Vignette

Mahrukh Niazi 1003948204

11/3/2020

```r
set.seed(1)

source("functions.R")
```

This vignette covers how to produce a cross-stitch pattern for a chosen image. The basis of this process is performing k-means clustering analysis on an image data set to partition the image into different regions of pixels of similar attributes in order to represent the image in a cross-stitch pattern. This process is broken down into four main functions as explained below. A picture of Iron Man will be used to demonstrate the process.

Figure 1: Iron Man

Firstly, the function `process_image()` is used to perform k-means clustering on the chosen image for a set of initial k number of cluster centers that the user chooses. It takes `image_file_name` and `k_list` as inputs, where `k_list` is the set of initial cluster centers the user chooses. A set of k values is given in order to determine how many cluster centers are needed (i.e. determine the best k value). For the Iron Man example the function is used as follows: `process_image("iron man.jpg", k_list = c(2, 5, 8, 10))`. The image is loaded into the function and since it is a `cimg` object, it needs to be converted into a dataset that can be used for clustering. This is done through the `as.data.frame()` function and is structured with coloumns `x`,

`y`, `R`, `G`, `B`, where `(x, y)` are the coordinates of the pixel values and `RGB` are the corresponding RGB values for the pixel values.

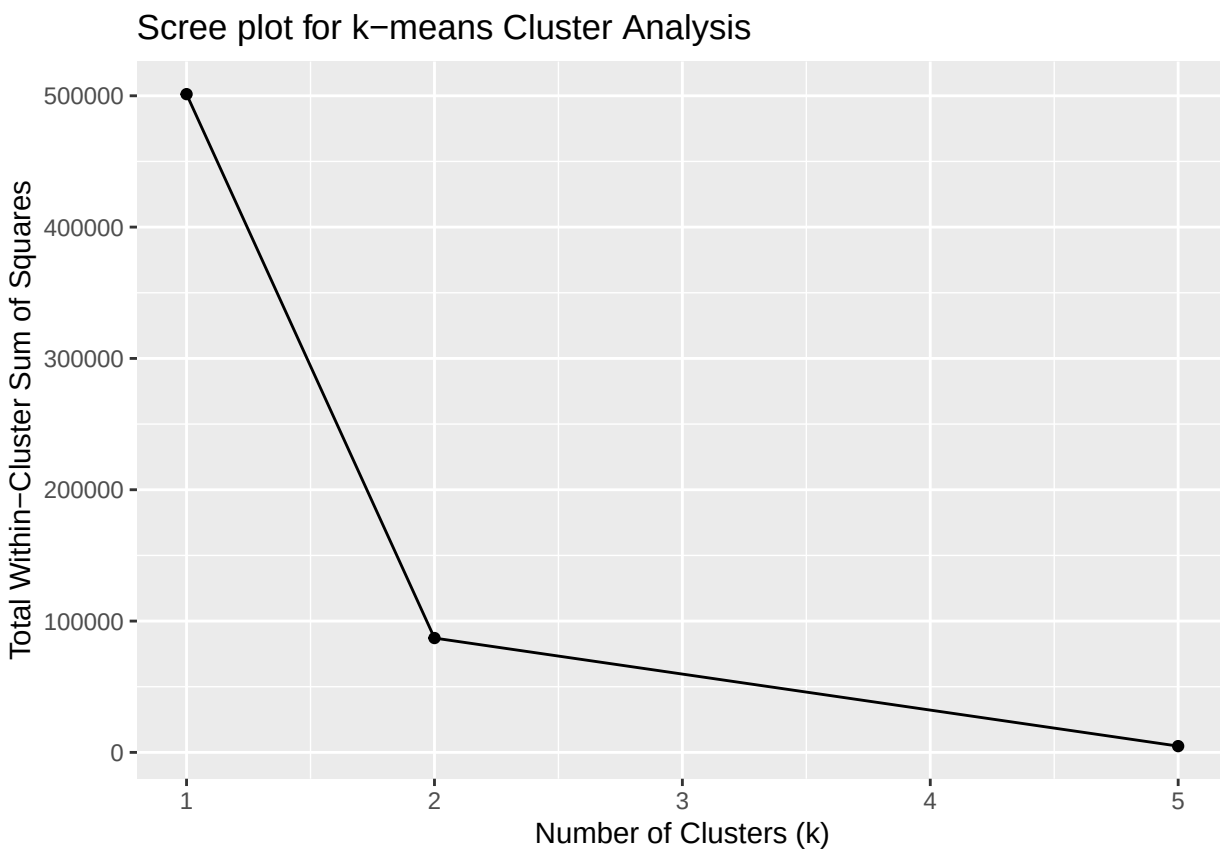
Once the data frame is formed, `process_image` computes k-means clustering for each k value in `k_list`, such that there is k-means clustering output for `k = 1`, `k = 2`, and `k = 5`, each with `nstart = 4` so that 4 initial random centroids are generated for each k and the initial configuration that best minimizes the cost function is selected. A higher value of `nstart` was not used as the code would take long to run. `process_image` also computes the tidied cluster centers for better data structure and analysis, the associated RGB values for the cluster centers, and the nearest DMC thread color information for the cluster centers, all for each k. `process_image` compiles this information, along with the values of k in `k_list` and the image data frame, into a large list. The output of `process_image` is assigned as `cluster_info` to be used in subsequent functions: `cluster_info <- process_image("iron man.jpg", k_list = c(1, 2, 5))`.

```
cluster_info <- process_image("iron man.jpg", k_list = c(1, 2, 5))
```

`cluster_info` now stores the k-means output for the different values of k, which can be used to chose the most best k (number of clusters) value. This is determined using the functions `scree_plot()` and `color_strips()`.

The `scree_plot()` function determines the best k value by plotting the k-means objective function as a function of k. This function plots the total within-cluster sum of squares values obtained from the k-means output for each k in `cluster_info` against the values of k, which is also stored in `cluster_info`. For the Iron Man example, the function is used as follows: `scree_plot(cluster_info)` and produces the following screeplot:

```
scree_plot(cluster_info)
```



The `scree_plot(cluster_info)` output shows that the elbow of the curve is at k = 8. This indicates that k = 5 is possibly the most optimal value as most of the variation is explained at this point. The plot shows

that adding more clusters beyond this 5 does not better model the data and instead may lead to over fitting.

To confirm whether k = 5 is the best value, the `color_strips()` function provides a visual representation of the clusterings, which provides further insight on the best value of k. `color_strips()` uses the DMC color information for each k stored in `cluster_info` and plots the hex codes of the DMC values to produce color strips for each k. For the Iron Man example, the function is used as follows: `cluster_info(cluster_info)` and produces the following plots:

```
color_strips(cluster_info)
```

These plots show for k = 5 there are 5 distinct color shades so k = 5 is the best value of k out of the given list.

Based on the chosen k, the cross-stitch representation of the image can be produced using the `make_pattern()` function. `make_patter()` filters through `cluster_info` to extract the chosen k value's k-means output. This output along is augmented with the original image data frame, which is also stored in `cluster_info`, using `augment()` to produce a dataframe that is inputed in the `change_resolution()` function. Since, most images have too many pixels to create a realistic cross-stitch, the `change_resolution()` function is used to generate a lower resolution of the image. In the process, the `change_resolution()` function can drop a small cluster from the augmented data frame because it is not the most common cluster. `make_pattern()` then uses this low resolution image dataframe and the `x_size` argument to create its cross-stitch pattern using `ggplot()` complete with a legend that has the DMC thread color name and number for each cluster, and a guide grid. The `x_size` argument refers to the total number of possible stitches in the horizontal direction and is usually given the `x_size = 50` since higher values make the code run for long. Based on the `black_white` input given in `make_pattern()`, the cross_stitch pattern can be outputted in black and white (`black_white = TRUE`) or in color (`black_white = FALSE`). Additionally, the user can omit the background color of the cross-stitch by inputting the hex code of the corresponding background color in the `background_color` argument in `make_pattern()`. For the Iron Man example, the function is used as follow to produce the various cross-stitch patterns:

```
no_background_color <- make_pattern(cluster_info, k = 5, x_size = 50, black_white =
FALSE, background_color = NULL)
```

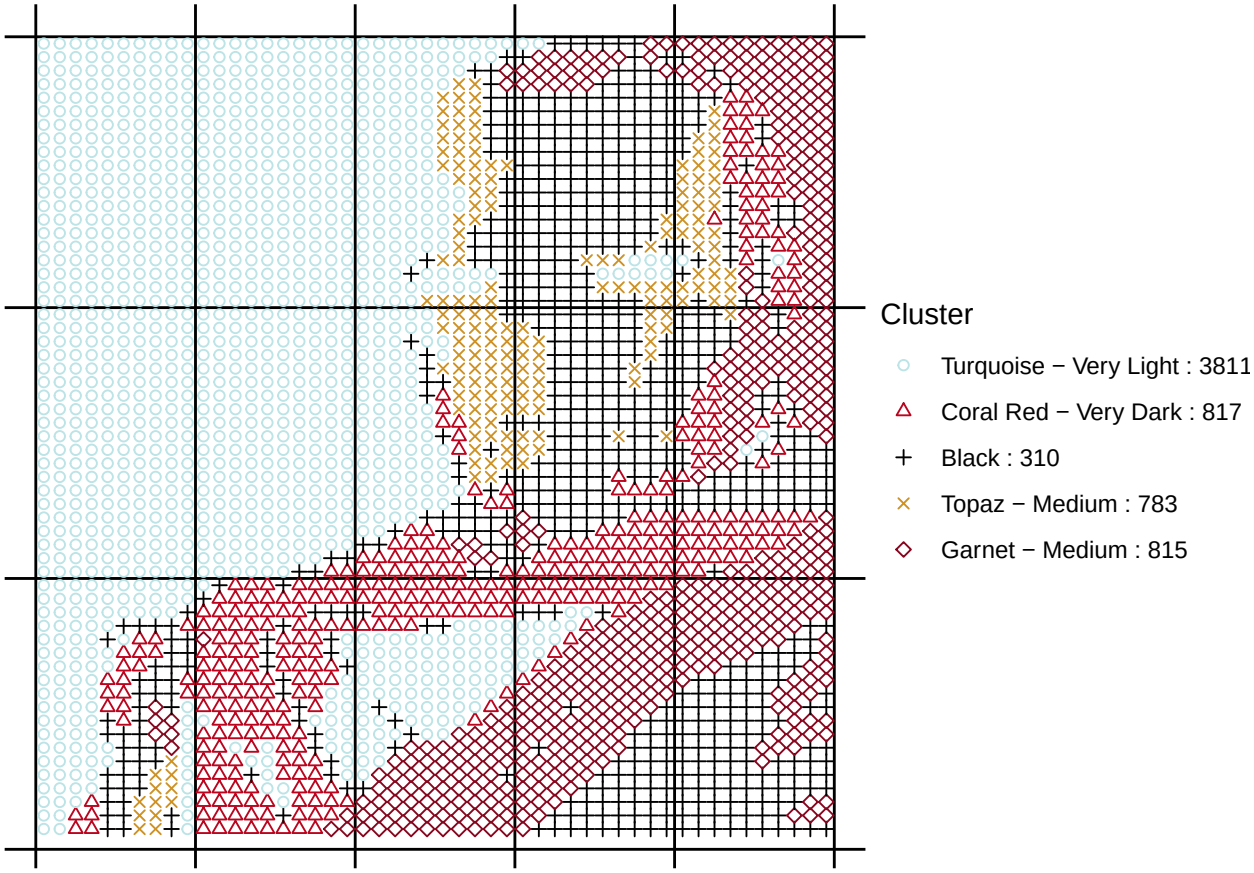
```
no_background_no_color <- make_pattern(cluster_info, k = 5, x_size = 50, black_white =
TRUE, background_color = NULL)
```

```
background_colour <- make_pattern(cluster_info, k = 5, x_size = 50, black_white = FALSE,
background_color = "#BCE3E6")
```

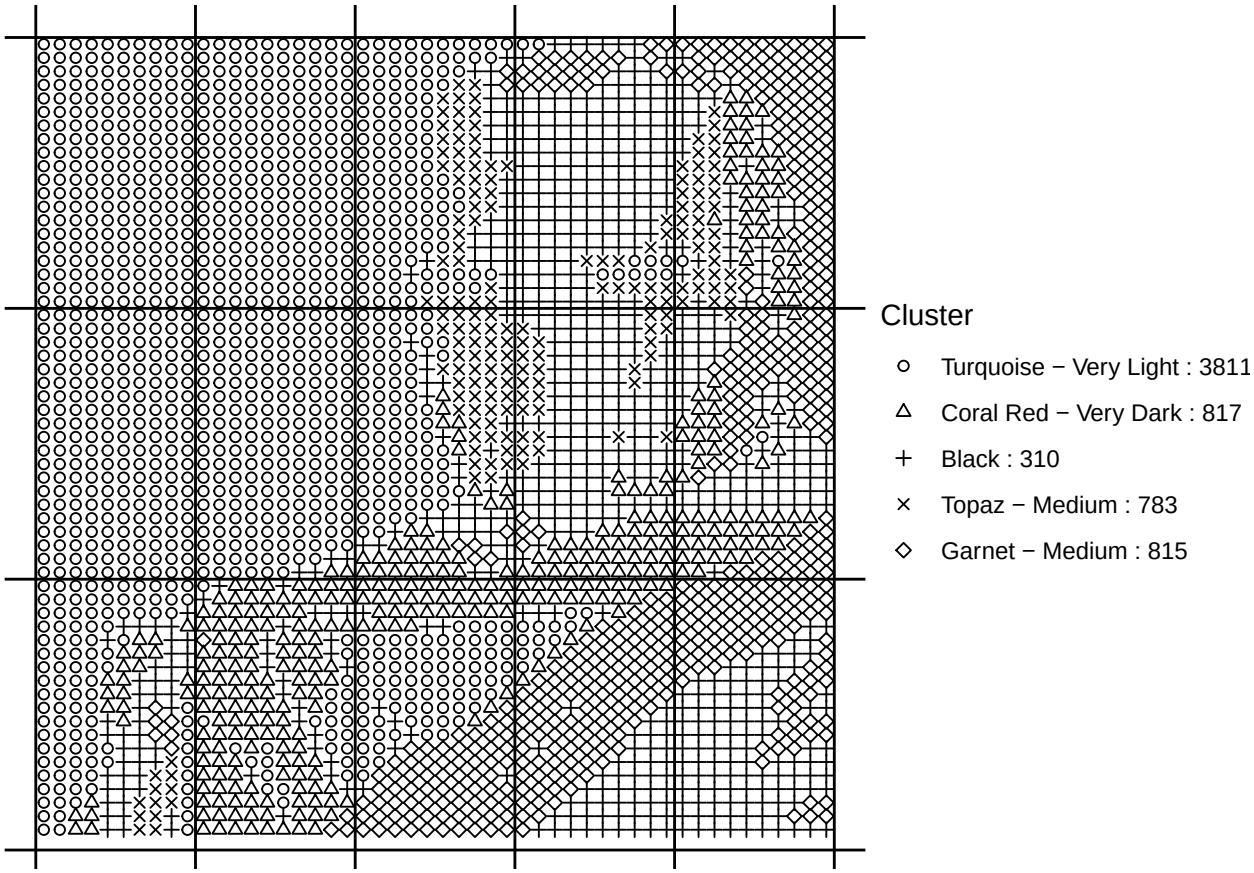
```
background_no_colour <- make_pattern(cluster_info, k = 5, x_size = 50, black_white =
```

```
TRUE, background_color = "#BCE3E6")
```

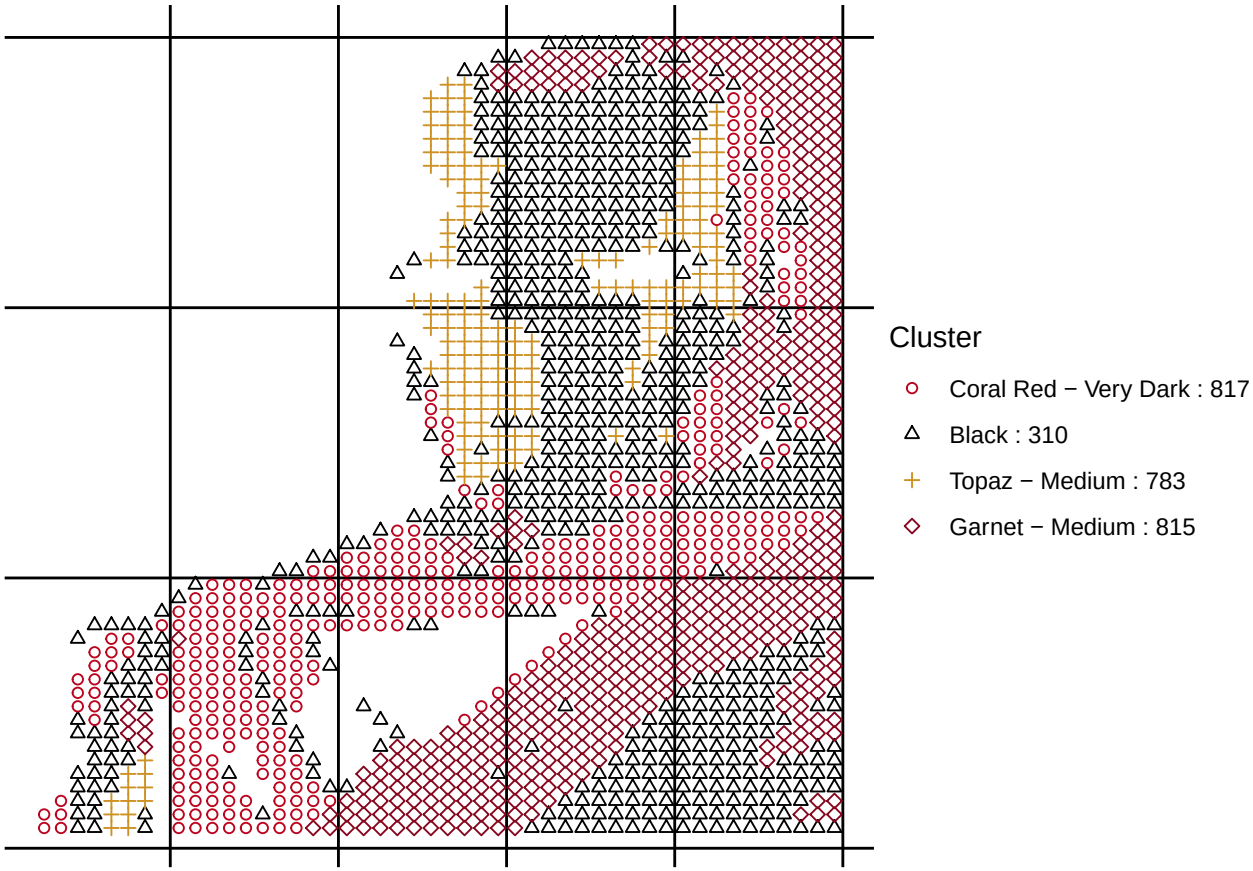
```
background_color <- make_pattern(cluster_info, k = 5, x_size = 50, black_white = FALSE, background_colo
background_color
```



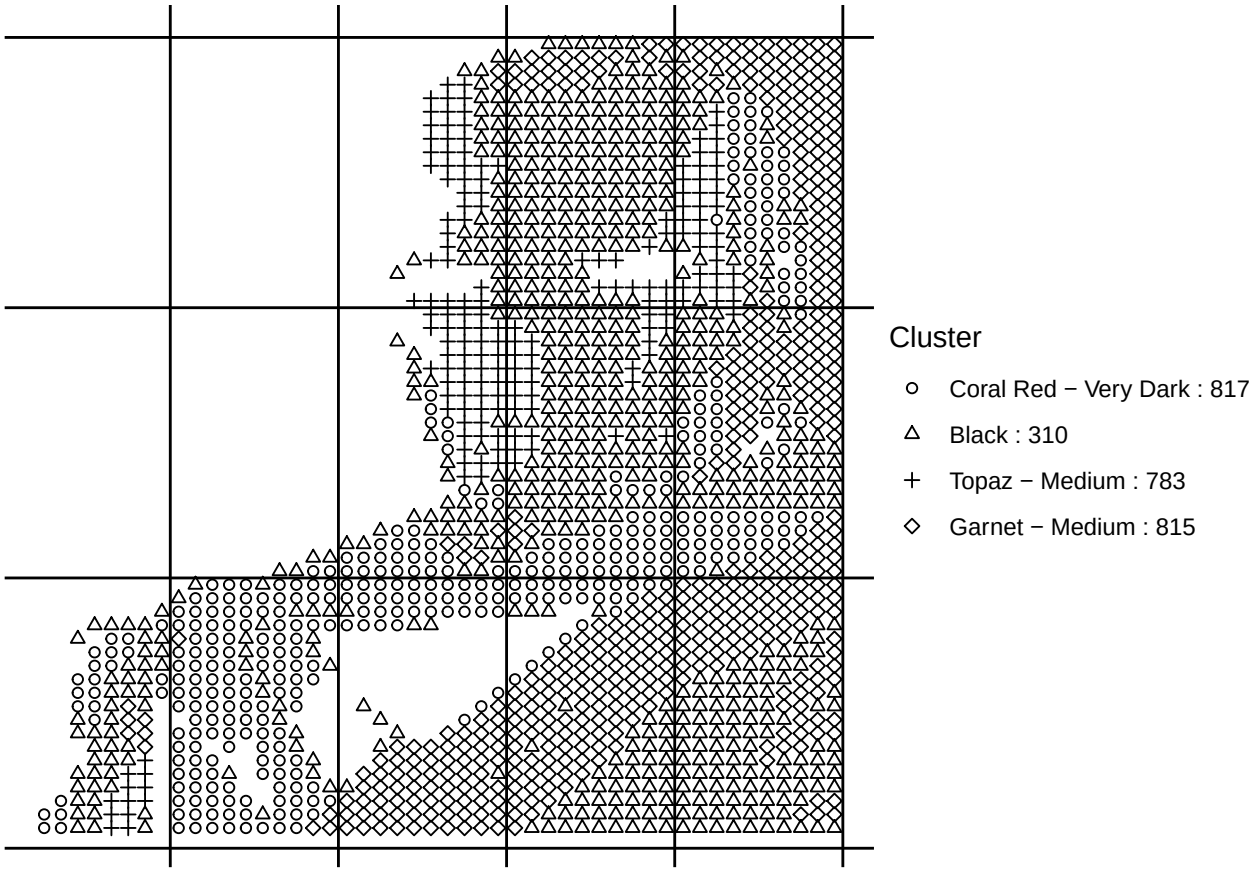

```
background_no_color <- make_pattern(cluster_info, k = 5, x_size = 50, black_white = TRUE, background_co
background_no_color
```




```
no_background_colour <- make_pattern(cluster_info, k = 5, x_size = 50, black_white = FALSE, background_c
no_background_colour
```




```
no_background_no_colour <- make_pattern(cluster_info, k = 5, x_size = 50, black_white = TRUE, backgroun
no_background_no_colour
```


Cluster
Coral Red − Very Dark : 817
Black : 310
Topaz − Medium : 783
Garnet − Medium : 815