Creating a Cross-Stitch From an Image

James F. Kanu

02/11/2020

```
source("functions.R")
```

Introduction to functions.R

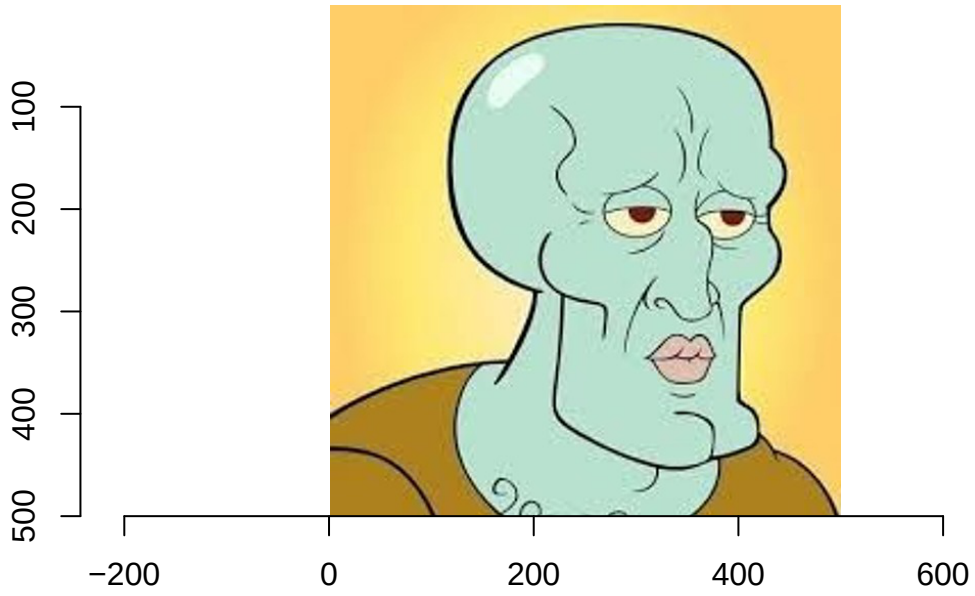
functions.R is a simple-to-run R file containing four primary functions used to create a cross-stitch from an image of your choosing. By using k-means clustering, our functions will pick out the main colours used in our image by grouping each pixel into a cluster. The number of cluster centres, or k, will be chosen by us, and will in turn be the number of DMC threads used to cross-stitch the image.

Processing an Image

Before we get into the 'knit-y gritty' of the functionality, let's first choose a picture; ideally with a focal interest and a simplistic range of colours. Although we will be using clustering to simplify the image, as most images often have too many colours to realistically cross-stitch, starting off with a simple image may help with a cross-stitch that better represents the original picture.

Here, I have chosen a simple picture of an objectively handsome model.

```
library(imager)
library(dplyr)
im <- imager::load.image("squid.jpg")
plot(im)
```



Now that we have our picture, the initial function that must be run to use the other functions is process_image(). This function takes in the image name and a vector of numbers that we think could be a good value for k to simplify the image. Using this data, it will turn our image into a dataframe and run k-means clustering using each entry in our vector of numbers.

Here, I used the numbers from 2 to 10 as possible number of cluster centres.

```
cluster_info <- process_image("squid.jpg", k_list = c(2:10))
```

What is returned is cluster_info: a list of two indices. The first index contains a tibble of the kmeans and tidied centres, with DMC colour information attached, of each of the entries in our k_list.

```
cluster_info[[1]]
```

```
## # A tibble: 9 x 3
##       k kclust   centres
##   <int> <list>   <list>
## 1     2 <kmeans> <tibble [2 x 8]>
## 2     3 <kmeans> <tibble [3 x 8]>
## 3     4 <kmeans> <tibble [4 x 8]>
## 4     5 <kmeans> <tibble [5 x 8]>
## 5     6 <kmeans> <tibble [6 x 8]>
## 6     7 <kmeans> <tibble [7 x 8]>
## 7     8 <kmeans> <tibble [8 x 8]>
## 8     9 <kmeans> <tibble [9 x 8]>
## 9    10 <kmeans> <tibble [10 x 8]>
```

The second index contains the image data of our model picture, with RGB values for each associated x and y coordinate. Here, I only displayed the first 6 rows as this dataframe contains 250,000 rows.

```
head(cluster_info[[2]])
```

```
##   x y R         G         B
## 1 1 1 1 0.8078431 0.4078431
## 2 2 1 1 0.8078431 0.4078431
## 3 3 1 1 0.8078431 0.4078431
## 4 4 1 1 0.8078431 0.4078431
## 5 5 1 1 0.8078431 0.4078431
## 6 6 1 1 0.8078431 0.4078431
```

Drawing a Scree Plot

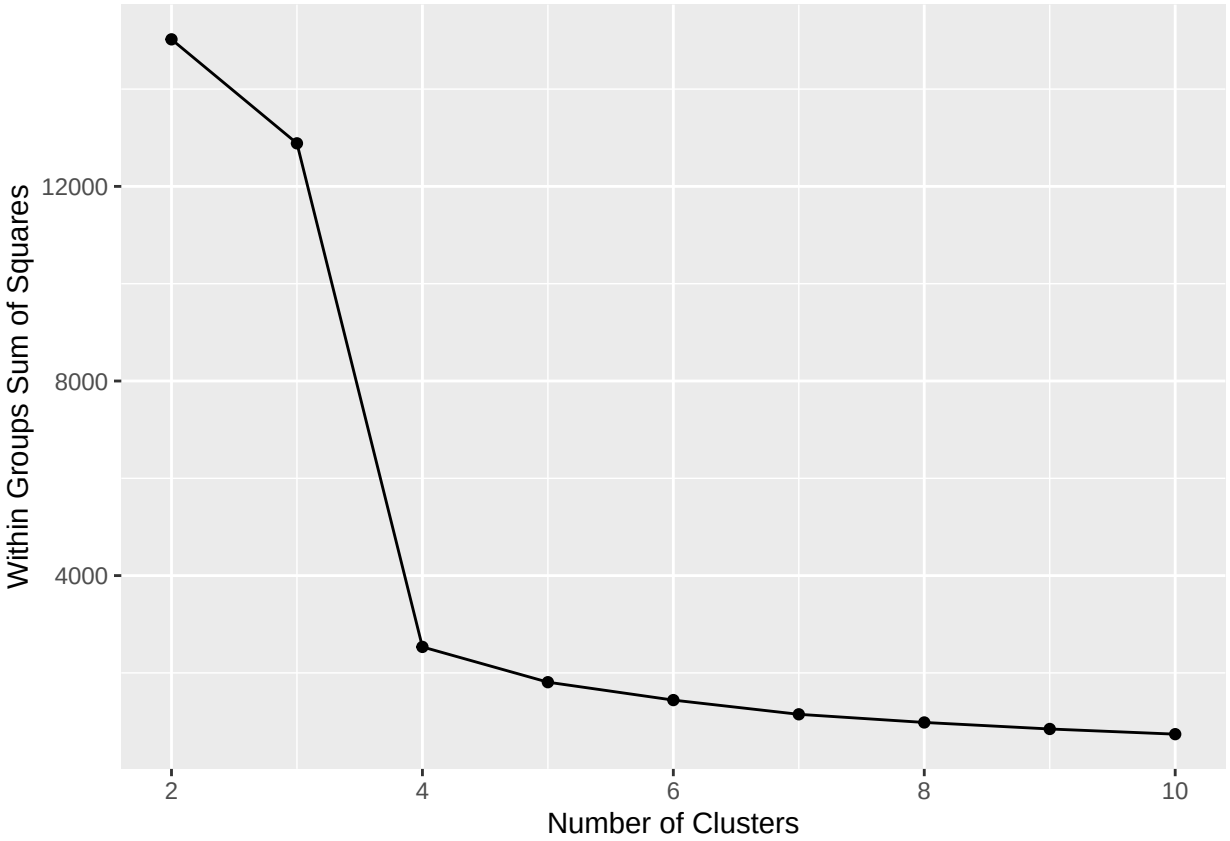
Now that we have some data to work with, we can use some of our other functions.

Having used a list of potential 'k's in our initial function, we can choose the best number of clusters centres from this list with a scree plot.

A scree plot plots the k-means objective function as a function of k. Since adding more colours would improve our approximation of the original picture, this plot will always be decreasing. However, it will hit a point where we won't benefit as much by adding more colours/cluster centres.

Let's see how our k_list looks on a scree plot.

```
scree_ex <- scree_plot(cluster_info)
scree_ex
```



In our scree plot, we want to choose a k that sits at the "elbow" of the plot. In other words, where we think that the plot stops reducing drastically and stabilizes. There are massive drops between 2 to 4, and slightly smaller incremental drops after.

It looks like the line stabilizes after 8, so for the rest of our walkthrough, lets use 8 as our value for k.

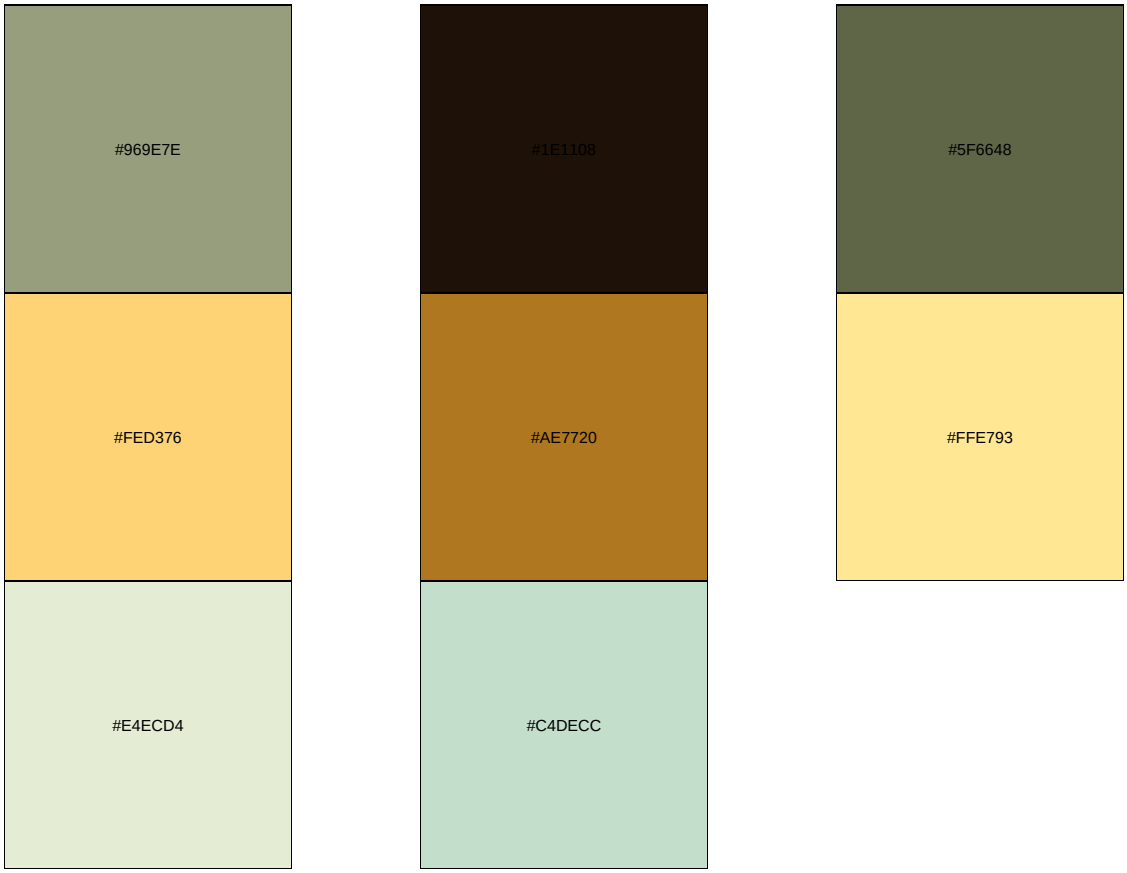
Now let's see what colours would be associated with this number of clusters.

Making Colour Strips

Using colour_strips(), another function using cluster_info, we can view the colours chosen by our clustering. The function uses the hex code of the DMC thread colours most associated with the colours in our picture, plotted in a colour strip. colour_strips() returns a list with each index corresponding to an entry in our original number vector.

Since 8 was the seventh entry in our k_list, we'll look at the seventh index of the returned list.

```
strip_ex <- colour_strips(cluster_info)
strip_ex[[7]]
```



Here are the colours that were chosen from our clustering with 8 centres. Remember, these hex codes aren't the ones used in our original image, but rather their closest approximation using DMC thread colours.

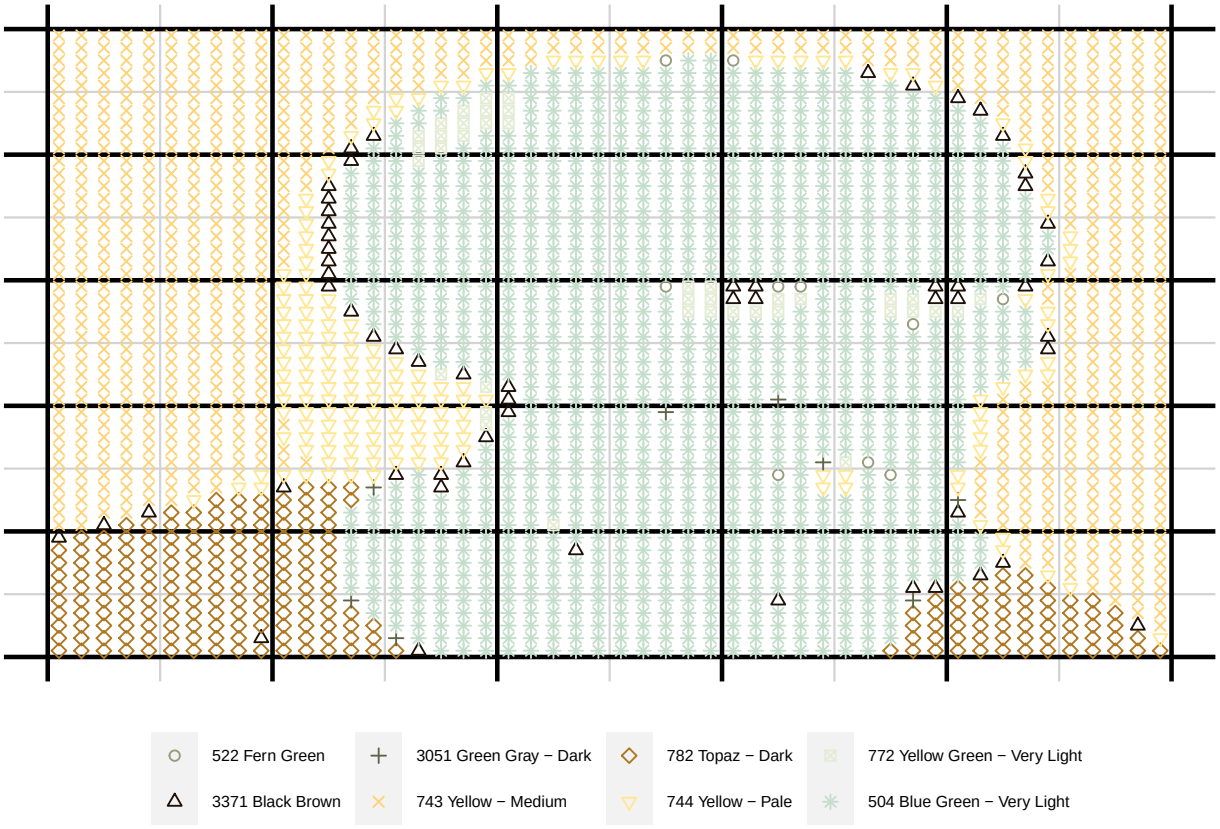
Scree plots can be misleading so it may be helpful to see our colours visualized and determine if there aren't enough colours to represent our image (our k is too small) or if there are colours that are too similar and may be unnecessary (our k is too large).

We see that there are two colours representing the background and another pair of similar colours. This may indicate that we can reduce our value of k, but let's stick to this number for now, as I want to keep some our model's facial details.

Stitching it All Together

Finally, let's make our cross-stitched image. The function make_pattern will create our image using cluster_data and a few more parameters. We will need to input our chosen number of clusters, as well as the number of stitches we want to use for the horizontal direction of the picture (the number of vertical stitches will be scaled accordingly). There are two additional optional parameters that are set to FALSE and NULL automatically, but first let's see what the function creates.

```
#Here I use 50 stitches for the horizontal direction.
stitch <- make_pattern(cluster_info, 8, 50)
stitch
```

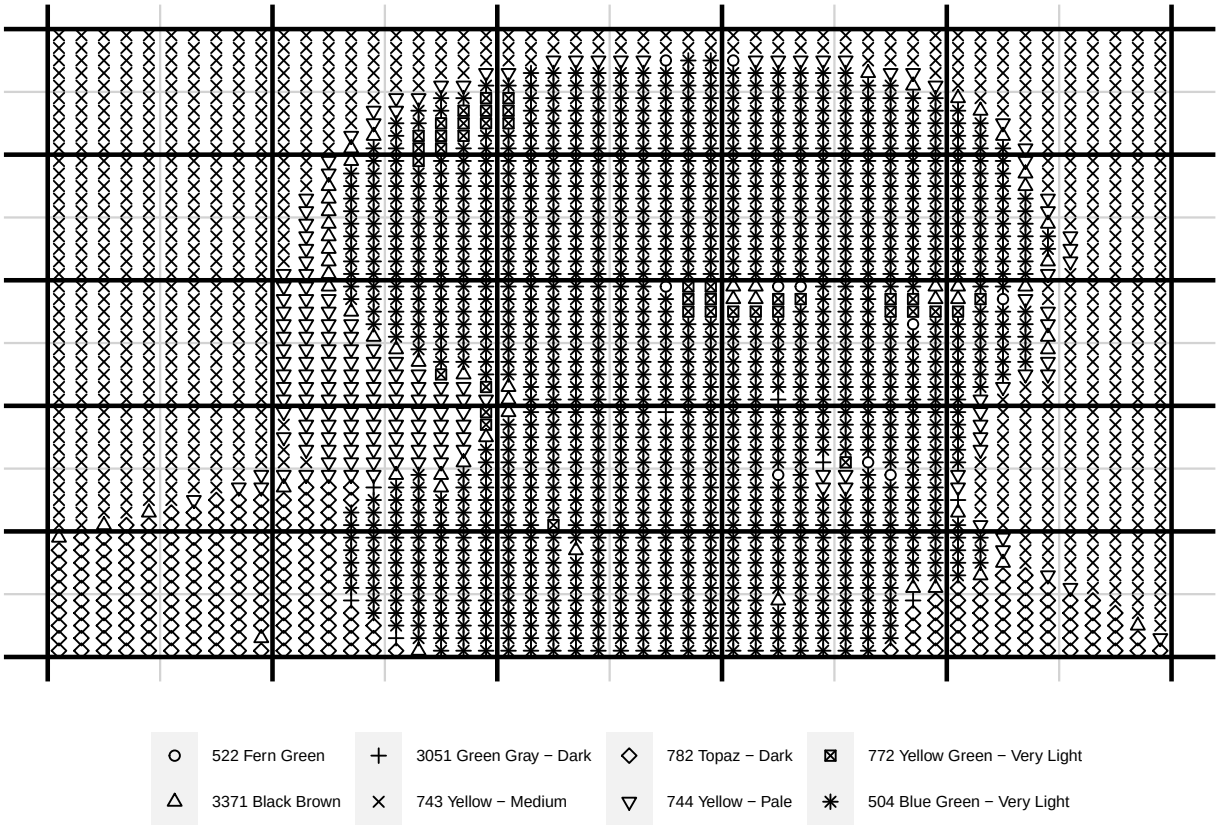


There's our handsome model! As we can see, each point in the stitch has a colour and shape associated to a DMC thread, and along with the added grid, this makes for a great cross-stitch guide.

Let's see what our additional parameters do.

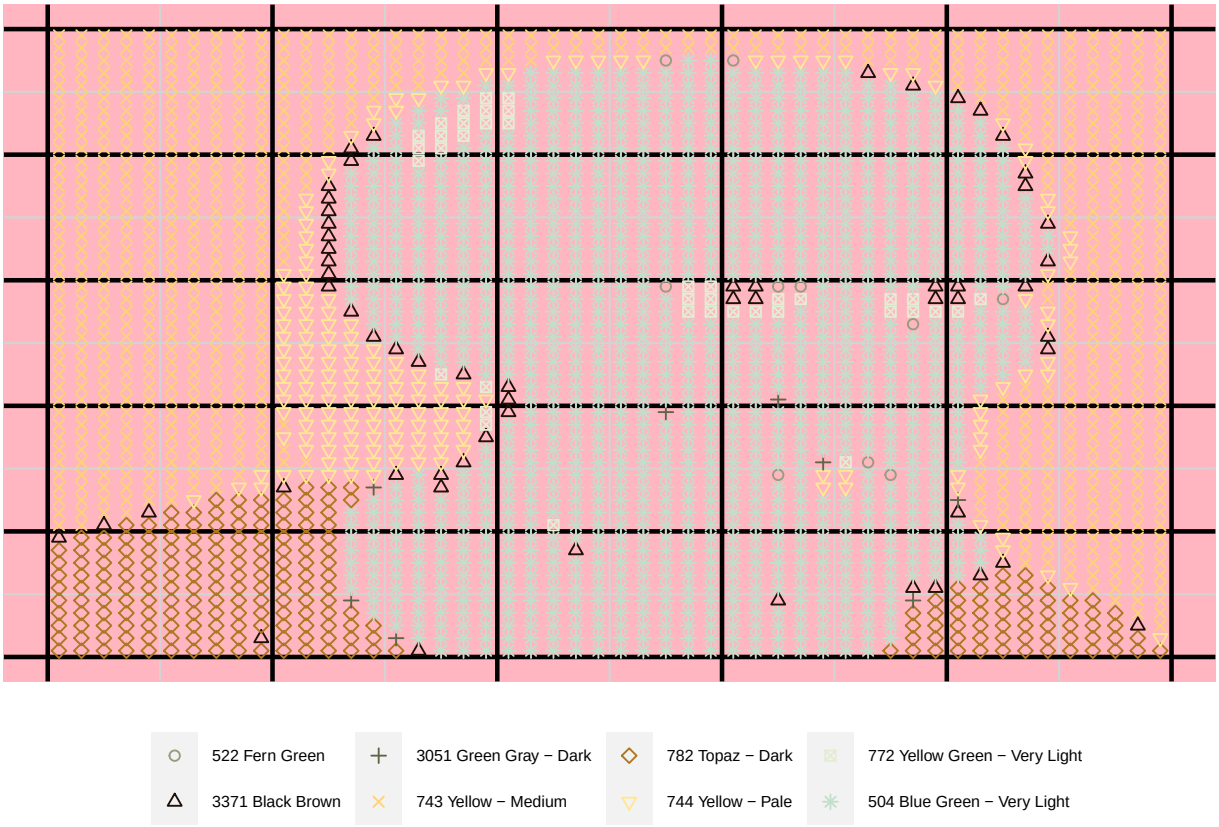
We can turn off colour and make our guide black and white by setting the black_white parameter to TRUE.

```
stitch <- make_pattern(cluster_info, 8, 50,black_white = TRUE)
stitch
```



We can also change the color of the background by setting the background_colour parameter to a hexcode or colour name.

```
stitch <- make_pattern(cluster_info, 8, 50, background_colour = "light pink")
stitch
```



And there you have it: you now know how to use functions.R. Experiment with your value of k using information from scree_plot and colour_strips; you may find it easier to stitch a pattern with more or less colours than your initial choice of k.

Happy Stitching!