Creating a Cross-Stitch using K-means Clustering

Bhavya Shah

03/11/2020

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

LEARNING OBJECTIVE

Summarize clustering characteristics and estimate the best number of clusters for a cross-stitch of an image. After which use the best number of clusters to plot a cross-stitch. For this article we will be using a portrait of Liza Minnelli by Andy Warhol. The image is shown below.

```
im <- imager::load.image("sample.jpg")
plot(im)
```

INTRODUCTION

This article only requires the tidyverse, tidymodels, imager, dmc (a package for choosing thread colour), scales, cowplot, and sp packages.

K-means clustering serves as a useful tool to help us cluster points in our data by replacing them with a representative for that cluster making it easier for us to identify things in our data and for exploratory data analysis. For example: Consider a map and different areas on it, if we wanted to identify water v/s land we could use k-means clustering to cluster data points representative of water together and do the same for land making it easier for us to tell future programs how to differentiate between the two.

LETS BEGIN!

Clustering

We start by taking a look at the “process_image” function which takes an image as the first input “image_file_name” and the second input, a list of possible ‘k’ values “k_list”.

We first load the image using the “imager” package and convert it to a data frame. This data frame essentially contains data points which represent the (x,y) coordinate of each pixel in the image and their corresponding RGB values. We then get rid of the x and y coordinates because we won’t be needing them for our clustering as of now.

```
im <- imager::load.image(image_file_name)
tidy_dat <- as.data.frame(im, wide = "c") %>% rename(R = c.1, G = c.2, B = c.3)
dat <- select(tidy_dat,c(-x,-y))
```

We then create a tibble from the k_list values to feed them into our k-means calls so that we can run k-means clustering for each value of k in k_list. After each k-means call we take the output and create three different tibbles each telling us something different about our clustering.

tidied - This gives us the cluster centers for each ‘k’ and the correspoinding “dmc” colour values and the “hex” colour values for them
glance - This gives us a one line summary for the clustering for each ‘k’ where we get the witin cluster sum of squares and the total sum of squares which will be useful to tell the variation between the ‘k’ values and choose the best one
augmented - Finally this is the data frame we get that assigns each data pixel in the image to a cluster which we will need later to replace the colour of that pixel with the cluster center colour that represents that cluster.

The list that this function returns is in the form of “cluster_info” below.

```
kclusts <- tibble(k = k_list) %>% mutate(kclust = map(k, ~kmeans(x = dat , centers =
.x, nstart=25)),
                                         tidied = map(kclust, tidy),
                                         glanced = map(kclust, glance),
                                         augmented = map(kclust, augment, tidy_dat))
cluster_info <- list(kclusts, tidied_clusters, cluster_summaries, clusterings)
```

After we cluster our data we begin to clean up our output a bit. We unpack each of the three tibbles mentioned above and clean the data up a bit and store them in three different data frames called tidied_clusters, cluster_summaries, and clusterings, where they represent the data from the tidied, glanced and augmented columns from our ‘kclusts’ data frams. We then store all these data frames in a list.

```
cluster_info <- process_image(image_file_name = "sample.jpg", k_list = c(2,4,6,8,10))
```

The next step in our path to building a good cross-stitch is to now choose the best number of clusters, i.e. the best ‘k’, from our possible k values in ‘k_list’.

Scree Plot

One way to do this is by plotting a “scree plot” of the number of clusters against the total sum of squares within each cluster. This tells us the variation in the data for that ‘k’ and we can choose the best ‘k’ accordingly. We are looking for the value of ‘k’ after which the variation stops decreasing alot for the next set of ‘k’ values. We call this the **elbow point** in our scree plot.

To do this we can use our “scree_plot”, all we need to give it is our output from the “process_image” function.

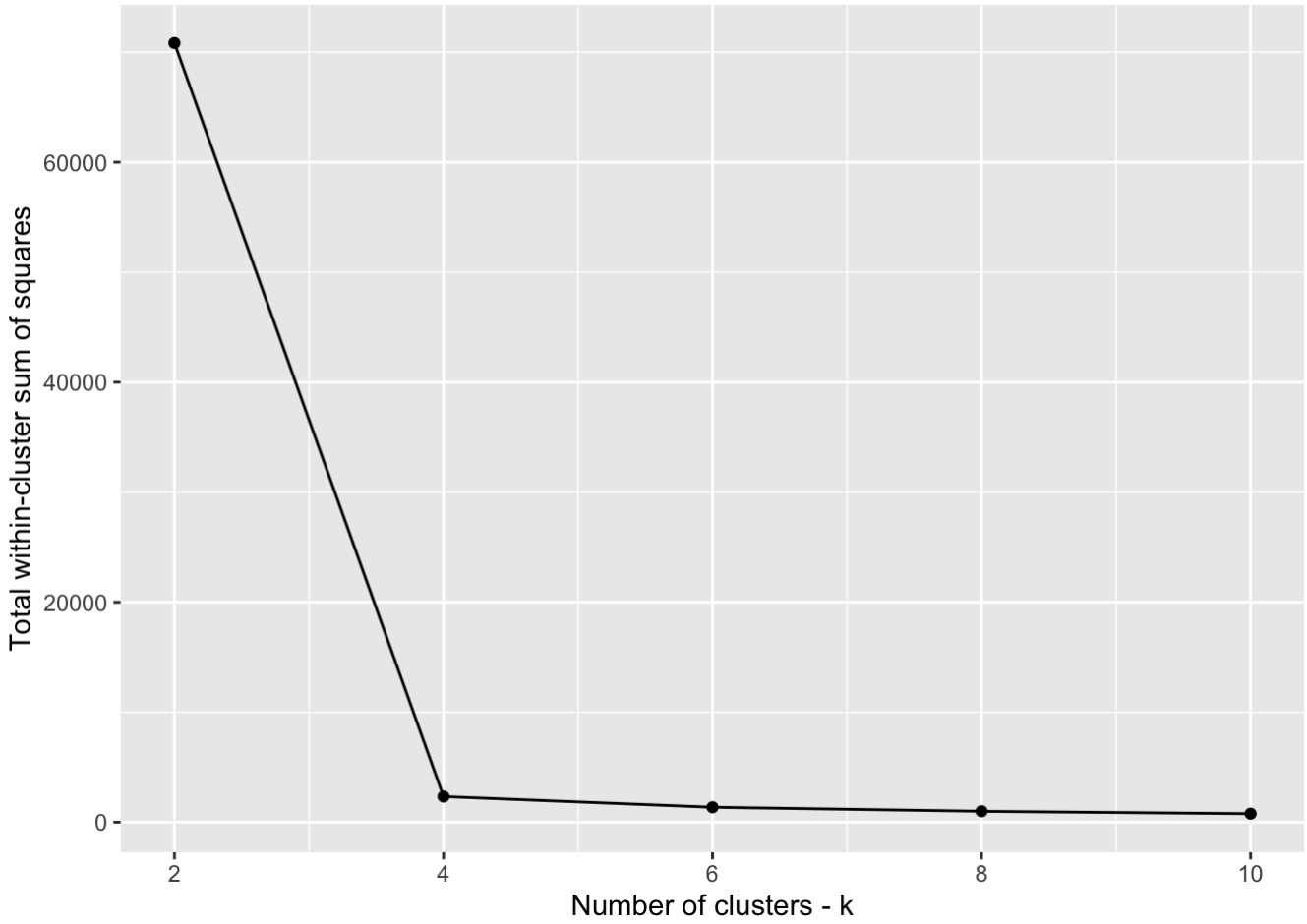
The scree plot is plotted using the “cluster_summaries” data frame from our “process_image” output as it contains the data about the sum of squares. This is the data frame we get by runnig:

```
cluster_summaries <- cluster_info[[3]]
```

We then plot the scree plot using:

The scree plot for our clustering on the Liza Minneli image is as shown below the code.

```
scree_plot(cluster_info)
```



We see that the plot has an obvious looking elbow point at 4 clusters but we can also see a slight noticeable decrease from 4 to 6 and then no noticable decrease after 6 clusters. Since we can’t be sure if we should choose 4 or 6 clusters we need another way to decide. This leads us to our next function.

Colour Strips

Since we could not tell from the scree plot we can use the “colour_strips” function to visually see the cluster centers for each ‘k’ and see how they differ to help us choose the right ‘k’.

For this function we need the “tidied_clusters” data frame from out “cluster_info” since it has the information about the cluster centers. Now lets take a look at the output for the colour strips for our Liza Minnelli image.

```
colour_strips(cluster_info)
```

We see that compared to 6 clusters the colour strips for 4 clusters are missing two colours which aren't as similar to any of the other colours in the 4 clusters to be ignored. So this tells us that we must take a look at 6 or more clusters. Now lets compare 6 to 8 and 10. We see that in 8 and 10 there are 2 or 3 clusters with similar colours which very evidently are representative of the skin. For example with 8 clusters, 2 of the beige-ish clusters are very similar and are evidently clusters for skin colour. We also see that the 2 greyish and greyish green colour clusters for 8 clusters are very similar. This tells us that in both cases we can do with with one colour being representative of the two clusters instead. So this tells us that 6 clusters after all are good enough for this image. So now we have our best number of clusters for our image, i.e. the best 'k' value, which is **6**.

Cross-Stitch

Now we have our data, all clustered and we have the best value for 'k'. So we can now move on to using the "make_pattern" function to plot our cross-stitch. The "make_pattern" function takes in 5 arguments. The first is the "cluster_info" which is the clustering data from "process_image". The second is the number of clusters we want to use, which we were able to identify from the scree_plot and colour_strips functions above. The next argument is the number of points we want in the x-axis direction in our cross-stitch. The last 2 arguments are if we want the cross-stitch to be black and white, and the background color which if not NULL will be used to remove the background colour from the cross-stitch of the image.

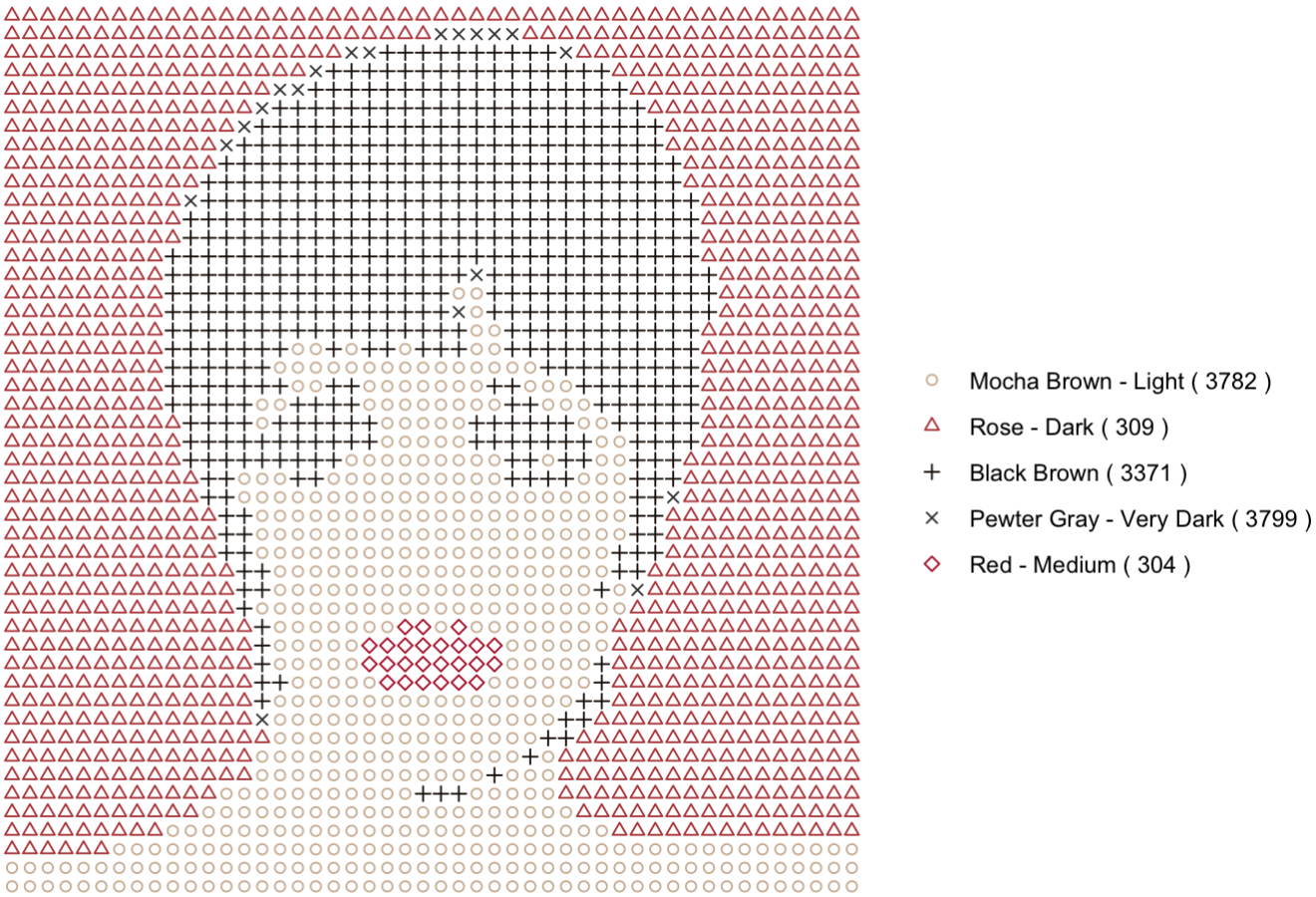
This function takes in the augmented data of the image clustering we get from "process_image", and subsets that data to only have data realted to the 'k' value we input as an argument. We also get the centers for the image for our chosen 'k' value from the output of the "process_image" function.

There are 4 different types of outputs we can get from this function.

The 1st:

This is the output we get when we do not want a black and white image and we do not want to get rid of the background colour.

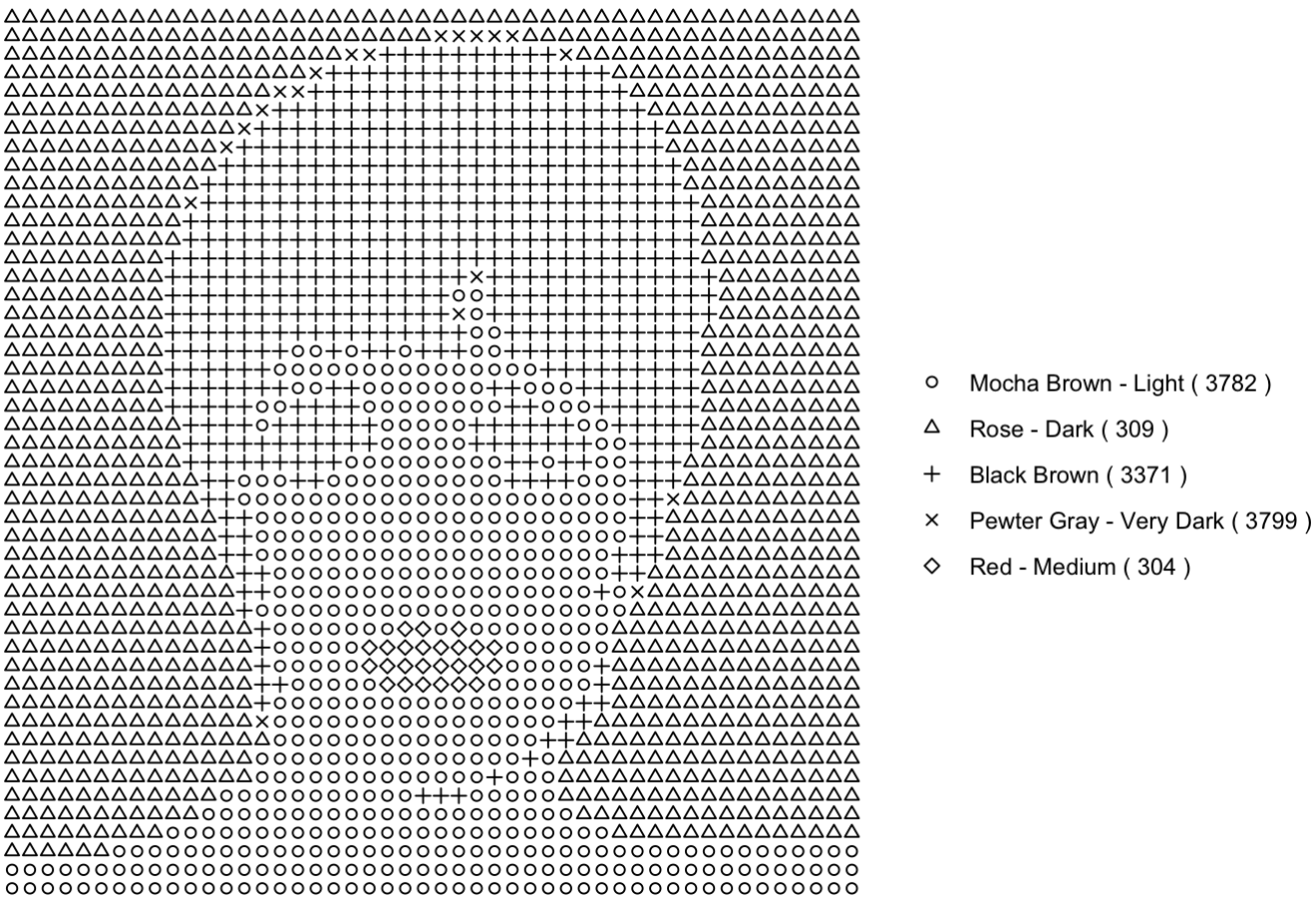
```
cross_stitch <- make_pattern(cluster_info, 6, 50, FALSE, NULL)
plot(cross_stitch)
```



The 2nd:

This is the output we get when we want a black and white image and we do not want to get rid of the background colour.

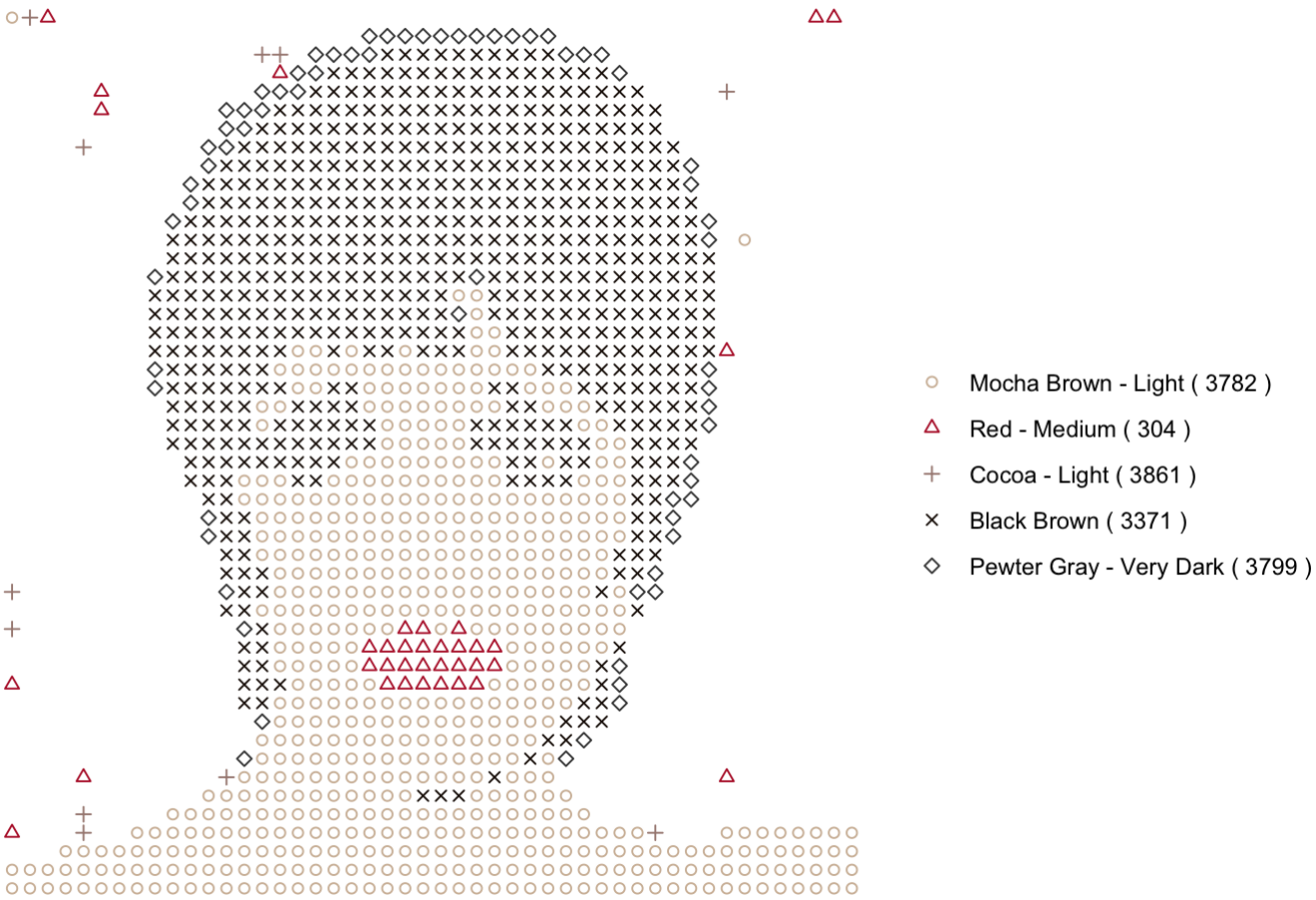
```
cross_stitch <- make_pattern(cluster_info, 6, 50, TRUE, NULL)
plot(cross_stitch)
```



The 3rd:

This is the output we get when we do not want a black and white image and we want to get rid of the background colour.

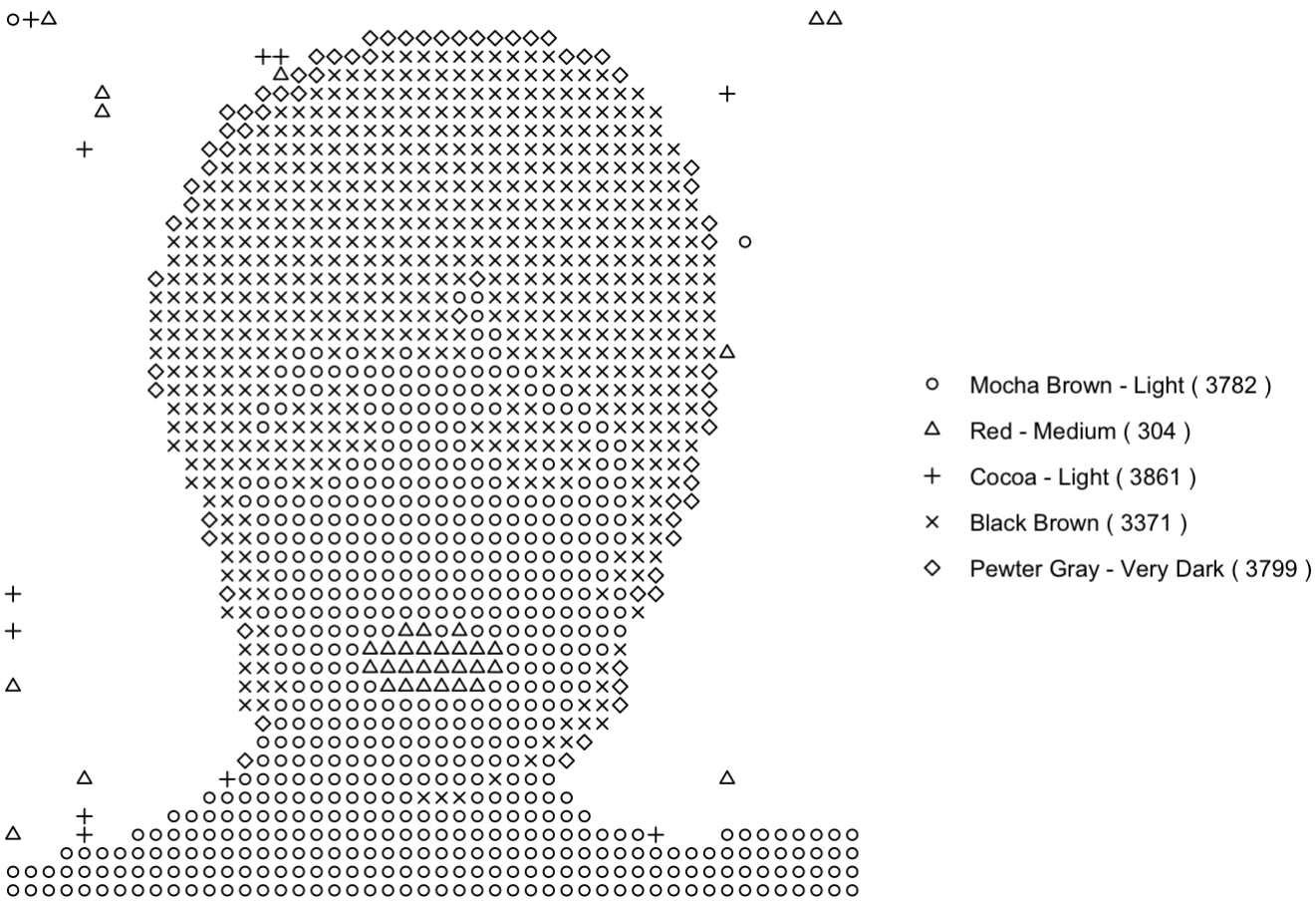
```
cross_stitch <- make_pattern(cluster_info, 6, 50, FALSE, "#BA4A4A")
plot(cross_stitch)
```



The 4th:

This is the output we get when we want a black and white image and we want to get rid of the background colour.

```
cross_stitch <- make_pattern(cluster_info, 6, 50, TRUE, "#BA4A4A")
plot(cross_stitch)
```



Using the steps described above we can use these 4 functions to get ourselves a beautiful cross-stitch of an image