

Abstract-ify

<https://tinyurl.com/abstractify>

Our project is in the form of a Shiny Application which takes as input an image file from the user. From there, it simplifies the image into a user-selected number of distinct colors and outputs the abstract image, the image outline, color analytics, and custom color palettes.

Introduction

Everyone has the ability to express themselves through art, but some of us lack the technical skills to create an aesthetically-pleasing piece. Novice artists can use paint-by-numbers as an elegant solution - one can practice their artistic skills while still creating something that elicits pride. Unfortunately, paint-by-numbers has a major flaw - the need to use someone else's template restricts the imagination of the artist and limits the possible subjects that one can create. What if an artist could choose their own image and transform it into a personal paint-by-numbers kit with a set number of colors and an outline? Starting from this motivation, we started investigating R's ability to manipulate images and the colors within them. We are really excited by the idea of simplified images using only a concrete number of colors. Additionally, we like the concept of creating outlines of images for recoloring, or enjoying as is. Blank outlines create endless opportunities for creativity and experimentation using different hues, shades, mediums and more. Finally, we wanted to see what we could do once we extracted the simplified colors. In class, we have frequently discussed how to choose an appropriate color palette for a particular data visualization and were exposed to many packages that provide such palettes. We are especially excited by the ability to add a custom color palette for possible data visualization applications. Our project is in the form of a Shiny Application which takes as input an image file from the user. From there, it simplifies the image into a user-selected number of distinct colors and outputs the abstract image, the image outline, color analytics, and custom color palettes.

In order to showcase our color palette, we use visualizations constructed from three datasets used previously in the course: the YouGov brexit.csv survey results, the BA_degrees awards data, and the TidyTuesday dog_travel.csv dog adoption figures. In each of these datasets, we use the same variables and plot types as the class has used in past homeworks. As sample images, we have included four head shots of our team, an abstract art piece, and a Simpsons image.

Image Processing

Paint-by-numbers was invented in the early 1950s and relied upon very abstract art styles that required few distinct colors. As the public became more demanding - seeking templates for famous pieces and more realistic styles - the color requirements began to grow as well. Even today, paint-by-number kits struggle to show both high levels of detail and realistic low-contrast color shifts in the same kit without an unwieldy number of distinct shades. Our application approaches the paint-by-numbers process with color minimization as a priority - the number of desired colors is a user-input and has a maximum of ten. Our choice to minimize colors means that our application cannot preserve all of the fine detail in realistic images and thus creates a more abstract version of the original. For demonstration, we will be using this image of Marge Simpson.



Color Simplification

In order to reduce the image to the selected number of colors, we employ k-means clustering to find the most significant colors in the image. The `colordistance` package includes a helpful wrapper function, `getKMeanColors()`, that tailors base R's `kMeansCluster()` function for clustering pixels. The `extractClusters()` function then operates on the k-means object and returns a dataframe wherein each row is a color cluster with the RGB values and the size of the cluster. `getKMeanColors()` also returns a one-dimensional vector with values ranging from 1 to the number of clusters; this maps individual pixels to their cluster. We then perform a `left_join()` to match the numbered clusters in this vector with their R, G, and B values, concatenate these three components, and use base R's `matrix()` function to turn this into the image that is displayed. While we are able to make heavy use of `colordistance` to extract an image's colors and map the clustered colors to each pixel, the process of creating the outlines is done through our own set of functions. Our example image was created using 6 colors.



Outline Creation

Outline creation is performed via the `outline_func()` and `plot_outline()` functions. The former is given the k-means color cluster object from earlier and the pixel dimensions of the original image. From there, `outline_func()` transforms the cluster vector into a matrix with the original's dimensions. `plot_outline()` operates on this matrix differently depending on size - small images that lack fine detail are essentially rounded to either black or white. Larger images are passed to a more extensive `boundary_func()` function that iterates through each pixel. If the pixel next to a pixel from a different color cluster, it is colored black to denote a boundary between distinct colors. Otherwise, each pixel is white.



Color Analytics

Our Color Palette Info tab contains three figures of interest - a Colorspace plot of the original image's RGB values in 3D, a reactive table with the clustered colors, and an R-Squared figure for the simplification process. The RGB figure is constructed via

Hex Code	Percentage of pixels	Nearest named color
#FFFFFF	82.1%	White
#CFDC9D	6.8%	Yellow-green (Crayola)
#0E77BA	5.3%	Star command blue
#F5CC24	2.9%	Jonquil
#161412	1.8%	Black chocolate
#9C6C48	1.1%	Liver chestnut

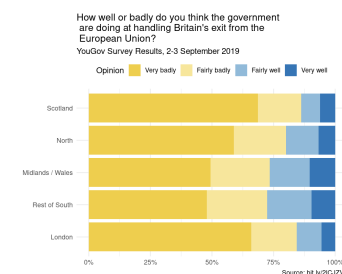
The goodness of fit, or R-Squared, associated with this pixelation is 97.7%

the `colordistance` wrapper function `plotPixels()` applied to the original image. Diagnostically, the RGB plot allows us to see the distribution of color in the original image and offers insight into how many clusters are required to adequately process the original. Contrastingly, the reactive table shows the color palette after our simplification process. For each clustered color, the nearest named color column calculates the Euclidean color distance between it and each of 898 colors in a named list that we downloaded from Wikipedia (ie. the sum of the squared distances between respective RGB components). Each color's closest match is displayed here, which helps users remember their desired colors more easily than with hex codes alone. The R-squared value is the link between the two and serves as a metric of how well the original image's colors are captured by the chosen number of clusters. Increasing the number of clusters will always increase R-Squared, but there are diminishing returns and performance losses for doing so. Our example images are best simplified with a minimum of three colors, but show only small improvements for using more than six distinct hues.

Abstract-ified Plot

Custom Palettes & Example Visualizations

In this tab, users can visualize the set of extracted colors in various sample plots and compare to commonly used packages for color palettes in R. The header of this page displays the selected hex codes separated by quotation marks and commas for easy insertion into a color or fill scale in `ggplot()`. There is also a button to copy this text to the clipboard directly. The body of this page contains a series of similar plots, each only differing by color palette. Users can click the radio buttons to toggle through three plots, each labeled by the type of color scale most appropriate for visualizing the data: "Categorical", "Sequential", and "Diverging." For sequential and diverging plots, users also have the option to choose which colors from the palette should represent the low and high ends of the color scale on the plot. The default values for these ends are the lightest and darkest hex codes from the extracted palette, respectively, as determined by the sum of the RGB components. For sequential plots, they can also adjust the darkness of the gray "Other" category. After manipulating these values, users can compare the generated plot against the default palettes from commonly used color packages in R. For all plots, these include base `ggplot2` colors, the `RColorBrewer`



package, and the colorspace package. The viridis package is also displayed only for categorical and sequential plots, as there is no default diverging color scale from the viridis package.

Discussion

We were originally inspired to create our application by paint-by-numbers, but as the project developed we began to see the difficulty of creating image outlines that could actually be filled by a human - often, the drawn regions are so small and numerous that a number couldn't even be displayed in all of them. We have identified three common features or patterns in an image that can produce poor simplifications and outlines: overall lack of contrast, fractal-like patterns, and blur effects. Low contrast photos have color clusters that are closer together and thus harder for our k-means process to distinguish. Fractal patterns are processed normally, but tend to create snakey, unfillable outlines that would not be practical for a paint-by-numbers. Blur effects are very poorly handled by our approach - since blurring both destroys boundary information and widens the spectrum of colors, it causes poor performance in both our color simplification and outline functions. Aside from specific patterns, our application is also severely limited by image size - processing time and RAM needs increase rapidly as the resolution of the user image rises. In order to cap runtimes to a reasonable level, we recommend using images that are less than 200KB. The final limitation of our application is that it does not produce its outlines deterministically. Each time that one varies the number of distinct colors on a small image, the simple boundary function will return outlines with significant changes. On large images, however, the outlines are much more stable and tend to emphasize the same features in the original image. The discrepancy lies in the difference between the two outlining functions - larger images contain enough information to create outline boundaries based on color shifts between pixels while small images must predict the boundaries based on the individual pixel values.

References

Data

The following data sources were used, following inspiration from class projects.
State-list.csv, created using R's built-in state.abb and state.name objects

[Brexit Data](#)

[Degrees Data](#)

[Dog Travel Data](#)

Code

[Named list of colors](#)

[bw_colors\(\) white/black cutoffs](#)

[Image download in Shiny](#)

Images

[Block Art](#)

[Marge Simpson](#)

Other sample images from personal use.

Contextual Info

[Paint-by-numbers](#)

APPENDIX

App Layout

Our app uses a sidebar with an image upload and a slider input for the user's preferred number of colors

Home Tab

Here, the original image is shown as well as brief instructions

Abstract-ify

Click "Browse..." to replace the default image with one of your own

Number of colors: (Most images look best with 3-5!)

1 5 10

1 2 3 4 5 6 7 8 9 10

Upload an image:*

Browse... No image selected

*Please limit photo size to <200 KB. If needed use [this website](#) to shrink your photo.

Or

Choose an Image

Marge Simpson


Home Simplified Output Outline Color Palette: Info Color Palette: Plotting

About

Welcome to Abstract-ify

How to use: select one our preset images or upload one of your own. Use the slider bar to select how many colors you would like to see in your simplified image. You can explore the various tabs to see the simplified version of your image and the outline version. You can also see some sample color palettes to use in ggplot!

Your image:



Simplified Output

Abstract-ify

Click "Browse..." to replace the default image with one of your own

Number of colors: (Most images look best with 3-5!)

1 5 10

1 2 3 4 5 6 7 8 9 10

Upload an image:*

Browse... No image selected

*Please limit photo size to <200 KB. If needed use [this website](#) to shrink your photo.


Or

Choose an Image

Marge Simpson

Home Simplified Output Outline Color Palette: Info Color Palette: Plotting

About



Download modified image

Here, the image with clustered colors is shown with a download option.

Abstract-ify

Click "Browse..." to replace the default image with one of your own

Number of colors: (Most images look best with 3-5!)

1

5

10

12345678910

Upload an image:*

Browse...

No image selected

*Please limit photo size to <200 KB. If needed use [this website](#) to shrink your photo.

Or

Choose an Image

Marge Simpson

Home


Simplified Output

Outline

Color Palette: Info

Color Palette: Plotting

About



Outline

Here, the image's abstract outline is shown.

Color Palette: Info

Abstract-ify

Click "Browse..." to replace the default image with one of your own

Number of colors: (Most images look best with 3-5!)

1

5

10

12345678910

Upload an image:*

Browse...

No image selected

*Please limit photo size to <200 KB. If needed use [this website](#) to shrink your photo.

Or

Choose an Image

Marge Simpson

Home

Simplified Output

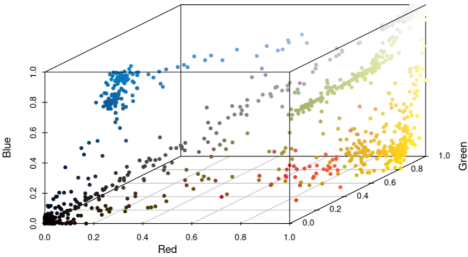
Outline

Color Palette: Info

Color Palette: Plotting

About

Colorspace Plot



Hex Code	Percentage of pixels	Nearest named color
#FFFFFF	82.1%	White
#CDDA9B	7%	Yellow-green (Crayola)
#1077B8	5.4%	Star command blue
#F0BE26	3.3%	Orange-yellow
#221E18	2.1%	Eerie black

The goodness of fit, or R-Squared, associated with this pixelation is 96.9%

Here, we present a three dimensional RGB plot of the original pixels, a reactable table of the clustered colors, and a goodness of fit metric (R-Squared).

Color Palette: Plotting

Abstract-ify

Click "Browse..." to replace the default image with one of your own

Number of colors: (Most images look best with 3-5!)

1 2 3 4 5 6 7 8 9 10

Upload an image:

Browse... No image selected

*Please limit photo size to <200 KB. If needed use [this website](#) to shrink your photo.

Or

Choose an image

Marge Simpson

Home Simplified Output Outline Color Palette: Info Color Palette: Plotting About

On this page, you can test out the color palette generated from your image in-use in `ggplot()`.

Paste the hexcodes to recreate the palette.

(If you have chosen alternative "low" or "high" colors for the Sequential or Diverging plots, you may need to reorder the hex codes for use.)

"#221E18", "#1077B8", "#F0BE26", "#CDDA9B", "#FFFFFF"

Copy to Clipboard

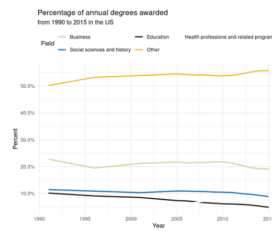
Choose Plot Type:

☒ Categorical

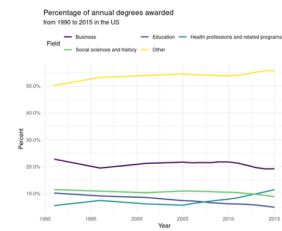
☐ Sequential

☐ Diverging

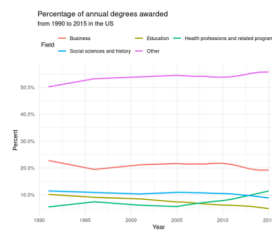
Abstract-ified Plot



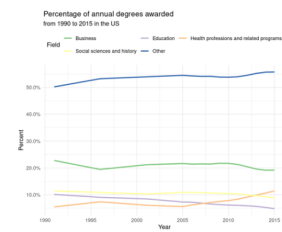
Viridis Plot



Base color Plot



RColorBrewer Plot



Colorspace Plot



Here, we show a set of five identical plots with color palettes from base ggplot2, common palette packages, and our own Abstract-ified custom palette. These five plots are reactive to the plot type buttons and adjust their source data and plot type to fit the needs of the chosen palette.