**My (not bad) stock photo source**

**Pexels metadata for the win**

Where to find information related to stock photos? I already used Pexels, a website providing CC0 pictures which is quite nice. My goal was to obtain the titles and the tags of stock photos of “data science”:

**Responsible webscraping**

When researching this post I discovered that Pexels has an API, documented . but this API does not get you the title nor the tags associated to a picture so only webscraping could get me what I needed.

Webscraping is a powerful tool allowing one to rectangle webpages but with great power comes great responsability. Being *able* to scrape a webpage does not mean you are *allowed* to. You could get sued or your IP could get blocked. I am far from being an expert but I often read where I learnt about rOpenSci’s robotstxt package that does “robots.txt file parsing and checking for R” which in plain language means it checks for you what a webpage legally allows you to do. See below,

# how I'll find pictures

robotstxt::paths\_allowed("https://www.pexels.com/search")

## [1] TRUE

# where tags live

robotstxt::paths\_allowed("https://www.pexels.com/photo")

## [1] TRUE

robots.txt files often also tell you how often you can hit a page by defining a “crawling delay”. Sadly Pexels robots.txt doesn’t:

robotstxt::get\_robotstxt("https://www.pexels.com")

## Sitemap: https://s3.amazonaws.com/pexels/sitemaps/sitemap.xml.gz

But Bob Rudis, who was patient and nice enough to answer my questions, told me that I should probably respect the rate limit defined in Pexels API docs. “Do not abuse the API. The API is rate-limited to 200 requests per hour and 20,000 requests per month.” As I recently explained, these days to limit rate of a function I use the very handy ratelimitr package.

limited\_get <- ratelimitr::limit\_rate(httr::GET,

ratelimitr::rate(200, 60\*60),# not more than 200 times an hour

ratelimitr::rate(1, 5))#not more than 1 time every 5 seconds

**Elegant webscraping**

At the time of the two aforelinked blog posts I had used RSelenium to scroll down and get the download link of many pictures, but an elegant and cool alternative using query parameters, on which I’ll build in this post.

Code Chunks - Query Parameters Method of Rainbowing Set of Pictures

|  |
| --- |
| library(V8) |
|  | library(xml2) |
|  | library(httr) |
|  | library(rvest) |
|  | library(stringi) |
|  | library(tidyverse) |
|  |  |
|  | get\_page <- function(num=1, seed=Sys.Date()) { |
|  |  |
|  | GET( |
|  | url = "https://www.pexels.com/search/nature/", |
|  | query = list( |
|  | page=num, |
|  | format="js", |
|  | seed=seed |
|  | ) |
|  | ) -> res |
|  |  |
|  | stop\_for\_status(res) |
|  |  |
|  | x <- content(res) |
|  | x <- stri\_replace\_first\_regex(x, "^.\*beforeend','\\\\n\\\\n", "'") |
|  | x <- stri\_replace\_last\_regex(x, "\\\\n\\\\n'\\);rowG.\*$", "'") |
|  |  |
|  | ctx <- v8() |
|  |  |
|  | pg <- read\_html(ctx$eval(x)) |
|  |  |
|  | data\_frame( |
|  | preview\_href = html\_attr(html\_nodes(pg, "img"), "src"), |
|  | full\_href = sprintf("https://www.pexels.com%s", html\_attr(html\_nodes(pg, "a"), "href")), |
|  | title = html\_attr(html\_nodes(pg, "a"), "title") |
|  | ) |
|  |  |
|  | } |

I first re-wrote the function to get all 15 pictures of each page of results.

get\_page <- function(num = 1, seed = 1) {

message(num)

limited\_get(

url = "https://www.pexels.com/search/data science/",

query = list(

page=num,

seed=seed

)

) -> res

httr::stop\_for\_status(res)

pg <- httr::content(res)

tibble::tibble(

url = rvest::html\_attr(rvest::html\_nodes(pg, xpath = "//a[@class='js-photo-link']"), "href"),

title = rvest::html\_attr(rvest::html\_nodes(pg, xpath = "//a[@class='js-photo-link']"), "title"),

tags = purrr::map(url, get\_tags)

)

}

I re-wrote it because I needed the “href” and because it seems that the structure of each page changed a bit since the day on which the gist was written. To find out I had to write “a[@class=’js-photo-link’]” I inspected the source of a page.

Then I wrote a function getting tags for each picture.

get\_tags <- function(url){

message(url)

url <- paste0("https://www.pexels.com", url)

res <- limited\_get(url)

httr::stop\_for\_status(res)

pg <- httr::content(res)

nodes <- rvest::html\_nodes(pg, xpath = '//a[@data-track-label="tag" ]')

rvest::html\_text(nodes)

}

And finally I got results for 20 pages. I chose 20 without thinking too much. It seemed enough for my needs, and each of these pages had pictures.

ds\_stock <- purrr::map\_df(1:20, get\_page)

ds\_stock <- unique(ds\_stock)

ds\_stock <- tidyr::unnest(ds\_stock, tags)

I got 300 unique pictures.

**What’s in a data science stock photo?**

Now that I have all this information at hand, I can describe data science stock photos!

**Data science tags**

library("ggplot2")

library("ggalt")

library("hrbrthemes")

tag\_counts <- dplyr::count(ds\_stock, tags, sort = TRUE)[1:10,]

dplyr::mutate(tag\_counts,

tags = reorder(tags, n)) %>%

ggplot() +

geom\_lollipop(aes(tags, n),

size = 2, col = "salmon") +

hrbrthemes::theme\_ipsum(base\_size = 16,

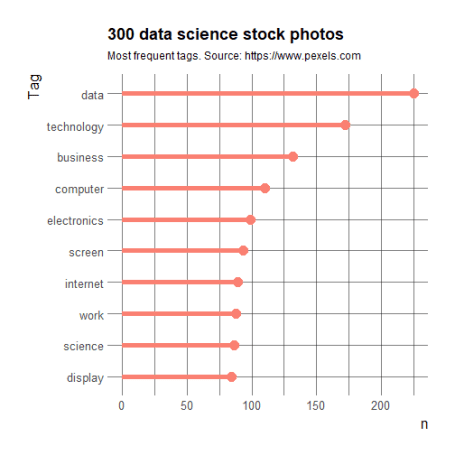
axis\_title\_size = 16) +

xlab("Tag") +

ggtitle("300 data science stock photos",

subtitle = "Most frequent tags. Source: https://www.pexels.com") +

coord\_flip()



So the most common tags are data, technology, business and computer. Not too surprising!

**Data science scenes**

Now, let’s have a look at *titles* that are in general more descriptive of what’s happening/present on the photo (i.e. is the computer near a cup of coffee or is someone working on it). One issue was probably my having too few titles: only half of pictures have titles!

stopwords <- rcorpora::corpora("words/stopwords/en")$stopWords

ds\_stock %>%

dplyr::filter(!is.na(title)) %>%

dplyr::select(title) %>%

unique() %>%

tidytext::unnest\_tokens(bigram, title,

token = "ngrams", n = 2) %>%

tidyr::separate(bigram, c("word1", "word2"), sep = " ",

remove = FALSE) %>%

dplyr::filter(!word1 %in% stopwords) %>%

dplyr::filter(!word2 %in% stopwords)%>%

dplyr::count(bigram, sort = TRUE) %>%

dplyr::mutate(bigram = reorder(bigram, n)) %>%

head(n = 10)%>%

ggplot() +

geom\_lollipop(aes(bigram, n),

size = 2, col = "salmon") +

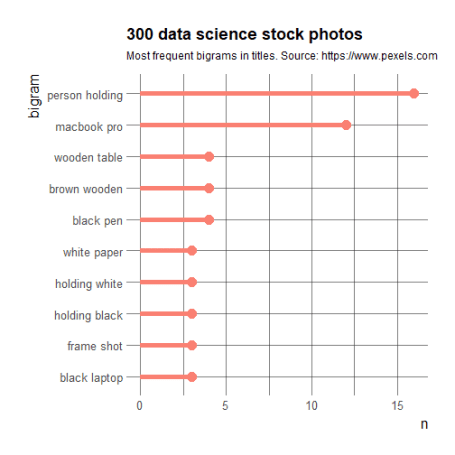
hrbrthemes::theme\_ipsum(base\_size = 16,

axis\_title\_size = 16) +

ggtitle("300 data science stock photos",

subtitle = "Most frequent bigrams in titles. Source: https://www.pexels.com")+

coord\_flip()



So there’s a lot of holding computer happening, and these laptops are either black or white… And well Macbook Pro probably looks more professional?

Code Chunks - Rainbowing a Set Of Pictures

**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 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.

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 had learnt how to do that when preparing , but this time instead of using the same colour every time (R-Ladies’ official purple), 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 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.

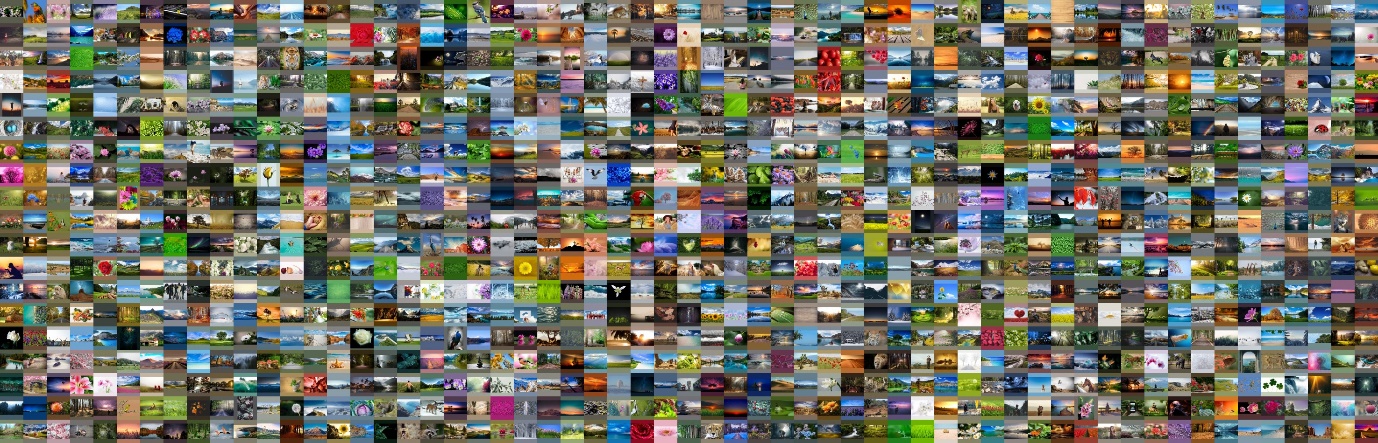
I first tested it using a random approach added to the data.frame created in the next section, and show the result here to give an idea of the variety of pictures. For many of them, however, the main colour that you can see in their border is greyish.

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  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

}

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"))

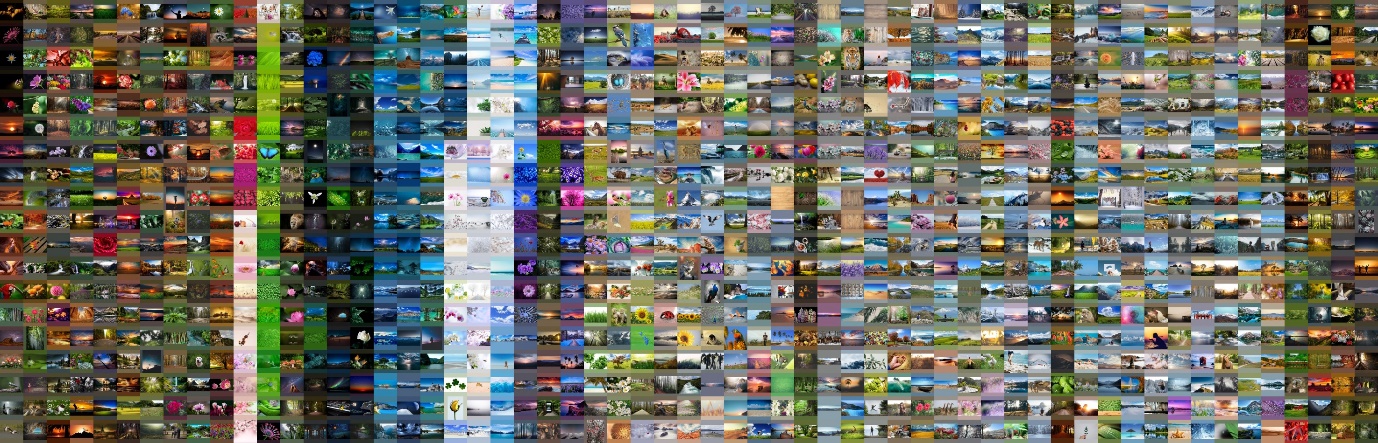
pics\_info <- dplyr::mutate(pics\_info,

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?

**Hold my laptop and watch…**

my trying to find a good post conclusion! In this post, I tried to responsibly and elegantly scrape rich photo metadata from Pexels to characterize stock photos of data science. Using tags, and most common bigrams in titles, I found that data science stock photos are associated with data, business and computers; and that they often show people holding computers. Now, you’ll excuse me while I try and comfort my poor pink laptop that feels a bit too un-data-sciency.