# A new data processing workflow for R: dplyr, magrittr, tidyr, ggplot2

Posted on January 13, 2015 by zev@zevross.com · 8 Comments

Over the last year I have changed my data processing and manipulation workflow in R dramatically. Thanks to some great new packages like `dplyr`, `tidyr` and `magrittr` (as well as the less-new `ggplot2`) I've been able to streamline code and speed up processing. Up until 2014, I had used essentially the same R workflow (`aggregate`, `merge`, `apply/tapply`, `reshape` etc) for more than 10 years. I have added a few improvements over the years in the form of functions in packages `doBy`, `reshape2` and `plyr` and I also flirted with the package `data.table` (which I found to be much faster for big datasets but the syntax made it difficult to work with) – but the basic flow has remained remarkably similar. Until now…

Given how much I've enjoyed the speed and clarity of the new workflow, I thought I would share a quick demonstration.

In this example, I am going to grab data from a sample SQL database provided by Google via Google BigQuery and then give examples of manipulation using `dplyr`, `magrittr` and `tidyr` (and `ggplot2` for visualization).

## 0) Install the packages!

In the examples, I am using the most up-to-date development versions of the packages – all of which come from GitHub. The links here provide instructions: dplyr (v0.4), magrittr (v1.5) and `tidyr` (v0.2). We also installed the latest version of bigrquery (v0.1). You should also have ggplot2 installed (we're using v1.0). For the most part, using the versions on the R website (CRAN) will work fine, but there are a few new features that will not be available to you.

## 1) Grab sample data from Google's BigQuery

The package `dplyr` has several great functions to connect directly to external databases. We discuss this functionality in our previous post on dplyr. One of the functions is designed to extract data from Google's BigQuery. This function (`src_bigquery`) uses code from the package `bigrquery` and I found that running the raw functions in `bigrquery` – `query_exec` in particular – worked more smoothly so that's what I use below. Google has several nice sample tables to choose from. We will use the word index for the works of Shakespeare.

If you would rather skip the step of extracting data from Google, I've placed the full dataset on GitHub (warning that it's relatively big, ~6mb). If you want to follow along exactly as you see below you will need an account with Google and you can find details on how to do this (it takes 5 minutes) here.

In this example, I write the SQL query as usual and execute using the function `query_exec`. The

Web2PDF
converted by Web2PDFConvert.com

'dark-mark-818' is the name of my project on Google, your project name will be different.

```
library(dplyr)
library(bigrquery)
sql<-"select * from [publicdata:samples.shakespeare]"
shakespeare <-query_exec(sql, project = "dark-mark-818",max_pages=Inf)
```

When you run the `query_exec()` function you will be asked if you want to cache your Google authorization credentials. Choose "Yes" by typing "1". Your browser will open and you will need to click the "Accept" button after which the query will be processed. In my case, this took 34.7 seconds and the data looks like below.

```
head(shakespeare)
##        word word_count          corpus corpus_date
## 1      hive          1 loverscomplaint        1609
## 2 plaintful          1 loverscomplaint        1609
## 3       Are          1 loverscomplaint        1609
## 4      Than          1 loverscomplaint        1609
## 5  attended          1 loverscomplaint        1609
## 6      That          7 loverscomplaint        1609
str(shakespeare)
## 'data.frame':    164656 obs. of  4 variables:
##  $ word       : chr  "hive" "plaintful" "Are" "Than" ...
##  $ word_count : int  1 1 1 1 1 7 1 1 1 1 ...
##  $ corpus     : chr  "loverscomplaint" "loverscomplaint" "loverscomplaint" "lovers
##  $ corpus_date: int  1609 1609 1609 1609 1609 1609 1609 1609 1609 1609 ...
```

You can see that this is a relatively simple table. Now we have the data we're ready to use the magic of `dplyr`, `tidyr` and `magrittr`.

## 2) `dplyr`: tools for working with data frames

There are multiple instances of words due to differences in case (lower, upper and proper case) and this gives us a very un-sexy introduction to the use of dplyr. Let's take a quick look at the repetition using one word which we know occurs frequently:

```
head(filter(shakespeare, tolower(word)=="henry"))
##    word word_count        corpus corpus_date
## 1 HENRY        122 kinghenryviii        1612
## 2 Henry          7 kinghenryviii        1612
## 3 HENRY        113  3kinghenryvi        1590
## 4 Henry         63  3kinghenryvi        1590
## 5 HENRY          1 rapeoflucrece        1609
## 6 HENRY        122 kingrichardii        1595
```

Here we use the `filter` verb to extract records and, yes it seems words are repeated, we need to do something about the repeated words. We will aggregate by lower case word, corpus and corpus date, summing all the instances of the word using `dplyr`'s functions.

Web2PDF
converted by Web2PDFConvert.com

```
shakespeare<-mutate(shakespeare, word=tolower(word))
grp<-group_by(shakespeare, word, corpus, corpus_date)
shakespeare<-summarize(grp, word_count=sum(word_count))
head(filter(shakespeare, tolower(word)=="henry"))
## Source: local data frame [6 x 4]
## Groups: word, corpus
##
##    word       corpus corpus_date word_count
## 1 henry 1kinghenryiv        1597        255
## 2 henry 1kinghenryvi        1590        103
## 3 henry 2kinghenryiv        1598        133
## 4 henry 2kinghenryvi        1590        162
## 5 henry 3kinghenryvi        1590        176
## 6 henry   kinghenryv        1599        194
```

Better, each occurrence of "henry" is in a different work of Shakespeare now (no repetition). Now let's use `dplyr` here to take a quick look at what the most and least popular words are by computing the total times each word occurs across all of Shakespeare's works. We will use `dplyr` at its most basic to start, grouping by word, summing occurrences (`total`), counting the number of Shakespeare works they occur in (`count`) and arranging by total occurrences (in descending order).

```
grp <- group_by(shakespeare, word)
cnts <- summarize(grp, count=n(), total = sum(word_count))
word.ount <- arrange(cnts, desc(total))

##Source: local data frame [6 x 3]
##
##  word count total
##1  the    42 29801
##2  and    42 27529
##3    i    42 21029
##4   to    42 20957
##5   of    42 18514
##6    a    42 15370
##6    a    42 15370
```

OK, this worked like a charm and, in my opinion, is cleaner than using `base` functions (getting variable names right with `aggregate`, for example, is a pain) but there are a lot of unnecessary keystrokes and we create some unecessary interim objects (`grp`, `cnts`). Let's try a different way.

## 3a) `magrittr`: streamline your code with basic piping

Piping is built into `dplyr`. Originally the author of `dplyr` (Hadley Wickham) used piping of the form `%.%` but, starting with version 0.2, he adopted piping using magrittr (%>%). Note, though, that the new pipe %<>% is not in version 0.4 of `dplyr` so I load the package `magrittr` separately below. We can use pipes to re-write the code like this:

Web2PDF
converted by Web2PDFConvert.com

```
library(magrittr)
word.count <- group_by(shakespeare, word) %>%
  summarize(count=n(), total = sum(word_count)) %>%
  arrange(desc(total))

head(word.count)
## Source: local data frame [6 x 3]
##
##   word count total
## 1  the    42 29801
## 2  and    42 27529
## 3    i    42 21029
## 4   to    42 20957
## 5   of    42 18514
## 6    a    42 15370
```

Much simpler! The basic idea is that the result from one operation gets piped to the next operation. This saves us the pain of creating interim objects and even saves us from having to give the interim pieces a name. See above where instead of:

```
summarize(grp, count=n(), total = sum(word_count))
```

we simply write

```
summarize(count=n(), total = sum(word_count))
```

We can do this because the pipe passes the interim object straight to the `summarize` function.

## 3b) `magrittr`: streamline more using the compound assignment pipe

You see above that the most common words in Shakespeare are dull (e.g., the, I, and, to, of). So let's see if the words might be more interesting if we limit to words with more then 4 characters and, crucially, limit to words that do NOT occur in all works of Shakespeare (eliminating 'the', 'I' etc).

Using the `dplyr` syntax we've already used we might write code like:

```
word.count <- filter(word.count, nchar(word)>4, count<42)
```

This works fine with one exception that has always bothered me. We are repeating the name of the table. Thanks to Stefan Milton Bache, the `magrittr` author, we can now drop these keystrokes using what he calls the compound assignment operator. This code can be re-written as:

Web2PDF
converted by Web2PDFConvert.com

```
# note that you should not have to load the magrittr library
# separately from dplyr but the %<>% operator is very new
word.count %<>% filter(nchar(word)>4, count<42)
head(word.count)
## Source: local data frame [6 x 3]
##
##     word count total
## 1  enter    39  2406
## 2  first    41  1465
## 3  henry    13  1311
## 4  speak    40  1194
## 5 exeunt    37  1061
## 6  queen    35  1005
```

This essentially says “take the shakespeare object and pipe it to filter. Then pipe it back and write over the original object”. Less code! The words are more interesting. They don't rank with some of Shakespeare's most interesting words (zounds, zwaggered) but better...

## 4) `tidyr`: tidy and re-structure your data

On my computer monitor I have a tiny bit of example code that reminds me how to use the `base` function `reshape`. I can never seem to remember how to specify the various arguments. The packages `reshape` and `reshape2` by Hadley Wickham were designed to more simply restructure data. Hadley now has a new package, `tidyr`, designed to help re-arrange data and, importantly, to integrate with `dplyr` and `magrittr`.

In this mini-example, let's go back and filter the original data to just the top 8 words in our new list (and we will drop the `corpus_date` column).

```
top8<-word.count$word[1:8]
top8 <- filter(shakespeare, word%in%top8)%>%
  select(-corpus_date)

## Note that Hadley Wickham suggests simpler code as
## top8<-shakespeare %>% semi_join(head(word.count,8))

head(top8)
## Source: local data frame [6 x 3]
## Groups: word, corpus
##
##    word               corpus word_count
## 1 death         1kinghenryiv         17
## 2 death         1kinghenryvi         44
## 3 death         2kinghenryiv         23
## 4 death         2kinghenryvi         57
## 5 death         3kinghenryvi         39
## 6 death allswellthatendswell         17
```

OK, now we will illustrate how easy it to restructure data with `tidyr` using the `spread` verb. With our data, I think a 'wide' format makes more sense and might show some initial patterns.

Web2PDF
converted by Web2PDFConvert.com

```
library(tidyr)
top8.wide<- spread(top8, word, word_count)
head(top8.wide, n=10)
## Source: local data frame [10 x 9]
##
##                  corpus death enter exeunt first great henry queen speak
## 1          1kinghenryiv    17    63     28    19    22   255     3    29
## 2          1kinghenryvi    44    83     36    38    30   103    10    18
## 3          2kinghenryiv    23    65     32    30    25   133     1    40
## 4          2kinghenryvi    57    84     40    40    31   162   103    22
## 5          3kinghenryvi    39    78     34    25    19   176   131    45
## 6  allswellthatendswell    17    56     26   117    46    NA     3    42
## 7    antonyandcleopatra    32   112     55    47    48    NA    47    38
## 8           asyoulikeit     7    54     27    27    12    NA     1    28
## 9        comedyoferrors     9    40     14    13     9    NA    NA    14
## 10           coriolanus    19    94     45   139    33    NA     1    55
```

Note that there is a nice argument called fill which can be used to assign missing values. In this case we should not have NAs, instead 0 would be more appropriate since this is the number of word occurrences:

```
top8.wide<- spread(top8, word, word_count, fill=0)
head(top8.wide, n=10)
## Source: local data frame [10 x 9]
##
##                  corpus death enter exeunt first great henry queen speak
## 1          1kinghenryiv    17    63     28    19    22   255     3    29
## 2          1kinghenryvi    44    83     36    38    30   103    10    18
## 3          2kinghenryiv    23    65     32    30    25   133     1    40
## 4          2kinghenryvi    57    84     40    40    31   162   103    22
## 5          3kinghenryvi    39    78     34    25    19   176   131    45
## 6  allswellthatendswell    17    56     26   117    46     0     3    42
## 7    antonyandcleopatra    32   112     55    47    48     0    47    38
## 8           asyoulikeit     7    54     27    27    12     0     1    28
## 9        comedyoferrors     9    40     14    13     9     0     0    14
## 10           coriolanus    19    94     45   139    33     0     1    55
```

Excellent. By the way, “exeunt” is a stage direction telling characters to leave the stage.

For comparison, how might we do this with the more traditional `reshape` function:

Web2PDF
converted by Web2PDFConvert.com

```
top8<-data.frame(top8) # needs to be a data frame
tmp<-reshape(top8, v.names="word_count", idvar="corpus", timevar="word", direction="w
head(tmp)
##                 corpus word_count.death word_count.enter word_count.exeunt
## 1         1kinghenryiv               17               63                28
## 2         1kinghenryvi               44               83                36
## 3         2kinghenryiv               23               65                32
## 4         2kinghenryvi               57               84                40
## 5         3kinghenryvi               39               78                34
## 6 allswellthatendswell               17               56                26
##   word_count.first word_count.great word_count.henry word_count.queen
## 1               19              22             255                3
## 2               38              30             103               10
## 3               30              25             133                1
## 4               40              31             162              103
## 5               25              19             176              131
## 6              117              46              NA                3
##   word_count.speak
## 1               29
## 2               18
## 3               40
## 4               22
## 5               45
## 6               42
```

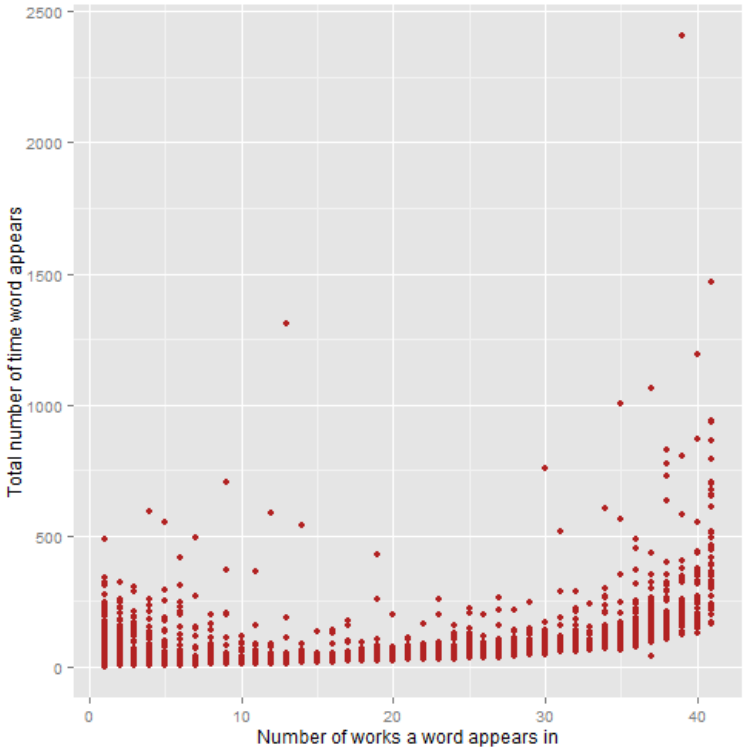
Not only is this confusing and much more code but we still have work to do to rename the columns using `gsub`.

## 5) `ggplot2`: visualize your data

The package `ggplot2` is not nearly as new as `dplyr`, `magrittr` and `tidyr` but it completes the list of components in my new workflow. We have a much more extensive post on ggplot2 (which happens to be our most popular post) and we will use some of those tricks to take a look at the Shakespeare data. We will focus on the total number of word occurrences against the number of works they occur in.

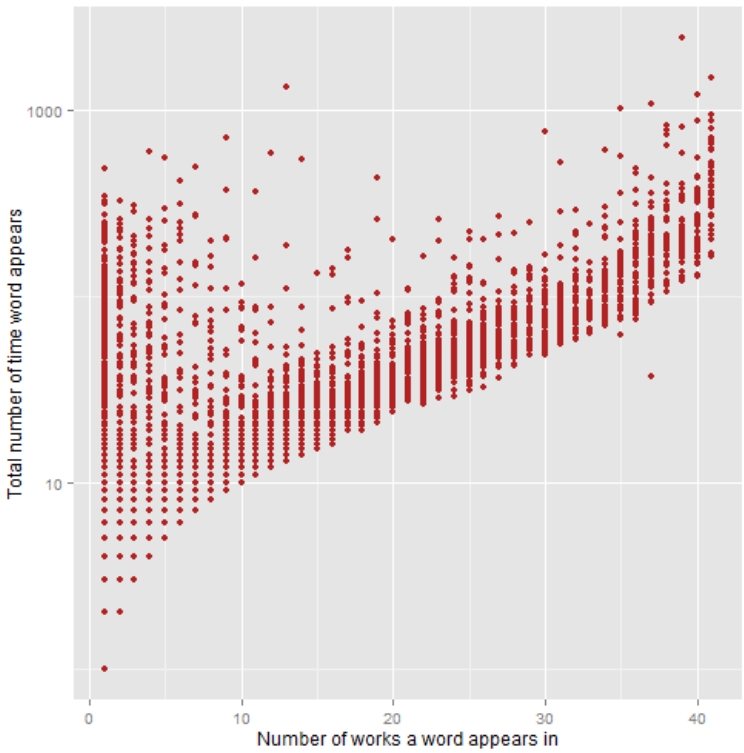
```
library(ggplot2)
ggplot(word.count, aes(count, total))+geom_point(color="firebrick")+
  labs(x="Number of works a word appears in",
       y="Total number of times word appears")
```

Web2PDF
converted by Web2PDFConvert.com




Given that we have some words that occur much, much more than others, let's use a log scale.

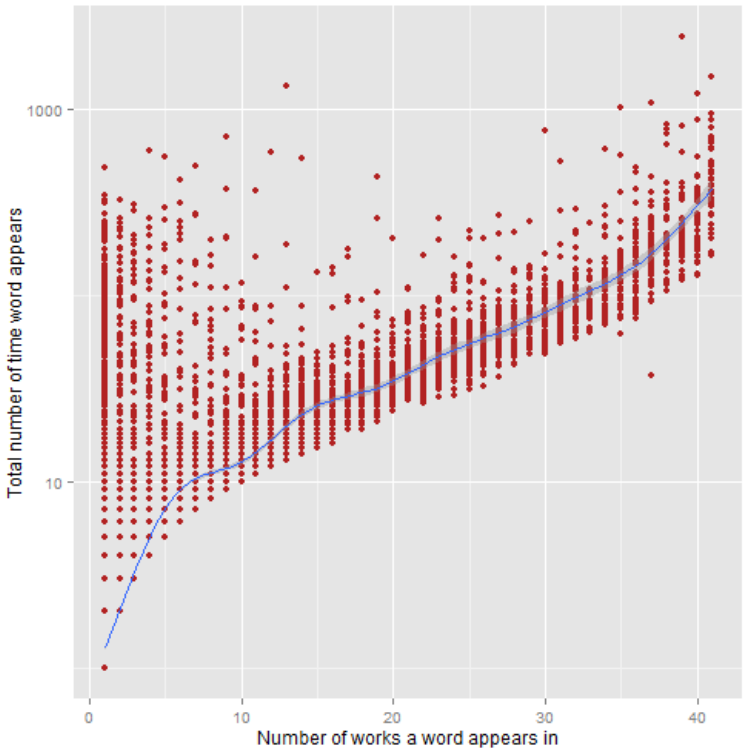
```
ggplot(word.count, aes(count, total))+geom_point(color="firebrick")+
  labs(x="Number of works a word appears in",
       y="Total number of times word appears")+
  scale_y_log10()
```




Much nicer. This plot, though, is very deceptive! If you draw a smooth through the points you might be surprised by what it does at the low end:

Web2PDF
converted by Web2PDFConvert.com

```
ggplot(word.count, aes(count, total))+geom_point(color="firebrick")+
  labs(x="Number of works a word appears in",
       y="Total number of times word appears")+
  scale_y_log10()+stat_smooth()
```



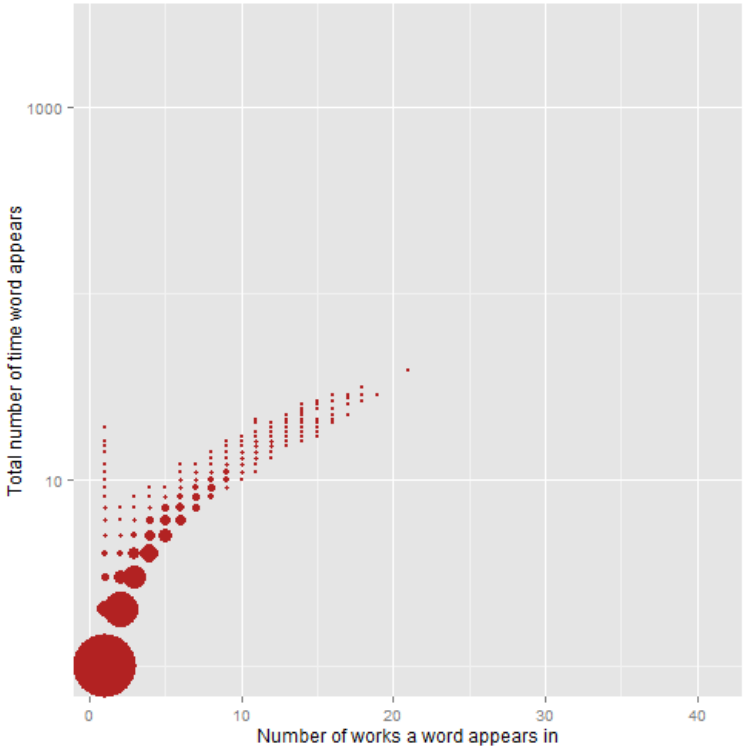

This is because words that appear in a limited number of works also occur much less overall. Let's try a couple of ways to see this starting with sizing the circles based on how many times a count/total combination occurs. We need to compute these stats and join (using `dplyr`'s `inner_join`):

```
word.stats1<-group_by(word.count, count, total)%>%
  summarize(
  cnttot=n())

head(word.stats1)
## Source: local data frame [6 x 3]
## Groups: count
##
##   count total cnttot
## 1     1     1   9834
## 2     1     2    546
## 3     1     3    158
## 4     1     4     74
## 5     1     5     38
## 6     1     6     38

word.count<-inner_join(word.count, word.stats1, by=c("count", "total"))
```

Now we can try sizing the points based on this new variable:

Web2PDF
converted by Web2PDFConvert.com

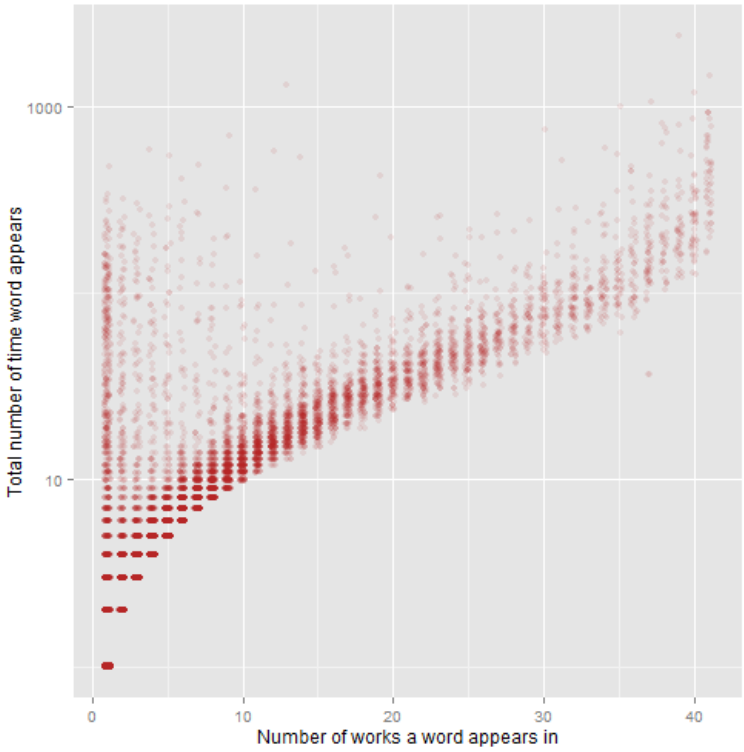
```
ggplot(word.count, aes(count, total, size=cnttot))+
  geom_point(color="firebrick")+
  labs(x="Number of works a word appears in",
       y="Total number of times word appears")+
  scale_y_log10()+
  scale_size_area(max_size=20)+ # scale circles
  theme(legend.position="none")# turn off legend
```




Wow, the words that occur one time in one of Shakespeare's works swamp all the others. That did not work. Let's try something else. How about jittering and adding opacity:

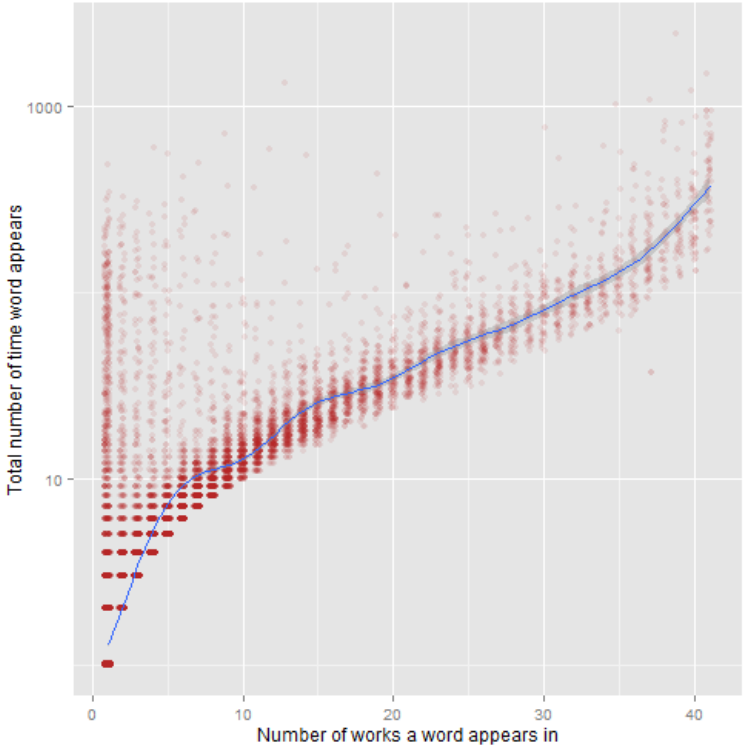
```
ggplot(word.count, aes(count, total))+
    geom_jitter(alpha=0.1,position = position_jitter(width = .2), color="firebrick")+
  labs(x="Number of works a word appears in",
       y="Total number of time word appears")+
  scale_y_log10()
```

Web2PDF
converted by Web2PDFConvert.com




OK, that is a lot more clear. Now if we add the smooth back in it may make more sense:

```
ggplot(word.count, aes(count, total))+
    geom_jitter(alpha=0.1,position = position_jitter(width = .2), color="firebrick")+
  labs(x="Number of works a word appears in",
       y="Total number of time word appears")+
  scale_y_log10()+
  stat_smooth()
```



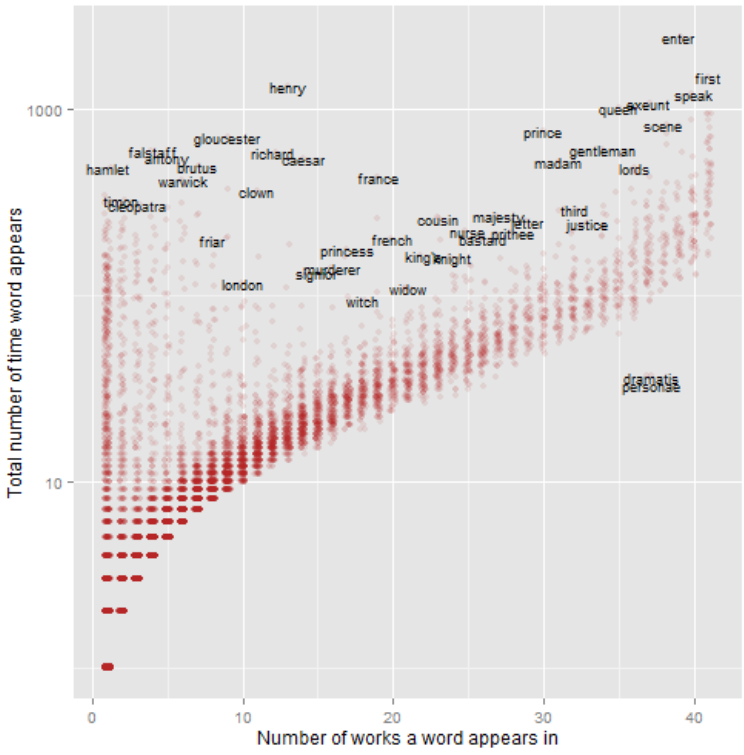

Definitely, it is now clear why the smooth dipped so much to 1/1. As a final step, let's label the words with the max occurrences in each bin. Plus, I'm curious what that word with very low total occurrences but occurs in most of Shakepeare's works is.

Web2PDF
converted by Web2PDFConvert.com

```
word.stats2<-group_by(word.count, count)%>%
  summarize(max=max(total), min=min(total))

head(word.stats1)
## Source: local data frame [6 x 3]
## Groups: count
##
##   count total cnttot
## 1     1     1   9834
## 2     1     2    546
## 3     1     3    158
## 4     1     4     74
## 5     1     5     38
## 6     1     6     38

word.count<-inner_join(word.count, word.stats2, by=c("count"))
```

Now we can place the labels:

```
ggplot(word.count, aes(count, total))+
    geom_jitter(alpha=0.1,position = position_jitter(width = .2), color="firebrick")+
  labs(x="Number of works a word appears in",
       y="Total number of time word appears")+
  scale_y_log10()+
  geom_text(data=filter(word.count, total==max),
            aes(count, total, label=word), size=3)+
    geom_text(data=filter(word.count, total==37 & min==37),
            aes(count, total, label=word), position=position_jitter(height=0.2), size
```




## Summary

Up until last year my R workflow was not dramatically different from when I started using R more than 10 years ago. Thanks to several R package authors, most notably Hadley Wickham,

Web2PDF
converted by Web2PDFConvert.com

my workflow has changed for the better using `dplyr`, `magrittr`, `tidyr` and `ggplot2`.

Posted in ggplot2, R

PREVIOUS POST
Optimize images for the web with one line of code using Grunt

NEXT POST
Using R to download and parse JSON: an example using data from an open data portal

## 8 responses

**Daniel**
January 14, 2015 at 1:38 pm

Fantastic post, thanks! Great to see integration through the Hadley projects for a very efficient, clean workflow. I've been using dplyr and ggplot2 but haven't incorporated tidyr yet. I am a big fan of the piping and hadn't seen the compound assignment operator. That seems really convenient because I do just overwrite the data frame 90% of the time when doing dplyr operations.

**zev@zevross.com**
January 14, 2015 at 1:42 pm

Glad you liked it and thanks for the note! Even if you've been using dplyr v 0.4 has some nice new functions.

**Joe**
January 30, 2015 at 10:21 am

Hi,

Nice piece.

I was wondering whether you can add mutate() to the piping chain. I would have thought you could do something like this:

```
shakespeare %>%
mutate(word = tolower(word)) %>%
group_by(word) %>%
summarize(count =n(), total = sum(word_count)) %>%
arrange(desc(total))
```

However, when I try to do that, the counts are completely off. "the" for instance, is said to appear in 101 works, "and" in 90, etc, which makes no sense at all to me. I'm sure I messed up the syntax somewhere, but can't figure out where. Thanks for any tips.

Web2PDF
converted by Web2PDFConvert.com

**Joe**
January 30, 2015 at 11:00 am

Sorry, I mean to add that the only way I could get it to work was by changing `count = n( )` to `count = n_distinct (corpus)`

**zev@zevross.com**
January 30, 2015 at 1:31 pm

Yes, you can add mutate to the pipeline. Offhand, I'm not seeing the issue in your code but perhaps start with a "fresh" version of the shakespeare data.frame and run these two chunks (in order) and see if you still have problems.

```
word.countA <- shakespeare %>%
  mutate(word = tolower(word)) %>%
  group_by(word) %>%
  summarize(count=n(), total = sum(word_count)) %>%
  arrange(desc(total))

head(word.countA)
#   word count total
# 1  the   101 29801
# 2  and    90 27529
# 3    i    43 21029
# 4   to    88 20957
# 5   of   108 18514
# 6    a    84 15370
< />
shakespeare<-mutate(shakespeare, word=tolower(word))
word.countB <- group_by(shakespeare, word) %>%
  summarize(count=n(), total = sum(word_count)) %>%
  arrange(desc(total))

head(word.countB)<
#   word count total<
# 1  the   101 29801<
# 2  and    90 27529<
# 3    i    43 21029<
# 4   to    88 20957<
# 5   of   108 18514<
# 6    a    84 15370
```

**Joe**
January 30, 2015 at 2:01 pm

Thanks for the answer. It look like the above code shows the same problem. Look at the count for "the": it's 101 and it should be 42, shouldn't it?

**zev@zevross.com**
January 30, 2015 at 2:35 pm

Absolutely! Sorry, in my haste to respond to you I didn't look at the question carefully enough. You can add the "mutate" piece of the code to the piping but you also have to add the other bits from the

Web2PDF
converted by Web2PDFConvert.com

initial code – this includes two group_by statements so here it is with mutate:

```
word.countA <- shakespeare %>%
  mutate(word=tolower(word))%>%
  group_by(word, corpus, corpus_date)%>%
  summarize(word_count=sum(word_count))%>%
  group_by(word) %>%
  summarize(count=n(), total = sum(word_count)) %>%
  arrange(desc(total))

#  word count total
#1  the    42 29801
#2  and    42 27529
#3    i    42 21029
#4   to    42 20957
#5   of    42 18514
#6    a    42 15370
```

**Joe**
January 30, 2015 at 2:59 pm

Thanks. That looks like the missing piece. I'll have to check whether using count = n_distinct(corpus) gives an equivalent answer (the first lines are at least the same). Using n_distinct for the corpus count seems more transparent to me than using two group group_by.

Comments are closed.

This is the home for posts on the more technical aspects of our work. For detail on our projects, clients, examples and more visit:

ZevRoss Spatial Analysis

Search ...

My Tweets

## Categories

- D3
- Data Visualization
- Database
- GDAL
- ggplot2
- GIS/Maps
- JavaScript
- LeafletJS
- PostGIS

Web2PDF
converted by Web2PDFConvert.com

- PostgreSQL
- Python
- R
- Regular Expressions
- Spatial
- Uncategorized
- Web Development
- Web Mapping
-

## Recent Comments

- Tung on Scrape website data with the new R package `rvest` (+ a postscript on interacting with web pages with `RSelenium`)
- zev@zevross.com on Scrape website data with the new R package `rvest` (+ a postscript on interacting with web pages with `RSelenium`)
- Patrick on Scrape website data with the new R package `rvest` (+ a postscript on interacting with web pages with `RSelenium`)
- steiger on Scrape website data with the new R package `rvest` (+ a postscript on interacting with web pages with `RSelenium`)
- zev@zevross.com on Scrape website data with the new R package `rvest` (+ a postscript on interacting with web pages with `RSelenium`)

© 2015 Technical Tidbits From Spatial Analysis & Data Science

Powered by WordPress & Themegraphy

Web2PDF
converted by Web2PDFConvert.com