.set_properties(**{'background-color': 'black',
                                           'color': 'lawngreen',
                                           'border-color': 'white'})
```

Out[41]:

	Name	Grade_Semester1	Grade_Semester2	Grade_Semester3	Tuition_Fees
0	John	80	88	95	5000
1	Emma	85	92	90	5500
2	Michael	90	87	88	6000
3	Sophia	92	85	92	6500
4	William	78	90	85	7000

set_table_attributes

Add label properteis to `table`

Bootstrap: Bootstrap is a popular CSS framework that provides a variety of table styles. Some of the Bootstrap table classes include:

- "table": Basic table styling.
- "table-striped": Alternating row colors.
- "table-bordered": Borders around cells and table.
- "table-hover": Highlight row on hover.
- "table-responsive": Enable horizontal scrolling for responsive tables.

Bulma: Bulma is a lightweight CSS framework with its own set of table classes. Some of the Bulma table classes include:

- "table": Basic table styling.
- "is-striped": Alternating row colors.
- "is-bordered": Borders around cells and table.
- "is-hoverable": Highlight row on hover.
- "is-fullwidth": Make the table full width.

Foundation: Foundation is another CSS framework with its own table classes. Some of the Foundation table classes include:

- "table": Basic table styling.

- "striped": Alternating row colors.
- "bordered": Borders around cells and table.
- "hover": Highlight row on hover.
- "responsive": Enable horizontal scrolling for responsive tables.

In [42]:
```
df.head().style.set_table_attributes('class = "table-bordered"')
```

Out[42]:

	Name	Grade_Semester1	Grade_Semester2	Grade_Semester3	Tuition_Fees
0	John	80	88	95	5000
1	Emma	85	92	90	5500
2	Michael	90	87	88	6000
3	Sophia	92	85	92	6500
4	William	78	90	85	7000

Style apply function

The `Styler` object in pandas provides `apply()` and `applymap()` methods that allow you to define more complex styling functions. These functions can be applied to specific cells, rows, columns, or the entire table.

Applying functions

In pandas Styler, there are two methods for applying functions to style the DataFrame:

- `Styler.applymap`: This method applies a function to every element of the DataFrame.
- `Styler.apply`: This method applies a function to either columns, rows, or the entire DataFrame, based on the axis parameter.

Here is a brief explanation of each method:

- `Styler.applymap`: This method applies a function to every individual element in the DataFrame. It is useful when you want to style each element independently based on its value. For example, you can define a function that returns a CSS string based on the value of each element, and then apply this function using `applymap()

In [43]:
```python
def apply_even_columns_style(value):
    if isinstance(value, (int, float)):
        return 'color: blue'
    else:
        return ''

df.style.applymap(apply_even_columns_style, subset=pd.IndexSlice[:, ::2])
```

Out[43]:

	Name	Grade_Semester1	Grade_Semester2	Grade_Semester3	Tuition_Fees
0	John	80	88	95	5000
1	Emma	85	92	90	5500
2	Michael	90	87	88	6000
3	Sophia	92	85	92	6500
4	William	78	90	85	7000
5	Olivia	88	95	90	5500
6	James	90	82	88	6000

- `Styler.apply` : This method applies a function to either columns, rows, or the entire DataFrame based on the axis parameter. By default, `axis=0` applies the function to each column, while `axis=1` applies it to each row. If you want to apply the function to the entire DataFrame, we can set axis=None.

In [44]:
```python
def highlight_gt90(series):
    return ['color: red' if value >= 90 else 'color: black' for value in series]

df.loc[:, 'Grade_Semester1':'Grade_Semester3'].style.apply(lambda series: highlight_gt90(series))
```

Out[44]:

	Grade_Semester1	Grade_Semester2	Grade_Semester3
0	80	88	95
1	85	92	90
2	90	87	88
3	92	85	92
4	78	90	85
5	88	95	90
6	90	82	88

In [45]:

```
def highlight_max(row):
    return ['background-color: yellow' if value == row.max() else '' for value in row]

df.iloc[:, 1:4].style.apply(highlight_max, axis = 1)
```

Out[45]:

	Grade_Semester1	Grade_Semester2	Grade_Semester3
0	80	88	95
1	85	92	90
2	90	87	88
3	92	85	92
4	78	90	85
5	88	95	90
6	90	82	88

Clear Style

In [46]:

```
styled_df = df.style.background_gradient()
styled_df.clear()
styled_df
```

Out[46]:

	Name	Grade_Semester1	Grade_Semester2	Grade_Semester3	Tuition_Fees
0	John	80	88	95	5000
1	Emma	85	92	90	5500
2	Michael	90	87	88	6000
3	Sophia	92	85	92	6500
4	William	78	90	85	7000
5	Olivia	88	95	90	5500
6	James	90	82	88	6000

Export as Excel File

```
#Export to Excel
df.style.to_excel('grades.xlsx')

# Using the specified engine, openpyxl provides better compatibility with styles
df.style.to_excel('grades.xlsx', engine = 'openpyxl')

# Specify the sheet name
dfs.to_excel('grades.xlsx', sheet_name = 'Sheet1')

# Specify missing value representation
dfs.to_excel('grades.xlsx', na_rep='-')

# Floating point number format, the following example converts 0.1234 to 0.12
dfs.to_excel('grades.xlsx', float_format = "%.2f")

# Include only these two columns
dfs.to_excel('grades.xlsx', columns = ['Grade_Semester1', 'Grade_Semester2'])

# Exclude the header
dfs.to_excel('grades.xlsx', header = False)

# Exclude the index
dfs.to_excel('grades.xlsx', index = False)
```

```
# Specify the index label, multiple labels for multi-level index
dfs.to_excel('grades.xlsx', index_label = ['Name'])

# Specify the starting row and starting column
dfs.to_excel('grades.xlsx', startrow = 2, startcol = 3)

# Do not merge cells
dfs.to_excel('grades.xlsx', merge_cells = False)

# Specify the encoding
dfs.to_excel('grades.xlsx', encoding = 'utf-8')

# Infinity representation (Excel does not have a native representation for infinity)
dfs.to_excel('grades.xlsx', inf_rep = 'inf')

# Display more information in the error log
dfs.to_excel('grades.xlsx', verbose = True)

#Specify the bottom row and right column to freeze
dfs.to_excel('grades.xlsx', freeze_panes = (0,2))
```

```
In [ ]:
```