now down a few collages from R using magick: [the faces of #rstats Twitter](http://www.masalmon.eu/2017/03/19/facesofr/), [We R-Ladies](http://livefreeordichotomize.com/2017/07/18/the-making-of-we-r-ladies/) with Lucy D’Agostino McGowan, The faces of #rstats Twitter and holiday card collages were arranged at random, while the We R-Ladies one was a mosaic forming the R-Ladies logo. I got the idea to up my collage skills by trying to learn how to arrange pics by their main colour, like a rainbow. The verb rainbow doesn’t exist, and [“rainbowing”](https://en.wikipedia.org/wiki/Rainbowing) doesn’t mean ordering by colour, but I didn’t let this stop me.

It was the occasion to grab some useful knowledge about colours, not useless for someone who did not even know about [Pantone’s Colors of the Year](https://en.wikipedia.org/wiki/Pantone#Color_of_the_Year) a few weeks ago…

*This post has nothing to do with Kesha’s new album. However, you can listen to it while reading since it’s so good, but maybe switch to something older from her when I use “$”.*

**Getting some pics to play with**

The first pictures I tried to arrange were all the pictures ever posted by R-Ladies local chapters on their Twitter account. While it was fun to grab them all, it was not very interesting to play with them as so many of them were pictures of screens. I therefore grabbed “nature” pictures from [Pexels](https://www.pexels.com/) using the same method [as when creating the Bubblegum Puppies]. I chose “nature” as a keyword because 1) it lead to many hits 2) it offered a good variety of colours.

library("rvest")

library("RSelenium")

library("magrittr")

rD <- rsDriver()

remDr <- rD[["client"]]

# open the webpage

remDr$navigate("https://www.pexels.com/search/nature/")

# scroll down

for(i in 1:130){

remDr$executeScript(paste("scroll(0,",i\*10000,");"),

args = list("dummy"))

# be nice and wait

Sys.sleep(1)

}

# https://www.pexels.com/faq/

page\_content <- remDr$getPageSource()

remDr$close()

get\_link\_from\_src <- function(node){

xml2::xml\_attrs(node)["src"] %>%

as.character() %>%

stringr::str\_replace("\\?h.\*", "")

}

xtract\_pic\_links <- function(source) {

css <- '.photo-item\_\_img'

read\_html(source[[1]]) %>%

html\_nodes(css) %>%

purrr::map\_chr(get\_link\_from\_src)

}

links <- xtract\_pic\_links(page\_content)

links <- links[1:1400]

# save

dir.create("nature")

save\_pic <- function(url){

Sys.sleep(1)

name <- stringr::str\_replace(url, ".\*\\/", "")

try(magick::image\_read(url) %>%

magick::image\_write(paste0("nature/", name)),

silent = TRUE)

}

purrr::walk(links, save\_pic)

**Extracting the main colour and making pics size-compatible**

In the following code, I extracted the main colour from each pic using [Russel Dinnage’s method](http://www.mepheoscience.com/colourful-ecology-part-1-extracting-colours-from-an-image-and-selecting-them-using-community-phylogenetics-theory/) as presented [in this blog post from David Smith](http://blog.revolutionanalytics.com/2015/03/color-extraction-with-r.html). Before that I had to install two packages from Github, [rblocks](https://github.com/ramnathv/rblocks) and [rPlotter](https://github.com/woobe/rPlotter).

This code also serves another role: since I wanted to paste pics together at some point, I decided to make them all of the same dimensions by adding a border with magick. I used the main colour I’d just extracted. The most important points to make a picture a square are to know magick::image\_info gives you the height and width of an image… and to somehow understand geometry which was embarrassingly a hurdle when I did that.

The code to extract colours didn’t work in a few cases which I did not investigate a lot: I had downloaded more pics than what I needed because I had experienced the issue when working with R-Ladies meetups pics, and had seen it was because of seemingly bicolor pics.

dir.create("formatted\_pics")

format\_image <- function(path){

image <- magick::image\_read(path)

info <- magick::image\_info(image)

# find in which direction I need to add pixels

# to make this a square

direction <- ifelse(info$height > info$width,

"height", "width")

scale\_number <- as.numeric(info[direction]/500)

image <- magick::image\_scale(image, paste0(info["width"]/scale\_number,

"x",

info["height"]/scale\_number))

newinfo <- magick::image\_info(image)

# colours

colours <- try(rPlotter::extract\_colours(path, num\_col = 1), silent = TRUE)

# one pic at least was problematic

if(!is(colours, "try-error")){

colour <- colours[1]

image <- magick::image\_border(image, colour, paste0((500-newinfo$width)/2, "x",

(500-newinfo$height)/2))

info <- magick::image\_info(image)

# odd numbers out!

if(info$height/2 != floor(info$height/2)){

image <- magick::image\_crop(image, "0x500+0")

}

if(info$width/2 != floor(info$width/2)){

image <- magick::image\_crop(image, "500x0+0")

}

magick::image\_write(image,

stringr::str\_replace(path, "nature", "formatted\_pics"))

tibble::tibble(path = path,

colour = colour)

}else{

NULL

}

}

pics\_main\_colours <- purrr::map\_df(dir("nature", full.names = TRUE), format\_image)

readr::write\_csv(pics\_main\_colours, path = "pics\_main\_colours.csv")

And because I’m apparently a bad planner, I had to reduce pictures afterwards.

# we need smaller images

reduce\_image <- function(path){

magick::image\_read(path) %>%

magick::image\_scale("50x50!") %>%

magick::image\_write(path)

}

purrr::walk(dir("formatted\_pics", full.names = TRUE),

reduce\_image)

**Preparing a function to order and paste pictures**

This function has a collage part which you might recognize from my [the faces of #rstats Twitter](http://www.masalmon.eu/2017/03/19/facesofr/) blog post, and a ordering pictures according to a variable part that’s new and uses a bit of tidy eval… Maybe I’ll really learn tidy eval this year! pics\_info needs to be a data.frame with the path to pictures and well the variable one wants to use to order them.

library("rlang")

make\_column <- function(i, files, no\_rows){

magick::image\_read(files[(i\*no\_rows+1):((i+1)\*no\_rows)]) %>%

magick::image\_append(stack = TRUE) %>%

magick::image\_write(paste0("cols/", i, ".jpg"))

}

make\_collage <- function(pics\_info, no\_rows, no\_cols, ordering\_col){

pics\_info <- dplyr::arrange(pics\_info, !!!syms(ordering\_col))

pics\_info <- pics\_info[1:(no\_rows\*no\_cols),]

pics\_info <- dplyr::mutate(pics\_info, column = rep(1:no\_cols, each = no\_rows))

pics\_info <- dplyr::group\_by(pics\_info, column) %>%

dplyr::arrange(!!!syms(ordering\_col)) %>%

dplyr::mutate(row = 1:no\_rows) %>%

dplyr::ungroup()

pics\_info <- dplyr::arrange(pics\_info, column, row)

dir.create("cols")

purrr::walk(0:(no\_cols-1), make\_column, files = pics\_info$path,

no\_rows = no\_rows)

banner <- magick::image\_read(dir("cols/", full.names = TRUE)) %>%

magick::image\_append(stack = FALSE)

unlink("cols", recursive = TRUE)

return(banner)

}

The function returns a magick object that one can then write to disk as a PNG for instance.

I first tested it using a random approach added to the data.frame created in the next section, and show the result here to give an idea of the variety of pictures. For many of them, however, the main colour that you can see in their border is greyish.

set.seed(42)

pics\_info <- dplyr::mutate(pics\_info, random = sample(1:nrow(pics\_info), nrow(pics\_info)))

make\_collage(pics\_info, 19, 59, "random") %>%

magick::image\_write("data/2018-01-07-rainbowing-banner\_random.png")

**Testing a first (bad) approach: using hue**

Once I had the main colour as an hex code, I had no idea how to order the colours and thought a good idea would be to use hue, which is the main wave length in a colour. Most observed colours are a mix of wave lengths unless you’re using a laser for instance. To get hue from a colour identified by its hex code, one needs two functions: colorspace::hex2rgb and grDevices::rgb2hsv. The latter one outputs hue, saturation and value. Hue is the main wavelength, saturation the amount of that wavelength in the colour and value the amount of light in the colour. The smaller the saturation, the less representative the hue is of the main colour. Add to that the fact that the main colour can also be only a little representative of your original picture… Ordering by hue isn’t too perfect, but I tried that anyway.

# now work on getting the hue and value for all pics

# create a data.frame with path, hue, value

get\_values <- function(path, pics\_main\_colours){

print(path)

# get main color

colour <- pics\_main\_colours$colour[pics\_main\_colours$path == stringr::str\_replace(path,

"formatted\_pics",

"nature")]

# translate it

rgb <- colorspace::hex2RGB(colour)

values <- grDevices::rgb2hsv(t(rgb@coords))

tibble::tibble(path = path,

hue = values[1,1],

saturation = values [2,1],

value = values[3,1])

}

# all values

pics\_col <- purrr::map\_df(dir("formatted\_pics", full.names = TRUE),

get\_values, pics\_main\_colours)

make\_collage(pics\_info, 19, 59, "hue") %>%

magick::image\_write("banner\_hue.png")

So this is not too pretty. Blue and green pictures seem to cluster together but there are very dark pictures which we’d intuitively put aside.

So I thought a bit more and decided to first assign main colours to a reference colour and then order pictures based on this…

**Choosing a better approach: RGB and distances**

The first challenge was to choose reference colours which’d be a rainbow slices. I could have looked up RGB codes corresponding to [ROYGBIV (red, orange, yellow, green, blue, indigo and violet.)](https://en.wikipedia.org/wiki/ROYGBIV) but I had read about [xkcd colors survey](https://blog.xkcd.com/2010/05/03/color-survey-results/) in [this interesting post](https://blog.algolia.com/how-we-handled-color-identification/) and therefore decided to use XKCD colors, available in R via the xkcdcolors package. I chose to use the 18 most common ones, and add black to that lot. It was no longer really a rainbow, I agree. The colors present in the pictures were ordered by hand by my husband, and I like his choices.

Then to assign each pic to a reference colour via its main colour, I calculated the Euclidian distance between that colour and all reference colours to find the closes reference colours, using the RGB values.

library("xkcdcolors")

library("magrittr")

# version of colorspace::hex2RGB returning a df

hex2rgb <- function(hex){

result <- colorspace::hex2RGB(hex)@coords

}

# https://stackoverflow.com/questions/45328221/unnest-one-column-list-to-many-columns-in-tidyr

colors <- tibble::tibble(name = c(xcolors()[1:18], "black"),

hex = name2color(name),

rgb = purrr::map(hex, hex2rgb)) %>%

dplyr::mutate(rgb = purrr::map(rgb, tibble::as\_tibble)) %>%

tidyr::unnest()

# for each colour I want the closest one.

find\_closest\_colour <- function(hex, colors){

test <- tibble::tibble(hex = hex) %>%

dplyr::mutate(rgb = purrr::map(hex, hex2rgb),

rgb = purrr::map(rgb, tibble::as\_tibble)) %>%

tidyr::unnest()

distance <- stats::dist(rbind(test[, c("R", "G", "B")],

colors[, c("R", "G", "B")]))

colors$name[which.min(as.matrix(distance)[,1][2:(nrow(colors) + 1)])]

}

imgs\_col <- dplyr::mutate(pics\_main\_colours,

xkcd\_col = purrr::map\_chr(colour, find\_closest\_colour,

colors = colors))

readr::write\_csv(imgs\_col, path = "imgs\_xkcd\_col.csv")

Once I had this information about each pic, I could order the pictures, after having defined the order of the reference colours.

pics\_info <- readr::read\_csv("imgs\_xkcd\_col.csv")

pics\_info <- dplyr::mutate(pics\_info,

path = stringr::str\_replace(path, "nature", "formatted\_pics"))

pics\_info <- dplyr::mutate(pics\_info,

xkcd\_col = factor(xkcd\_col, ordered = TRUE,

levels = c("black","brown","red","magenta","pink",

"lime green","green","dark green","teal",

"light blue","sky blue","blue","purple","grey")))

This looks much better, but of course the initial set (and maybe the used extraction method as well) don’t provide for enough colours to make this extremely pretty. I’m not sure how useful this end product is, but hey I got to look at pretty landscapes full of colours from my grey rainy city, and learnt a lot along the way… Besides, maybe *you* will find a cool use case of some of the colour methods featured here and will tell me about it in the comments?