Both my osqueryr and macthekinfe packages have had a few updates and I wanted to put together a fun example (it being Friday, and all) for what you can do with them. All my packages are now on GitHub and GitLab and I’ll be maintaining them on both so I can accommodate the comfort-level of any and all contributors but will be prioritizing issues and PRs on GitLab ahead of any other platform. Having said that, I’ll mark non-CRAN packages with a # notcran comment in the source views so you know you need to install it from wherever you like to grab sketch packages from.

One table that osquery makes available under macOS is an inventory of all “apps” that macOS knows about. Previous posts have shown how to access these tables via the dplyr interface I built for osquery, but they involved multiple steps and as I started to use it more regularly (especially to explore the macOS 10.14 beta I’m running) I noticed that it could use some helper functions. One in particular — osq\_expose\_tables() — is pretty helpful in that it handles all the dplyr boilerplate code and makes table(s) available in the global environment by name. It takes a single table name or regular expression and then exposes all matching entities. While the function has a help page, it’s easier just to see it in action. Let’s expose the apps table:

library(osqueryr) # notcran

library(tidyverse)

osq\_expose\_tables("apps")

apps

## # Source: table [?? x 19]

## # Database: OsqueryConnection

## applescript\_enab… bundle\_executable bundle\_identifier bundle\_name bundle\_package\_…

##

## 1 0 1Password 6 com.agilebits.onep… 1Password 6 APPL

## 2 0 2BUA8C4S2C.com.agil… 2BUA8C4S2C.com.agi… 1Password m… APPL

## 3 1 Adium com.adiumX.adiumX Adium APPL

## 4 1 Adobe Connect com.adobe.adobecon… Adobe Conne… APPL

## 5 1 Adobe Illustrator com.adobe.illustra… Illustrator… APPL

## 6 "" AIGPUSniffer com.adobe.AIGPUSni… AIGPUSniffer APPL

## 7 "" CEPHtmlEngine Helper com.adobe.cep.CEPH… CEPHtmlEngi… APPL

## 8 "" CEPHtmlEngine com.adobe.cep.CEPH… CEPHtmlEngi… APPL

## 9 "" LogTransport2 com.adobe.headligh… LogTranspor… APPL

## 10 "" droplet "" Analyze Doc… APPL

## # ... with more rows, and 14 more variables: bundle\_short\_version ,

## # bundle\_version , category , compiler , copyright ,

## # development\_region , display\_name , element , environment ,

## # info\_string , last\_opened\_time , minimum\_system\_version , name ,

## # path

There’s tons of info on all the apps macOS knows about, some of which are system services and “helper” apps (like Chrome’s auto-updater). One field — last\_opened\_time — caught my eye and I thought it would be handy to see which apps had little use (i.e. ones that haven’t been opened in a while) and which apps I might use more frequently (i.e. ones with more recent “open” times). That last\_open\_time is a fractional POSIX timestamp and, due to the way osquery created the schemas, it’s in a character field. That’s easy enough to convert and then arrange() the whole list in descending order to let you see what you use most frequently.

But, this is R and we can do better than a simple table or even a DT::datatable().

I recently added the ability to read macOS property lists (a.k.a. “plists”) to mactheknife by wrapping a Python module (plistlib). Since all (OK, “most”) macOS apps have an icon, I thought it would be fun to visualize the last opened frequency for each app using the app icons and ggplot2. Unfortunately, the ImageMagick (and, thus the magick package) cannot read macOS icns files, so you’ll need to do a brew install libicns before working with any of the remaining code since we’ll be relying on a command-line utility from that formula.

Let’s get the frontmatter out of the way:

library(sys)

library(magick)

library(osqueryr) # notcran

library(mactheknife) #notcran

library(ggimage)

library(hrbrthemes)

library(ggbeeswarm)

library(tidyverse)

osq\_expose\_tables("apps")

# macOS will use a generic app icon when none is present in an app bundle; this is the location and we'll

# need to use it when our plist app spelunking comes up short

default\_app <- "/System/Library/CoreServices/CoreTypes.bundle/Contents/Resources/GenericApplicationIcon.icns"

Next, we'll:

* collect the apps table locally
* filter out system-ish things (which we really don't care about for this post)
* convert the last used time to something useful (and reduce it to a day resolution)
* try to locate the property list for the app and read the path to the app icon file, substituting the generic one if not found (or other errors pop up):

select(apps, name, path, last\_opened\_time) %>%

collect() %>%

filter(!str\_detect(path, "(^/System|usr|//System|/Library/|Helper|/Contents/|\\.service$)")) %>%

mutate(lop\_day = as.Date(anytime::anytime(as.numeric(last\_opened\_time)))) %>%

mutate(icon = map\_chr(path, ~{

p <- read\_plist(file.path(.x, "Contents", "Info.plist"))

icns <- p$CFBundleIconFile[1]

if (is.null(icns)) return(default\_app)

if (!str\_detect(icns, "\\.icns$")) icns <- sprintf("%s.icns", icns)

file.path(.x, "Contents", "Resources", icns)

})) -> apps\_df

apps\_df

## # A tibble: 274 x 5

## last\_opened\_time name path lop\_day icon

##

## 1 1529958322.11297 1Password 6.app /Applications/1Password … 2018-06-25 /Applications/1Password 6.…

## 2 1523889402.80918 Adium.app /Applications/Adium.app 2018-04-16 /Applications/Adium.app/Co…

## 3 1516307513.7606 Adobe Connect.app /Applications/Adobe Conn… 2018-01-18 /Applications/Adobe Connec…

## 4 1530044681.76677 Adobe Illustrator.app /Applications/Adobe Illu… 2018-06-26 /Applications/Adobe Illust…

## 5 -1.0 Analyze Documents.app /Applications/Adobe Illu… 1969-12-31 /Applications/Adobe Illust…

## 6 -1.0 Make Calendar.app /Applications/Adobe Illu… 1969-12-31 /Applications/Adobe Illust…

## 7 -1.0 Contact Sheets.app /Applications/Adobe Illu… 1969-12-31 /Applications/Adobe Illust…

## 8 -1.0 Export Flash Animation.app /Applications/Adobe Illu… 1969-12-31 /Applications/Adobe Illust…

## 9 -1.0 Web Gallery.app /Applications/Adobe Illu… 1969-12-31 /Applications/Adobe Illust…

## 10 -1.0 Adobe InDesign CC 2018.app /Applications/Adobe InDe… 1969-12-31 /Applications/Adobe InDesi…

## # ... with 264 more rows

Since I really didn't feel like creating a package wrapper for libicns, we're going to use the sys package to make system calls to convert the icns files to png files. We *really* don't want to do this repeatedly for the same files if we ever run this again, so we'll setup a cache directory to hold our converted pngs.

Apps can (and, usually do) have multiple icons with varying sizes and are not guaranteed to have every common size available. So, we'll have the libicns icns2png utility extract *all* the icons and use the highest resolution one, using magick to reduce it to a 32x32 png bitmap.

# setup the cache dir -- use whatever you want

cache\_dir <- path.expand("~/.r-icns-cache")

dir.create(cache\_dir)

# create a unique name hash for more compact names

mutate(apps\_df, icns\_png = map\_chr(icon, ~{

hash <- digest::digest(.x, serialize=FALSE)

file.path(cache\_dir, sprintf("%s.png", hash))

})) -> apps\_df

# find the icns2png program

icns2png <- unname(Sys.which("icns2png"))

# go through each icon file

pb <- progress\_estimated(length(apps\_df$icns\_png))

walk2(apps\_df$icon, apps\_df$icns\_png, ~{

pb$tick()$print() # progress!

if (!file.exists(.y)) { # don't create it if it already exists

td <- tempdir()

# no icon file == use default one

if (!file.exists(.x)) .x <- default\_app

# convert all of them to pngs

sys::exec\_internal(

cmd = icns2png,

args = c("-x", "-o", td, .x),

error = FALSE

) -> res

rawToChar(res$stdout) %>% # go through icns2png output

str\_split("\n") %>%

flatten\_chr() %>%

keep(str\_detect, " Saved") %>% # find all the extracted icons

last() %>% # use the last one

str\_replace(".\* to /", "/") %>% # clean up the filename so we can read it in

str\_replace("\\.$", "") -> png

# read and convert

image\_read(png) %>%

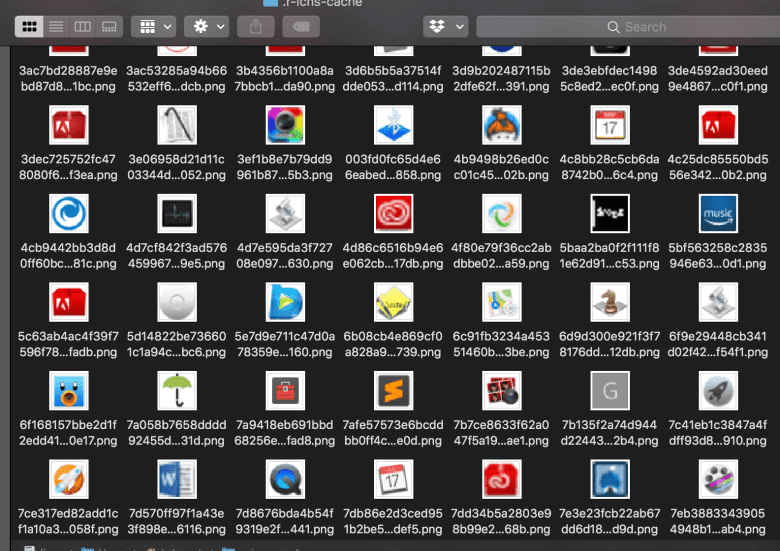
image\_resize(geometry\_area(32, 32)) %>%

image\_write(.y)

}

})

You can open up that cache directory with the macOS finder to find all the extracted/converted pngs.

[](https://rud.is/b/2018/07/06/visualizing-macos-app-usage/screen-shot-2018-07-06-at-14-35-03/)

Now, we're on the final leg of our app-use visualization journey.

Some system/utility apps have start-of-epoch dates due to the way the macOS installer tags them. We only want "recent" ones so I set an arbitrary cutoff date of the year 2000. Since many apps would have the same last opened date, I wanted to get a spread out layout "for free". One way to do that is to use ggbeeswarm::position\_beswarm():

filter(apps\_df, lop\_day > as.Date("2000-01-01")) %>%

ggplot() +

geom\_image(

aes(x="", lop\_day, image = icns\_png), size = 0.033,

position = position\_quasirandom(width = 0.5)

) +

geom\_text(

data = data\_frame(

x = c(0.6, 0.6),

y = as.Date(c("2018-05-01", "2017-09-15")),

label = c("More recently used ↑", "Not used in a while ↓")

),

aes(x, y, label=label), family = font\_an, size = 5 , hjust = 0,

color = "lightslategray"

) +

labs(x = NULL, y = "Last Opened Time") +

labs(

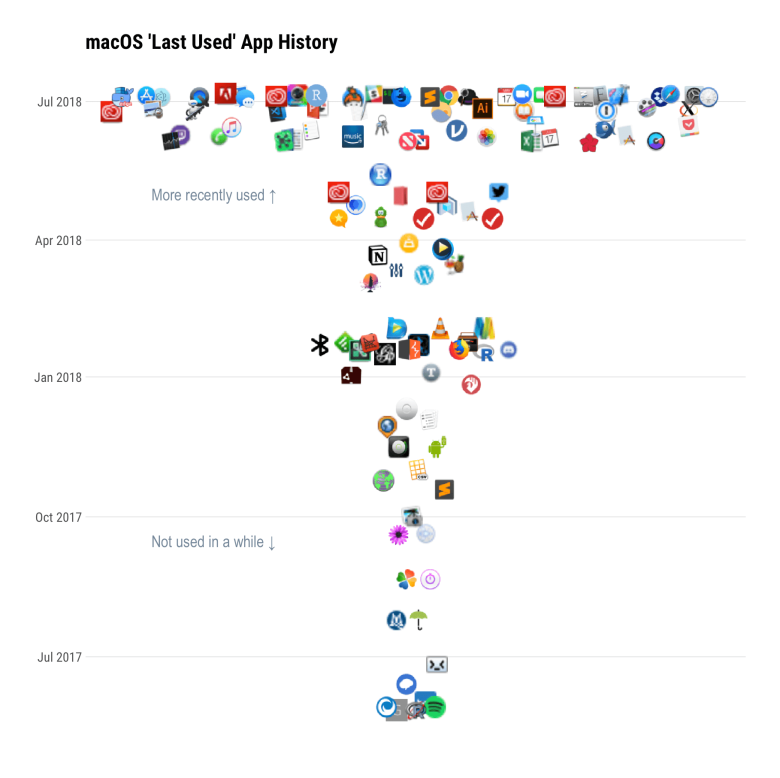
x = NULL, y = NULL,

title = "macOS 'Last Used' App History"

) +

theme\_ipsum\_rc(grid="Y") +

theme(axis.text.x = element\_blank())

[](https://rud.is/b/2018/07/06/visualizing-macos-app-usage/app-lod-tree-1/)

There are tons of other ways to look at this data and you can use the osquery daemon to log this data regularly so you can get an extra level of detail. An interesting offshot project would be to grab the latest RStudio dailies and see if you can wrangle a sweet D3 visualization from the app data we collected. Make sure to drop a comment with your creations in the comments.