Week 8: Interactive data graphics (with Shiny) and dealing with text data

Vicki Hertzberg

March 1, 2017

Today we are going to go through a few topics. We are going to start with interactive data graphics, and we will talk about making Shiny apps in R. We will also talk about dealing with text data. If there is time, we will talk some about analyzing microbiome data. I'm adding this file to make it able to commit.

Interactive Data Graphics

With Web 2.0, web browsers became more complex in the mid-00's along with an increased demand for interactive data visualizations in a browser. To date, all we have discussed are static images, but there are tools that make it easy (or easier) to make interactive data graphics.

One of these is JavaScript. With JavaScript, the computations are taking place on the *client side*, not on the host's web server.

The current state of the art for client-side dynamic data graphics on the web is D3, a JavaScript Library. D3 stands for "data-driven documents."

How to do with with R then? The developers of R Studio have come to the rescue with the package, which allows R developers to create packages that render graphics in HTML using D3. In other words, R developers can make use of D3 without having to learn D3. Moreover, since this is happening on the R Studio side, R users can embed these graphics in annotated web documents.

One such tool is Plot.ly, which is really a project to develop the ability to generate data graphics between R, Python, and other tools. It is based on the `plotly.js` JavaScript library. In R we get the functionality of Plot.ly by using the `plotly` package.

An especially attractive feature of `plotly` is that it can convert any `ggplot2` object into a `plotly` object using the `ggplotly()` function. It supports the following capabilities:

- *brushing*: marking selected points
- *mouse-over*: points display additional information when the mouse hovers over them
- *panning*: moving across the viewing pane in a parallel direction
- *zooming*: moving into an image.

Let's do an example. The package `babynames` contains a dataset listing the names of all babies born in the US since 1880. We will use this dataset to determine the frequency with which babies were given the names of one of the members of the Beatles over time.

```
# Create dataframe by grabbing the data then filtering
library(ggplot2)
library(plotly)
```

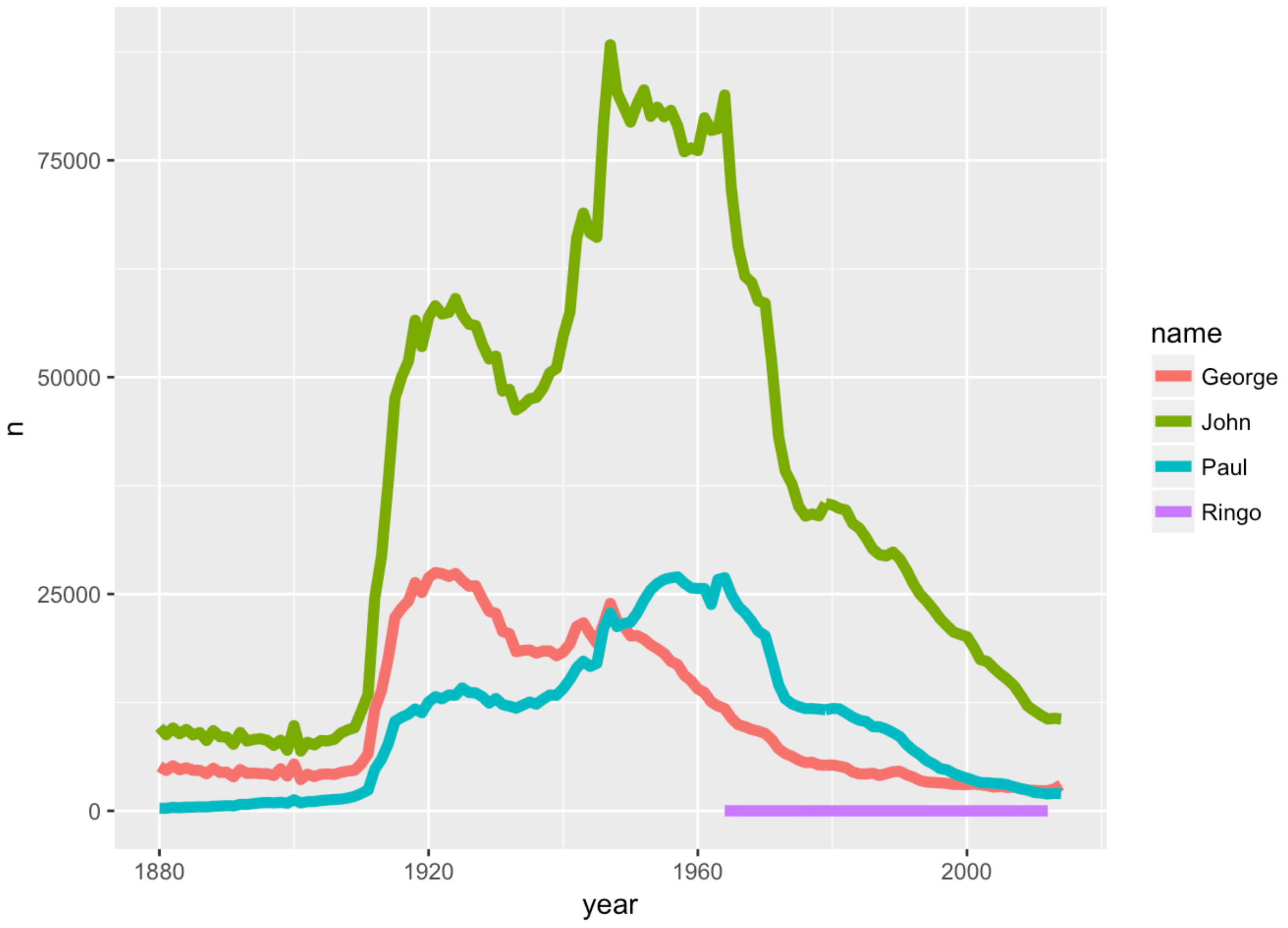
```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

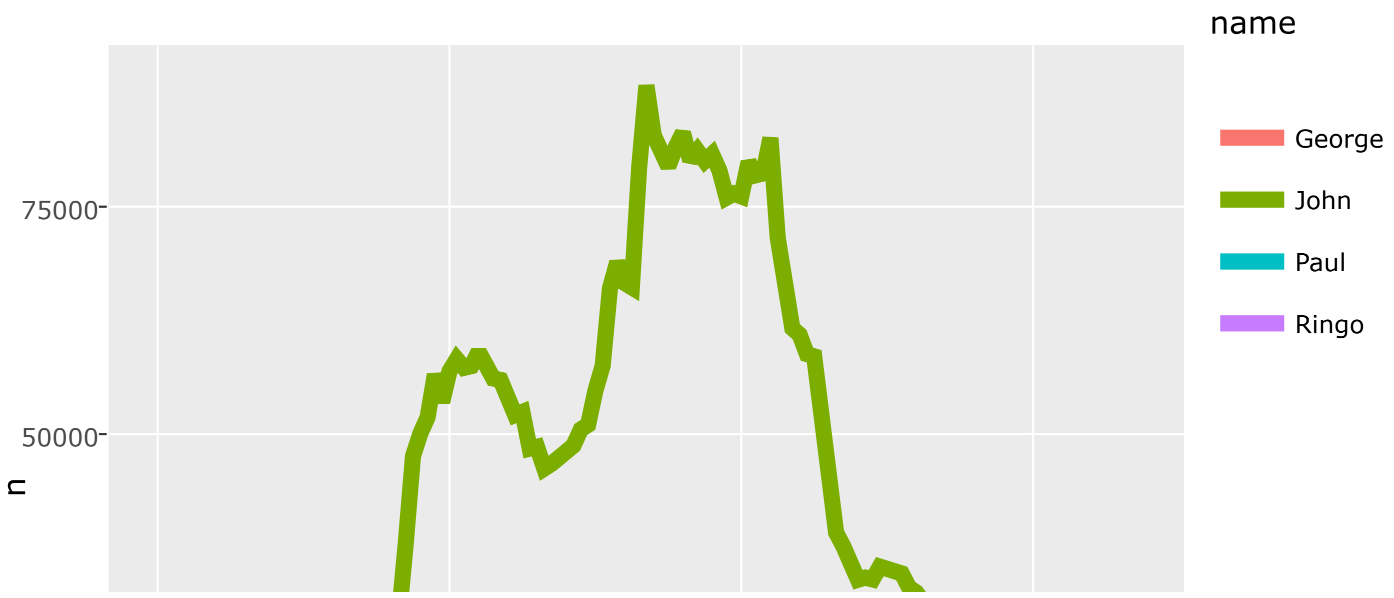
```
## The following object is masked from 'package:stats':
##
##     filter
```

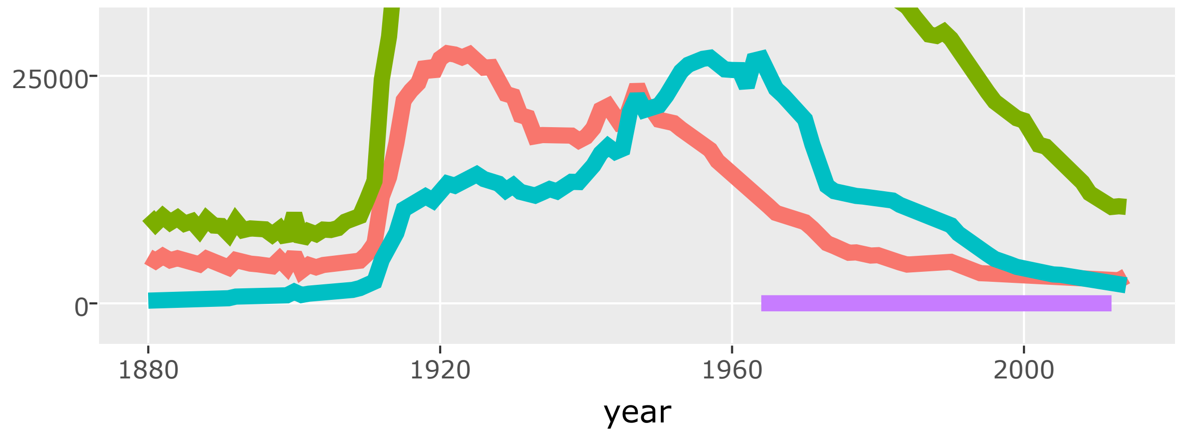
```
## The following object is masked from 'package:graphics':
##
##     layout
```

```
library(babynames)
Beatles <- babynames %>%
  filter(name %in% c("John", "Paul", "George", "Ringo") & sex == "M")

# Build the plot

beatles_plot <- ggplot(data = Beatles, aes(x=year, y = n)) +
  geom_line(aes(color=name), size=2)
beatles_plot
```



```
# Make it interactive

ggplotly(beatles_plot)
```





So use the tools for mousing, brushing, etc.

Another of the htmlwidgets is the DT (i.e., DataTables) package that makes data tables interactive. Let's look at an example with our Beatles names data.

```
# Build a dynamic table

library(DT)
datatable(Beatles, options = list(pageLength = 25))
```

Show 25 entries Search:

	year	sex	name	n	prop
1	1880	M	John	9655	0.0815456081081081
2	1880	M	George	5126	0.0432939189189189
3	1880	M	Paul	301	0.00254222972972973
4	1881	M	John	8769	0.0809814931107089
5	1881	M	George	4664	0.0430719219829338
6	1881	M	Paul	291	0.00268737763658528
7	1882	M	John	9557	0.0783148820401039
8	1882	M	George	5193	0.0425540632451878
9	1882	M	Paul	397	0.00325321839174649
10	1883	M	John	8894	0.0790718349928876
11	1883	M	George	4736	0.0421052631578947

12	1883	M	Paul	358	0.0031827880512091
13	1884	M	John	9388	0.0764862596850278
14	1884	M	George	4961	0.0404184420853667
15	1884	M	Paul	422	0.00343813395686853
16	1885	M	John	8756	0.0755172621973833
17	1885	M	George	4674	0.0403115216435095
18	1885	M	Paul	428	0.00369134173372317
19	1886	M	John	9026	0.075822615737435
20	1886	M	George	4671	0.0392385816651406
21	1886	M	Paul	466	0.00391461765274149
22	1887	M	John	8110	0.0741899482225516
23	1887	M	George	4284	0.0391898567429606
24	1887	M	Paul	449	0.0041074336315568
25	1888	M	John	9248	0.071188841333867

Showing 1 to 25 of 430 entries Previous 1 2 3 4 5 … 18 Next

Pretty cool, huh!

Another tool for dynamic visualization is the `\textcolor{red}{ggvis}` package. This package is *not* built using the D3 or frameworks. Let's use it to create a visualization of the proportion of male babies named John as a function of the number of names over time, such that the user can mouse over a values to see the year, number, and proportion.

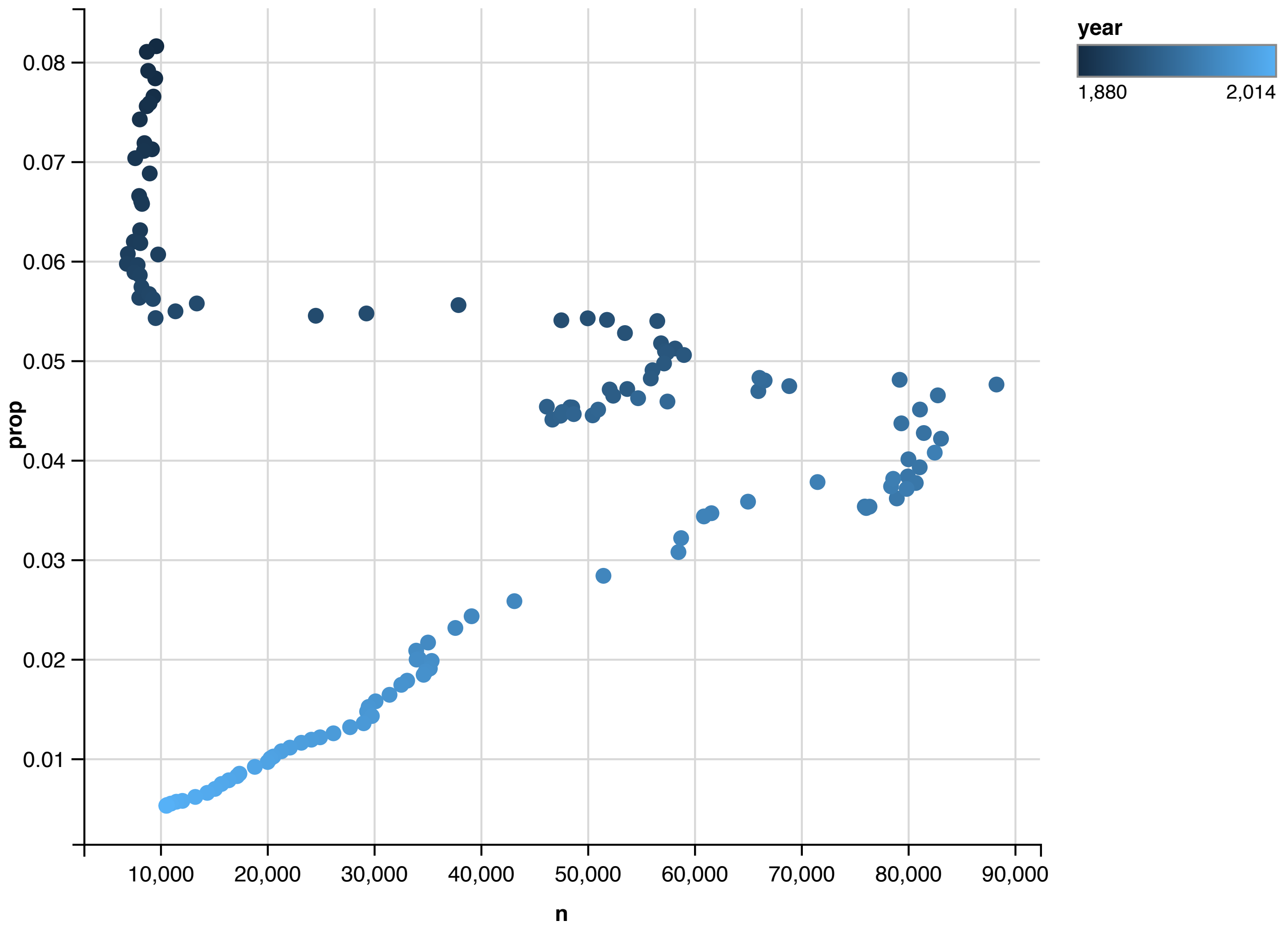
```
library(ggvis)
```

```
##
## Attaching package: 'ggvis'
## The following objects are masked from 'package:plotly':
##
##     add_data, hide_legend
```

```
## The following object is masked from 'package:ggplot2':
##
##     resolution
```

```
# Find out how many males named John
John <- filter(Beatles, name=="John")

# Find out how many males
all_values <- function(x){
  if (is.null(x)) return(NULL)
  row <- John[John$year == x$year, ]
  paste0(names(row), ": ", format(row), collapse = "<br />")
}

John %>%
  ggvis(~n, ~prop, fill = ~year) %>%
  layer_points() %>%
  add_tooltip(all_values, "hover")
```

```
## Warning: Can't output dynamic/interactive ggvis plots in a knitr document.
## Generating a static (non-dynamic, non-interactive) version of the plot.
```



And then there is *Shiny*

Shiny is an R package that can turn your analyses into interactive web applications. It is beyond the scope of the course, not because it is at an advanced level, but more that it is a topic for which we need more time to teach, and we don't have that time in the syllabus. We are not going to tell you *how* to make a shiny application here, but we are going to show you a couple of examples and tell you how to learn more.

Example 1

See (https://cpelat.shinyapps.io/mass/ (https://cpelat.shinyapps.io/mass/))[https://cpelat.shinyapps.io/mass/ (https://cpelat.shinyapps.io/mass/)] for disease surveillance in France.

Example 2

See (https://miningthedetails.shinyapps.io/knn-dashboard-shiny-plotly/ (https://miningthedetails.shinyapps.io/knn-dashboard-shiny-plotly/))[https://miningthedetails.shinyapps.io/knn-dashboard-shiny-plotly/ (https://miningthedetails.shinyapps.io/knn-dashboard-shiny-plotly/)] for data mining to predict heart disease.

The reason that we bring up Shiny is that with another 3 hours or so, you could build your own Shiny app. You have the skills now in terms of R coding, it would just take Melinda and me a couple of hours to get you through all of the steps necessary to do so. If you are interested, there are a variety of online tutorials, see

- (http://shiny.rstudio.com/tutorial/ (http://shiny.rstudio.com/tutorial/))[http://shiny.rstudio.com/tutorial/ (http://shiny.rstudio.com/tutorial/)]
- (http://deanattali.com/blog/building-shiny-apps-tutorial/ (http://deanattali.com/blog/building-shiny-apps-tutorial/))[http://deanattali.com/blog/building-shiny-apps-tutorial/ (http://deanattali.com/blog/building-shiny-apps-tutorial/)]

Working with Text

Machines ==> Good at storing text, not so good at understanding text

Humans ==> Good at understanding text, not so good at storing text

To process text, you need an extra set of wrangling skills. We are now going to introduce how to - *ingest* text - create *corpora* (collections of text document) - use *regular expressions* to automate text searches

We are going to use text mining techniques to explore the play *The Tragedy of Hamlet, Prince of Denmark*, aka *Hamlet*. To do this, we are going to pull the text from the Project Gutenberg site, (http://www.gutenberg.org (http://www.gutenberg.org))[http://www.gutenberg.org (http://www.gutenberg.org)]. Project Gutenberg has over 53,000 electronic books and other text documents that you can download *for free*.

```
# Get the text of Hamlet

library(RCurl)
```

```
## Loading required package: bitops
```

```
hamlet_url <- "http://www.gutenberg.org/cache/epub/1787/pg1787.txt"
Hamlet_raw <- RCurl::getURL(hamlet_url)
```

Note that the object `Hamlet_raw` is a *single string* of text containing the entire play.

To work with this, we are going to have to split this string into a vector of strings, and we will do this using the function `strsplit()`. We will also have to specify the end of line character(s), which in this case are: `\r\n`.

```
# split string at end of line codes (\r\n); this returns a list - we only want the fi
rst element in the list
hamlet <- strsplit(Hamlet_raw, "\r\n")[[1]]
length(hamlet)
```

```
## [1] 5154
```

Now let's examine the text:

```
#examine the text - list out some elements
hamlet[350:360]
```

```
##  [1] "  Hor. Stay! Speak, speak! I charge thee speak!"
##  [2] "                                                     Exit Ghost."
##  [3] "  Mar. 'Tis gone and will not answer."
##  [4] "  Ber. How now, Horatio? You tremble and look pale."
##  [5] "    Is not this something more than fantasy?"
##  [6] "    What think you on't?"
##  [7] "  Hor. Before my God, I might not this believe"
##  [8] "    Without the sensible and true avouch"
##  [9] "    Of mine own eyes."
## [10] "  Mar. Is it not like the King?"
## [11] "  Hor. As thou art to thyself."
```

Notice that in this new `hamlet` object there are two spaces between the start of a line, then an abbreviation for the speaker's name, then another space. We can take advantage of this and other patterns to quantify the ideas within the text. As an example, let's see how many times Hamlet speaks.

```
# get the number of lines that Hamlet speaks
hamlet_lines <- grep("  Ham. ", hamlet, value = TRUE)
length(hamlet_lines)
```

```
## [1] 358
```

```
# inspect the data
head(hamlet_lines)
```

```
## [1] "  Ham. [aside] A little more than kin, and less than kind!"
## [2] "  Ham. Not so, my lord. I am too much i' th' sun."
## [3] "  Ham. Ay, madam, it is common."
## [4] "  Ham. Seems, madam, Nay, it is. I know not 'seems.'"
## [5] "  Ham. I shall in all my best obey you, madam."
## [6] "  Ham. O that this too too solid flesh would melt,"
```

Look at the difference if we don't consider the space after the abbreviation in our expression to evaluate:

```
# get the number of lines that Hamlet speaks
hamlet_lines <- grep("  Ham.", hamlet, value = TRUE)
length(hamlet_lines)
```

```
## [1] 364
```

```
# inspect the data
head(hamlet_lines)
```

```
## [1] "  Hamlet, son to the former, and nephew to the present king."
## [2] "  Ham. [aside] A little more than kin, and less than kind!"
## [3] "  Ham. Not so, my lord. I am too much i' th' sun."
## [4] "  Ham. Ay, madam, it is common."
## [5] "  Ham. Seems, madam, Nay, it is. I know not 'seems.'"
## [6] "  Ham. I shall in all my best obey you, madam."
```

We will see that happens down below.

The `grep()` function is what you use when you want to use R to find a needle in a haystack. The first argument to the function is a *regular expression* (i.e., a pattern) that you want to find, and the second argument is the character vector in which you want to find the patterns. Note that if you do not include "value = TRUE", the function will return the indices of the haystack in which the needles were found. To illustrate, let's look for Ophelia's lines;

```
# find Ophelia's lines
ophelia_lines <- grep(" Oph. ", hamlet)
length(ophelia_lines)
```

```
## [1] 58
```

```
head(ophelia_lines)
```

```
## [1] 794 801 837 882 886 896
```

The function `grepl()` uses the same syntax but returns instead a logical vector as long as the haystack. For example,

```
#illustrate differences between grep and grepl
length(grep("  Ham. ", hamlet))
```

```
## [1] 358
```

```
length(grepl("  Ham. ", hamlet))
```

```
## [1] 5154
```

To extract the piece of each matching line that actually matched, use the `str_extract()` function from the `stringr` package.

```
# Extract the lines that match
library(stringr)
pattern <- "  Ham. "
grep(pattern, hamlet, value = TRUE) %>%
  str_extract(pattern) %>%
  head()
```

```
## [1] "  Ham. " "  Ham. " "  Ham. " "  Ham. " "  Ham. " "  Ham. "
```

Regular expressions are very powerful and very commonly used in many different environments. Understanding the concept of regular expressions will pay off in the long term.

Regular Expression Syntax

- *.* is a metacharacter that matches any character. If you want to search for the literal values of a metacharacter, you have to use two backslashes in R.

```
# Example of use of . as a metacharacter

hamlet_lines <- grep("  Ham.", hamlet, value = TRUE)
length(hamlet_lines)
```

```
## [1] 364
```

```
hamlet_lines <- grep("  Ham. ", hamlet, value = TRUE)
length(hamlet_lines)
```

```
## [1] 358
```

```
hamlet_lines <- grep("  Ham\\.", hamlet, value = TRUE)
length(hamlet_lines)
```

```
## [1] 358
```

- *Character sets:* Use brackets to define sets of characters to match.

```
# Example os use of character sets

head(grep("H[b-z]", hamlet, value = TRUE))
```

```
## [1] "**Etexts Readable By Both Humans and By Computers, Since 1971**"
## [2] "*These Etexts Prepared By Hundreds of Volunteers and Donations*"
## [3] "  Horatio, friend to Hamlet."
## [4] "  Ber. He."
## [5] "    If you do meet Horatio and Marcellus,"
## [6] "                     Enter Horatio and Marcellus."
```

This notation will result in each occurrence of H followed by any small letter except “a”.

- *Alternation:* For this we use the symbol `|` within parentheses.

```
# Example of alternation

head(grep("  H(a|o)", hamlet, value = TRUE))
```

```
## [1] "  Hamlet, son to the former, and nephew to the present king."
## [2] "  Horatio, friend to Hamlet."
## [3] "  Hor. Friends to this ground."
## [4] "  Hor. A piece of him."
## [5] "  Hor. Tush, tush, 'twill not appear."
## [6] "  Hor. Well, sit we down,"
```

So you see that using the `H(a|o)` alternation allows us to pick up any lines in which occurs the sequence " Ha" or " Ho“.

- *Anchors:* Use `^` to anchor a pattern to the beginning of a text, and use `$` to anchor it to the end.

```
# Example of anchor at the beginning
head(grep("^  H[b-z]", hamlet, value = TRUE))
```

```
## [1] "  Horatio, friend to Hamlet."
## [2] "  Hor. Friends to this ground."
## [3] "  Hor. A piece of him."
## [4] "  Hor. Tush, tush, 'twill not appear."
## [5] "  Hor. Well, sit we down,"
## [6] "  Hor. Most like. It harrows me with fear and wonder."
```

- *Repetitions:* Specify the number of times that we want a certain pattern to occur.
    - `?` means zero or one time

  - `*` means zero or more times
  - `+` means one or more times

```
# Examples of repetitive patterns

head(grep("  $H(a|o)", hamlet, value = TRUE))
```

```
## character(0)
```

```
# Examples of repetitive patterns

head(grep("  *H(a|o)", hamlet, value = TRUE))
```

```
## [1] "This etext is a typo-corrected version of Shakespeare's Hamlet,"
## [2] "The Tragedy of Hamlet, Prince of Denmark"
## [3] "        Mail:  Prof. Michael Hart"
## [4] "  Hamlet, son to the former, and nephew to the present king."
## [5] "  Horatio, friend to Hamlet."
## [6] "  Gertrude, Queen of Denmark, mother to Hamlet."
```

```
# Examples of repetitive patterns

head(grep("  +H(a|o)", hamlet, value = TRUE))
```

```
## [1] "  Hamlet, son to the former, and nephew to the present king."
## [2] "  Horatio, friend to Hamlet."
## [3] "  Hor. Friends to this ground."
## [4] "  Hor. A piece of him."
## [5] "  Hor. Tush, tush, 'twill not appear."
## [6] "  Hor. Well, sit we down,"
```

So how do we use these techniques to analyze the text?

- What can we learn by noting who speaks when? _ When does each character speak as a function of the line number in the play?

```
# Assign the characters
Hamlet <- grepl("  Ham\\.", hamlet)
Ophelia <- grepl("  Oph\\.", hamlet)
Polonius <- grepl("  Pol\\.", hamlet)
Gertrude <- grepl("  Queen\\.", hamlet)
Laertes <- grepl("  Laer\\.", hamlet)
Claudius <- grepl("  King\\.", hamlet)
Horatio <- grepl("  Hor\\.", hamlet)
Fortinbras <- grepl("  For\\.", hamlet)

length(Hamlet)
```

```
## [1] 5154
```

```
length(Ophelia)
```

```
## [1] 5154
```

```
length(Polonius)
```

```
## [1] 5154
```

```
length(Gertrude)
```

```
## [1] 5154
```

```
length(Laertes)
```

```
## [1] 5154
```

```
length(Claudius)
```

```
## [1] 5154
```

```
length(Horatio)
```

```
## [1] 5154
```

```
length(Fortinbras)
```

```
## [1] 5154
```

```
sum(Hamlet)
```

```
## [1] 358
```

```
sum(Ophelia)
```

```
## [1] 58
```

```
sum(Polonius)
```

```
## [1] 86
```

```
sum(Gertrude)
```

```
## [1] 74
```

```
sum(Laertes)
```

```
## [1] 62
```

```
sum(Claudius)
```

```
## [1] 106
```

```
sum(Horatio)
```

```
## [1] 108
```

```
sum(Fortinbras)
```

```
## [1] 2
```

Before we can use these data, we want to convert the *logical* vectors into *numeric* vectors, then tidy the data. There is also a bunch of unwanted text at the beginning and end of the raw text, and we want to get rid of it so that our analyses just pertain to the corpus of the play itself, which starts at line 274 and extends to line 5130.

```
# Let's tidy up the dataset
library(tidyverse)
```

```
## Loading tidyverse: tibble
## Loading tidyverse: tidyr
## Loading tidyverse: readr
## Loading tidyverse: purrr
## Loading tidyverse: dplyr
```

```
## Conflicts with tidy packages ----------------------------------------------
```

```
## complete(): tidyr, RCurl
## filter():   dplyr, plotly, stats
## lag():      dplyr, stats
```

```
speaker_freq <- data.frame(Hamlet, Polonius, Claudius, Horatio, Ophelia, Gertrude, La
ertes) %>%
  mutate(line=1:length(hamlet)) %>%
  gather(key = "character", value = "speak", -line) %>%
  mutate(speak = as.numeric(speak)) %>%
  filter(line > 273 & line < 5131)

glimpse(speaker_freq)
```

```
## Observations: 33,999
## Variables: 3
## $ line      <int> 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 28...
## $ character <chr> "Hamlet", "Hamlet", "Hamlet", "Hamlet", "Hamlet", "H...
## $ speak     <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
```

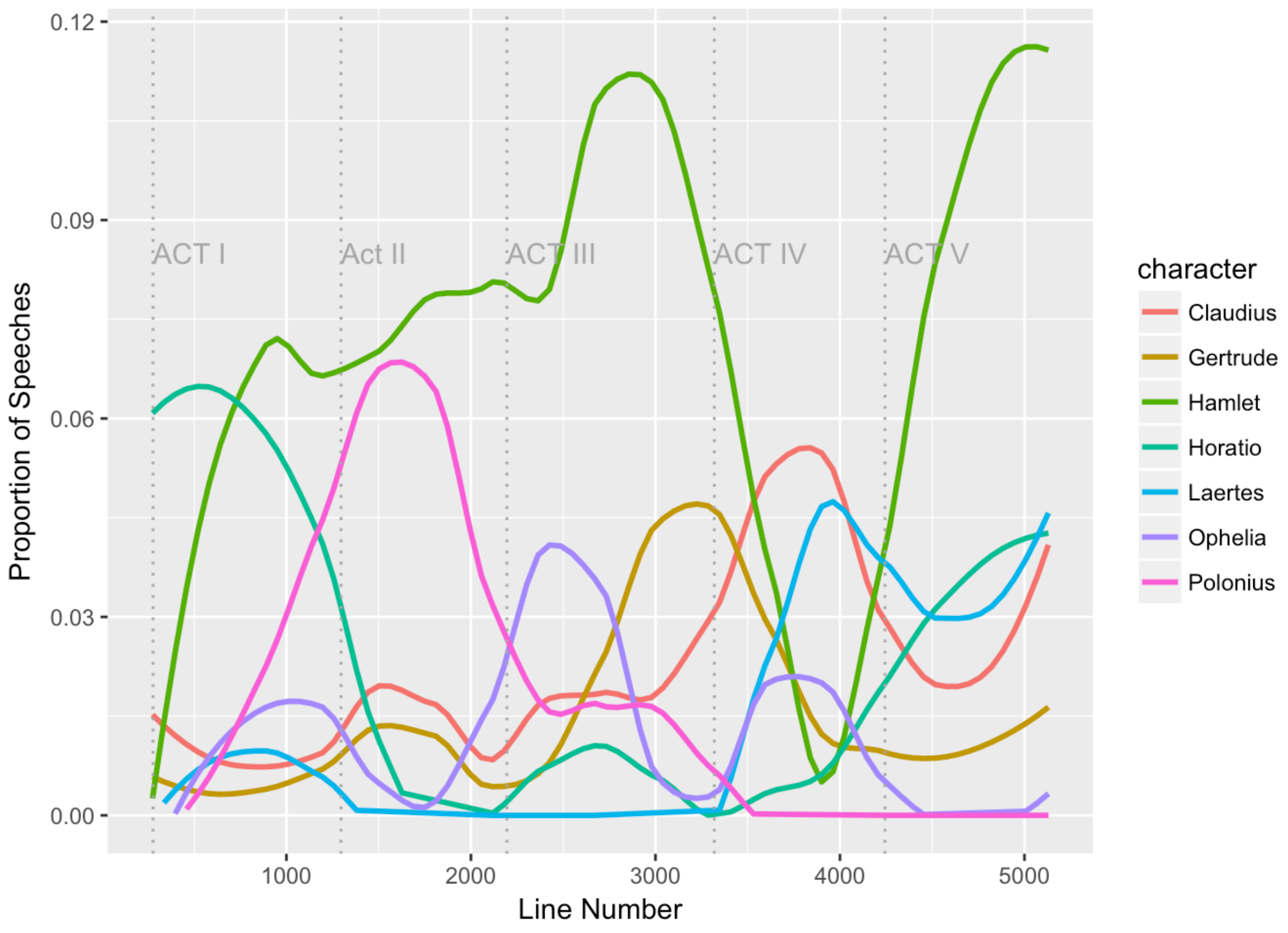
Another thing to do before we do the deep dive: Let’s create some context of helpful information, namely the lines at which each Act starts and stops.

```
# Delineate the Acts
acts_idx <- grep("^A[C|c][T|t] [I|V]+", hamlet)
acts_labels <- str_extract(hamlet[acts_idx], "^A[C|c][T|t] [I|V]+")
acts <- data.frame(line=acts_idx, labels = acts_labels)
```

Now let's see when these 4 characters are speaking throughout the play:

```
# Plot the lines at which each character speaks
ggplot(data = speaker_freq, aes(x=line, y=speak)) +
  geom_smooth(aes(color=character), method = "loess", se = 0, span = 0.4) +
  geom_vline(xintercept = acts_idx, color = "darkgray", lty = 3) +
  geom_text(data = acts, aes(y=0.085, label = labels), hjust = "left", color = "darkg
ray") +
  ylim(c(0, NA)) +
  xlab("Line Number") +
  ylab("Proportion of Speeches")
```

```
## Warning: Removed 54 rows containing missing values (geom_smooth).
```



We can also ingest text by scraping it from the internet. Let’s look at the discography of David Bowie, as listed on Wikipedia:

```
# Grab the table of David Bowie recordings from Wikipedia
library(rvest)
```

```
## Loading required package: xml2
```

```
##
## Attaching package: 'rvest'
```

```
## The following object is masked from 'package:readr':
##
##     guess_encoding
```

```
library(tidyverse)
library(methods)
url <- "https://en.wikipedia.org/wiki/List_of_songs_recorded_by_David_Bowie#Z"
tables <- url %>%
  read_html() %>%
  html_nodes(css = "table")
tables
```

```
## {xml_nodeset (2)}
## [1] <table class="wikitable sortable plainrowheaders">\n<tr>\n<th scope= ...
## [2] <table class="nowraplinks vcard hlist collapsible autocollapse navbo ...
```

```
songs <- html_table(tables[[1]])
glimpse(songs)
```

```
## Observations: 418
## Variables: 4
## $ Song             <chr> "\"1917\"", "\"1984\"", "\"5:15 The Angels Ha...
## $ Writer(s)        <chr> "Bowie, Gabrels !Bowie, Reeves Gabrels", "Bow...
## $ Original release <chr> "\"Thursday's Child\" single", "Diamond Dogs"...
## $ Year             <chr> "1999", "1974", "2002", "1987", "1998", "1991...
```

First let’s clean this up a bit.

```
# Clean the data

songs <- songs %>%
  mutate(Song = gsub('\\"', "", Song), Year = as.numeric(Year)) %>%
  rename(songwriters = `Writer(s)`)
```

```
## Warning in eval(substitute(expr), envir, enclos): NAs introduced by
## coercion
```

It appears that Bowie recorded 418 songs. Who wrote all of them?

```
# Count the number of different songwriters
pattern <- "Bowie"
bowie_wrote <- grepl(pattern, songs$songwriters)
sum(bowie_wrote)
```

```
## [1] 352
```

Another important concept in text mining is to calculate the *term frequency - inverse document frequency (tf-idf)*, also called the *document term matrix*. The frequency of term *t* in document *d* is denoted as *t f (t,d)* and is equal to the number of times that the term *t* appears in document *d*. The *inverse document frequency* measures the prevalence of a term across a set of documents,

$$idf(t, D) = log \frac{|D|}{|\{d \in D : t \in d\}}$$

Finally, we also use *tf.idf(t,d,D)* which we calculate as

$$tf(t, D)xidf(t, D)$$

This is frequently use in search engines, when the relevance of a particular word is needed across a corpus.

Commonly used words like “the” and “a” will appear in every document. Thus, their inverse document frequencies will be zero, and their *tf.idf* will be zero regardless of the term frequency. Documents with high *tf.idf* scores for a particular term will contain that term many times relative to its appearance across many documents, lending increased relevance of that document to the search term of concern.

The `DocumentTermMatrix()` function in the `tm` (text mining) package will create a document term matrix with one row per document and one column per term. Let’s find the term frequency information in the titles of the songs that Bowie sang.

```
library(tm)
```

```
## Loading required package: NLP
```

```
##
## Attaching package: 'NLP'
```

```
## The following object is masked from 'package:ggplot2':
##
##     annotate
```

```
song_titles <- VCorpus(VectorSource(songs$Song)) %>%
  tm_map(removeWords, stopwords("english")) %>%
  DocumentTermMatrix(control = list(weighting = weightTfIdf))
```

```
## Warning in weighting(x): empty document(s): 91
```

```
findFreqTerms(song_titles, 25)
```

```
## [1] "machine)" "you"
```

We see that songs from the album *Tin Machine* have “(Tin Machine)” appended to their name. Let’s fix that then look at the term frequency information again. Looking through the titles, we see two more problems to fix, so let’s do those as well.

```
# A bit of clean up
pattern <- "\\(Tin Machine\\)"
songs$Song <- sub(pattern, "", songs$Song)
pattern2 <- "\\(Live\\)"
songs$Song <- sub(pattern2, "", songs$Song)
pattern5 <- "Bowie\\)"
songs$Song <- sub(pattern5, "", songs$Song)
pattern6 <- "!"
songs$Song <- sub(pattern6, "", songs$Song)
pattern4 <- "Segue"
songs$Song <- sub(pattern4, "", songs$Song)
pattern7 <- "\\(Segue\\)"
songs$Song <- sub(pattern7, "", songs$Song)
pattern8 <- "the"
songs$Song <- sub(pattern8, "", songs$Song)
pattern9 <- "The"
songs$Song <- sub(pattern9, "", songs$Song)
pattern10 <- "David"
songs$Song <- sub(pattern10, "", songs$Song)
pattern11 <- "and"
songs$Song <- sub(pattern11, "", songs$Song)

pattern3 <- "\\(\\)"
songs$Song <- sub(pattern3, "", songs$Song)

songs$Song[353] <- "Tin Machine"

# Run the frequencies again
song_titles <- VCorpus(VectorSource(songs$Song)) %>%
  tm_map(removeWords, stopwords("english")) %>%
  DocumentTermMatrix(control = list(weighting = weightTfIdf))
```

```
## Warning in weighting(x): empty document(s): 91
```

```
findFreqTerms(song_titles, 25)
```

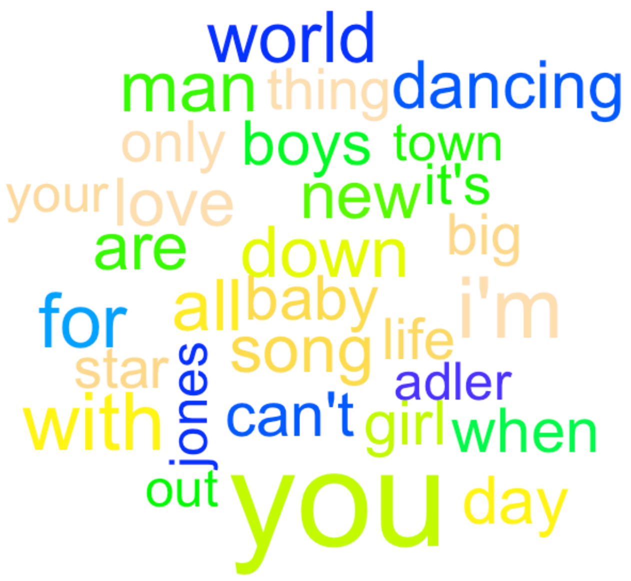
```
## [1] "you"
```

Another way to look at text is to create a *word cloud*, which you can think of as a multivariate histogram for words. You will be surprised, I'm sure, to learn that R has a `wordcloud` package that will allow you to create this object.

```
# Create wordcloud from Bowie song titles

library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

```
library(RColorBrewer)
wordcloud(VCorpus(VectorSource(songs$Song)), max.words = 30, scale = c(4, 1), colors
= topo.colors(n=30), random.color = TRUE)
```


world
man
thing
dancing
only
boys
town
your
love
new
it's
big
are
down
for
all
baby
life
i'm
star
jones
song
adler
with
can't
girl
when
out
you
day