# Style your Pandas DataFrame and Make it Stunning

Kaustubh Gupta

Last Updated : 12 Oct, 2024

Pandas is a widely-used data science library that presents data in table format, similar to Excel. Just like in Excel, you can customize tables by adding colors and highlighting important values. The Pandas Style API allows for similar styling within dataframes to enhance presentation and make data more visually appealing. This article covers the features of Pandas styling, built-in functions, customizations, and advanced usage for improving dataframe aesthetics using `df.style`.

***This article was published as a part of the Data Science Blogathon.***

## Table of contents

- Add Captions

## What is Pandas Style?

A pandas dataframe is a tabular structure comprising rows and columns. One prevalent environment for data-related tasks is Jupyter notebooks, which are web-based, platform-independent integrated development environments (IDEs). In Jupyter notebooks, the pandas style of the dataframe is achieved through the use of HTML tags and CSS for rendering. Consequently, you have the flexibility to customize the appearance of these web elements.

We will see this in action in upcoming sections. For now, let's create a sample dataset and display the output dataframe.

Copy Code

```
import pandas as pd
import numpy as np
np.random.seed(88)
df = pd.DataFrame({'A': np.linspace(1, 10, 10)})
df = pd.concat([df, pd.DataFrame(np.random.randn(10, 4), columns=list('BCDE'))],
               axis=1)
df.iloc[3, 3] = np.nan
df.iloc[0, 2] = np.nan
print(df)
```

Doesn't this look boring to you? What if you transform this minimal table to this:

| | | | | | |
|---|---|---|---|---|---|
| 0 | 1.000000 | 0.106884 | nan | 0.956563 | 0.068411 |
| 1 | 2.000000 | 1.068514 | 0.997183 | -0.931548 | 0.730430 |
| 2 | 3.000000 | -0.171214 | -1.288691 | 1.061436 | -0.040501 |
| 3 | 4.000000 | -0.903721 | -1.554133 | nan | 0.200526 |
| 4 | 5.000000 | -0.747329 | 1.068081 | -0.277024 | 0.086557 |
| 5 | 6.000000 | -0.253221 | -1.212041 | 0.277273 | 0.552219 |
| 6 | 7.000000 | -0.467743 | -1.427493 | 0.885805 | 2.360634 |
| 7 | 8.000000 | -1.522006 | -0.215945 | 0.190327 | 0.722256 |
| 8 | 9.000000 | -0.870716 | 0.101749 | 0.555358 | 0.962261 |
| 9 | 10.000000 | 1.433499 | -0.701818 | -0.856952 | -0.858158 |

Image by Author (Made in Pandas)

The transformed table above has:

1. Maximum values marked yellow for each column
2. Null values marked red for each column
3. More appealing table style, better fonts for header, and increased font size.

Now, we will be exploring all the possible ways of styling the dataframe and making it similar to what you saw above, so let's begin!

## Styling the DataFrame

This accessor helps in the modification of the styler object (df.style), which controls the display of the dataframe on the web. Let's look at some of the methods to style the dataframe.

## 1. Highlight Min-Max values

The dataframes can take a large number of values but when it is of a smaller size, then it makes sense to print out all the values of the dataframe. Now, you might be doing some type of analysis and you wanted to highlight the extreme values of the data. For this purpose, you can add style to your dataframe that highlights these extreme values.

### For Highlighting Maximum Values

Chain ".highlight_max()" function to the styler object. Additionally, you can also specify the axis for which you want to highlight the values. (axis=1: Rows, axis=0: Columns – default).

```
df.style.highlight_max()
```

Copy Code

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2.000000 | 1.068514 | 0.997183 | -0.931548 | 0.730430 |
| 2 | 3.000000 | -0.171214 | -1.288691 | 1.061436 | -0.040501 |
| 3 | 4.000000 | -0.903721 | -1.554133 | nan | 0.200526 |
| 4 | 5.000000 | -0.747329 | 1.068081 | -0.277024 | 0.086557 |
| 5 | 6.000000 | -0.253221 | -1.212041 | 0.277273 | 0.552219 |
| 6 | 7.000000 | -0.467743 | -1.427493 | 0.885805 | 2.360634 |
| 7 | 8.000000 | -1.522006 | -0.215945 | 0.190327 | 0.722256 |
| 8 | 9.000000 | -0.870716 | 0.101749 | 0.555358 | 0.962261 |
| 9 | 10.000000 | 1.433499 | -0.701818 | -0.856952 | -0.858158 |

## For Highlighting Minimum Values

Chain ".highlight_min()" function to the styler object. Here also, you can specify the axis at which these values will be highlighted.

```
df.style.highlight_min()
```

Copy Code

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2.000000 | 1.068514 | 0.997183 | -0.931548 | 0.730430 |
| 2 | 3.000000 | -0.171214 | -1.288691 | 1.061436 | -0.040501 |
| 3 | 4.000000 | -0.903721 | -1.554133 | nan | 0.200526 |
| 4 | 5.000000 | -0.747329 | 1.068081 | -0.277024 | 0.086557 |
| 5 | 6.000000 | -0.253221 | -1.212041 | 0.277273 | 0.552219 |
| 6 | 7.000000 | -0.467743 | -1.427493 | 0.885805 | 2.360634 |
| 7 | 8.000000 | -1.522006 | -0.215945 | 0.190327 | 0.722256 |
| 8 | 9.000000 | -0.870716 | 0.101749 | 0.555358 | 0.962261 |
| 9 | 10.000000 | 1.433499 | -0.701818 | -0.856952 | -0.858158 |

Both Min-Max highlight functions support the parameter “color” to change the highlight color from yellow.

## 2. Highlight Null Values

Every dataset has some or the other null/missing values. These values should be either removed or handled in such a way that it doesn’t introduce any biasness. To highlight such values, you can chain the “.highlight_null()” function to the styler object. This function doesn’t support the axis parameter and the color control parameter here is “null_color” which takes the default value as “red”

```
df.style.highlight_null(null_color="green")
```

Copy Code

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2.000000 | 1.068514 | 0.997183 | -0.931548 | 0.730430 |
| 2 | 3.000000 | -0.171214 | -1.288691 | 1.061436 | -0.040501 |
| 3 | 4.000000 | -0.903721 | -1.554133 | nan | 0.200526 |
| 4 | 5.000000 | -0.747329 | 1.068081 | -0.277024 | 0.086557 |
| 5 | 6.000000 | -0.253221 | -1.212041 | 0.277273 | 0.552219 |
| 6 | 7.000000 | -0.467743 | -1.427493 | 0.885805 | 2.360634 |
| 7 | 8.000000 | -1.522006 | -0.215945 | 0.190327 | 0.722256 |
| 8 | 9.000000 | -0.870716 | 0.101749 | 0.555358 | 0.962261 |
| 9 | 10.000000 | 1.433499 | -0.701818 | -0.856952 | -0.858158 |

## set_na_rep()

Along with highlighting the missing values, they may be represented as “nan”. You can change the representation of these missing values using the set_na_rep() function. This function can also be chained with any styler function but chaining it with highlight_null will provide more details.

```
df.style.set_na_rep("OutofScope").highlight_null(null_color="orange")
```

Copy Code

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2.000000 | 1.068514 | 0.997183 | -0.931548 | 0.730430 |
| 2 | 3.000000 | -0.171214 | -1.288691 | 1.061436 | -0.040501 |
| 3 | 4.000000 | -0.903721 | -1.554133 | OutofScope | 0.200526 |
| 4 | 5.000000 | -0.747329 | 1.068081 | -0.277024 | 0.086557 |
| 5 | 6.000000 | -0.253221 | -1.212041 | 0.277273 | 0.552219 |
| 6 | 7.000000 | -0.467743 | -1.427493 | 0.885805 | 2.360634 |
| 7 | 8.000000 | -1.522006 | -0.215945 | 0.190327 | 0.722256 |
| 8 | 9.000000 | -0.870716 | 0.101749 | 0.555358 | 0.962261 |
| 9 | 10.000000 | 1.433499 | -0.701818 | -0.856952 | -0.858158 |

## 3. Create Heatmap within dataframe

Heatmaps are used to represent values with the color shades. The higher is the color shade, the larger is the value present. These color shades represent the intensity of values as compared to other values. To plot such a mapping in the dataframe itself, there is no direct function but the “styler.background_gradient()” workaround does the work.

```
df.style.background_gradient()
```

Copy Code

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2.000000 | 1.068514 | 0.997183 | -0.931548 | 0.730430 |
| 2 | 3.000000 | -0.171214 | -1.288691 | 1.061436 | -0.040501 |
| 3 | 4.000000 | -0.903721 | -1.554133 | nan | 0.200526 |
| 4 | 5.000000 | -0.747329 | 1.068081 | -0.277024 | 0.086557 |
| 5 | 6.000000 | -0.253221 | -1.212041 | 0.277273 | 0.552219 |
| 6 | 7.000000 | -0.467743 | -1.427493 | 0.885805 | 2.360634 |
| 7 | 8.000000 | -1.522006 | -0.215945 | 0.190327 | 0.722256 |
| 8 | 9.000000 | -0.870716 | 0.101749 | 0.555358 | 0.962261 |
| 9 | 10.000000 | 1.433499 | -0.701818 | -0.856952 | -0.858158 |

There are few parameters you can pass to this function to further customize the output generated:

1. **cmap:** By default, the “PuBu” colormap is selected by pandas You can create a custom matplotlib colormap and pass it to the camp parameter.
2. **axis:** Generating heat plot via rows or columns criteria, by default: columns
3. **text_color_threshold**: Controls text visibility across varying background colors.

## 4. Table Properties

As mentioned earlier also, the dataframe presented in the Jupyter notebooks is a table rendered using HTML and CSS. The table properties can be controlled using the “set_properties” method. This method is used to set one or more data-independent

Copy Code

```
df.style.set_properties(**{'border': '1.3px solid green',
                           'color': 'magenta'})
```

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 0 | 1.000000 | 0.106884 | nan | 0.956563 | 0.068411 |
| 1 | 2.000000 | 1.068514 | 0.997183 | -0.931548 | 0.730430 |
| 2 | 3.000000 | -0.171214 | -1.288691 | 1.061436 | -0.040501 |
| 3 | 4.000000 | -0.903721 | -1.554133 | nan | 0.200526 |
| 4 | 5.000000 | -0.747329 | 1.068081 | -0.277024 | 0.086557 |
| 5 | 6.000000 | -0.253221 | -1.212041 | 0.277273 | 0.552219 |
| 6 | 7.000000 | -0.467743 | -1.427493 | 0.885805 | 2.360634 |
| 7 | 8.000000 | -1.522006 | -0.215945 | 0.190327 | 0.722256 |
| 8 | 9.000000 | -0.870716 | 0.101749 | 0.555358 | 0.962261 |
| 9 | 10.000000 | 1.433499 | -0.701818 | -0.856952 | -0.858158 |

Also Read: [How to Create a Pandas DataFrame from Lists ?]

## 5. Create Bar Charts

Just as the heatmap, the bar charts can also be plotted within the dataframe itself. The bars are plotted in each cell depending upon the axis selected. By default, the axis=0 and the plot color are also fixed by pandas but it is configurable. To plot these bars, you simply need to chain the ".bar()" function to the styler object.

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2.000000 | 1.068514 | 0.997183 | -0.931548 | 0.730430 |
| 2 | 3.000000 | -0.171214 | -1.288691 | 1.061436 | -0.040501 |
| 3 | 4.000000 | -0.903721 | -1.554133 | nan | 0.200526 |
| 4 | 5.000000 | -0.747329 | 1.068081 | -0.277024 | 0.086557 |
| 5 | 6.000000 | -0.253221 | -1.212041 | 0.277273 | 0.552219 |
| 6 | 7.000000 | -0.467743 | -1.427493 | 0.885805 | 2.360634 |
| 7 | 8.000000 | -1.522006 | -0.215945 | 0.190327 | 0.722256 |
| 8 | 9.000000 | -0.870716 | 0.101749 | 0.555358 | 0.962261 |
| 9 | 10.000000 | 1.433499 | -0.701818 | -0.856952 | -0.858158 |

## 6. Control Precision

The current values of the dataframe have float values and their decimals have no boundary condition. Even the column “A”, which had to hold a single value is having too many decimal places. To control this behavior, you can use the “.set_precision()” function and pass the value for maximum decimals to be allowed.

```
df.style.set_precision(2)
```

| | | | | | |
|---|---|---|---|---|---|
| **1** | 2.00 | 1.07 | 1.00 | -0.93 | 0.73 |
| **2** | 3.00 | -0.17 | -1.29 | 1.06 | -0.04 |
| **3** | 4.00 | -0.90 | -1.55 | nan | 0.20 |
| **4** | 5.00 | -0.75 | 1.07 | -0.28 | 0.09 |
| **5** | 6.00 | -0.25 | -1.21 | 0.28 | 0.55 |
| **6** | 7.00 | -0.47 | -1.43 | 0.89 | 2.36 |
| **7** | 8.00 | -1.52 | -0.22 | 0.19 | 0.72 |
| **8** | 9.00 | -0.87 | 0.10 | 0.56 | 0.96 |
| **9** | 10.00 | 1.43 | -0.70 | -0.86 | -0.86 |

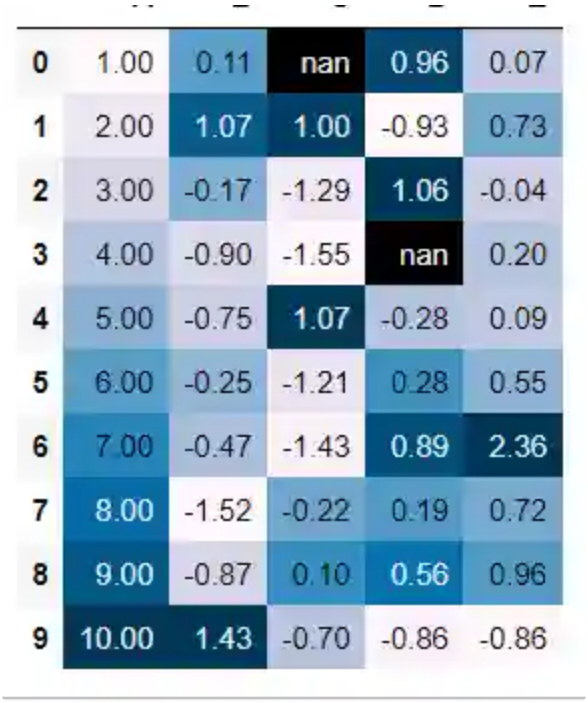
Now the dataframe looks clean.

## 7. Add Captions

Like every image has a caption that defines the post text, you can add captions to your dataframes. This text will depict what the dataframe results talk about. They may be some sort of summary statistics like pivot tables.

```
df.style.set_caption("This is Analytics Vidhya Blog").set_precision(2).backgrc
```

Copy Code



| | | | | | |
|---|---|---|---|---|---|
| 0 | 1.00 | 0.11 | nan | 0.96 | 0.07 |
| 1 | 2.00 | 1.07 | 1.00 | -0.93 | 0.73 |
| 2 | 3.00 | -0.17 | -1.29 | 1.06 | -0.04 |
| 3 | 4.00 | -0.90 | -1.55 | nan | 0.20 |
| 4 | 5.00 | -0.75 | 1.07 | -0.28 | 0.09 |
| 5 | 6.00 | -0.25 | -1.21 | 0.28 | 0.55 |
| 6 | 7.00 | -0.47 | -1.43 | 0.89 | 2.36 |
| 7 | 8.00 | -1.52 | -0.22 | 0.19 | 0.72 |
| 8 | 9.00 | -0.87 | 0.10 | 0.56 | 0.96 |
| 9 | 10.00 | 1.43 | -0.70 | -0.86 | -0.86 |

*(Here, different methods have been changed along with the caption method)*

## 8. Hiding Index or Column

As the title suggests, you can hide the index or any particular column from the dataframe. Hiding index from the dataframe can be useful in cases when the index doesn’t convey anything significant about the data. The column hiding depends on whether it is useful or not.

```
df.style.hide_index()
```

| | | | | |
|---|---|---|---|---|
| 1.000000 | 0.106884 | nan | 0.956563 | 0.068411 |
| 2.000000 | 1.068514 | 0.997183 | -0.931548 | 0.730430 |
| 3.000000 | -0.171214 | -1.288691 | 1.061436 | -0.040501 |
| 4.000000 | -0.903721 | -1.554133 | nan | 0.200526 |
| 5.000000 | -0.747329 | 1.068081 | -0.277024 | 0.086557 |
| 6.000000 | -0.253221 | -1.212041 | 0.277273 | 0.552219 |
| 7.000000 | -0.467743 | -1.427493 | 0.885805 | 2.360634 |
| 8.000000 | -1.522006 | -0.215945 | 0.190327 | 0.722256 |
| 9.000000 | -0.870716 | 0.101749 | 0.555358 | 0.962261 |
| 10.000000 | 1.433499 | -0.701818 | -0.856952 | -0.858158 |

## 9. Control Display Values

Using the styler object's ".format()" function, you can distinguish between the actual values held by the dataframe and the values you present. The "format" function takes in the format spec string that defines how individual values are presented.

You can directly specify the specification which will apply to the whole dataset or you can pass the specific column on which you want to control the display values.

```
df.style.format("{:.3%}")
```

| | | | | | |
|---|---|---|---|---|---|
| 1 | 200.000% | 106.851% | 99.718% | -93.155% | 73.043% |
| 2 | 300.000% | -17.121% | -128.869% | 106.144% | -4.050% |
| 3 | 400.000% | -90.372% | -155.413% | nan% | 20.053% |
| 4 | 500.000% | -74.733% | 106.808% | -27.702% | 8.656% |
| 5 | 600.000% | -25.322% | -121.204% | 27.727% | 55.222% |
| 6 | 700.000% | -46.774% | -142.749% | 88.580% | 236.063% |
| 7 | 800.000% | -152.201% | -21.595% | 19.033% | 72.226% |
| 8 | 900.000% | -87.072% | 10.175% | 55.536% | 96.226% |
| 9 | 1000.000% | 143.350% | -70.182% | -85.695% | -85.816% |

You may notice that the missing values have also been marked by the format function. This can be skipped and substituted with a different value using the “na_rep” (na replacement) parameter.

```
df.style.format("{:.3%}", na_rep="&&")
```

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 0 | 100.000% | 10.688% | && | 95.656% | 6.841% |
| 1 | 200.000% | 106.851% | 99.718% | -93.155% | 73.043% |
| 2 | 300.000% | -17.121% | -128.869% | 106.144% | -4.050% |
| 3 | 400.000% | -90.372% | -155.413% | && | 20.053% |
| 4 | 500.000% | -74.733% | 106.808% | -27.702% | 8.656% |
| 5 | 600.000% | -25.322% | -121.204% | 27.727% | 55.222% |
| 6 | 700.000% | -46.774% | -142.749% | 88.580% | 236.063% |
| 7 | 800.000% | -152.201% | -21.595% | 19.033% | 72.226% |

Although you have many methods to style your dataframe, it might be the case that your requirements are different and you need a custom styling function for your analysis. You can create your function and use it with the styler object in two ways:

1. **apply function:** When you chain the “apply” function to the styler object, it sends out the entire row (series) or the dataframe depending upon the axis selected. Hence, if you make your function work with the “apply” function, it should return the series or dataframe with the same shape and CSS attribute-value pair.
2. **apply map function:** This function sends out scaler values (or element-wise) and therefore, your function should return a scaler only with CSS attribute-value pair.

Let’s implement both types:

## Target: apply function

Copy Code

```
def highlight_mean_greater(s):
    '''
    highlight yellow is value is greater than mean else red.
    '''
    is_max = s > s.mean()
    return ['background-color: yellow' if i else 'background-color: red' for i in is_max
```

Copy Code

```
df.style.apply(highlight_mean_greater)
```

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2.000000 | 1.068514 | 0.997183 | -0.931548 | 0.730430 |
| 2 | 3.000000 | -0.171214 | -1.288691 | 1.061436 | -0.040501 |
| 3 | 4.000000 | -0.903721 | -1.554133 | nan | 0.200526 |
| 4 | 5.000000 | -0.747329 | 1.068081 | -0.277024 | 0.086557 |
| 5 | 6.000000 | -0.253221 | -1.212041 | 0.277273 | 0.552219 |
| 6 | 7.000000 | -0.467743 | -1.427493 | 0.885805 | 2.360634 |
| 7 | 8.000000 | -1.522006 | -0.215945 | 0.190327 | 0.722256 |
| 8 | 9.000000 | -0.870716 | 0.101749 | 0.555358 | 0.962261 |
| 9 | 10.000000 | 1.433499 | -0.701818 | -0.856952 | -0.858158 |

## Target: apply map function

Copy Code

```
def color_negative_red(val):
    """
    Takes a scalar and returns a string with
    the css property `'color: red'` for negative
    strings, black otherwise.
    """
    color = 'red' if val < 0 else 'black'
    return 'color: %s' % color
```

Copy Code

```
df.style.apply(color_negative_red)
```

| | | | | | |
|---|---|---|---|---|---|
| 1 | 2.000000 | 1.068514 | 0.997183 | -0.931548 | 0.730430 |
| 2 | 3.000000 | -0.171214 | -1.288691 | 1.061436 | -0.040501 |
| 3 | 4.000000 | -0.903721 | -1.554133 | nan | 0.200526 |
| 4 | 5.000000 | -0.747329 | 1.068081 | -0.277024 | 0.086557 |
| 5 | 6.000000 | -0.253221 | -1.212041 | 0.277273 | 0.552219 |
| 6 | 7.000000 | -0.467743 | -1.427493 | 0.885805 | 2.360634 |
| 7 | 8.000000 | -1.522006 | -0.215945 | 0.190327 | 0.722256 |
| 8 | 9.000000 | -0.870716 | 0.101749 | 0.555358 | 0.962261 |
| 9 | 10.000000 | 1.433499 | -0.701818 | -0.856952 | -0.858158 |

## Table Styles

These are styles that apply to the table as a whole, but don't look at the data. It is very similar to the set_properties function but here, in the table styles, you can customize all web elements more easily.

The function of concern here is the "set_table_styles" that takes in the list of dictionaries for defining the elements. The dictionary needs to have the selector (HTML tag or CSS class) and its corresponding props (attributes or properties of the element). The props need to be a list of tuples of properties for that selector.

The images shown in the beginning, the transformed table has the following style:

```
styles = [
```

Copy Code

```
    dict(selector="td", props=[("color", "#999"),
                               ("border", "1px solid #eee"),
                               ("padding", "12px 35px"),
                               ("border-collapse", "collapse"),
                               ("font-size", "15px")
                               ]),
    dict(selector="table", props=[
                                    ("font-family" , 'Arial'),
                                    ("margin" , "25px auto"),
                                    ("border-collapse" , "collapse"),
                                    ("border" , "1px solid #eee"),
                                    ("border-bottom" , "2px solid #00cccc"),
                                      ]),
    dict(selector="caption", props=[("caption-side", "bottom")])
]
```

And the required methods which created the final table:

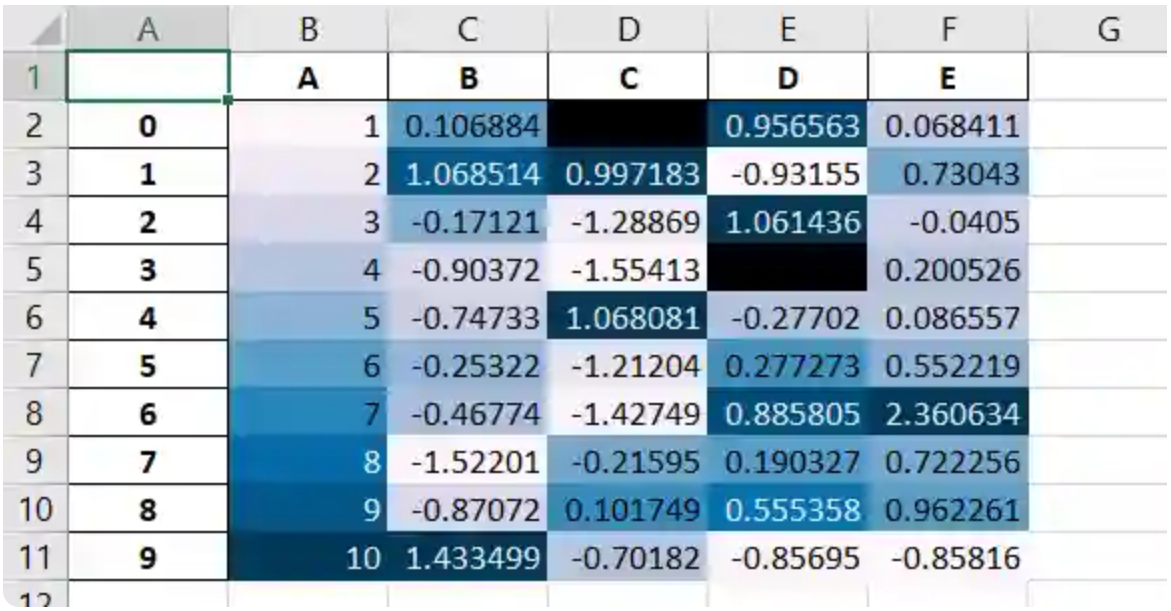
```
df.style.set_table_styles(styles).set_caption("Image by Author (Made in Pandas
```

Copy Code

You can store all the styling you have done on your dataframe in an excel file. The ".to_excel" function on the styler object makes it possible. The function needs two parameters: the name of the file to be saved (with extension XLSX) and the "engine" parameter should be "openpyxl".

```
df.style.set_precision(2).background_gradient().hide_index().to_excel('styled.
```

Copy Code



| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | | A | B | C | D | E | |
| 2 | 0 | 1 | 0.106884 | | 0.956563 | 0.068411 | |
| 3 | 1 | 2 | 1.068514 | 0.997183 | -0.93155 | 0.73043 | |
| 4 | 2 | 3 | -0.17121 | -1.28869 | 1.061436 | -0.0405 | |
| 5 | 3 | 4 | -0.90372 | -1.55413 | | 0.200526 | |
| 6 | 4 | 5 | -0.74733 | 1.068081 | -0.27702 | 0.086557 | |
| 7 | 5 | 6 | -0.25322 | -1.21204 | 0.277273 | 0.552219 | |
| 8 | 6 | 7 | -0.46774 | -1.42749 | 0.885805 | 2.360634 | |
| 9 | 7 | 8 | -1.52201 | -0.21595 | 0.190327 | 0.722256 | |
| 10 | 8 | 9 | -0.87072 | 0.101749 | 0.555358 | 0.962261 | |
| 11 | 9 | 10 | 1.433499 | -0.70182 | -0.85695 | -0.85816 | |

## How to apply style to DataFrame?

## Using `Styler.apply()`

- **What it does**: This method lets you apply styles to the entire DataFrame, either by rows or columns.
- **How to use it**:
  - Create a function that returns a style (like color) for each row or column.
  - Use `Styler.apply()` to apply your function.
- **Example**: If you want to color the text in a column based on its value, you can write a function that checks the value and returns a color.

- Use `Styler.applymap()` to apply your function to every cell.

- **Example**: If you want to change the background color of cells based on their values, this is the method to use.

## 3. Formatting Values

- **What it does**: This method allows you to change how numbers or text look in the DataFrame.
- **How to use it**:
  - Use `Styler.format()` to specify how you want certain columns to be displayed (like showing numbers with two decimal places).
- **Example**: You can format a column of prices to always show two decimal points.

## 4. Setting Table Styles

- **What it does**: This method lets you add CSS styles to the entire table.
- **How to use it**:
  - Use `Styler.set_table_styles()` to define styles for headers, rows, or cells.
- **Example**: You can set all table headers to have a bold font and a specific background color.

## Additional Cu

the highest value in a column.

- **Set Caption**: Use `Styler.set_caption()` to add a title to your table.

## Conclusion

In this detailed article, we saw all the built-in methods to style the dataframe. Then we looked at how to create custom styling functions and then we saw how to customize the dataframe by modifying it at HTML and CSS level. We also saw how to save our styled dataframe into excel files.

Hope you like the article! To enhance data presentation, use **pandas to HTML style** with **df.style**. The **pandas style** feature allows customization, enabling **df.style.format** for elegant outputs, making **df.style pandas** more visually appealing.

***The media shown in this article are not owned by Analytics Vidhya and are used at the Author's discretion.***

---

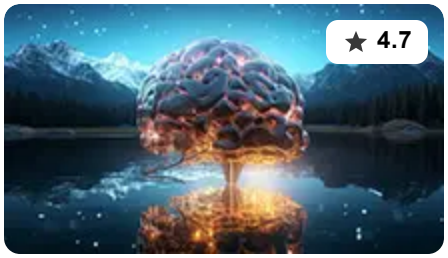
Kaustubh Gupta

Kaustubh Gupta is a skilled engineer with a B.Tech in Information Technology from Maharaja Agrasen Institute of Technology. With experience as a CS Analyst and Analyst Intern at Prodigal Technologies, Kaustubh excels in Python, SQL, Libraries.

significantly to the tech community through speaking sessions and workshops.

Beginner | Data Exploration | Programming | Project | Python | Python

## Free Courses




### Generative AI - A Way of Life

Explore Generative AI for beginners: create text and images, use top AI tools, learn practical skills, and ethics.


### Getting Started with Large Language Models

Master Large Language Models (LLMs) with this course, offering clear guidance in NLP and model training made simple.

chatbots with enterprise data.


## Improving Real World RAG Systems: Key Challenges & Practical Solutions

Explore practical solutions, advanced retrieval strategies, and agentic RAG systems to improve context, relevance, and accuracy in AI-driven applications.


## Microsoft Excel: Formulas & Functions

Master MS Excel for data analysis with key formulas, functions, and LookUp tools in this comprehensive course.

# Responses From Readers

What are your thoughts?...

Submit reply

Sagar

hide_index() doesn't seem to work. Ref last screenshot of the excel output

# Frequently Asked Questions

## Q1. What is pandas style?

A. Pandas styling refers to the capability in the Python library Pandas to apply formatting and styling to tabular data frames. It allows users to customize the visual representation of their data, such as changing cell colors, fonts, and highlighting specific values, making it easier to analyze and present data in a more visually appealing and informative manner.

## Q2. What is the code style of pandas?

- Build Your Brand & Audience
- Cash In on Your Knowledge
- Join a Thriving Community
- Level Up Your Data Science Game


## Flagship Courses

GenAI Pinnacle Program | AI/ML BlackBelt Courses

## Free Courses

## Popular Categories

Generative AI | Prompt Engineering | Generative AI Application | News | Technical Guides | AI Tools | Interview Preparation | Research Papers | Success Stories | Quiz | Use Cases | Listicles

## Generative AI Tools and Techniques

GANs | VAEs | Transformers | StyleGAN | Pix2Pix | Autoencoders | GPT | BERT | Word2Vec | LSTM | Attention Mechanisms | Diffusion Models | LLMs | SLMs | StyleGAN | Encoder Decoder Models | Prompt Engineering | LangChain | LlamaIndex | RAG | Fine-tuning | LangChain AI Agent | Multimodal Models | RNNs | DCGAN | ProGAN | Text-to-Image Models | DDPM | Document Question Answering | Imagen | T5 (Text-to-Text Transfer Transformer) | Seq2seq Models | WaveNet | Attention Is All You Need (Transformer Architecture)

## Popular GenAI Models

Llama 3.1 | Llama 3 | Llama 2 | GPT 4o Mini | GPT 4o | GPT 3 | Claude 3 Haiku | Claude 3.5 Sonnet | Phi 3.5 | Phi 3 | Mistral Large 2 | Mistral NeMo | Mistral-7b | Gemini 1.5 Pro | Gemini Flash 1.5 | Bedrock | Vertex AI | DALL.E | Midjourney | Stable Diffusion

## Data Science Tools and Techniques

Python | R | SQL | Jupyter Notebooks | TensorFlow | Scikit-learn | PyTorch | Tableau | Apache Spark | Matplotlib | Seaborn | Pandas | Hadoop | Docker | Git | Keras | Apache Kafka | AWS | NLP | Random Forest | Computer Vision | Data Visualization | Data Exploration | Big Data | Common Machine Learning Algorithms | Machine Learning

## Learn

Free Courses

AI&ML Program

GenAI Program

Agentic AI Program

## Engage

Community

Hackathons

Events

Podcasts

## Contribute

Become an Author

Become a Speaker

Become a Mentor

Become an Instructor

## Enterprise

Our Offerings

Trainings

Data Culture

AI Newsletter