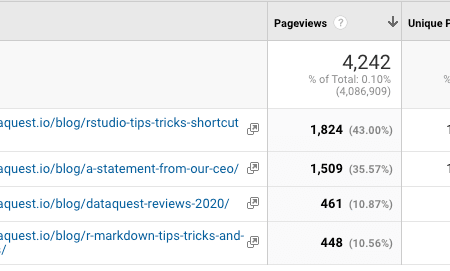
On the surface, it’s a fairly straightforward goal. With Google Analytics, you can quickly get just about any metric you need for your blog posts, for any date range.

But when it comes to comparing blog post performance, things get a bit trickier.

For example, let’s say we want to compare the performance of the blog posts we published



But wait… two blog posts with more than 1,000 pageviews were published earlier in the month, And the two with fewer than 500 pageviews were published at the end of the month. That’s hardly a fair comparison!

My first solution to this problem was to look up each post individually, so that I could make an even comparison of how each post performed in their first day, first week, first month, etc.

However, that required a lot of manual copy-and-paste work, which was extremely tedious if I wanted to compare more than a few posts, date ranges, or metrics at a time.

But then, I learned R, and realized that there was a much better way.

In this post, we’ll walk through how it’s done, so you can do my better blog post analysis for yourself!

**What we’ll need**

To complete this tutorial, you’ll need basic knowledge of R syntax and the tidyverse, and access to a Google Analytics account.

Not yet familiar with the basics of R? We can help with that!

You’ll also need the dyplr, lubridate, and stringr packages installed — which, as a reminder, you can do with the install.packages() command.

Finally, you will need a CSV of the blog posts you want to analyze. Here’s what’s in my dataset:

post\_url: the page path of the blog post  
post\_date: the date the post was published (formatted m/d/yy)  
category: the blog category the post was published in (optional)  
title: the title of the blog post (optional)

Depending on your content management system, there may be a way for you to automate gathering this data — but that’s out of the scope of this tutorial!

For this tutorial, we’ll use a manually-gathered dataset of the past ten Dataquest blog posts.

**Setting up the googleAnalyticsR package**

To access data from the Google Analytics API, we’ll use the excellent [googleAnalyticsR](https://code.markedmondson.me/googleAnalyticsR/) package

As described in the documentation, there are two “modes” to the googleAnalyticsR package. The first mode, which we’ll use here, is a “Try it out” mode, which uses a shared Google Project to authorize your Google Analytics account.

If you want to make this report a recurring tool for your blog or client, be sure to create your own Google Project, which will help keep the traffic on the shared Project to a minimum.

For now, though, we’ll stick with “Try it out” mode.

First, we’ll install the package using this code:

install.packages('googleAnalyticsR', dependencies = TRUE)

This installs the package, as well as the required dependencies.

Next, we’ll load the library, and authorize it with a Google Analytics account using the ga\_auth() function.

library(googleAnalyticsR)

ga\_auth()

When you run this code the first time, it will open a browser window and prompt you to log in to your Google account. Then, it will give you a code to paste into your R console. After that, it will save an authorization token so you only have to do this once!

Once you’ve completed the Google Analytics authorization, we’re ready to set up the rest of the libraries and load in our blog posts. We’ll also use dplyr::mutate() to change the post\_date to a Date class while we’re at it!

library(dplyr)

library(lubridate)

library(stringr)

library(readr)

blog\_posts <- read.csv("articles.csv") %>%

mutate(

post\_date = as.Date(post\_date, "%m/%d/%y") # changes the post\_date column to a Date

)

Here’s what the blog post data frame looks like:



Finally, to get data from your Google Analytics account, you will need the ID of the Google Analytics view you want to access. ga\_account\_list() will return a list of your available accounts.

accounts <- ga\_account\_list()

# select the view ID by view and property name, and store it for ease of use

view\_id <- accounts$viewId[which(accounts$viewName == "All Web Site Data" & accounts$webPropertyName == "Dataquest")]

# be sure to change this out with your own view and/or property name!

Now, we’re ready to do our first Google Analytics API requests!

**Accessing blog post data with googleAnalyticsR**

In this tutorial, our goal is to gather data for the first week each post was active, and compile it in a dataframe for analysis. To do this, we’ll create a function that runs a for loop and requests this data for each post in our blog\_posts dataframe.

So, let’s take a look at how to send a request to the Google Analytics API using googleAnalyticsR.

google\_analytics(view\_id,

date\_range = c(as.Date("2020-06-01"), as.Date("2020-06-30")),

metrics = c("pageviews"),

dimensions = c("pagePath")

)

This request has a few components. First, enter the view\_id, which we already stored from our ga\_accounts() dataframe.

Next, specify the date range, which needs to be passed in as a list of dates.

Then, we input the metrics (like pageviews, landing page sessions, or time on page) and dimensions (like page path, channel, or device). We can use any dimension or metric that’s available in the Google Analytics UI —

So, the request above will return a dataframe of all pageviews in June, by page path

But, in our case, we only want to retrieve pageviews for a specific page – so we need to filter on the pagePath dimension using a dimension filter, which looks like this:

page\_filter <- dim\_filter(dimension = "pagePath",

operator = "REGEXP",

expressions = "^[www.dataquest.io/blog/r-markdown-guide-cheatsheet/$](http://www.dataquest.io/blog/r-markdown-guide-cheatsheet/$)")

To use this filter in our request, googleAnalyticsR wants us to create a filter clause – which is how you would combine filters if you wanted to use multiple dimension filters. But in our case, we just need the one:

page\_filter\_clause <- filter\_clause\_ga4(list(page\_filter))

Now, let’s try sending a response with this filter:

google\_analytics(view\_id,

date\_range = c(as.Date("2020-07-01"), Sys.Date()),

metrics = c("pageviews"),

dimensions = c("pagePath"),

dim\_filters = page\_filter\_clause)

The result is a dataframe with the pageviews for the R Markdown post!

**Creating the for loop**

Now that we can gather data and filter it by dimension, we are ready to build out our function to run our for loop! The steps to the function are:

* Set up a data frame to hold the results
* Begin the loop based on the number of rows in the data frame
* Access the post URL and post date for each post
* Create a page filter based on the post URL
* Send a request to Google Analytics using the post\_date as the start date, and date the week later as the end date
* Add the post URL and pageview data to the final data frame

I also have added a print() command to let us know how far along the loop is (because it can take awhile) and a Sys.Sleep() command to keep us from hitting the Google Analytics API rate limit.

Here’s what that looks like all put together!

get\_pageviews <- function(posts) {

# set up dataframe to be returned, using the same variable names as our original dataframe

final <- tibble(pageviews = numeric(),

post\_url = character())

# begin the loop for each row in the posts dataframe

for (i in seq(1:nrow(posts))) {

# select the post URL and post date for this loop — also using the same variable names as our original dataframe

post\_url <- posts$post\_url[i]

post\_date <- posts$post\_date[i]

# set up the page filter and page filter clause with the current post URL

page\_filter <- dim\_filter(dimension = "pagePath",

operator = "REGEXP",

expressions = post\_url)

page\_filter\_clause <- filter\_clause\_ga4(list(page\_filter))

# send the request, and set the date range to the week following the date the post was shared

page\_data <- google\_analytics(view\_id,

date\_range = c(post\_date, post\_date %m+% weeks(1)),

metrics = c("pageviews"),

dim\_filters = page\_filter\_clause)

# add the post url to the returned dataframe

page\_data$post\_url <- post\_url

# add the returned data to the data frame we created outside the loop

final <- rbind(final, page\_data)

# print loop status

print(paste("Completed row", nrow(final), "of", nrow(posts)))

# wait two seconds

Sys.sleep(2)

}

return(final)

}

We could potentially speed this up with a “functional” in R, such as purrr::map(). The map() function takes a function as an input and returns a vector as output.

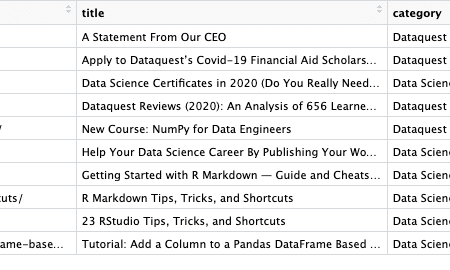
For this tutorial, though, we’ll use a for loop because it’s a bit less abstract.

Now, we’ll run the loop on our blog\_posts dataframe, and merge the results to our blog\_posts data.

recent\_posts\_first\_week <- get\_pageviews(blog\_posts)

recent\_posts\_first\_week <- merge(blog\_posts, recent\_posts\_first\_week)

recent\_posts\_first\_week



And that’s it! Now, we can get on to the good stuff — analyzing and visualizing the data.

**Blog post data, visualized!**

For demonstration, here’s a ggplot bar chart that shows how many pageviews each of our most recent 10 blog posts got in the first week after they were published:

library(ggplot2)

library(scales)

recent\_posts\_first\_week %>%

arrange(

post\_date

) %>%

mutate(

pretty\_title = str\_c(str\_extract(title, "^(\\S+\\s+\\n?){1,5}"), "..."),

pretty\_title = factor(pretty\_title, levels = pretty\_title[order(post\_date)])

) %>%

ggplot(aes(pretty\_title, pageviews)) +

geom\_bar(stat = "identity", fill = "#39cf90") +

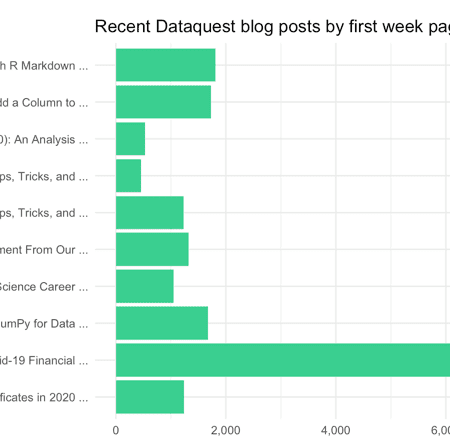
coord\_flip() +

theme\_minimal() +

theme(axis.title = element\_blank()) +

labs(title = "Recent Dataquest blog posts by first week pageviews") +

scale\_y\_continuous(labels = comma)



Now we can see how useful it is to be able to compare blog posts on “even footing”!