STA314 Assignment 1: Making a cross-stitch pattern for a given image

Ruo Ning Qiu

03 November, 2020

```
# Before we begin, let's set the seed to be a student number so it produces the same result
# every time.
set.seed(1004079631)
```

This vignette covers the process of using 4 functions in functions.R, `process_image()`, `scree_plot()`, `colour_strips()`, `make_pattern()` to generate a cross-stitch of a given image.

For demonstrating purpose, the image that we will use to generate the cross-stitch is the following:

We need the following library packages to run the functions, make sure you have installed them!

```
# This also checks if the required libraries are installed while loading the functions.
# If not, please follow the return error message and install the required libraries.
source("functions.R")
# load the libraries that we need, should be installed at this stage by previous check
library(imager)
library(tidyverse)
library(tidymodels)
library(sp)
library(scales)
library(cowplot)
library(dmc)
```

The first step is to use `process_image()` function to perform clustering on the image data by stating the argument `k_list`, a list of possible number of clusters that you want to try for the image given at a filepath by inputting the name of the image for `image_file_name` argument. Let's try to compute a k-means clustering with $1-10$ as the potential cluster number.

```
# let's try to compute a k-means clustering with the 1-10 clusters as the candidates
cluster_info <- process_image(image_file_name = "cat.jpg", k_list = c(1:10))
```

The output `cluster_info` has many information. For example, `cluster_info$clusterings` return a summary of all the clusterings we have tried on the image data, such as the varaince for each number of clusterings.

```
cluster_info$clusterings
```

```
## # A tibble: 10 x 6
##        k kclust   totss tot.withinss betweenss  iter
##    <int> <list>   <dbl>        <dbl>     <dbl> <int>
##  1     1 <kmeans> 5493.        5493. -2.71e-10     1
##  2     2 <kmeans> 5493.        1932.  3.56e+ 3     1
##  3     3 <kmeans> 5493.        1120.  4.37e+ 3     2
```

```
##  4     4 <kmeans> 5493.         756.  4.74e+ 3     3
##  5     5 <kmeans> 5493.         600.  4.89e+ 3     3
##  6     6 <kmeans> 5493.         500.  4.99e+ 3     5
##  7     7 <kmeans> 5493.         406.  5.09e+ 3     4
##  8     8 <kmeans> 5493.         336.  5.16e+ 3     5
##  9     9 <kmeans> 5493.         295.  5.20e+ 3     7
## 10    10 <kmeans> 5493.         262.  5.23e+ 3     6
```

`cluster_info[[k]]` returns two tibbles `dat` and `centres` for the k number of clusters implemented on the image. `dat` is all the data points after clustered and tidied with RGB values, where `centres` tells us detailed information about each cluster, such as the colour `col` of the clusters and their closest dmc colours `hex` and the name `dmc`.

```
# for example, k = 3
cluster_info[[3]]$dat
```

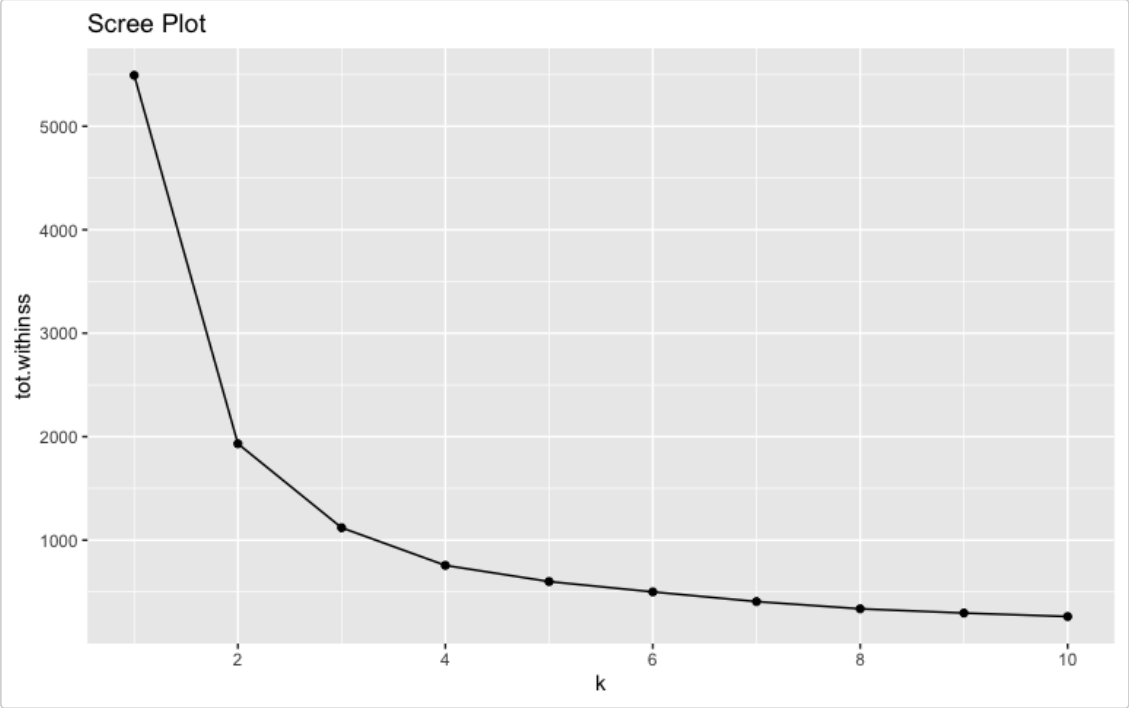
```
## # A tibble: 50,625 x 8
##        x     y     R     G     B cluster dmc                           hex
##    <int> <int> <dbl> <dbl> <dbl> <fct>   <chr>                         <chr>
##  1     1     1 0.671 0.502 0.369 1       Desert Sand - Very Dark (3772) #A06C50
##  2     2     1 0.671 0.502 0.369 1       Desert Sand - Very Dark (3772) #A06C50
##  3     3     1 0.675 0.506 0.373 1       Desert Sand - Very Dark (3772) #A06C50
##  4     4     1 0.678 0.510 0.376 1       Desert Sand - Very Dark (3772) #A06C50
##  5     5     1 0.682 0.514 0.380 1       Desert Sand - Very Dark (3772) #A06C50
##  6     6     1 0.682 0.514 0.380 1       Desert Sand - Very Dark (3772) #A06C50
##  7     7     1 0.686 0.518 0.384 1       Desert Sand - Very Dark (3772) #A06C50
##  8     8     1 0.686 0.518 0.384 1       Desert Sand - Very Dark (3772) #A06C50
##  9     9     1 0.678 0.510 0.376 1       Desert Sand - Very Dark (3772) #A06C50
## 10    10     1 0.678 0.510 0.376 1       Desert Sand - Very Dark (3772) #A06C50
## # … with 50,615 more rows
```

```
# for example, k = 3
cluster_info[[3]]$centres
```

```
## # A tibble: 3 x 9
##       R     G      B  size withinss cluster col     hex    dmc
##   <dbl> <dbl>  <dbl> <int>    <dbl> <fct>   <chr>   <chr>  <chr>
## 1 0.625 0.409 0.293  20926     414. 1       #9F684B #A06C… Desert Sand - Very D…
## 2 0.834 0.644 0.514  16485     355. 2       #D5A483 #C69F… Hazelnut Brown - Lig…
## 3 0.384 0.178 0.0975 13214     351. 3       #622D19 #6825… Rosewood - Dark (385…
```

The second step for generating a cross-stitch pattern is to determine the number of clusters that we are actually using. To do so, we need to have a look at the scree plot of the k-clusterings for $1 - 10$ clusters of the image (using the function `scree_plot()`) to narrow down our candidates for the cluster number.

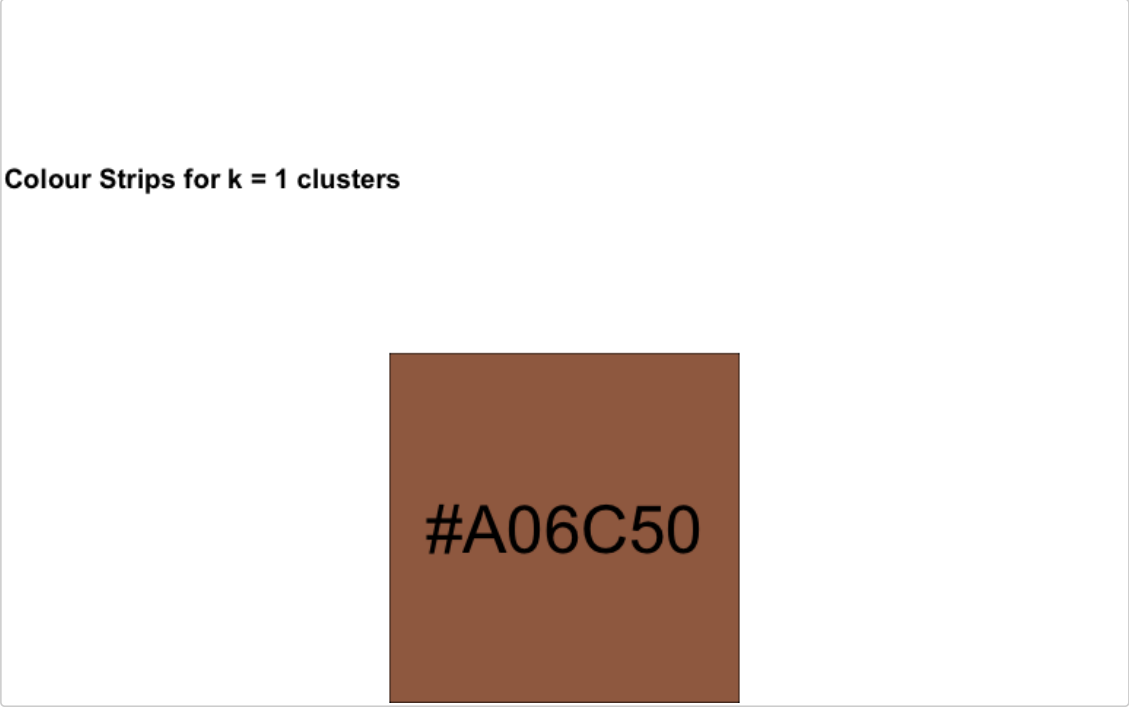
```
scree_plot(cluster_info)
```



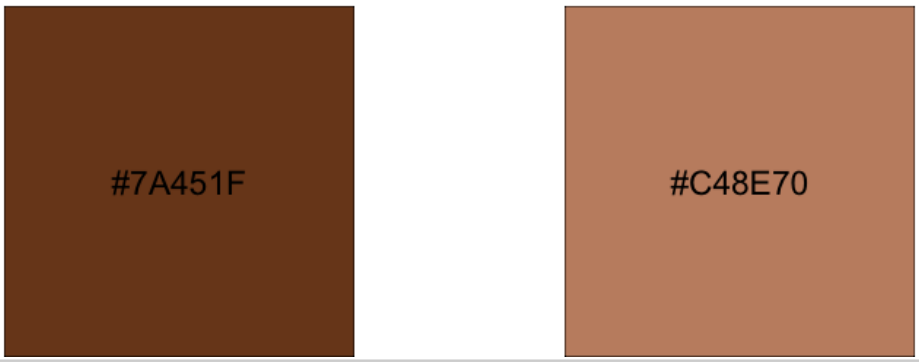
We can see that from the scree plot, the number of clusters should be 5 or 6 for the cat image, as the variance does not decrease as much after that number of clusters.

Let's have a look at the colour strips of the cluster centres for each cluster number to finalize the number of clusters that we will be using by the function `colour_strips()`.

```
colour_strips(cluster_info)
```



Colour Strips for k = 2 clusters



Colour Strips for k = 3 clusters

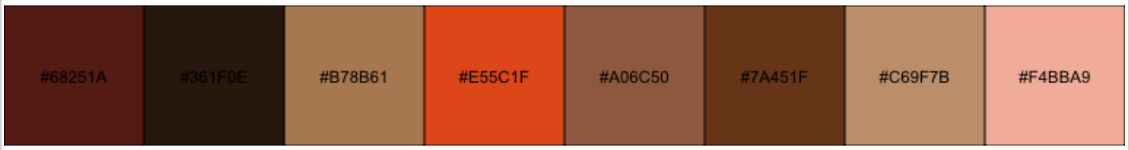
Colour Strips for k = 4 clusters

Colour Strips for k = 5 clusters

Colour Strips for k = 6 clusters

Colour Strips for k = 7 clusters

Colour Strips for k = 8 clusters

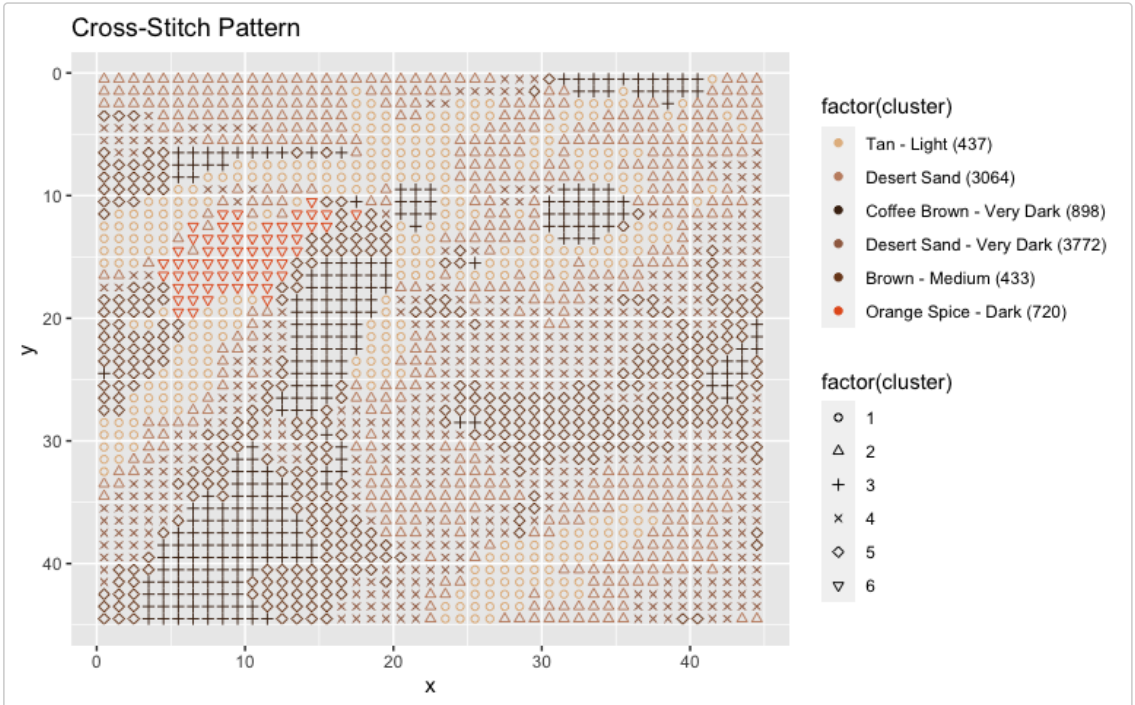


Colour Strips for k = 9 clusters

I will choose to pick 6 as the number of clusters since the colours are quite different (the orange is new for 6 clusters, which is a very distinct colour from the colours of the 5 clusters). This is more of a personal choice; it is quite challenging to actually pick the best number. I am only picking a reasonable number of clusters and I am uncertain that if this is the best pick.

After we have decided the number of clusters, let's get to the exiting part — to actually compute the cross-stitch pattern for the cat image! The function `make_pattern()` will take care of it for us, by inputting `cluster_info`, the chosen number of clusters `k = 6`, and the (approximate) total number of possible stitches in the horizontal direction `x_size` that you would like.

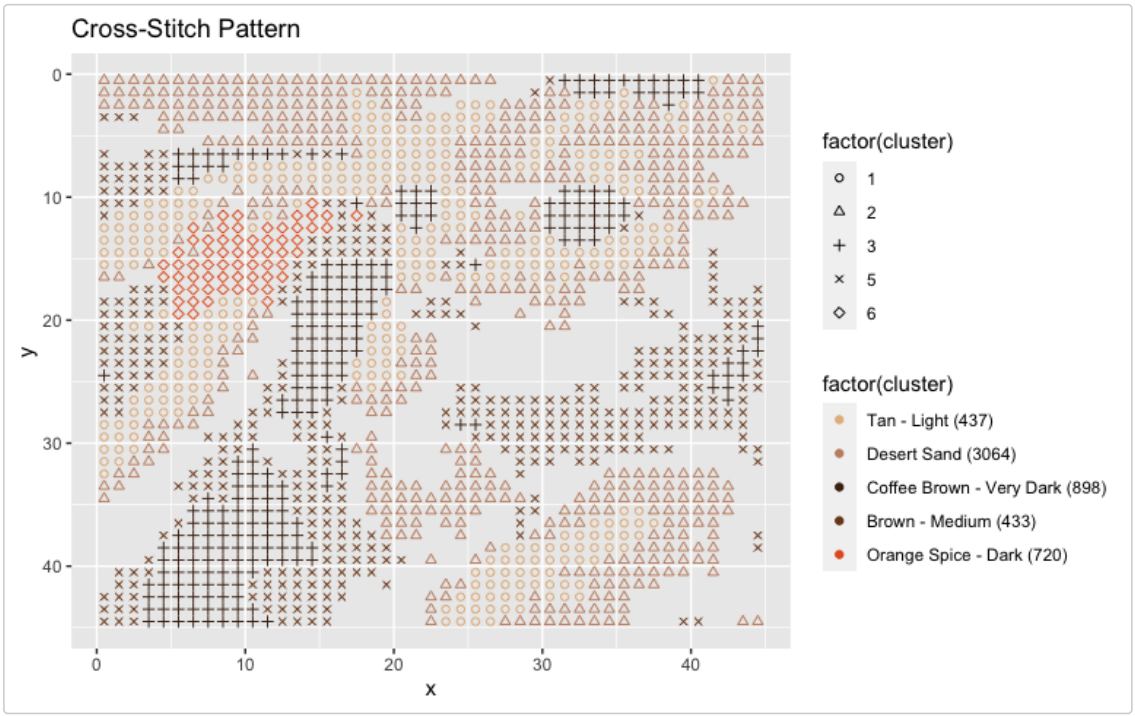
```
# I picked x_size to be 50 for example
make_pattern(cluster_info, k = 6, x_size = 50)
```



If you are nearsighted like me, just take off your glasses and/or squeeze your eyes, you can see that the cross-stitch pattern that we generated is so similar to the original picture (with some imaginations). Let's load the original image again to compare.

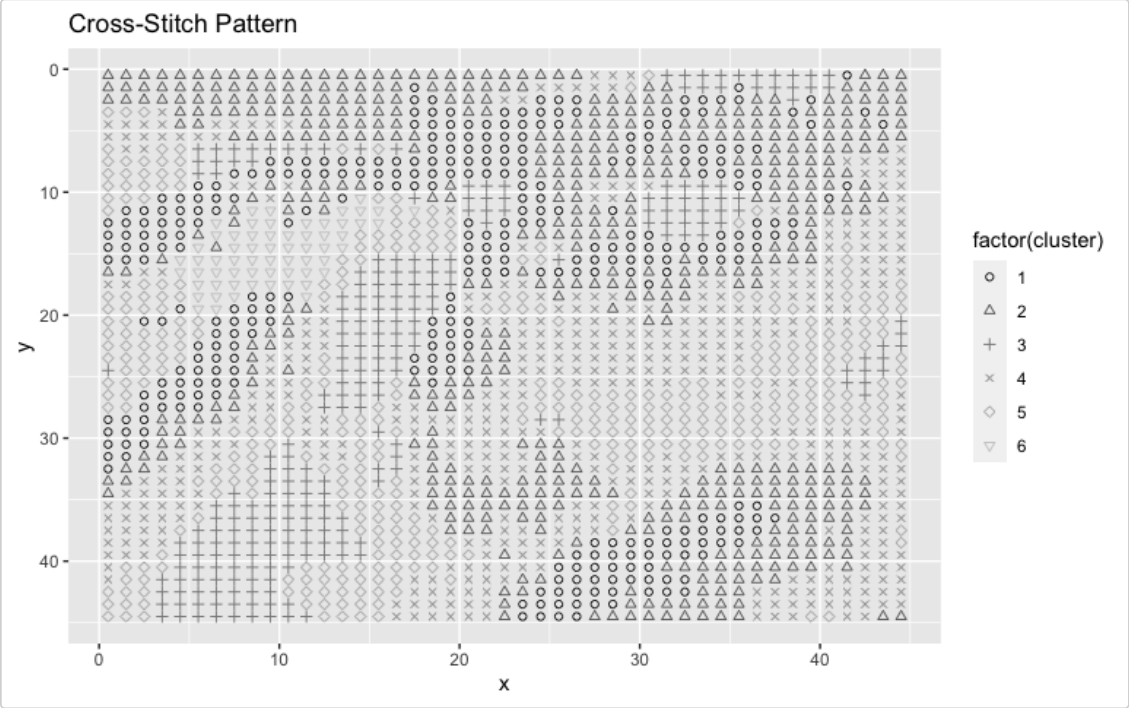
We can also generate a cross-stitch that does not have the background colour stitched, for example, we can remove the “Dessert Sand - Very Dark (3772)” colour from our cat image by entering the string argument `background_colour` as the number id in the bracket of the colour in the legend, in this case, the string `"3772"`.

```
make_pattern(cluster_info, k = 6, x_size = 50, background_colour = "3772")
```



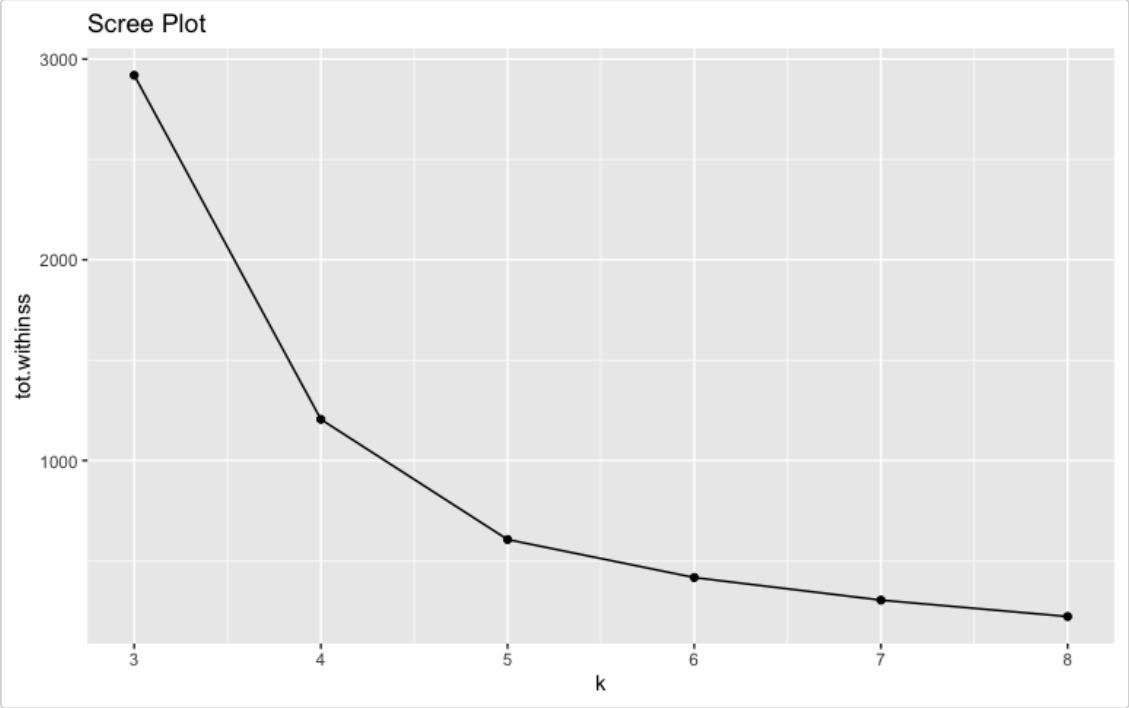
Moreover, if you are in the mood of creating a sketching style, we can make a black and white cross-stitch pattern by letting the argument `black_white` to be `TRUE`.

```
source("functions.R")
make_pattern(cluster_info, k = 6, x_size = 50, black_white = TRUE)
```



Since this example cat image does not have a clear background colour, let's use another image to demonstrate the usage of argument `background_colour` better (in the spirit of Halloween, the new image is again a cat picture with pumpkin because you can tell by now that I am a cat person).

```
# let's try to compute a k-means clustering with the k_list clusters number candidate
# I eyeball that it should have at least 3 to 8 colours of clusters
k_list2 = c(3:8)
cluster_info2 <- process_image("pumpkin_cat.jpg", k_list2)
scree_plot(cluster_info2)
```



We can see that the number of clusters should be 5 or 6 from this new scree plot.

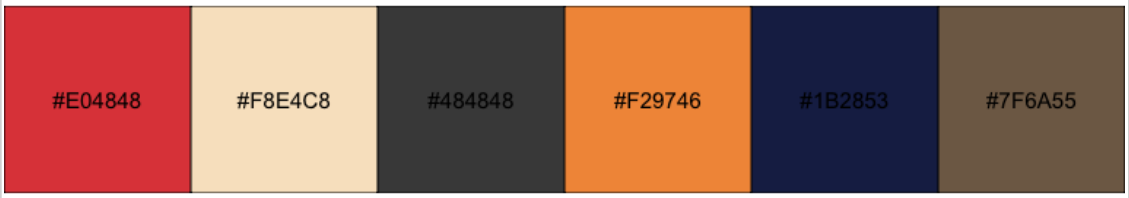
Let's have a look at the colour strips of the cluster centres to decide the number of clusters that we should use to make the cross-stitch pattern.

```
colour_strips(cluster_info2)
```

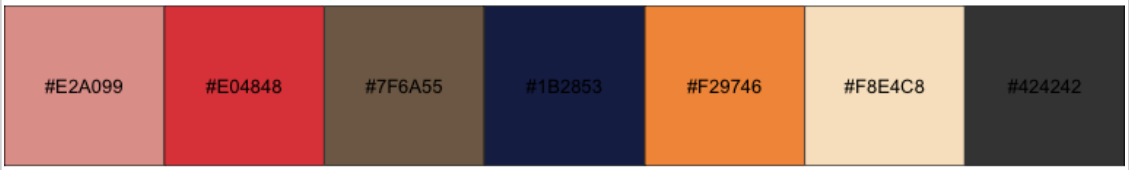
Colour Strips for k = 4 clusters

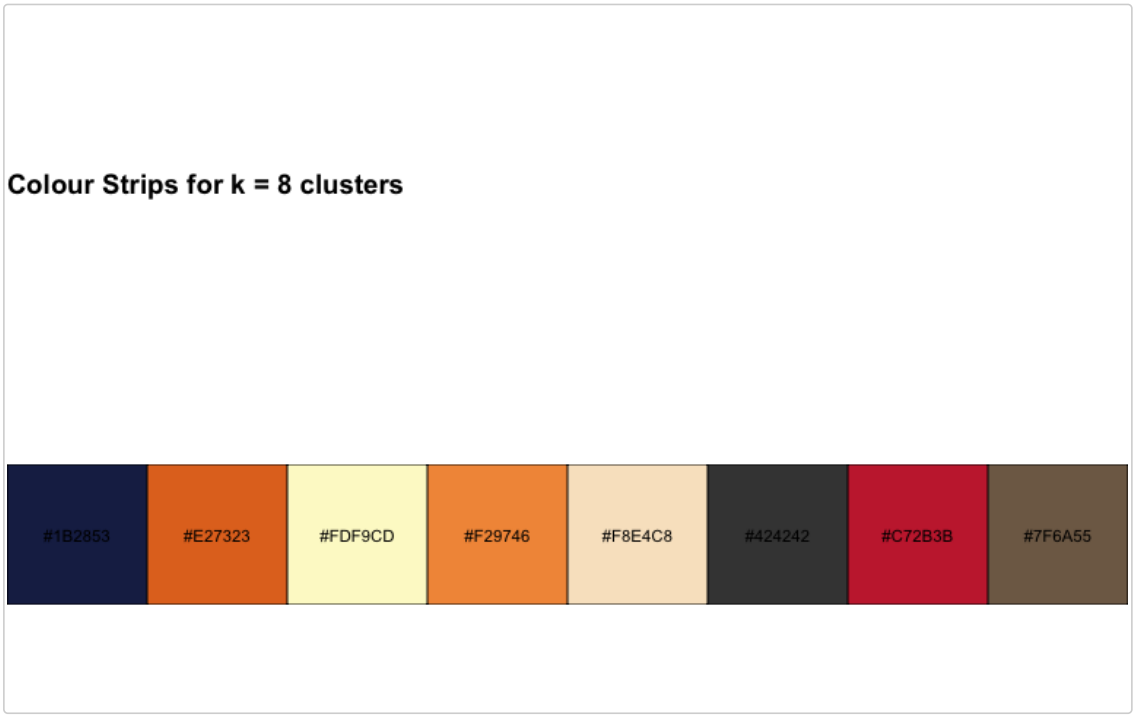
Colour Strips for k = 5 clusters

Colour Strips for k = 6 clusters



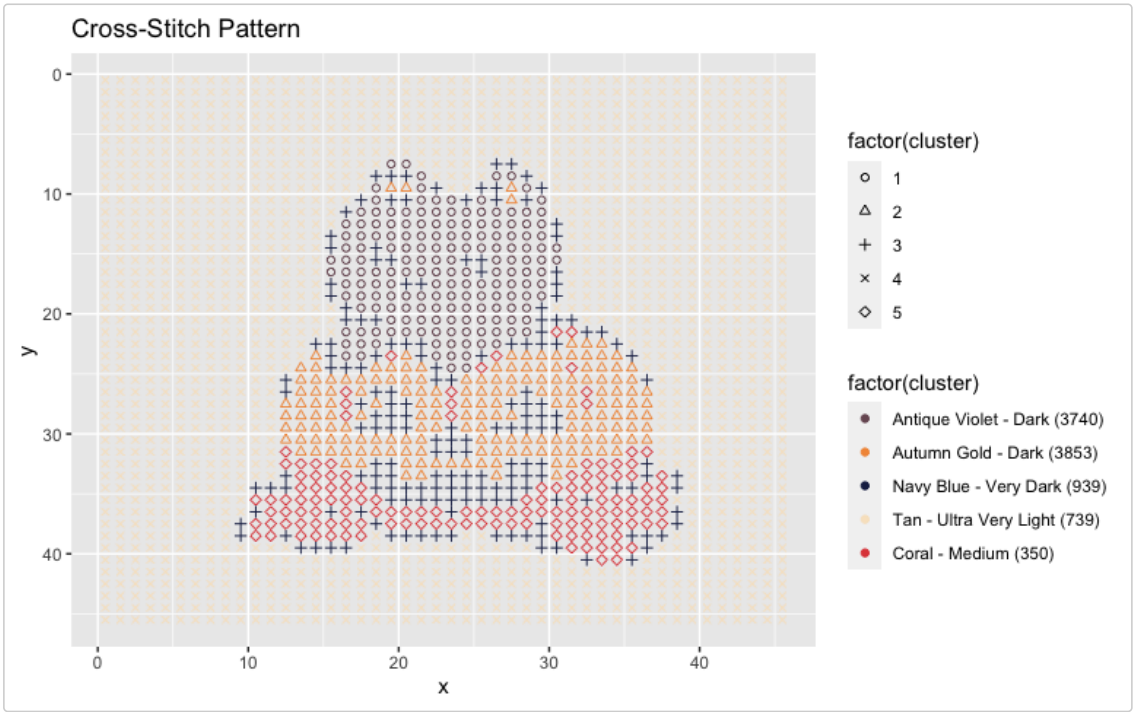
Colour Strips for k = 7 clusters





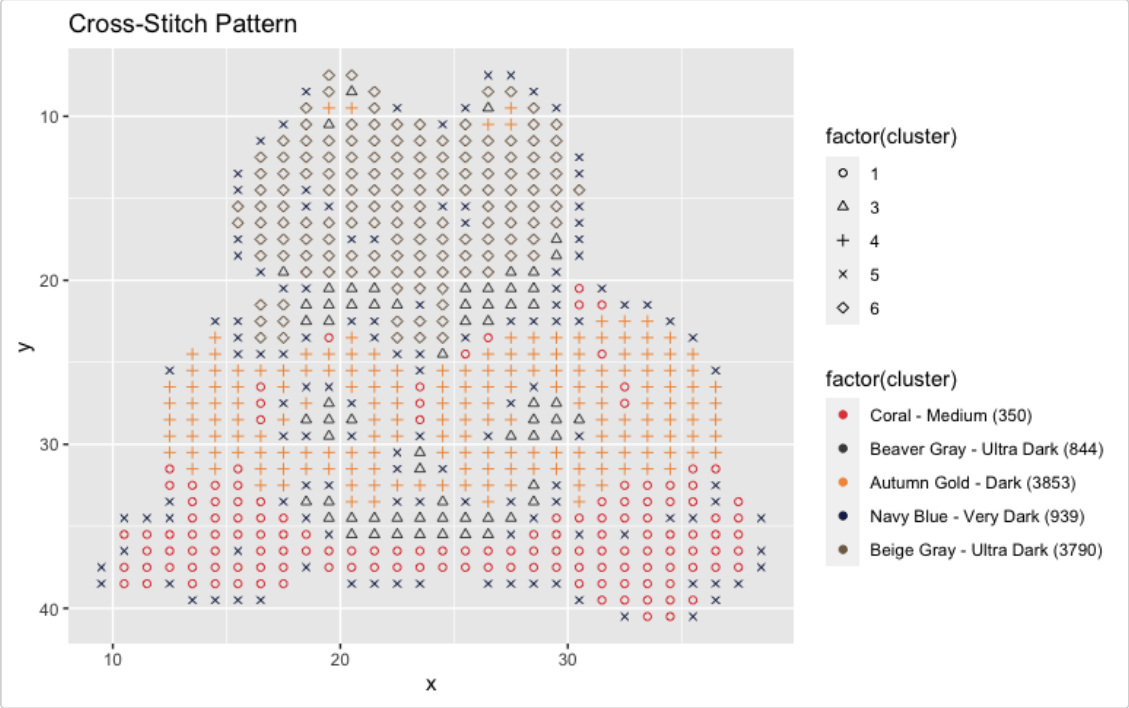
I will pick $k = 5$ clusters since the added colours for 6 clusters become similar colours of grey $(\#7F6A55, \#484848)$. Let's get to the exiting part again, to actually compute the cross-stitches pattern!

```
make_pattern(cluster_info2, k = 5, x_size = 50)
```



We can clearly see that the tan colour is the background colour that we do not need to stitch (one can stitch onto a tan colour cloth, save themselves from stitching so many entries). Let's remove the background colour "Tan - Ultra Very Light (739)" colour from our pumpkin cat image by entering the argument `background_colour = "739"`, which is the number in the bracket in the legend as previously mentioned.

```
make_pattern(cluster_info2, k = 6, x_size = 50, background_colour = "739")
```



Now we only have the pumpkin and the cat, no more stitches for the background.

This is the end of the tutorial for how to generate a cross-stitch pattern of a given image using the 4 functions in functions.R. If you are curious why there's an extra function called `change_resolution()` in the functions.R file, it is because `make_pattern()` uses it to lower the quality of the image for a better performance of making stitching patterns.

I believe that you are an expert now on how to generate cross-stitch pattern for an image. It's time for you to start stitching! Enjoy :)