Seeing the Forest Despite the Trees: Analyzing the usage of databases and journal packages

NCLA RTSS Webinar, June 5, 2020

Danica Lewis, Collections & Research Librarian for Life Sciences

Heidi Tebbe, Collections & Research Librarian for Engineering and Data Science

Setup

This is an R Markdown Notebook. When you execute code within the notebook, the results will appear beneath the code. To view the HTML file created by this notebook, click the *Preview* button in RStudio or press *Cmd+Shift+K*. Or, download the PDF file saved in the GitHub reposotory and view a rendered version of this file there.

In this notebook we'll be sharing a much more in-depth view of the process of cleaning COUNTER 5 usage data for visualization in R.

First, we'll need to set up our working directory, so R knows where to look for the data we'll be reading in. Make sure that you download both this .Rmd file and the associated usage data, and that they are both stored in the same folder. In R Notebooks, the working directory is automatically set to be wherever the notebook is, which is convenient for us!

It's also good to keep a list of the packages you use in this script up near the top, that way when you try to run it again you can double check that they are all installed and correctly loaded.

```r
require(tidyverse)
```

```
## Loading required package: tidyverse

## -- Attaching packages ---------------------------------------------------------------------------

## v ggplot2 3.3.0     v purrr   0.3.4
## v tibble  3.0.1     v dplyr   0.8.5
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0

## Warning: package 'tibble' was built under R version 3.6.2

## Warning: package 'purrr' was built under R version 3.6.2

## -- Conflicts ------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
require(dplyr)
require(readxl)
```

```
## Loading required package: readxl
```

```r
require(lubridate)
```

```
## Loading required package: lubridate
```

```
## Warning: package 'lubridate' was built under R version 3.6.2
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:dplyr':
##
##     intersect, setdiff, union
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

I like to use the require() function to do this, because it will give you a warning instead of an error if a package doesn't exist and needs to be installed.

If you do need to install a package, you can use the install.packages() function. In this notebook I've turned those commands into comments, just so RStudio doesn't try to install duplicate copies and get confused. If you need to run an installation, just remove the #s.

```r
#install.packages("tidyverse")
# install.packages("dplyr")
# install.packages("readxl")
# install.packages("lubridate")
```

Once we've checked our working directory and have the right packages ready to go, we can start loading, cleaning, and visualizing our data!

Journal COUNTER Data

Read in the COUNTER data and save it to a dataframe called "counter"

The example data from our presentation was from a very big database, and gets really unweildy really fast, and also we can't really share the raw data. So we invented a practice dataset for a much *much* smaller journal package, which hopefully will be easier to work with if you want to try running the code live. There's also a much shorter date range for the same reasons. In this case, maybe we're looking at a brief trial of a small package to decide if we want to subscribe, rather than evaluating an existing subscription.

Since the data we're using, and a lot of data that vendors share, is in an excel sheet, the read_excel() function from the readxl package is going to be the best function to read in our data. It has a lot of extra features that help you deal with special excel challenges, like workbooks with multiple sheets. We only have one sheet and by default read_excel() will load the first sheet, but we left that argument in so you can see where you would specify which sheet you want to load.

Additionally, COUNTER has a lot of metadata at the top of the sheet, which is great for keeping track of when the data was collected and for what, but it gets in the way of doing an analysis. We're going to skip those first 12 rows and get right to the data!

```
counter <- read_excel("NCLA20_counterData.xlsx", sheet = 1, skip = 12)
```

Format the COUNTER dataframe so that it is "tidy" data

The Tidyverse is a series of packages designed for data science by Hadley Wickham. They're really useful, but they assume that your data is "tidy." One of the biggest challenges of working with collections data is that it is frequently not tidy, and before you can begin any analysis you have to reformat it.

Typically, data from vendors or COUNTER reports are "wide" data. The date variable is usually stored as a column header, with the number of uses as the values. So lets take a look at our data.

```
show(counter)
```

```
## # A tibble: 10 x 10
##    Title Access_Type   YOP Section_Type Reporting_Perio~ `43586` `43617` `43647`
##    <chr> <chr>       <dbl> <chr>                   <dbl>   <dbl>   <dbl>   <dbl>
##  1 Jour~ Controlled   2019 Article                   542     113      93      82
##  2 Jour~ OA_Gold      2019 Article                    21       9       0       2
##  3 Jour~ Controlled   2019 Article                   174      30      49      36
##  4 Jour~ OA_Gold      2019 Article                    39      15       8       9
##  5 Jour~ Controlled   2019 Article                     4       0       4       0
##  6 Jour~ Controlled   2019 Article                    81      13       7      14
##  7 Jour~ OA_Gold      2019 Article                     0       0       0       0
##  8 Jour~ Controlled   2020 Article                     1       0       0       0
##  9 Jour~ OA_Gold      2020 Article                     2       0       0       0
## 10 Jour~ Controlled   2019 Article                     4       1       0       0
## # ... with 2 more variables: `43678` <dbl>, `43709` <dbl>
```

There's a lot going on here and it's not all super good.

The column headers after Reporting_Period_Total are dates, but because excel realized they were dates and gave them strange custom formatting, they're importing as nonsense. We're going to have to address this in a moment, but more pressing is the fact that this data isn't tidy.

Our variables are Title, Access_Type, YOP, Section_Type, Reporting_Period_Total, Date, and Use. But Date and Use don't show up as columns, they're stored as headers and values. So our first step will have to be to gather our wide data into a long data format.

We'll do this with the gather() function, saving a new version of the counter dataframe that is long instead of wide. If you wanted to keep your original wide counter data, possibly for later or as a backup while you experiment, you could just create a new dataframe called something like counterLong instead of saving over your dataframe. Inside the gather function, we say what dataframe we want to use, what the key column should be called, what the values should be called, and what columns we want to be affected - which here is all the columns except 1 through 5.

```
counter <- gather(counter, date, use, -c(1:5))
show(counter)
```

```
## # A tibble: 50 x 7
##    Title     Access_Type   YOP Section_Type Reporting_Period_Total date    use
##    <chr>     <chr>       <dbl> <chr>                         <dbl> <chr> <dbl>
##  1 Journal A Controlled   2019 Article                         542 43586   113
##  2 Journal A OA_Gold      2019 Article                          21 43586     9
##  3 Journal B Controlled   2019 Article                         174 43586    30
##  4 Journal B OA_Gold      2019 Article                          39 43586    15
##  5 Journal C Controlled   2019 Article                           4 43586     0
##  6 Journal D Controlled   2019 Article                          81 43586    13
##  7 Journal D OA_Gold      2019 Article                           0 43586     0
##  8 Journal D Controlled   2020 Article                           1 43586     0
##  9 Journal D OA_Gold      2020 Article                           2 43586     0
## 10 Journal E Controlled   2019 Article                           4 43586     1
## # ... with 40 more rows
```

Now our data is long, with 50 observations instead of 10. But now each observation is for a specific title on a specific date, measuring our variables Title, Access_Type, YOP, Section_Type, Reporting_Period_Total, Date, and Use. This didn't really make intuitive sense to me at the start, but it makes a lot of sense to the tidyverse.

Finally we can adress the problems excel caused with our dates. The easiest solution we've found for this is to convert the date column to numeric values (right now R's storing them as characters because it has no clue what on earth excel is going on about) and then converting those numeric values to date values. We will also have to clarify the start date, which for excel is 1899-12-30.

```r
counter$date <- as.numeric(counter$date)
counter$date <- as.Date(counter$date, origin = "1899-12-30")
glimpse(counter)
```

```
## Rows: 50
## Columns: 7
## $ Title                  <chr> "Journal A", "Journal A", "Journal B", "Jour...
## $ Access_Type            <chr> "Controlled", "OA_Gold", "Controlled", "OA_G...
## $ YOP                    <dbl> 2019, 2019, 2019, 2019, 2019, 2019, 2019, 20...
## $ Section_Type           <chr> "Article", "Article", "Article", "Article", ...
## $ Reporting_Period_Total <dbl> 542, 21, 174, 39, 4, 81, 0, 1, 2, 4, 542, 21...
## $ date                   <date> 2019-05-01, 2019-05-01, 2019-05-01, 2019-05...
## $ use                    <dbl> 113, 9, 30, 15, 0, 13, 0, 0, 0, 1, 93, 0, 49...
```

The function glimpse() is from the tidyverse as well, and will give you a nice overview of what's going on behind the scenes with your dataframe, including what data type each column is stored as. As we can see above, date is finally formatted and stored as a date that makes more sense to us *and* to R.

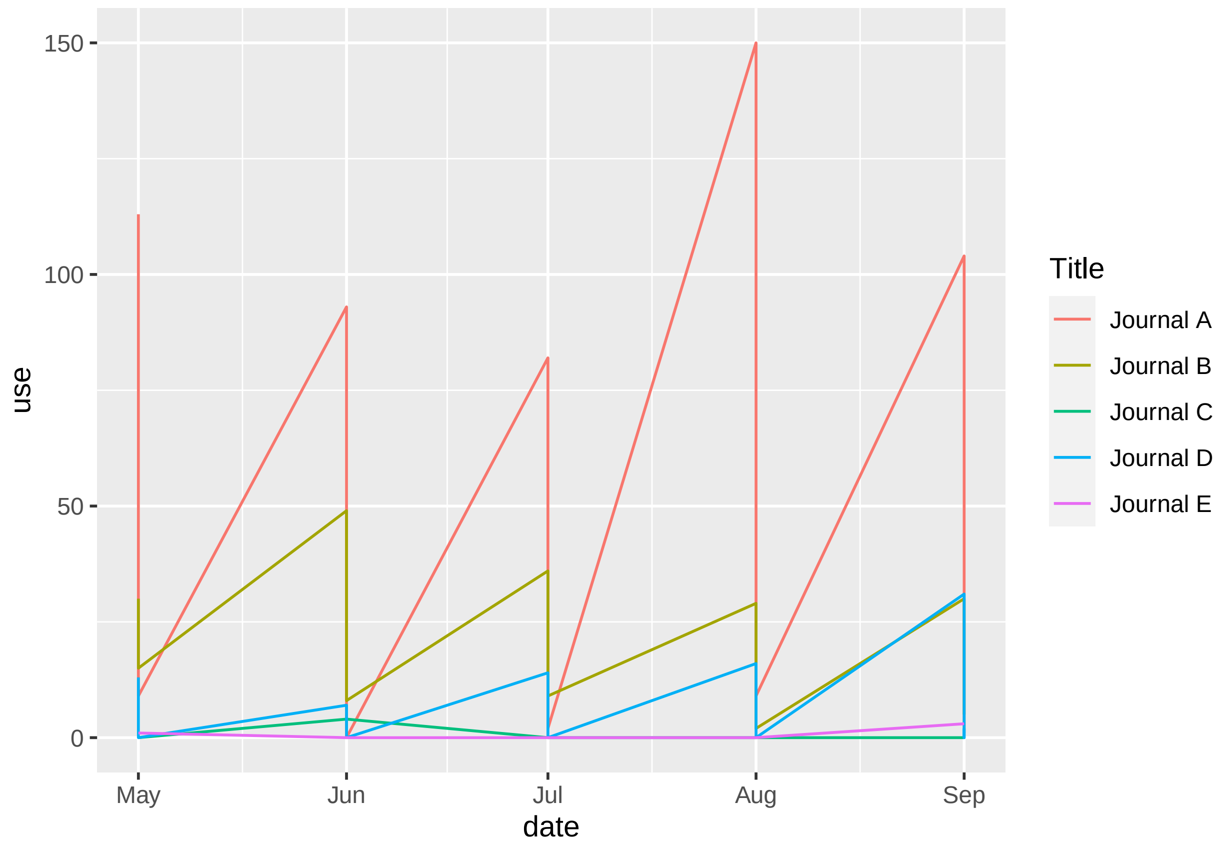
Our Title variable is still stored as a character, but we'll want to group observations of the same title together, and character variables don't really want to do that. Factors however do, so one more quick conversion and then we'll be all set to go!

```r
counter$Title <- as.factor(counter$Title)
```

COUNTER data exploratory visualization

In our example scenario where we're looking at data coming out of a short trial of a new package, probably one of the first things we should look at is the number of uses each title is getting.

```
ggplot(counter, aes(x = date, y = use, color = Title))+
  geom_line()
```



This does not look like the line graph we were expecting to get, and it's because there's still some underlying mess in our data that we didn't notice and therefore haven't cleaned up. Our variable "Access_Type" means that for each month each title has *two* data points, which creates the jagged lines. So now we actually need to *spread* our data, and turn Controlled use and OA_Gold use into their own columns. At this point Reporting_Period_Total is just going to get in the way and mess everything up, so I'm getting rid of if first by subsetting the data. I can always calculate it if I want it down the line.

```
counter <- counter[-5]
counter <- spread(counter, Access_Type, use)
show(counter)
```

```
## # A tibble: 30 x 6
##    Title       YOP Section_Type date       Controlled OA_Gold
##    <fct>     <dbl> <chr>        <date>          <dbl>   <dbl>
##  1 Journal A  2019 Article      2019-05-01        113       9
##  2 Journal A  2019 Article      2019-06-01         93       0
##  3 Journal A  2019 Article      2019-07-01         82       2
##  4 Journal A  2019 Article      2019-08-01        150       9
##  5 Journal A  2019 Article      2019-09-01        104       1
##  6 Journal B  2019 Article      2019-05-01         30      15
##  7 Journal B  2019 Article      2019-06-01         49       8
##  8 Journal B  2019 Article      2019-07-01         36       9
##  9 Journal B  2019 Article      2019-08-01         29       2
## 10 Journal B  2019 Article      2019-09-01         30       5
## # ... with 20 more rows
```

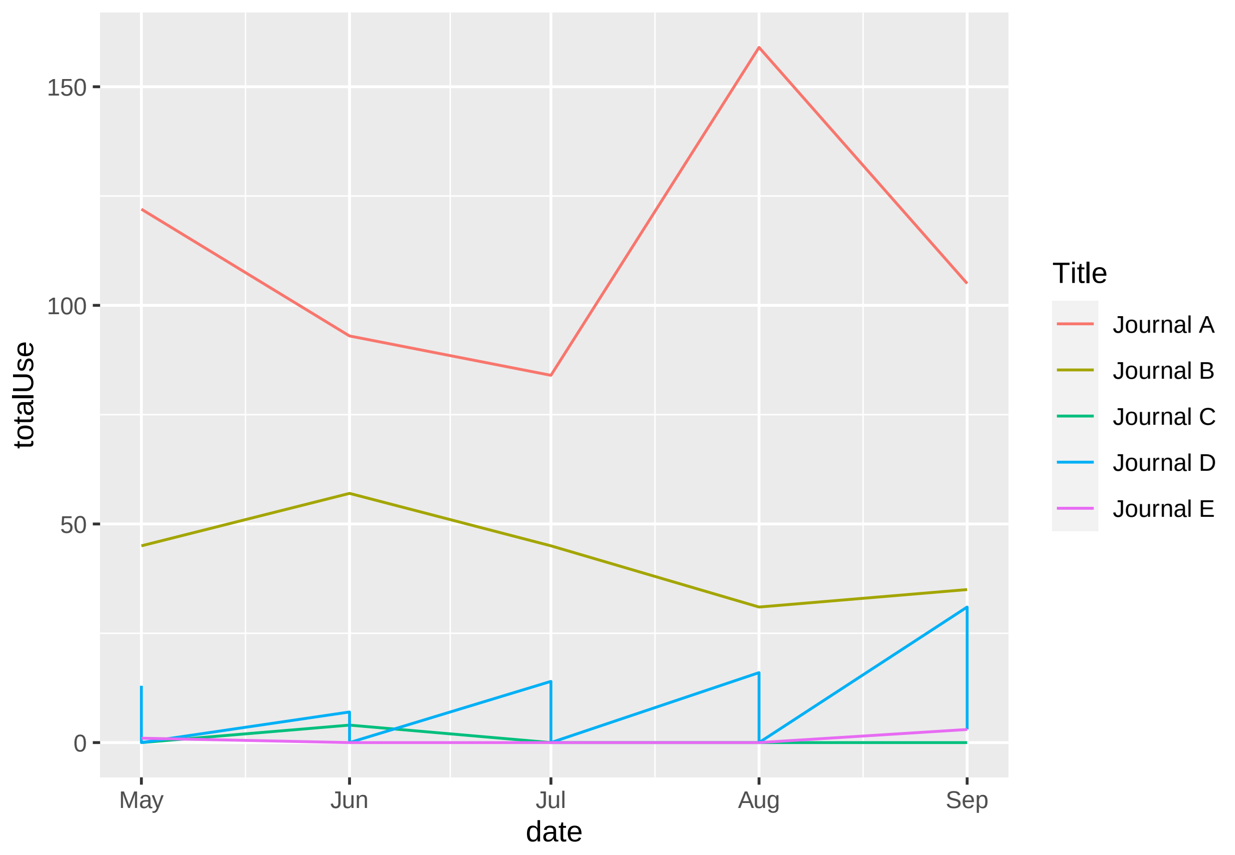
What was that about wanting a total use value down the line? Now I'll create a total use column by adding the Controlled and OA_Gold columns together, because I don't feel too fussy right now about where the use comess from.

```
counter$totalUse <- rowSums(cbind(counter$Controlled, counter$OA_Gold), na.rm = TRUE)
show(counter)
```

```
## # A tibble: 30 x 7
##    Title       YOP Section_Type date       Controlled OA_Gold totalUse
##    <fct>     <dbl> <chr>        <date>          <dbl>   <dbl>    <dbl>
##  1 Journal A  2019 Article      2019-05-01        113       9      122
##  2 Journal A  2019 Article      2019-06-01         93       0       93
##  3 Journal A  2019 Article      2019-07-01         82       2       84
##  4 Journal A  2019 Article      2019-08-01        150       9      159
##  5 Journal A  2019 Article      2019-09-01        104       1      105
##  6 Journal B  2019 Article      2019-05-01         30      15       45
##  7 Journal B  2019 Article      2019-06-01         49       8       57
##  8 Journal B  2019 Article      2019-07-01         36       9       45
##  9 Journal B  2019 Article      2019-08-01         29       2       31
## 10 Journal B  2019 Article      2019-09-01         30       5       35
## # ... with 20 more rows
```

And the graph:

```
ggplot(counter, aes(x = date, y = totalUse, color = Title))+
  geom_line()
```

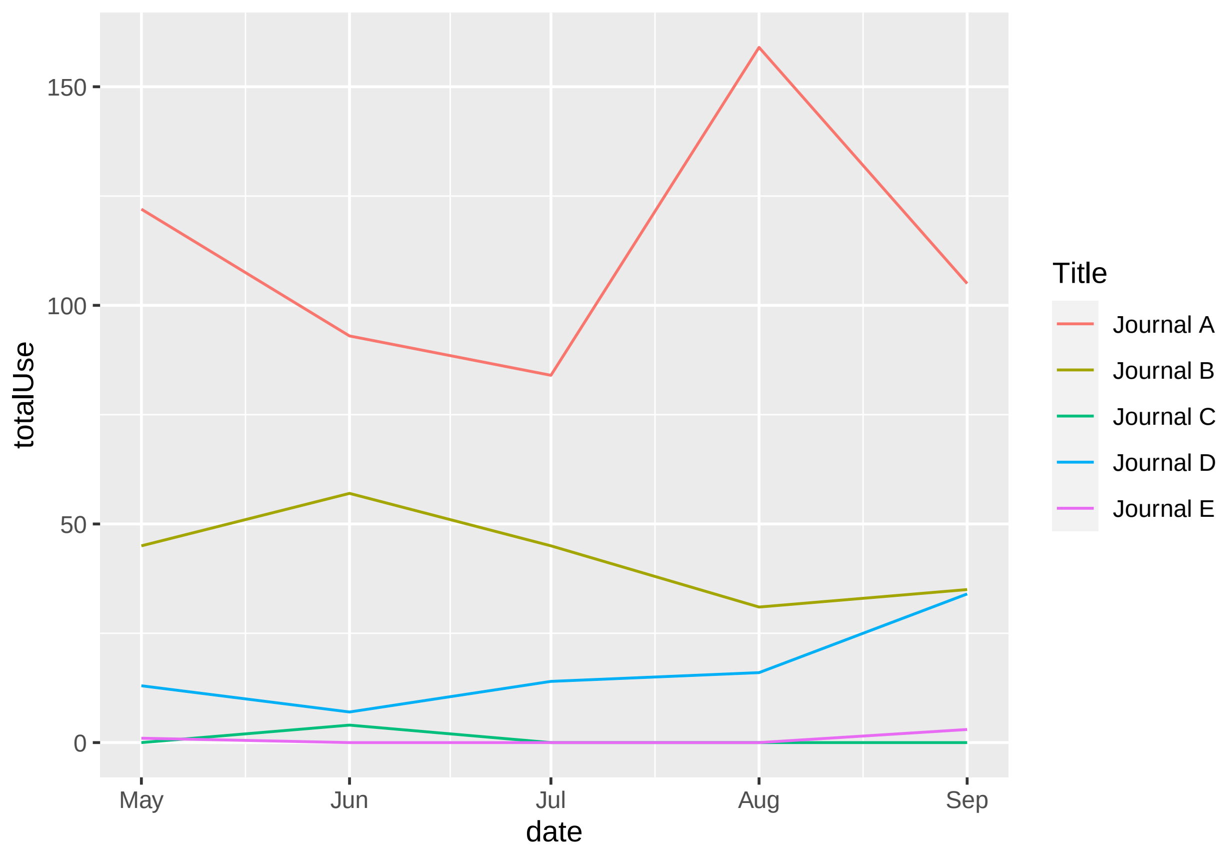


That fixed some of it, but Journal D is still misbehaving. Upon review our "YOP" variable is also duplicating things by separating out uses to content published in 2019 and content published in 2020. Even though our hypothetical trial ran May-September 2019. It would seem that, for Journal D, 2020 came early. For now I don't care about the year of publication, so I'm going to combine use for 2019 and 2020 content and focus just on total use. I'll get two more columns I don't care so much about, and add the rows that were for 2020 use to the 2019 use rows using aggregate().

```
counter <- counter[-c(2:3)]
counter <- aggregate(counter[c(3:5)], by=counter[c(1:2)], sum)
```

Now our graph should come out correctly!

```
ggplot(counter, aes(date, totalUse, color = Title))+
  geom_line()
```



But what if we *do* care about the access type? I might feel pretty dumb for deleting and merging that column. But! I never touched the original file. So I'll just copy paste my code to the point where I started doing things I now regret and start from there. If I were feeling responsible I would maybe name the dataframe something different this time. If I were really REALLY responsible I would name it something descriptive, brief, and easy to type.

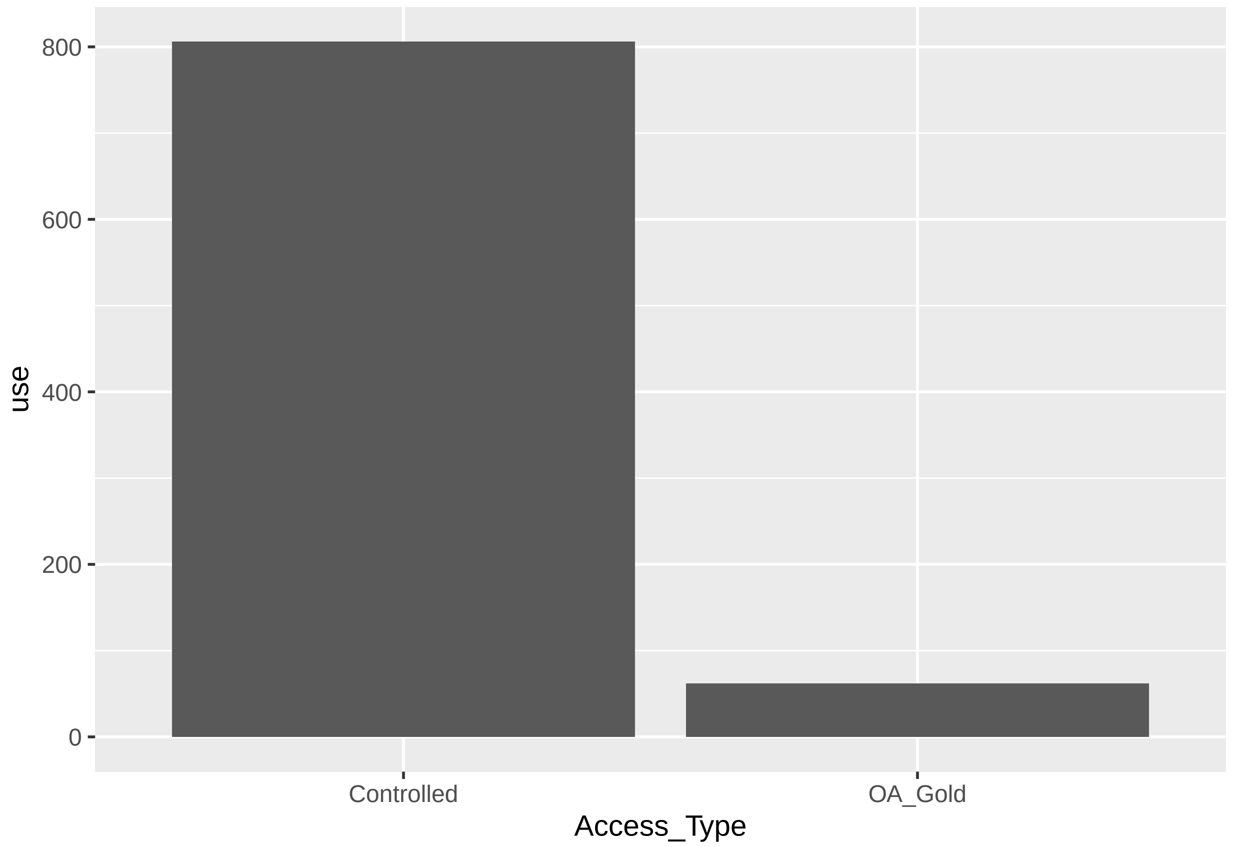
```
counterA <- read_excel("NCLA20_counterData.xlsx", skip = 12)

#reformat the counter dataframe
counterA <- gather(counterA, date, use, -c(1:5))
counterA$date <- as.numeric(counterA$date)
counterA$date <- as.Date(counterA$date, origin = "1899-12-30")
counterA$Title <- as.factor(counterA$Title)
show(counterA)
```

```
## # A tibble: 50 x 7
##    Title    Access_Type   YOP Section_Type Reporting_Period_To~ date         use
##    <fct>    <chr>       <dbl> <chr>                       <dbl> <date>     <dbl>
##  1 Journal~ Controlled   2019 Article                       542 2019-05-01   113
##  2 Journal~ OA_Gold      2019 Article                        21 2019-05-01     9
##  3 Journal~ Controlled   2019 Article                       174 2019-05-01    30
##  4 Journal~ OA_Gold      2019 Article                        39 2019-05-01    15
##  5 Journal~ Controlled   2019 Article                         4 2019-05-01     0
##  6 Journal~ Controlled   2019 Article                        81 2019-05-01    13
##  7 Journal~ OA_Gold      2019 Article                         0 2019-05-01     0
##  8 Journal~ Controlled   2020 Article                         1 2019-05-01     0
##  9 Journal~ OA_Gold      2020 Article                            2 2019-05-01     0
## 10 Journal~ Controlled   2019 Article                            4 2019-05-01     1
## # ... with 40 more rows
```

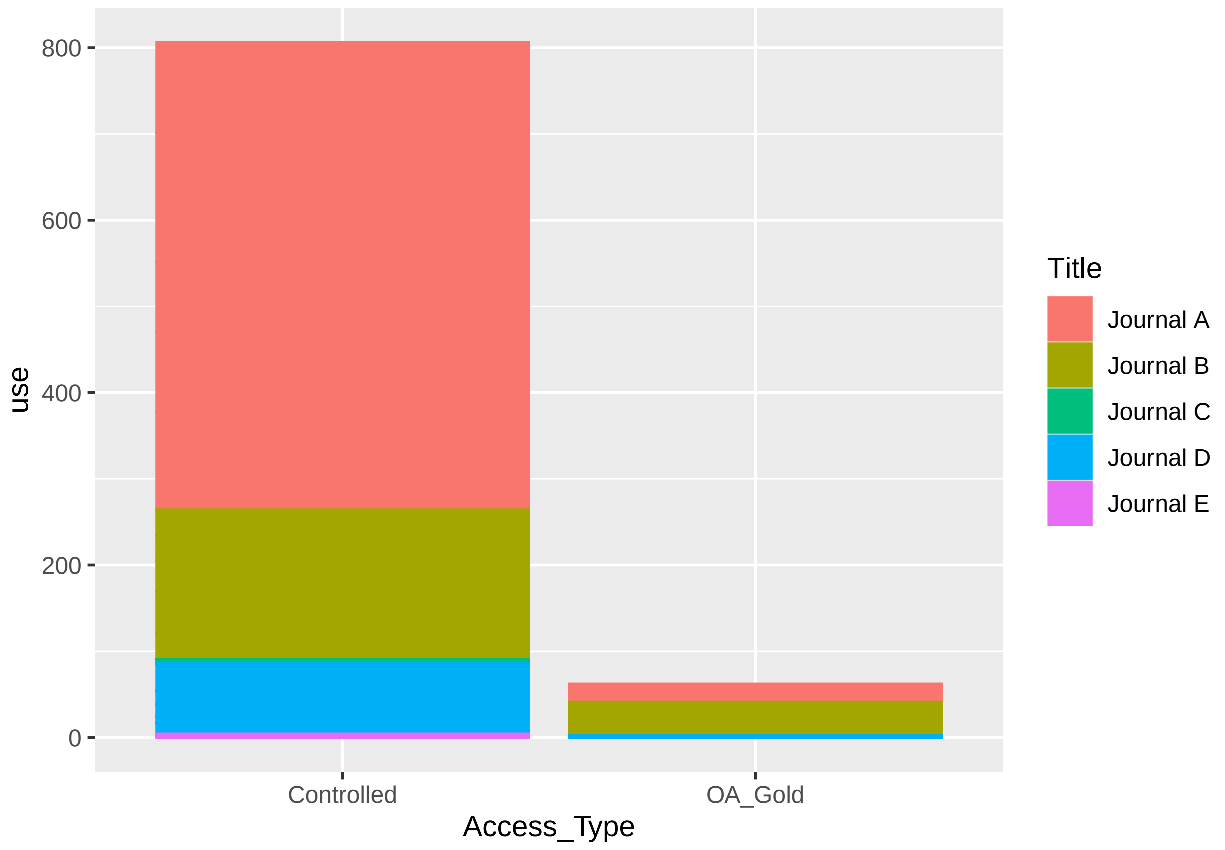
Very nostalgic.

```
ggplot(counterA, aes(Access_Type, use))+
  geom_col()
```



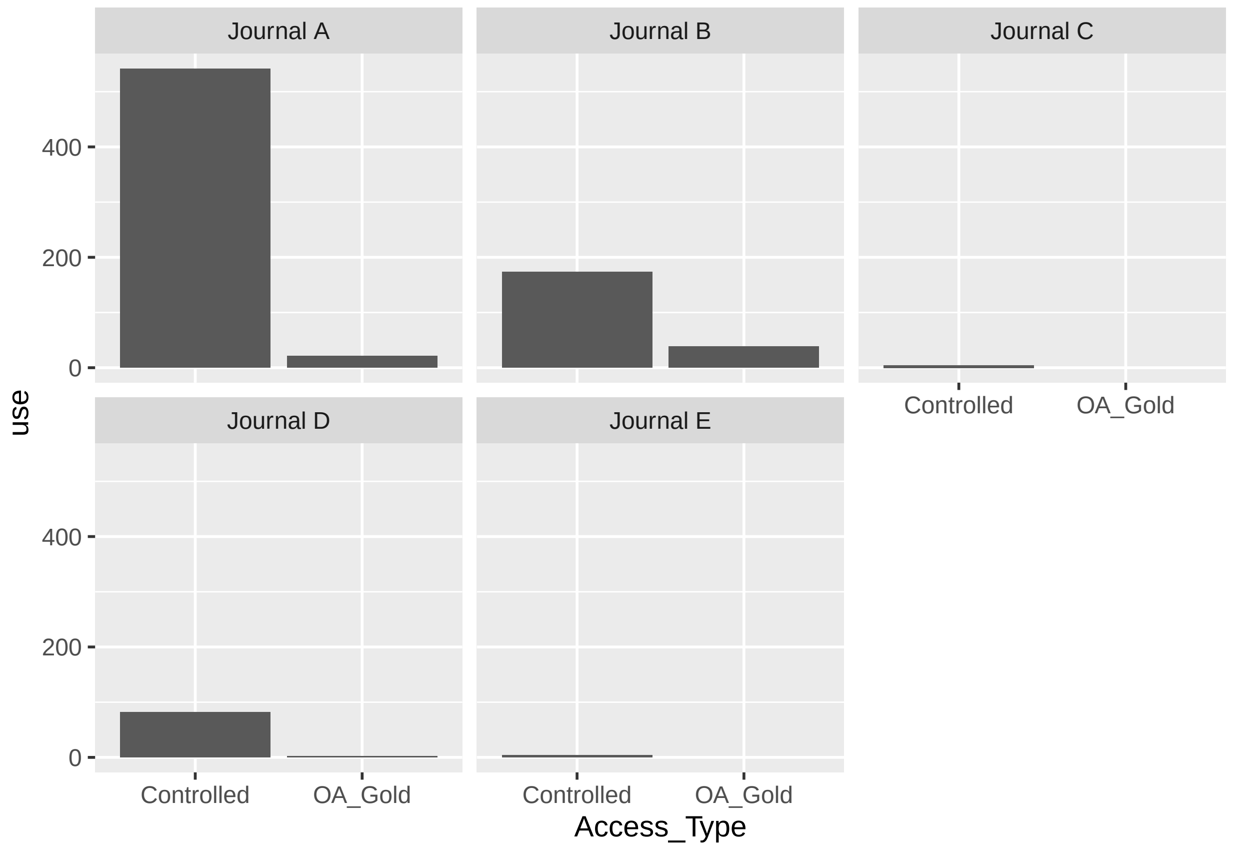
And if I want to see the breakdown by title? I've got a couple options

```
ggplot(counterA, aes(Access_Type, use, color=Title, fill=Title))+
  geom_col()
```



Or I could make a bunch of little graphs for each title

```
ggplot(counterA, aes(Access_Type, use))+
  geom_col()+
  facet_wrap(vars(Title))
```



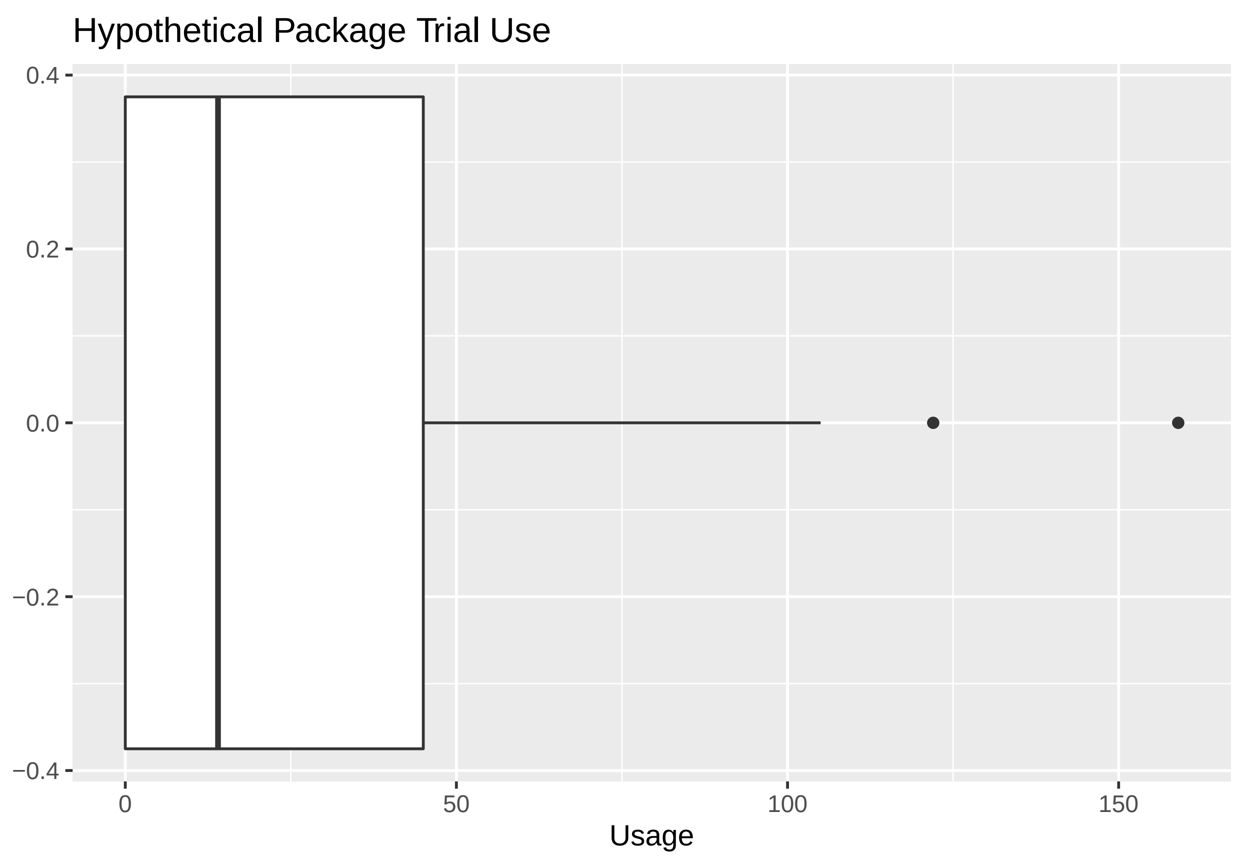
Visualizing the "Averages"

There's few enough titles in this example that it's really not all too difficult to get a visual "sense" of how the package is performing. But let's try a couple of summary visualizations on this data.

Boxplot

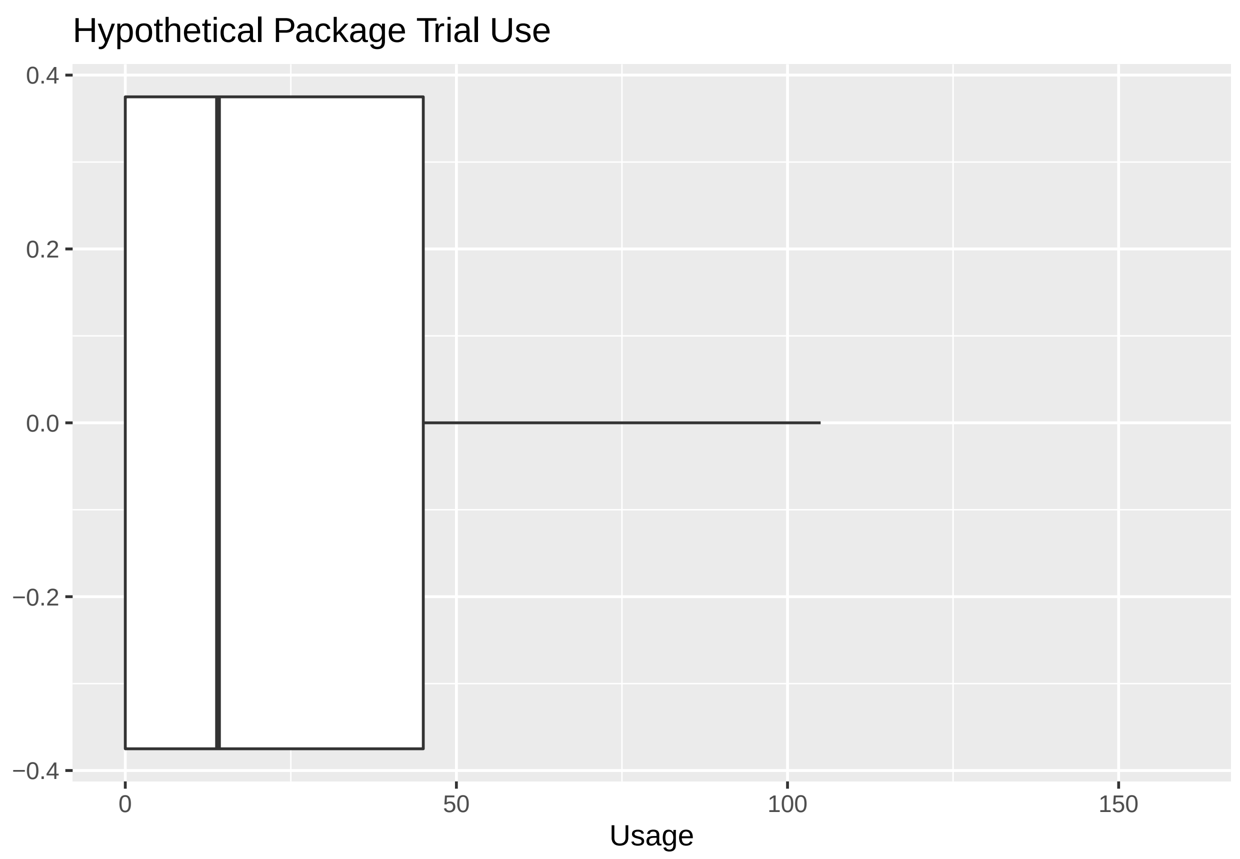
First with outliers

```
ggplot(counter, aes(y=totalUse))+
  geom_boxplot()+
  labs(title = "Hypothetical Package Trial Use", x= NULL, y = "Usage") +
  coord_flip()+
  theme(legend.position = "none")
```



and then without:

```
ggplot(counter, aes(y=totalUse))+
  geom_boxplot(outlier.shape = NA)+
  labs(title = "Hypothetical Package Trial Use", x= NULL, y = "Usage") +
  coord_flip()+
  theme(legend.position = "none")
```



So we can confirm that we are evaluating a skewed distribution. Visualizations give me personally the best impression of what's going on in a given dataset, but for those who prefer to look at numbers we can also run summary statistics on our totalUse column.

```
summary(counter$totalUse)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00    0.00   14.00   34.72   45.00  159.00
```

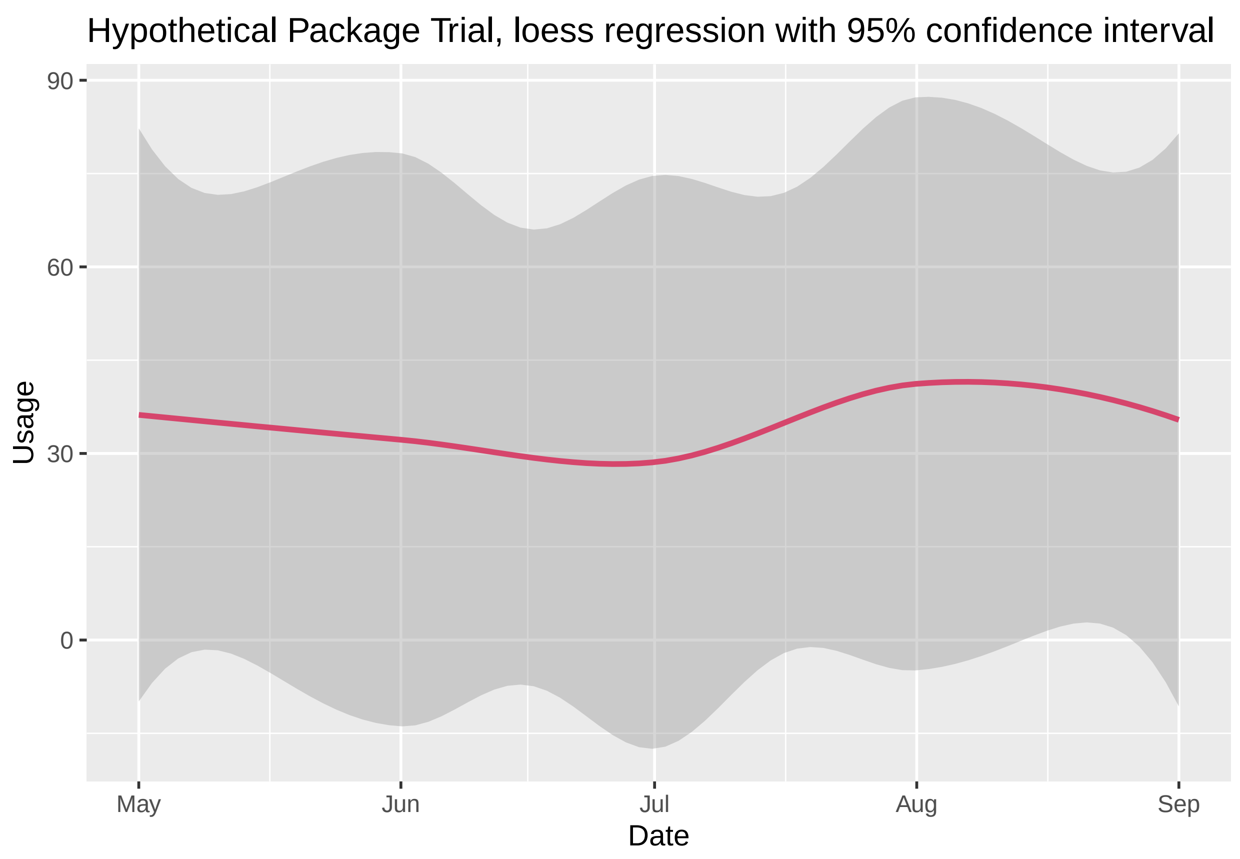
The "1st Qu." value represents the first quartile, or the first 25% of a distribution, and the 3rd Qu., or third quartile, is the 75% point. It's also really easy to tell from these numbers that evaluating the package based on a median value will look very different than evaluating based on a mean value.

Smooth Means

So what does the mean performance look like? R is able to generate a Smooth Means regression visualization, and because our distribution is highly skewed we'll do one with a loess regression, which is designed to handle highly variable data.

```
ggplot(counter, aes(date, totalUse)) +
      geom_smooth(method = "loess", color = "#D6456CFF") +
      labs(title = "Hypothetical Package Trial, loess regression with 95% confidence interval", x = "Da
      theme(legend.position="none")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



There’s a lot of variation inside that 95% confidence interval, but we can see the mean trend more clearly now, and maybe unsurprisingly can see a downward trend through the summer, and a bump once school starts in August.

Thanks!

Hopefully this was a useful introduction to working with and visualizing COUNTER 5 data!