STA314 Assignment 1

Henry Shiffer 1003866881

Objective: Using K-means clustering to make a cross-stitch pattern of an image.

To create a cross-stitch pattern, most images have too many colours to conveniently do a cross-stitch. So a k-means clustering would be useful to simplify the image to make a cross-stitch possible.

To do this, we must make use of these packages.

```r
#devtools::install_github("sharlagelfand/dmc")
library(dmc)
library(imager)
library(tidyverse)
library(tidymodels)
library(dplyr)
library(cowplot)
```

First, we use the function **process_image(image_file_name, k_list)**. This function takes an image file name, and a vector of K values that we must specify. For this example, we'll use an Andy Warhol screen print image of Marilyn Monroe as this image has stylized, flat, block colours that are distinct and should work well for a clustering of colours. When we look at the image, there appears to be about 5 or 6 colours, which may lead to about 6 clusters of colours for our cross stitch pattern. However, we do not know that for sure, so we should input a range of K cluster centre values to see which K would work best for our image. For this example, I input a vector of K values from 1 to 7

```r
set.seed(12345)
cluster_info <- process_image <- function(image_file_name = "AndyWarhol.jpg", k_list){
 ## process_image(image_file_name, k_list) takes an image file the user provides as a JPEG,
 ## and a user specified list of k values in a vector. process_image
 ## then computes a k-means clustering for each k in k_list and stores the
 ## important information extracted in a tibble for each k value. The
 ## information in the output of this function is then used as the input
 ## for each other function.
 ##
 ##
 ##
 ## Input:
 ##  -image_file_name: a PNG or JPEG
 ##  -k_list: vector full of cluster center k values
 ## Output:
 ##  -cluster_info: A tibble derived from k_means that has
 ##                 a clustering computed for each value in k_list.
 ##                 The tibble has the k values in k_list, their
 ##                 associated RGB values, size, withinss, the cluster size,
 ##                 col, DMC_num, DMC_name, Hex, a tibble with the
 ##                 data with the glance data and a tibble with the
 ##                 augmented data.
 ##
```

```r
##
## Example:library(tidyverse)
##         library(tidymodels)
##         library(dplyr)
##         library(dmc)
##         vec_list = c(4, 5, 6)
##         image_for_function = "AndyWarhol.jpg"
##         cluster_info = process_image(image_for_function, vec_list)
##
##
## getting the data from the image file:
mm_image = imager::load.image(image_file_name)
mm_image_data <- as.data.frame(mm_image, wide = "c") %>% rename(R = c.1, G = c.2, B = c.3)

## creating a tibble of information for t:
kclusts_info_tib <- tibble(k = k_list)
for (cluster_val in k_list){
  kclusts_info_tib <- add_row(kclusts_info_tib, k =cluster_val)
  kclusts_info_tib <- unique(kclusts_info_tib)
}

## computing the clusterings
kclusts_info_tib <- add_column(kclusts_info_tib,
                              k_clustering = map(kclusts_info_tib$k,
                                                 ~kmeans(select(mm_image_data, -x, -y),
                                                         centers = .x, nstart = 20)))



## add the tidy data column
kclusts_info_tib <- add_column(kclusts_info_tib,
                              tidy_clusterings = map(kclusts_info_tib$k_clustering, tidy))



## add columns with DMC information to the data frame
kclusts_info_tib$tidy_clusterings <- lapply(kclusts_info_tib$tidy_clusterings,
                                           transform,
                                           col = rgb(R,G,B))
tided_clusters_with_colours <- kclusts_info_tib$tidy_clusterings
tided_clusters_with_colours <- unnest(kclusts_info_tib,
                                     cols = tidy_clusterings)
cluster_colours <- c(tided_clusters_with_colours$col)
dmc_cluster_colours <- sapply(cluster_colours, dmc)
dmc_list <- dmc_cluster_colours[c(1,2,3), c(1:sum(k_list))]
dmc_data_frame <- data.frame(matrix(unlist(dmc_list),
                                   nrow= sum(k_list),
                                   byrow=T),
                            stringsAsFactors=FALSE)
colnames(dmc_data_frame) <- c("DMC_num", "DMC_name", "Hex")
final_clusters <- tided_clusters_with_colours %>% mutate(DMC_num = dmc_data_frame$DMC_num)
final_clusters <- final_clusters %>% mutate(DMC_name = dmc_data_frame$DMC_name)
final_clusters <- final_clusters %>% mutate(Hex = dmc_data_frame$Hex)
tided_clusters_with_colours <- final_clusters
```

```
  ## add glanced data column
  tided_clusters_with_colours <- add_column(tided_clusters_with_colours,
                                            glanced = map(tided_clusters_with_colours$k_clustering,
                                                          glance))
  ## add augmented data column
  tided_clusters_with_colours <- add_column(tided_clusters_with_colours,
                                            augmented = map(tided_clusters_with_colours$k_clustering,
                                                            augment,
                                                            mm_image_data))



  return(tided_clusters_with_colours)
 }
y = process_image("AndyWarhol.jpg", c(1, 2, 3, 4, 5, 6, 7))
```

```
## Warning: Quick-TRANSfer stage steps exceeded maximum (= 24535000)

## Warning: Quick-TRANSfer stage steps exceeded maximum (= 24535000)

## Warning: The `x` argument of `as_tibble.matrix()` must have unique column names if `.name_repair` is
## Using compatibility `.name_repair`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

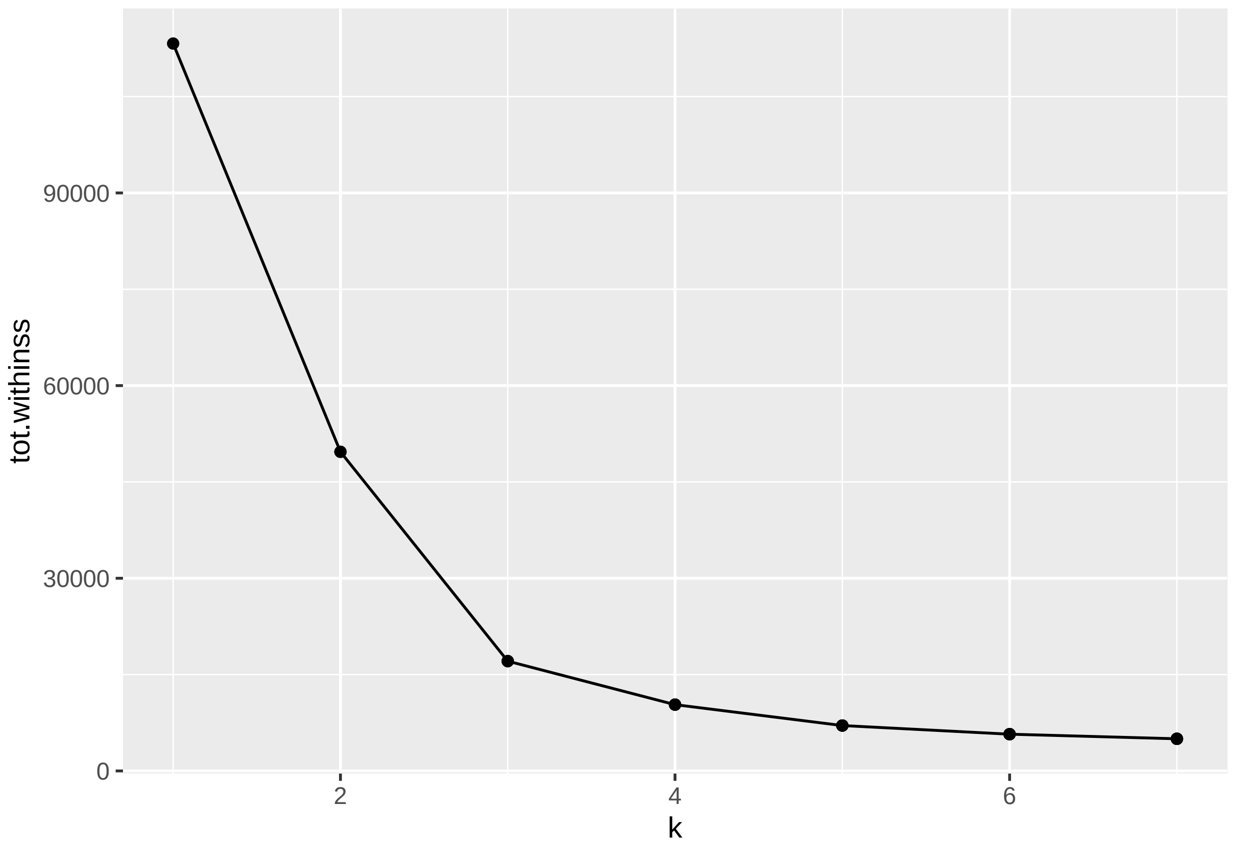
```
glimpse(y)
```

```
## Rows: 28
## Columns: 14
## $ k            <dbl> 1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5, 6, 6, 6, ...
## $ k_clustering <list> [<1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ R            <dbl> 0.73975650, 0.18089458, 0.86375723, 0.88108148, 0.8307...
## $ G            <dbl> 0.54581640, 0.10689115, 0.64320548, 0.59256640, 0.7254...
## $ B            <dbl> 0.36939282, 0.08849522, 0.43171860, 0.21848345, 0.7906...
## $ size         <int> 490700, 89106, 401594, 254146, 149338, 87216, 113099, ...
## $ withinss     <dbl> 113259.3486, 3944.3099, 45743.3916, 10579.7882, 2984.8...
## $ cluster      <fct> 1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, 3, 4, 5, 1, 2, 3, ...
## $ col          <chr> "#BD8B5E", "#2E1B17", "#DCA46E", "#E19738", "#D4B9CA",...
## $ DMC_num      <chr> "3828", "938", "728", "3852", "554", "938", "3820", "9...
## $ DMC_name     <chr> "Hazelnut Brown", "Coffee Brown - Ultra Dark", "Topaz"...
## $ Hex          <chr> "#B78B61", "#361F0E", "#E4B468", "#CD9D37", "#DBB3CB",...
## $ glanced      <list> [<tbl_df[1 x 4]>, <tbl_df[1 x 4]>, <tbl_df[1 x 4]>, <...
## $ augmented    <list> [<tbl_df[490700 x 6]>, <tbl_df[490700 x 6]>, <tbl_df[...
```

When we input these values into **process_image**, and compute a clustering for each K value, we get out **cluster_info**. This is a tibble that contains the information derived from using the **k_means()** function in R that will be needed to input into the other functions.

Second, we use the function **scree_plot(cluster_info)**. This function produces and plots a scree plot for K values 1 through 7. Why a scree plot is useful, is that it can help determine the optimal K to choose to cluster the dataset. In the scree plot, we plot the total within sum of squares (tot.withinss) for each K value on the Y-axis, and the values of K in k_list on the X-axis. We are looking towards decreasing within group variability, so we are looking for the K that has a significant drop in the total within sum of squares.

```
scree_plot <- function(cluster_info){
  ## scree_plot(cluster_info) takes the output from
  ## process_image and plots the total within sum of squares
  ## of each value of k, by the corresponding k value in k_list.
  ## From this we are able to gage an appropriate k value to use,
  ## by looking for the k value where the line stops decreasing rapidly.
  ##
  ##
  ## Input:
  ##  - cluster_info: the information that is the output of
  ##                  process_image(image_file_name, k_list).
  ##
  ## Output:
  ##  - a ggplot graph that has each k value in k_list on the
  ##    x-axis, and the total within sum of squares of each
  ##    k value on the y-axis.
  ##
  ##
  ## Example:library(tidyverse)
  ##         library(tidymodels)
  ##         library(dplyr)
  ##         cluster_info = process_image("AndyWarhol.jpg", c(1, 2, 3, 4, 5))
  ##         scree_plot(cluster_info)


  ## getting the data from the image file:
  cluster_info_data <- cluster_info

  ## unnesting the column that corresponds to glanced
  clusterings <-
  cluster_info_data %>%
  unnest(cols = c(glanced))

  ## create the ggplot with k on the x-axis
  ## and tot.withinss on the y-axis
  scree_plotted <- ggplot(clusterings, aes(k, tot.withinss)) + geom_line() + geom_point()
return(scree_plotted)
}
x = process_image("AndyWarhol.jpg", c(1, 2, 3, 4, 5, 6, 7))
scree_plot(x)
```



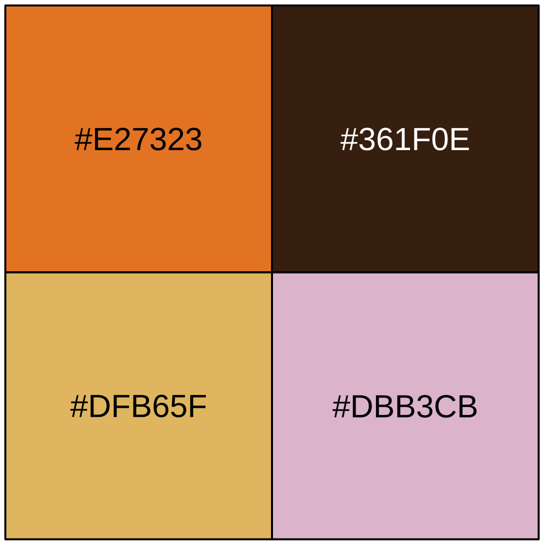
In the plot, we see that the drop in variability from K value 1 to K value 2 is very large, and the drop from K value 2 to K value 3 is still large, but this drop starts to get considerably smaller from K value 3 to K value 4 and from K value 4 to K value 5. When we go from K value 5 to K value 6, we practically don't see any drop. This suggests that once we hit K value 5, any K value beyond that is not necessary as it does not drop our variability significantly. This suggests that from the scree-plot, the optimal cluster K to choose to cluster the dataset seems to be about 5.

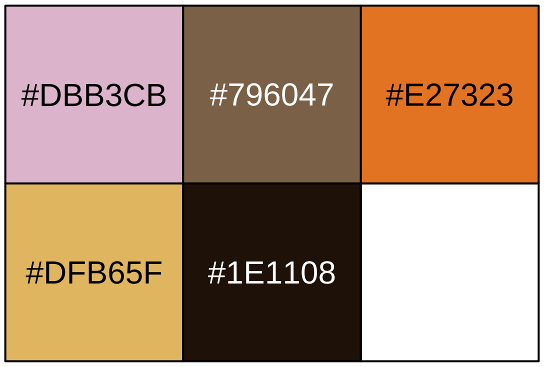
Third, we can use the function **colour_strips(cluster_info)** to verify the appropriate K to choose by looking at the cluster DMC colours that correspond to that K value. From the **scree_plot(cluster_info)** function above, we observed that 5 may be the optimal cluster K to choose to use to cluster the image colours. We can check to see if 5 is indeed optimal by looking at the cluster DMC colours associated with K equal to 4, 5, and 6.

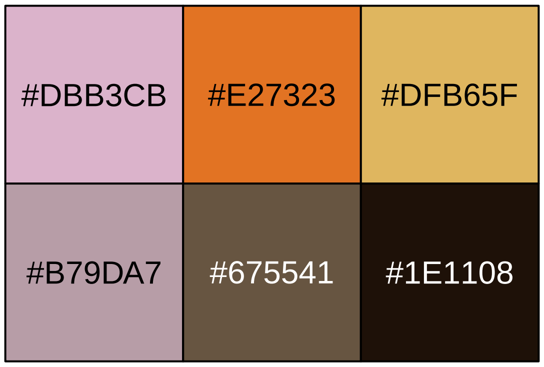
```r
colour_strips <- function(cluster_info){
  ## colour_strips produces colour strips with the DMC
  ## colour that is closest to the cluster centre colour
  ## for each k value in k_list.
  ##
  ## Input:
  ##  - cluster_info: the information that is the output of
  ##                  process_image(image_file_name, k_list).
  ## Output:
  ## - A DMC colour strip for each value of k in k_list. For example
  ##   a k_list of c(2, 3) would output two colour strips, the first
  ##   with 2 colours, and the second with 3 colours.
  ##
  ## Example:
  ##        library(dmc)
  ##          library(tidyverse)
  ##          library(tidymodels)
  ##          library(dplyr)
  ##          library(cowplot)
  ##           x <- process_image("AndyWarhol.jpg", c(2, 3))
  ##           colour_strips(x)
  ##
  ##

  ## get data from cluster_info
  cluster_info_data <- cluster_info
  ## get the k values out for a loop
  k <- cluster_info_data$k
  k <- as.vector(unique(k))
  ## loop over the values of K and show the colour strip for each K
  for (cluster_val in k){
      k_clust <- subset(cluster_info_data, k %in% c(cluster_val))
      cluster_colour <- k_clust$Hex
      show_col(cluster_colour)
  }
}

x = process_image("AndyWarhol.jpg", c(4, 5, 6))
```

```
## Warning: Quick-TRANSfer stage steps exceeded maximum (= 24535000)
```

```
colour_strips(x)
```







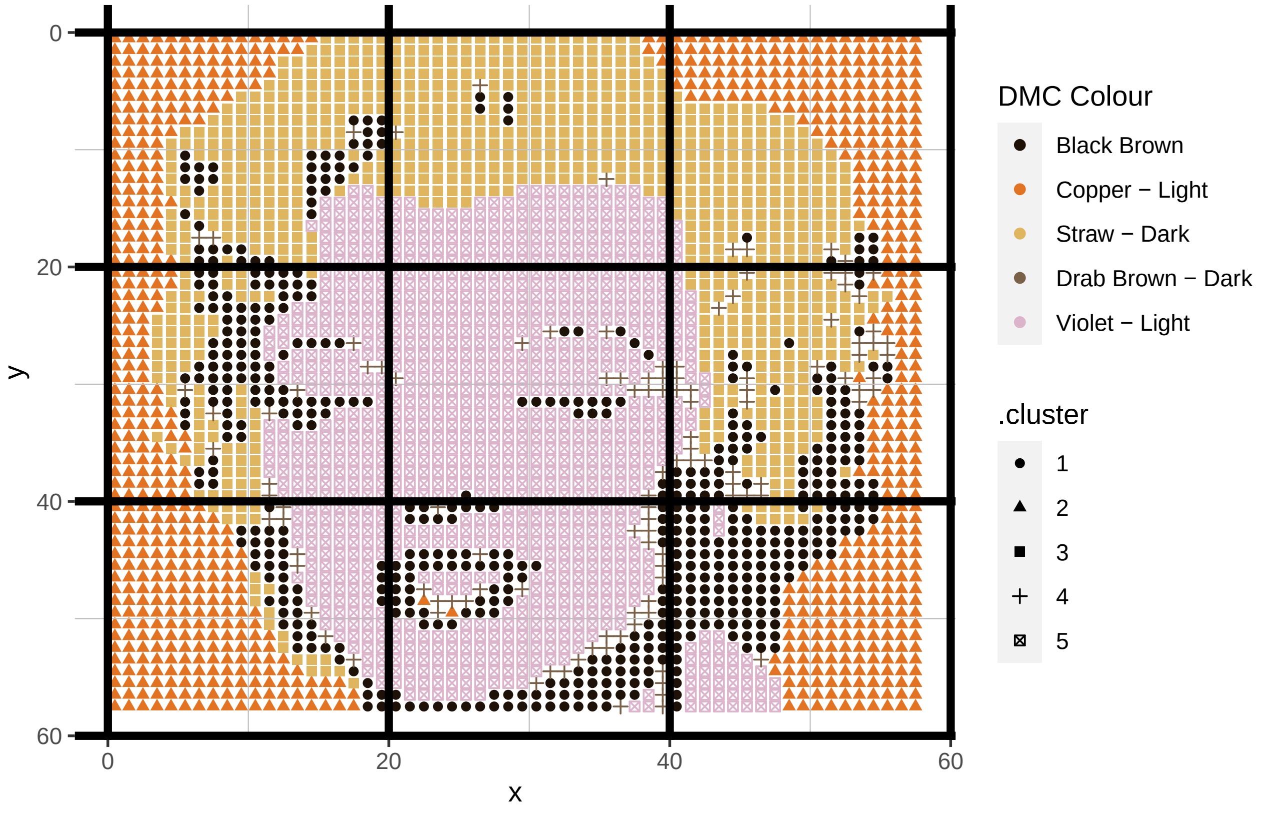
Looking at the colour strips, it seems that the colour strip with 5 DMC colours, corresponding to K equal to 5 has an appropriate amount of colours. K equal to 4 seems to be close to the appropriate amount of colours, as each colour is distinct in the strip, but this may be a too little amount of texture to make out the shadows in the Andy Warhol screen print of Marlyin's face. We also see that with K equal to 6, there are 2 violet colours that are very similar, and one may not be needed. However looking at K equal to 5, it seems to be a middle ground between K equal to 4 and K equal to 6. The added dark brown-sand colour may provide some texture to the shadows along Marylin's face along with the black colour.

```r
change_resolution <- function(image_df, x_size)
{
  ## change_resolution(image_df, x_size) subsamples an image to produce
  ## a lower resolution image. Any non-coordinate columns in the data
  ## frame are summarized with their most common value in the larger
  ## grid cell.
  ##
  ## Input:
  ## - image_df: A data frame in wide format. The x-coordinate column MUST
  ##             be named 'x' and the y-coordinate column MUST be named 'y'.
  ##             Further columns have no naming restrictions.
  ## - x_size:   The number of cells in the x-direction. The number of cells
  ##             in the vertical direction will be computed to maintain the
  ##             perspective. There is no guarantee that the exact number
  ##             of cells in the x-direction is x_size
  ##
  ## Output:
  ## - A data frame with the same column names as image_df, but with fewer
  ##   entries that corresponds to the reduced resolution image.
  ##
  ## Example:
  ##   library(imager)
  ##   library(dplyr)
  ##   fpath <- system.file('extdata/Leonardo_Birds.jpg',package='imager')
  ##   im <- load.image(fpath)
  ##   im_dat<- as.data.frame(im,wide = "c") %>% rename(R = c.1, G = c.2, B = c.3) %>%
  ##            select(x,y,R,G,B)
  ##   agg_image <- change_resolution(im_dat, 50)

  if(!require(sp)) {
    stop("The sp packages must be installed. Run install.packages(\"sp\") and then try again.")
  }
  if(!require(dplyr)) {
    stop("The dplyr packages must be installed. Run install.packages(\"dplyr\") and then try again.")
  }

  sp_dat <- image_df
  gridded(sp_dat) = ~x+y

  persp = (gridparameters(sp_dat)$cells.dim[2]/gridparameters(sp_dat)$cells.dim[1])
  y_size = floor(x_size*persp)
  orig_x_size = gridparameters(sp_dat)$cells.dim[1]
  orig_y_size = gridparameters(sp_dat)$cells.dim[2]

  x_res = ceiling(orig_x_size/x_size)
  y_res = ceiling(orig_y_size/y_size)

  gt = GridTopology(c(0.5,0.5), c(x_res, y_res),
                    c(floor(orig_x_size/x_res), floor(orig_y_size/y_res)))
  SG = SpatialGrid(gt)
  agg = aggregate(sp_dat, SG, function(x) names(which.max(table(x)))[1] )
  agg@grid@cellsize <- c(1,1)
  df <- agg %>% as.data.frame %>% rename(x = s1, y = s2)  %>% select(colnames(image_df))

  return(df)

}
```

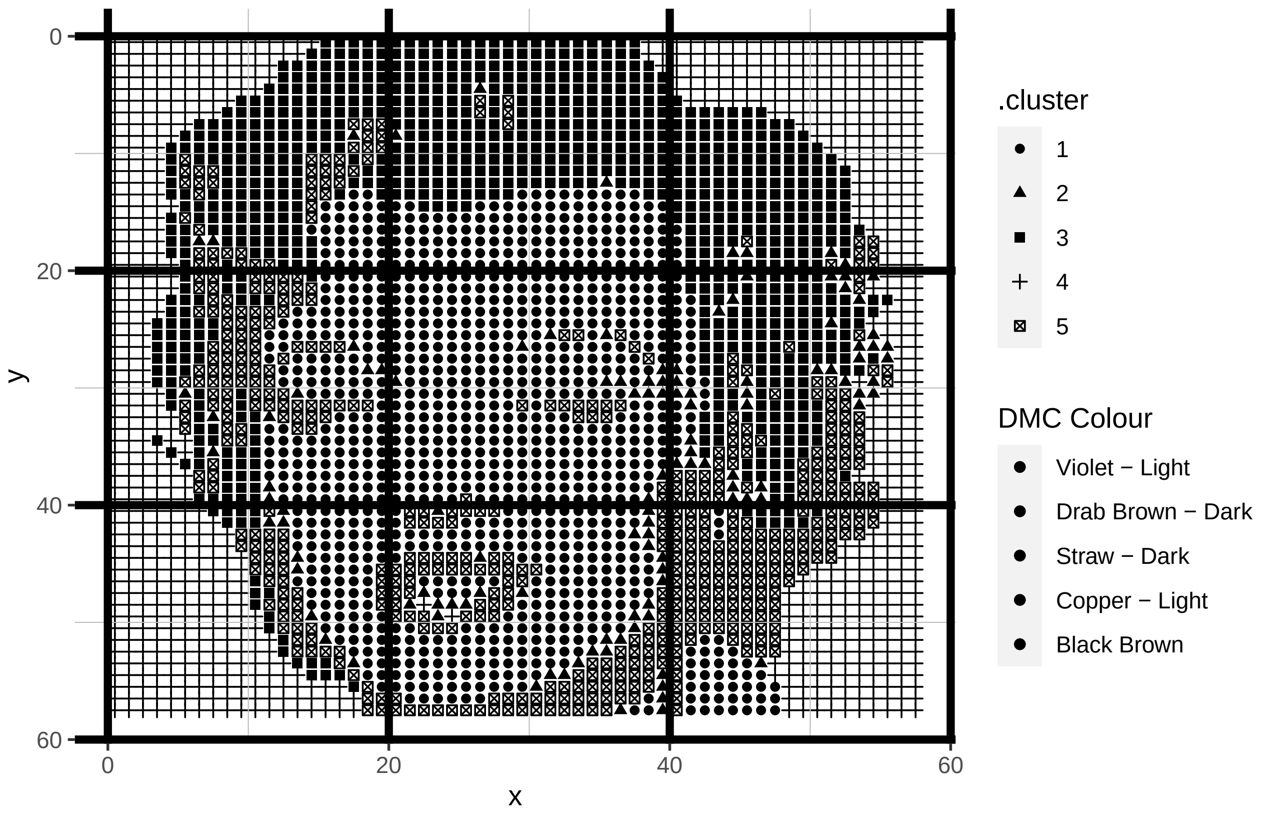
Finally, we can use the function **make_pattern(cluster_info, k, x_size, black_white = FALSE, background_colour = NULL)** to get our cross-stitch pattern of the image we inputted. For the function, we must specify a specific K value that we choose from looking at the output of **scree_plot(cluster_info)**, and **colour_strips(cluster_info)**. From the output of both of these functions, it appears that the optimal K value is 5, so we should input 5 as the K value into the make_pattern function. We will also specify our x_size as 60, which means our pattern will be 60 x 60. For this function, we can also specify whether or not the pattern should include colour, or be in black and white. We can also specify if the pattern should include the image background colour. The default is that the background colour of the image is included in the cross stitch pattern.

```
make_pattern <- function(cluster_info, k, x_size, black_white = FALSE, background_colour = NULL){
  ## make_pattern(cluster_info, k, x_size, black_white, background_colour)
  ## plots the cross stitch pattern. The user specifies a specific k value,
  ## size of the pattern, and whether the plot should include colour
  ## or be black and white. The user can also specify if the pattern should
  ## include background colour.
  ##
  ##
```

```r
## Input:
##  - cluster_info: the information that is the output of
##                  process_image(image_file_name, k_list).
##  - k: The chosen cluster size by the user.
##  - x_size: The number of cells in the x-direction.
##            The (approximate) total number of possible stitches
##            in the horizontal direction.
##  - black_white: (logical) Print the pattern in black and
##    white (TRUE) or print the pattern colour by having the default (FALSE).
##  - background_colour: The colour of the background, which should not be stitched in the
##    pattern. (Default is to not have a colour).
##
##
## Output:
##  - A ggplot that has the plotted cross stitch pattern of the image,
##    with a legend that corresponds to each cluster symbol with the
##    associated cluster colour name. The plotted pattern image also
##    includes a black grid ontop of it.
##
##
## Example:library(dmc)
##         library(tidyverse)
##         library(tidymodels)
##         library(dplyr)
##         library(cowplot)
##         cluster_info <- process_image("AndyWarhol.jpg", c(6))
##         make_pattern(cluster_info, 6, 75, TRUE)

## get the data frame with the cluster information at the specific k
cluster_info_data <- cluster_info("AndyWarhol.jpg", k)
cluster_data <- cluster_info_data$augmented
cluster_info_dataframe <- as.data.frame(cluster_data)
cluster_info_data1 <- select(cluster_info_dataframe, c(x,y,.cluster))


## apply the change_resolution function to change the resolution of the data
changed_resolution_data <- change_resolution(cluster_info_data1, x_size = x_size)


## create a frame of colour information
colour_frame <- tibble(cluster = c(1:k), name = c(cluster_info_data$DMC_name),
                    col = c(cluster_info_data$Hex))
## create a theme for the plotted image which includes a grid line for the plot
mytheme <- theme(panel.grid.major = element_line(colour="black", size = (1.5)),
               panel.grid.minor = element_line(size = (0.2), colour="grey"),
               panel.grid = element_line(color = "black"),
  panel.ontop = TRUE, panel.background = element_rect(color = NA, fill = NA))

## create the plotted image with colour
plotted = ggplot(changed_resolution_data, aes(x=x, y=y)) + mytheme +
  geom_point(aes(col = .cluster, shape = .cluster)) +
  scale_colour_manual(name = "DMC Colour", values = colour_frame %>% select(cluster, col) %>% deframe
                     label =  colour_frame %>% select(cluster, name) %>% deframe) +
    scale_y_reverse() + mytheme

  ## create the plotted image in black and white
  if (black_white == TRUE){
    vec = rep("black", k)
    plotted = ggplot(changed_resolution_data, aes(x=x, y=y)) +
    geom_point(aes(col = .cluster, shape = .cluster)) +
    scale_colour_manual(name = "DMC Colour", values = vec,
                        label =  colour_frame %>% select(cluster, name) %>% deframe) +
    scale_y_reverse() + mytheme}




return(plotted)

}
info = process_image("AndyWarhol.jpg", c(5))
plotted = make_pattern(info, 5, 60, FALSE)
plotted
```



```
plotted2 = make_pattern(info, 5, 60, TRUE)
plotted2
```



Above we can see the final cross-stitch pattern of Marylin in both Colour and in Black and White. Both images above include the background colour in the plot, have K equal to 5, and are 60 x 60.

We can interpret the legend by looking at the cluster number and the DMC colour name associated to the colour at the same position in the list. For example, the first DMC Colour name listed corresponds to the first cluster shape listed, and so on.

Concluding Statement:

Overall, we saw that it is difficult to find an optimal K value, which is usually the case with K-means clustering as this must be specified by the user. However, the process of choosing an inital list of K values, plotting a scree-plot of the values, and looking at the cluster colours corresponding to the those K values, is a good process to discern how many DMC colours should be included in the cross-stitch pattern.