**Transform your Pandas Dataframes: Styles, 🎨 Colors, and 😎 Emojis**

[[Romy Mendez](https://medium.com/@romina.elena.mendez?source=post_page-----bf938d6e98a2--------------------------------)](https://medium.com/@romina.elena.mendez?source=post_page-----bf938d6e98a2--------------------------------)

[Romy Mendez](https://medium.com/@romina.elena.mendez?source=post_page-----bf938d6e98a2--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2F3d0947d520ac&operation=register&redirect=https%3A%2F%2Fmedium.com%2F%40romina.elena.mendez%2Ftransform-your-pandas-dataframes-styles-colors-and-emojis-bf938d6e98a2&user=Romy+Mendez&userId=3d0947d520ac&source=post_page-3d0947d520ac----bf938d6e98a2---------------------post_header-----------)

9 min read

·

Jan 2

375

4

In the following section of this article, we will explore a method to add colors and styles to Pandas DataFrames. Our focus will be on the application of colors and emojis, utilizing approaches similar to the popular conditional formatting commonly employed in pivot tables within spreadsheets. Through this strategy, we aim to enhance the presentation of our data, making the exploration and understanding of information not only informative but also visually appealing

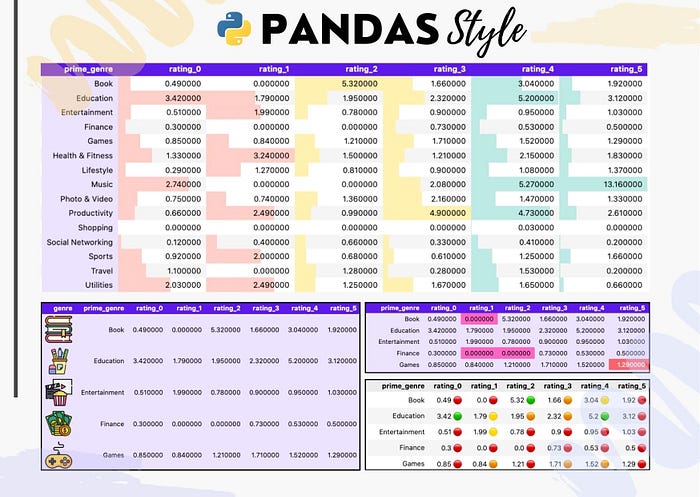


Image generated using [PartyRock](https://partyrock.aws/)

**What is Pandas Style?**

**Pandas Styler** is a module within the Pandas library that provides methods for creating HTML-styled representations of DataFrames. This feature allows for the customization of the visual appearance of DataFrames during their visualization. The core functionality of Pandas Styler lies in the ability to highlight, color, and format cells based on specific conditions, facilitating the visual identification of patterns and trends in datasets.

Also, Pandas Styler stands out for its capability to assist in the design of DataFrames or series by generating visual representations using HTML and CSS. This functionality simplifies the creation of attractive and customized data presentations, enhancing the visualization experience, and enabling a more intuitive interpretation of the information contained in the datasets.



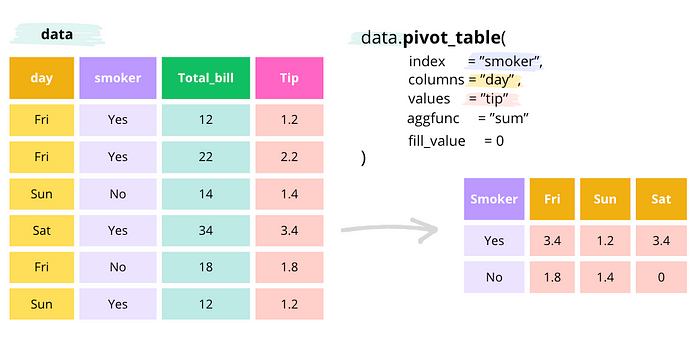
Next we have the code with we are going to create a pivot table using a set of data and from this you will begin to give it different styles and conditional formats such as can be seen in the previous image.

**Pivot Tables**

The **pivot table** is a tabular data structure that provides a summarized overview of information from another table, organizing the data based on one variable and displaying values associated with another variable. In this specific scenario, the pivot table organizes the data according to the ‘smoker’ column and presents the total sum of tips, categorized by the days on which clients consume in the restaurant

**Example**

The following example shows the `pivot\_table` method with the ‘tips’ DataFrame



import pandas as pd  
import seaborn as sns  
# create the tips dataframe   
data = sns.load\_dataset('tips')  
data\_pivot = pd.pivot\_table(data,  
 index='smoker',  
 columns='day',  
 values='total\_bill',  
 aggfunc='sum').reset\_index()  
data\_pivot

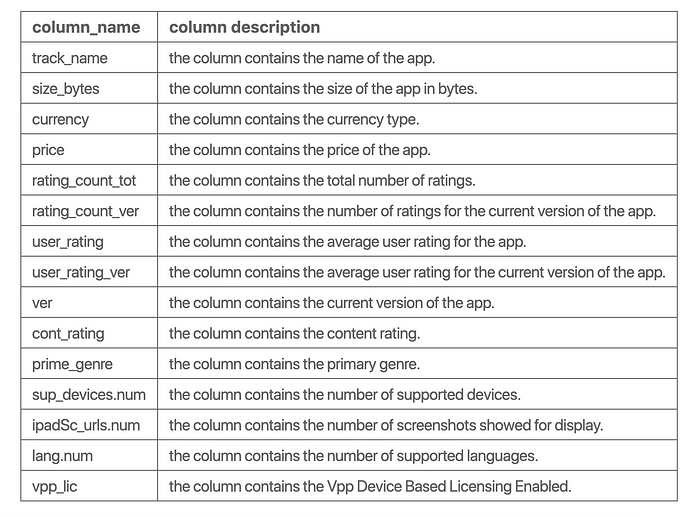
| smoker | Thur | Fri | Sat | Sun |  
--|--------|--------|--------|--------|---------|  
0 | Yes | 326.24 | 252.20 | 893.62 | 458.28 |  
1 | No | 770.09 | 73.68 | 884.78 | 1168.88 |

**Dataframe: Apple Store apps**

In this analysis, we will use the \*\*’🍎 Apple Store apps’\*\* DataFrame to explore the creation of pivot tables and customization of table styles. This dataset provides detailed insights into Apple App Store applications, covering aspects from app names to specifics like size, price, and ratings. Our objective is to efficiently break down the information while applying styles that enhance the presentation and comprehension of data effectively.

The dataset was downloaded from [Kaggle](https://www.kaggle.com/ramamet4/app-store-apple-data-set-10k-apps) and it contains more than 7000 Apple iOS mobile application details. It is important to note that the data was collected in July 2017.

**Data Schema overview**



**Create Dataframe**

In the following code chunk, we will create a DataFrame by reading the CSV file.

import pandas as pd  
import numpy as np  
import math  
import matplotlib.pyplot as plt  
import warnings  
# Deactivate pandas warning  
warnings.filterwarnings('ignore')  
print("Python Libraries version:")  
print(' - '\*20)  
print("Pandas version: ", pd.\_\_version\_\_)  
print("Numpy version: ", np.\_\_version\_\_)  
print("Matplotlib version: ", plt.matplotlib.\_\_version\_\_)

Python Libraries version:  
 - - - - - - - - - - - - - - - - - - - -   
Pandas version: 2.1.3  
Numpy version: 1.26.1  
Matplotlib version: 3.8.1

# Create a dataframe from a csv file  
# You can download the file from the following link https://github.com/r0mymendez/pandas-styles  
path='data/AppleStore.csv'  
data =pd.read\_csv(path,sep=';')

**Pivot Table**

In the next step, we are going to create a pivot table from a DataFrame.

# Pivot table  
# filter the data to keep only the top 15 genres  
top\_genre = data.value\_counts('prime\_genre')[:15].index.tolist()  
tmp = data.loc[data['prime\_genre'].isin(top\_genre),['prime\_genre','user\_rating','price']]  
# create a new column with the rating rounded to the nearest integer  
tmp['user\_rating'] = [f'rating\_{str(math.trunc(item))}' for item in tmp['user\_rating']]  
# create a pivot table  
tmp\_pivot = (  
 pd.pivot\_table(  
 data = tmp,  
 columns='user\_rating',  
 index='prime\_genre',  
 values='price',  
 aggfunc='mean',  
 fill\_value=0  
 ).reset\_index().round(2)  
)  
# rename the columns  
tmp\_pivot.columns.name=''  
# print the pivot table  
tmp\_pivot



**Styling with Pandas**

Now we will explore the `style` module in Pandas, that enables us to enhance the visual presentation of DataFrames. The `style` module provides a differents of options to modify the appearance of the data, allowing us to customize aspects such as:

* **Coloring Cells**: Apply different colors based on cell values or conditions.
* **Highlighting**: Emphasize specific rows, columns, or values.
* **Formatting**: Adjust the format of the displayed values, including precision and alignment.
* **Bar Charts**: Represent data with horizontal or vertical bar charts within cells.

**🎨 Styling: Setting Background Color for Headers**

In this section, we will apply styles to both the titles and the table. Therefore we use background colors to highlight the headers and the rest of the table.

# Styling: Changing Background Color for Column Headers  
headers = {  
 'selector': 'th.col\_heading',  
 'props': 'background-color: #5E17EB; color: white;'  
}  
index\_style = {  
 'selector': 'th.index\_name',  
 'props': 'background-color: #5E17EB; color: white;'  
}  
tmp\_pivot\_style = (  
 tmp\_pivot  
 .style  
 .set\_table\_styles([headers,index\_style])  
 .set\_properties(\*\*{'background-color': '#ECE3FF','color': 'black'})  
)  
tmp\_pivot\_style



**🎨 Style: Setting the background color for a particular cell**

In following code snippet illustrates how to set a custom background color for a particular cell in our DataFrame using pandas styling.

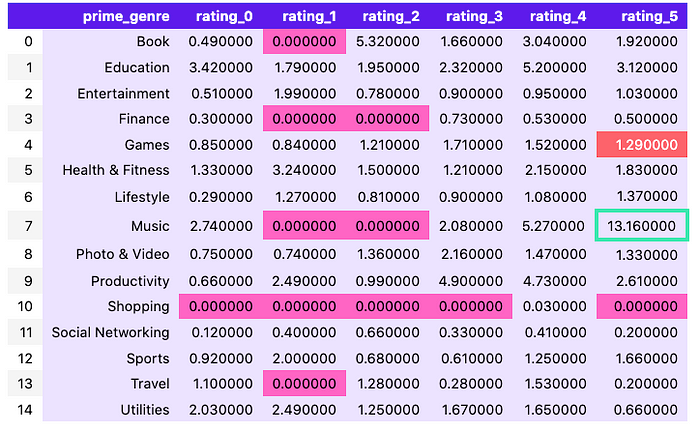
(  
 tmp\_pivot  
 .style  
 .set\_table\_styles([headers, index\_style])  
 .set\_properties(\*\*{'background-color': '#ECE3FF', 'color': 'black'})  
 .set\_properties(\*\*{'background-color': '#FD636B', 'color': 'white'},subset=pd.IndexSlice[4, 'rating\_5'])  
)



**🎨 Style: Setting the background color for max/min values in the dataframe**

Now, we will focus on highlighting the maximum and minimum values in our DataFrame. For this reason, we will assign distinctive background colors to these extreme values, facilitating a quicker and more intuitive understanding of the dataset. The code snippet below demonstrates how to implement this stylistic enhancement.

# select the columns that start with 'rating\_'  
columns = tmp\_pivot.columns[tmp\_pivot.columns.str.startswith('rating\_')]  
# get the max and min values  
max\_value = tmp\_pivot[columns].max().max()  
min\_value = tmp\_pivot[columns].min().min()  
# Establecer el estilo para la celda con el valor máximo  
max\_style = f'border: 4px solid #3BE8B0 !important;'  
# Establecer el estilo para la celda con el valor mínimo  
min\_style = f'background-color: #FF66C4; '  
(  
 tmp\_pivot  
 .style  
 .set\_table\_styles([headers, index\_style])  
 .set\_properties(\*\*{'background-color': '#ECE3FF', 'color': 'black'})  
 .set\_properties(\*\*{'background-color': '#FD636B', 'color': 'white'}, subset=pd.IndexSlice[4, 'rating\_5'])  
 .applymap(lambda x: max\_style if x == max\_value else '')  
 .applymap(lambda x: min\_style if x == min\_value else '', subset=columns)  
)



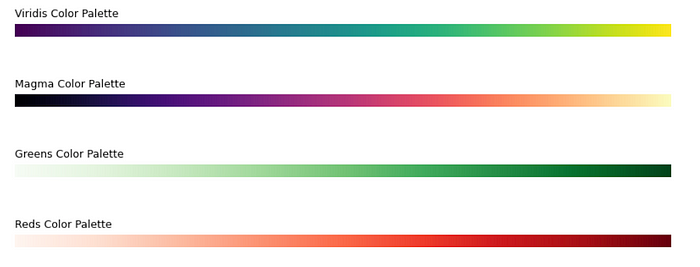
**🎨 Style: Color Background Gradients**

In the upcoming section, we will delve into the concept of color maps, representing a spectrum of colors arranged in a gradient. A colormap, essentially a palette of colors, consists of distinctive denominations, with the most popular ones being \*\*[‘viridis,’ ‘magma,’ ‘Greens,’ ‘Reds’]\*\*.

The primary objective behind creating these color spectrums is to enhance the visual representation of data. Each color in the gradient carries specific nuances, contributing to a more nuanced data visualization experience.

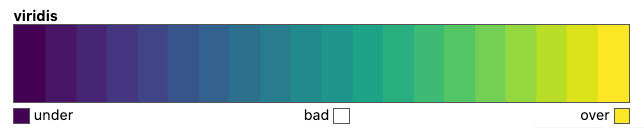
For an extensive array of color options, you can explore the [matplotlib colormaps](<https://matplotlib.org/3.1.0/tutorials/colors/colormaps.html>) link.

import matplotlib.pyplot as plt  
import numpy as np  
# Define the colormap  
for cmap\_item in ['viridis', 'magma','Greens','Reds']:  
 cmap = plt.get\_cmap(cmap\_item)  
 # Create a color gradient  
 gradient = np.linspace(0, 1, 256).reshape(1, -1)  
# Display the color palette  
 plt.figure(figsize=(10, 0.2))  
 plt.imshow(gradient, aspect='auto', cmap=cmap)  
 plt.axis('off')  
 plt.title(f'{cmap\_item.capitalize()} Color Palette', loc='left', fontsize=9)  
 plt.show()

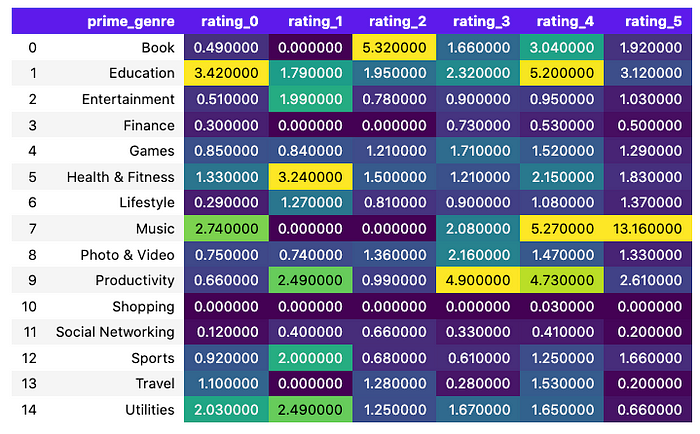


***Viridis palette*** *Now, we will apply a color gradient to our pivot table, allowing you to observe how it is colored using the Viridis palette. In this context, lighter colors signify larger values within the distribution, while darker shades correspond to smaller values in the distribution. This approach provides a visual representation that intuitively conveys the magnitude of the data, making it easier to discern patterns and variations across the dataset.*

plt.get\_cmap('viridis',lut=20)



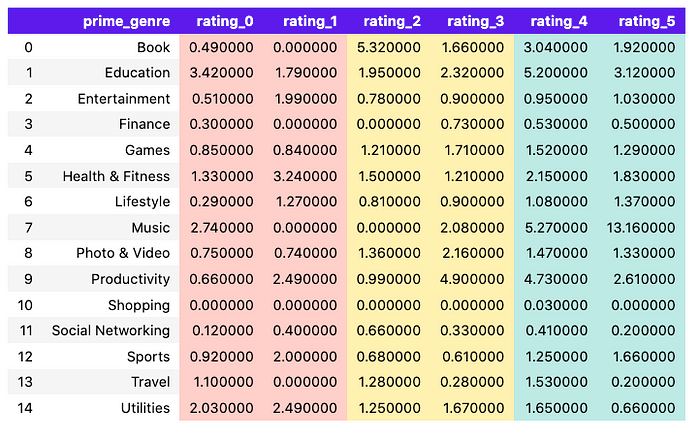
(  
 tmp\_pivot  
 .style  
 .set\_table\_styles([headers, index\_style])  
 .background\_gradient(cmap='viridis',subset=columns)  
)



**🎨 Style: Color Background in columns**

In the next code chunk, we will enhance the visual representation of our pivot table by introducing distinct color backgrounds to specific columns. This technique aids in better highlighting and categorizing data, making it easier to draw insights from the table.

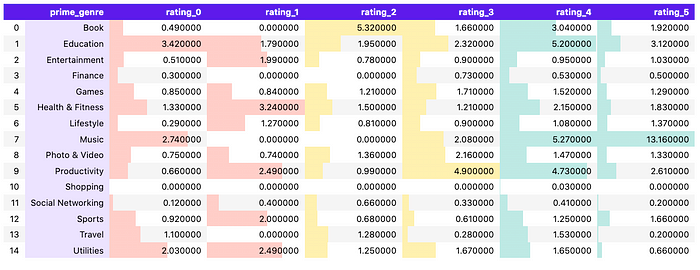
(  
 tmp\_pivot  
 .style  
 .set\_table\_styles([headers, index\_style])  
 .set\_properties(\*\*{'background-color': '#FFCFC9','color':'black'},subset=['rating\_0','rating\_1'])  
 .set\_properties(\*\*{'background-color': '#FFF1B0','color':'black'},subset=['rating\_2','rating\_3'])  
 .set\_properties(\*\*{'background-color': '#BEEAE5','color':'black'},subset=['rating\_4','rating\_5'])  
)



**🎨 Style: Color Bar**

In this section, we will implement the style.bar function to introduce a dynamic color bar into our DataFrame. The color bar provides a visual representation of data values, assigning varying colors to different data ranges.

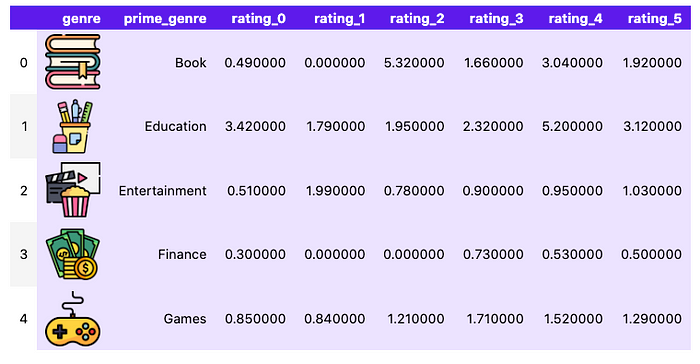
(  
 tmp\_pivot  
 .style  
 .set\_table\_styles([headers, index\_style])  
 .set\_properties(\*\*{'background-color': '#ECE3FF', 'color': 'black'})  
 .set\_properties(\*\*{'background-color': 'white','color':'black'},subset=columns)  
 .bar(color='#FFCFC9',subset=['rating\_0','rating\_1'])  
 .bar(color='#FFF1B0',subset=['rating\_2','rating\_3'])  
 .bar(color='#BEEAE5',subset=['rating\_4','rating\_5'])  
)



**🎨 Style: Image in Columns**

In this section, we explore the enhancement of data representation by adding an image to an additional column. This approach provides an alternative method to elevate the visual impact of the data being presented. These images can serve as icons, represent brands, or convey additional visual elements to captivate and engage the audience.

# create a function to add an image to the dataframe depending on the genre  
def add\_image(image\_name):  
 img\_url = f"img/icons/img\_{image\_name}.png"  
 width = "width: 50px"  
 height = "height: 50px"  
 text\_align ="center"  
 return f'{width};{height}; content: url({img\_url}); text-align:{text\_align}'  
  
# apply the function to the dataframe  
styled\_df = (  
 tmp\_pivot  
 .head(5)  
 .reset\_index()  
 .rename({'index': 'genre'}, axis=1)  
 .style.applymap(add\_image, subset=pd.IndexSlice[:, ['genre']])  
 .set\_table\_styles([headers, index\_style])  
 .set\_properties(\*\*{'background-color': '#ECE3FF', 'color': 'black'})  
)  
  
# display the dataframe with the images  
display(styled\_df)

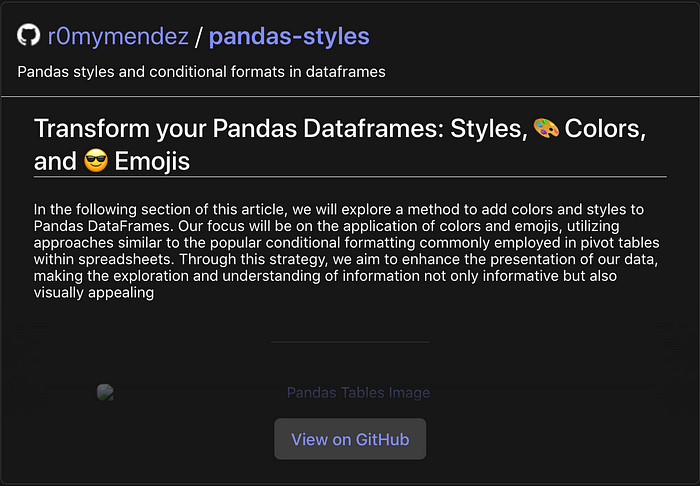


***Disclaimer: Issues with Notebook Cache*** *During the creation of this content, I encountered difficulties related to the notebook cache. Despite making changes to the images, the visualization did not update correctly. Even after attempting to restart the kernel and clear the cell output, the problem persisted. The only effective solution I found was to change the file names of the images, thus avoiding unexpected cache behavior.  
  
It’s important to note that these issues may be specific to the Jupyter Notebooks environment and may not reflect inherent limitations in the code or libraries used. While I tried to address this problem, I did not find a complete solution and opted for an alternative fix by changing the file names.  
  
If you have suggestions or additional solutions, I would be delighted to learn and improve this process.*

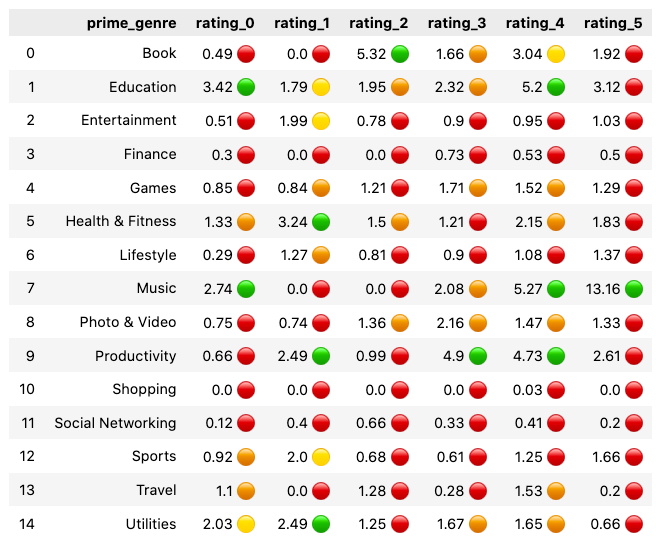
**🎨 Style: Emoji Representation Based on Percentile Values**

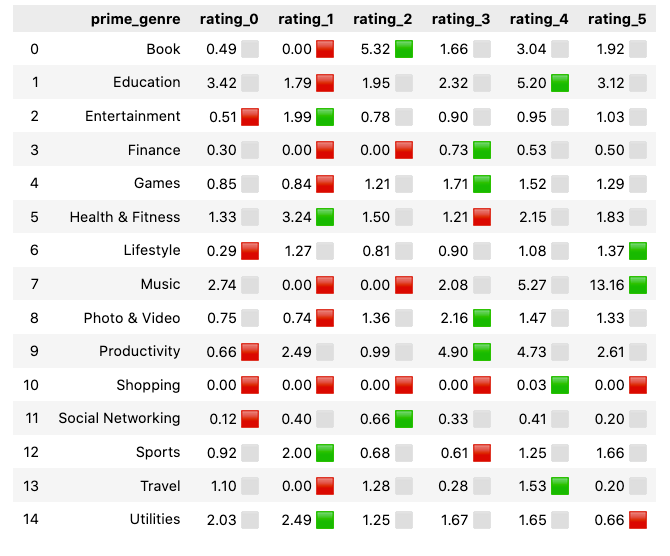
In this section, we delve into the creative use of emojis based on percentile values, offering a distinctive approach to elevate data representation. By incorporating diverse emojis, we enhance the visual impact of the data. Specifically, we employ circles and squads as emojis to bring nuanced expressions to our data points.

If you’d like to view the code for creating this style, it’s available in my [GitHub repository](https://github.com/r0mymendez/pandas-styles/). Feel free to check it out and give it a star if you find it helpful! ⭐️



[GitHub repository](https://github.com/r0mymendez/pandas-styles/)





**📚 References**

If you want to learn…

* [🐼 Pandas Style Documentation](https://pandas.pydata.org/docs/user_guide/style.html)

[Data Science](https://medium.com/tag/data-science?source=post_page-----bf938d6e98a2---------------data_science-----------------)

[Data Visualization](https://medium.com/tag/data-visualization?source=post_page-----bf938d6e98a2---------------data_visualization-----------------)

[Technology](https://medium.com/tag/technology?source=post_page-----bf938d6e98a2---------------technology-----------------)

[Python](https://medium.com/tag/python?source=post_page-----bf938d6e98a2---------------python-----------------)

[Programming](https://medium.com/tag/programming?source=post_page-----bf938d6e98a2---------------programming-----------------)