There’s a *yuge* chance you’re reading this post (at least initially) on R-Bloggers right now (though you should also check out [R Weekly](https://rweekly.org/) and add their live feed to your RSS reader pronto!). It’s a central “watering hole” for R folks and is read by many (IIRC over 20,000 Feedly users have it in their OPML).

I’m *addicted* to [Feedly](https://feedly.com/) and waited *years* for them to publish their API. [They have](https://developer.feedly.com/) and there will eventually be a package for it (go for it if you want to get’er done before me since I won’t have time to do it justice for a while). As just parenthetically noted, I’ve started work on one and have scaffolded *just enough* to give R folks a present: almost 5 years of R-Bloggers data — posts, engagement rates, authors, etc). *But*, you’ll have to put up with some expository, first.

**Digging In**

We’ll need some packages to help this expository and extraction. Plus, you’ll need to go to <https://developer.feedly.com/> to get your developer token (NOTE: this requires a “Pro” account *or* a regular account and you manually doing the OAuth dance to get an access token; any final “Feedly package” by myself or others will likely use OAuth) and store it in your ~/.Renviron in FEEDLY\_ACCESS\_TOKEN.

I’ve sliced and diced bits from the (non-published) fledgling package to give a peek behind the API covers. There’s *plenty* of exposition in the following code block comment header to describe what it does:

#' Simplifying some example package setup for this non-pkg example

.pkgenv <- new.env(parent=emptyenv())

.pkgenv$token <- Sys.getenv("FEEDLY\_ACCESS\_TOKEN")

#' In reality, this is more complex since the non-toy example has to

#' refresh tokens when they expire.

.feedly\_token <- function() {

return(.pkgenv$token)

}

#' Get a chunk of a Feedly "stream"

#'

#' For the purposes of this short example, consider a

#' "stream" to be all the historical items in a feed.

#' (Note: the definition is more complex than that)

#'

#' Max "page size" (mad numbner of items returned in a single call)

#' is 1,000. For example simplicity, there's a blanket assumption

#' that if `continuation` is actually present, the caller is

#' savvy and asked for a large number of items (e.g. 10,000).

#' Therefore, assume we're paging by the thousands.

#'

#' @md

#' @param feed\_id the id of the stream (for this examplea feed id)

#' @param ct numnber of items to retrieve (API will only return 1,000

#' items for a single response and populate `continuation`

#' with a value that should be passed to subsequent calls

#' to page through the results; `ct` will be reset to 1,000

#' internally if this is the case)

#' @param continuation see `ct`

#' @references

#' @return for this example, an ugly `list`

feedly\_stream <- function(stream\_id, ct=100L, continuation=NULL) {

ct <- as.integer(ct)

if (!is.null(continuation)) ct <- 1000L

httr::GET(

url = "https://cloud.feedly.com/v3/streams/contents",

httr::add\_headers(

`Authorization` = sprintf("OAuth %s", .feedly\_token())

),

query = list(

streamId = stream\_id,

count = ct,

continuation = continuation

)

) -> res

httr::stop\_for\_status(res)

res <- httr::content(res, as="text")

res <- jsonlite::fromJSON(res)

res

}

We’ll grab 10,000 Feedly entries for the R-Bloggers feed stream:

r\_bloggers\_feed\_id <- "feed/http://feeds.feedburner.com/RBloggers"

rb\_stream <- feedly\_stream(r\_bloggers\_feed\_id, 10000L)

# preallocate space

streams <- vector("list", 10)

streams[1L] <- list(rb\_stream)

# gotta catch'em all!

idx <- 2L

while(length(rb\_stream$continuation) > 0) {

cat(".", sep="") # poor dude's progress par

feedly\_stream(

stream\_id = r\_bloggers\_feed\_id,

ct = 1000L,

continuation = rb\_stream$continuation

) -> rb\_stream

streams[idx] <- list(rb\_stream)

idx <- idx + 1L

}

cat("\n")

For those who aren’t used to piecing together bits from API’s like this (and for those who do not have a Pro account, those who didn’t want to write OAuth code or those who don’t use Feedly and cannot reproduce the post example), here’s some dissection:

str(streams, 1)

## List of 12

## $ :List of 7

## $ :List of 7

## $ :List of 7

## $ :List of 7

## $ :List of 7

## $ :List of 7

## $ :List of 7

## $ :List of 7

## $ :List of 7

## $ :List of 7

## $ :List of 7

## $ :List of 6 # No "continuation" in this one

str(streams[[1]], 1)

## List of 7

## $ id : chr "feed/http://feeds.feedburner.com/RBloggers"

## $ title : chr "R-bloggers"

## $ direction : chr "ltr"

## $ updated : num 1.52e+12

## $ alternate :'data.frame': 1 obs. of 2 variables:

## $ continuation: chr "15f457e2b66:160d6e:8cbd7d4f"

## $ items :'data.frame': 1000 obs. of 22 variables:

glimpse(streams[[1]]$items)

## Observations: 1,000

## Variables: 22

## $ id "XGq6cYRY3hH9/vdZr0WOJiPdAe0u6dQ2ddUFEsTqP10=\_1628f55fc26:7feb...

## $ keywords ["R bloggers", "R bloggers", "R bloggers", "R bloggers", "R b...

## $ originId "https://tjmahr.github.io/ridgelines-in-bayesplot-1-5-0-releas...

## $ fingerprint "f96c93f7", "9b2344db", "ca3762c8", "980635d0", "fbd60fac", "6...

## $ content c("

"Ridgelines in bayesplot 1.5.0", "Mathematical art in R", "R a...

## $ published 1.522732e+12, 1.522796e+12, 1.522714e+12, 1.522714e+12, 1.5227...

## $ crawled 1.522823e+12, 1.522809e+12, 1.522794e+12, 1.522793e+12, 1.5227...

## $ canonical [ c("feed/http://feeds.feedburner.com/RBloggers", "feed/h...

## $ author "Higher Order Functions", "David Smith", "R Views", "rOpenSci ...

## $ alternate [ c("At the end of March, Jonah Gabry and I released\nbay...

## $ visual c("feedly-nikon-v3.1", "feedly-nikon-v3.1", "feedly-nik...

## $ unread TRUE, TRUE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, F...

## $ categories [ 9, 37, 52, 15, 78, 35, 31, 9, 28, 2, 21, 8, 25, 11, 21, 29, 12...

## $ engagementRate 0.41, 1.37, 1.58, 0.45, 2.23, 0.97, 0.84, 0.23, 0.72, 0.05, 0....

## $ recrawled NA, NA, NA, NA, NA, NA, NA, NA, 1.522807e+12, NA, NA, NA, NA, ...

## $ tags [NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, ...

## $ decorations c("NA", "NA", "NA", "NA", "NA", "NA", "NA", "NA", "NA",...

## $ enclosure [NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, ...

That entries structure is defined [in the Feedly API docs](https://developer.feedly.com/v3/entries/).

We’ll extract the bits we want to use for the rest of the post and clean it up a bit:

map\_df(streams, ~{

select(.x$items, title, author, published, engagement) %>%

mutate(published = anytime::anydate(published / 1000)) %>% # overly-high-resolution timestamp

tbl\_df()

}) -> xdf

glimpse(xdf)

## Observations: 11,421

## Variables: 4

## $ title "Ridgelines in bayesplot 1.5.0", "Mathematical art in R", "R and T...

## $ author "Higher Order Functions", "David Smith", "R Views", "rOpenSci - op...

## $ published 2018-04-03, 2018-04-03, 2018-04-02, 2018-04-02, 2018-04-03, 2018-...

## $ engagement 9, 37, 52, 15, 78, 35, 31, 9, 28, 2, 21, 8, 25, 11, 21, 29, 12, 11...

Using an arbitrary “10,000” extract didn’t give us full months:

range(xdf$published)

## [1] "2013-05-31" "2018-04-03"

so we’ll filter out the incomplete bits and add in some additional temporal metadata:

xdf %>%

filter(

published > as.Date("2013-05-31"), # complete months

published < as.Date("2018-04-01")

) %>%

mutate(

year = as.integer(lubridate::year(published)),

month = lubridate::month(published, label=TRUE, abbr=TRUE),

wday = lubridate::wday(published, label=TRUE, abbr=TRUE),

ym = as.Date(format(published, "%Y-%m-01"))

) -> xdf

I’m only going to do some light analysis work with engagement data (how “popular” a post was) but the *full post summary and body content* is available in the data dump you’re going to get at the end (this is reminding me of the Sesame Street “[Monster at the End of This Book](https://en.wikipedia.org/wiki/The_Monster_at_the_End_of_This_Book%3A_Starring_Lovable%2C_Furry_Old_Grover)” story). That means enterprising folk can do some [tidy text mining](https://www.tidytextmining.com/) to cluster away some additional insights.

Thankfully, there’s not a ton of missing engagement data:

sum(is.na(xdf$engagement)) / nrow(xdf)

## [1] 0.06506849

broom::tidy((summary(xdf$engagement)))

## minimum q1 median mean q3 maximum na

## 1 0 5 20 69.27219 75 4785 741

Let’s look at post count over time, first:

count(xdf, ym) %>%

arrange(ym) %>%

ggplot(aes(ym, n)) +

ggforce::geom\_bspline0(color="lightslategray") +

scale\_x\_date(expand=c(0,0.5)) +

labs(

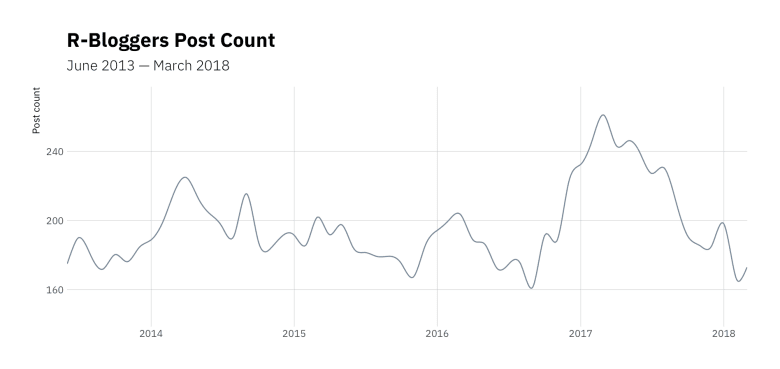
x=NULL, y="Post count",

title="R-Bloggers Post Count",

subtitle="June 2013 — March 2018"

) +

theme\_ipsum\_ps(grid="XY")

[](https://i0.wp.com/rud.is/b/wp-content/uploads/2018/04/rb-post-count.png?ssl=1)

It’ll be interesting to watch that over this year and compare 2017 to 2018 given how “hot” 2017 seems to have been. To turn a Mythbuster phrase: a neat “try this at home” exercise would be to tease out some “whys” for various spikes (which likely means some post content spelunking).

Let’s see if any days are more popular than others:

count(xdf, wday) %>%

ggplot(aes(wday, n)) +

geom\_col(fill="lightslategray", width=0.65) +

scale\_y\_comma() +

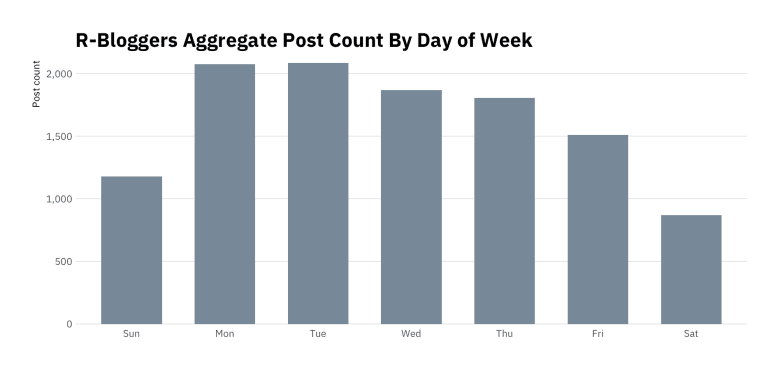
labs(

x=NULL, y="Post count",

title="R-Bloggers Aggregate Post Count By Day of Week"

) +

theme\_ipsum\_ps(grid="Y")

[](https://i1.wp.com/rud.is/b/wp-content/uploads/2018/04/rb-post-count-dow.png?ssl=1)

Weekends are sleepy and there are some “go-getters” at the beginning of the week. More “try this at home” would be to see if any individuals have “patterns” by day of week (or even time of day, since that’s also available in the published time stamp).

The summary() above told us we have a pretty skewed engagement distribution, but it’s always nice to visualise just how bad it is:

ggplot(xdf, aes(engagement)) +

geom\_density(aes(y=calc(count)), fill="lightslategray", alpha=2/3) +

scale\_x\_comma() +

scale\_y\_comma() +

labs(

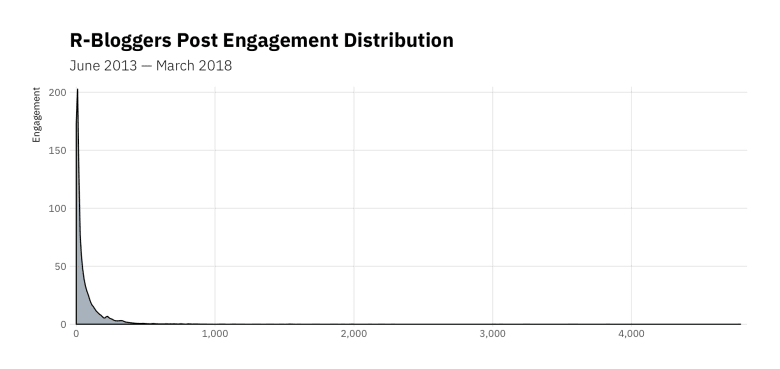
x=NULL, y="Engagement",

title = "R-Bloggers Post Engagement Distribution",

subtitle = "June 2013 — March 2018"

) +

theme\_ipsum\_ps(grid="XY")

[](https://i0.wp.com/rud.is/b/wp-content/uploads/2018/04/rb-engagement.png?ssl=1)

That graph is the story of my daily life dealing with internet data. Couldn’t even get a break when trying to have some fun. #sigh

We’ll close with the “all time top 10” based on total engagement:

count(xdf, author, wt=engagement, sort=TRUE)

## # A tibble: 1,065 x 2

## author n

## 1 David Smith 87381

## 2 Tal Galili 29302

## 3 Joseph Rickert 16846

## 4 DataCamp Blog 14402

## 5 DataCamp 14208

## 6 John Mount 13274

## 7 Francis Smart 8506

## 8 hadleywickham 8129

## 9 hrbrmstr 7855

## 10 Sharp Sight Labs 7620

## # ... with 1,055 more rows

[@revodavid](http://twitter.com/revodavid) is a blogging *machine*, and that top-spot is well-deserved given the plethora of interesting, useful and fun content he shares. And, it looks like *someone* only needs to blog a *bit more* this year to overtake [@hadley](http://twitter.com/hadley) (*I’m comin’ fer ya, Hadley!*).

**FIN**

As promised, you can get the data in a ~30MB RDS file via <https://rud.is/dl/r-bloggers-feedly-streams.rds> and can then use the extraction-to-data-frame example from above to work with the bits you care about.

Hopefully folks will have some fun with this and share their results!