Tidy text and sentiment analysis

JHU Data Science

Data Cleaning

In general, data cleaning is a process of investigating your data for inaccuracies, or recoding it in a way that makes it more manageable.

⚠️ MOST IMPORTANT RULE - LOOK 👀 AT YOUR DATA! ⚠️

String functions

Pasting strings with `paste` and `paste0`

Paste can be very useful for joining vectors together:

```
paste("Visit", 1:5, sep = "_")
```

```
[1] "Visit_1" "Visit_2" "Visit_3" "Visit_4" "Visit_5"
```

```
paste("Visit", 1:5, sep = "_", collapse = " ")
```

```
[1] "Visit_1 Visit_2 Visit_3 Visit_4 Visit_5"
```

```
paste("To", "is going be the ", "we go to the store!", sep = "day ")
```

```
[1] "Today is going be the day we go to the store!"
```

```
# and paste0 can be even simpler see ?paste0
paste0("Visit",1:5)
```

```
[1] "Visit1" "Visit2" "Visit3" "Visit4" "Visit5"
```

Paste Depicting How Collapse Works

```r
paste(1:5)
```

```
[1] "1" "2" "3" "4" "5"
```

```r
paste(1:5, collapse = " ")
```

```
[1] "1 2 3 4 5"
```

Useful String Functions

Useful String functions

- `toupper()`, `tolower()` - uppercase or lowercase your data:
- `str_trim()` (in the `stringr` package) or `trimws` in base
- will trim whitespace on ends
- `stringr::str_squish` - trims and replaces double spaces
- `nchar` - get the number of characters in a string

The `stringr` package

Like `dplyr`, the `stringr` package:

- Makes some things more intuitive
- Is different than base R
- Is used on forums for answers
- Has a standard format for most functions
- the first argument is a string like first argument is a `data.frame` in `dplyr`

'Find' functions: **`stringr`**

`str_detect`, `str_subset`, `str_replace`, and `str_replace_all` search for matches to argument pattern within each element of a character vector: they differ in the format of and amount of detail in the results.

- `str_detect` - returns `TRUE` if `pattern` is found
- `str_subset` - returns only the strings which pattern were detected
- convenient wrapper around `x[str_detect(x, pattern)]`
- `str_extract` - returns only strings which pattern were detected, but ONLY the pattern
- `str_replace` - replaces `pattern` with `replacement` the first time
- `str_replace_all` - replaces `pattern` with `replacement` as many times matched

Let's look at modifier for `stringr`

```
?modifiers
```

- `fixed` - match everything exactly
- `regexp` - default - uses **reg**ular **exp**ressions
- `ignore_case` is an option to not have to use `tolower`
- `boundary` - Match boundaries between things (e.g. words, sentences, characters).

Substringing

Very similar:

Base R

- `substr(x, start, stop)` - substrings from position start to position stop
- `strsplit(x, split)` - splits strings up - returns list!

`stringr`

- `str_sub(x, start, end)` - substrings from position start to position end
- `str_split(string, pattern)` - splits strings up - returns list!

Splitting String: base R

In base R, `strsplit` splits a vector on a string into a `list`

```
x <- c("I really", "like writing", "R code programs")
(y <- strsplit(x, split = " ")) # returns a list
```

```
[[1]]
[1] "I"      "really"

[[2]]
[1] "like"    "writing"

[[3]]
[1] "R"        "code"     "programs"
```

```
(y2 <- stringr::str_split(x, " ")) # returns a list
```

```
[[1]]
[1] "I"      "really"

[[2]]
[1] "like"    "writing"

[[3]]
[1] "R"        "code"     "programs"
```

Using a fixed expression

One example case is when you want to split on a period "`.`". In regular expressions `.` means **ANY** character, so

```
str_split("I.like.strings", ".")
```

```
[[1]]
 [1] "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
```

```
str_split("I.like.strings", fixed("."))
```

```
[[1]]
[1] "I"       "like"    "strings"
```

Use **`purrr`** or **`apply*`** to extract from string lists

```
sapply(y, dplyr::first) # on the fly
```

```
[1] "I"    "like" "R"
```

```
purrr::map_chr(y, nth, 2) # on the fly
```

```
[1] "really"  "writing" "code"
```

```
sapply(y, dplyr::last) # on the fly
```

```
[1] "really"   "writing"  "programs"
```

Boundary

We can use `boundary` in the case of `str_split` as well:

```
words <- c("These are   some words.")
str_count(words, boundary("word"))
```

```
[1] 4
```

```
# split with space
str_split(words, " ")[[1]]
```

```
[1] "These"  "are"    ""       ""       "some"   "words."
```

```
# split between word
str_split(words, boundary("word"))[[1]]
```

```
[1] "These" "are"   "some"  "words"
```

Splitting/Find/Replace and Regular Expressions

- R can do much more than find exact matches for a whole string
- Like Perl and other languages, it can use regular expressions.
- What are regular expressions?
- Ways to search for specific strings
- Can be very complicated or simple
- Highly Useful - think “Find” on steroids

A bit on Regular Expressions

- http://www.regular-expressions.info/reference.html
- They can use to match a large number of strings in one statement
- `.` matches any single character
- `*` means repeat as many (even if 0) more times the last character
- `?` makes the last thing optional
- `^` matches start of vector `^a` - starts with “a”
- `$` matches end of vector `b$` - ends with “b”

Beginning of line with ^

```
x = c("i think we all rule for participating",
      "i think i have been outed",
      "i think this will be quite fun actually",
      "it will be fun, i think")

str_detect(x, "^i think")
```

```
[1]  TRUE  TRUE  TRUE FALSE
```

End of line with $

```r
x = c("well they had something this morning",
      "then had to catch a tram home in the morning",
      "dog obedience school in the morning",
      "this morning I'll go for a run")

str_detect(x, "morning$")
```

```
[1]  TRUE  TRUE  TRUE FALSE
```

Character list with [ ]

```
x = c("Name the worst thing about Bush!",
      "I saw a green bush",
      "BBQ and bushwalking at Molonglo Gorge",
      "BUSH!!")

str_detect(x,"[Bb][Uu][Ss][Hh]")
```

```
[1] TRUE TRUE TRUE TRUE
```

Sets of letters and numbers

```r
x = c("7th inning stretch",
      "2nd half soon to begin. OSU did just win.",
      "3am - cant sleep - too hot still.. :(",
      "5ft 7 sent from heaven")

str_detect(x,"^[0-9][a-zA-Z]")
```

```
[1] TRUE TRUE TRUE TRUE
```

Negative Classes

I want to match NOT a `?` or `.` at the end of line (fixed with `[ ]`).

```
x = c("are you there?",
      "2nd half soon to begin. OSU did just win.",
      "6 and 9",
      "dont worry... we all die anyway!")

str_detect(x,"[^?.]$")
```

```
[1] FALSE FALSE  TRUE  TRUE
```

. means anything

```
x = c("these are post 9-11 rules",
      "NetBios: scanning ip 203.169.114.66",
      "Front Door 9:11:46 AM",
      "Sings: 0118999881999119725...3 !")

str_detect(x, "9.11")
```

```
[1] TRUE TRUE TRUE TRUE
```

means or

```
x = c("Not a whole lot of hurricanes.",
      "We do have floods nearly every day",
      "hurricanes swirl in the other direction",
      "coldfire is STRAIGHT!")

str_detect(x,"flood|earthquake|hurricane|coldfire")
```

```
[1] TRUE TRUE TRUE TRUE
```

Detecting phone numbers

```
x = c("206-555-1122","206-332","4545","test")

phone = "([2-9][0-9]{2})[- .]([0-9]{3})[- .]([0-9]{4})"

str_detect(x,phone)
```

```
[1]  TRUE FALSE FALSE FALSE
```

Read in Salary Data

```
suppressMessages({
  Sal = readr::read_csv(
    "https://raw.githubusercontent.com/muschellij2/adv_data_sci_2023/main/exam
    progress = FALSE)
})
raw_salary_data = Sal
head(Sal)
```

```
# A tibble: 6 × 7
  Name                JobTitle       AgencyID Agency HireDate AnnualSalary GrossP
  <chr>               <chr>          <chr>    <chr>  <chr>    <chr>        <chr>
1 Aaron,Keontae E     AIDE BLUE C… W02200   Youth… 06/10/2… $11310.00     $873.6
2 Aaron,Patricia G    Facilities/… A03031   OED-E… 10/24/1… $53428.00     $52868
3 Aaron,Petra L       ASSISTANT S… A29005   State… 09/25/2… $68300.00     $67439
4 Abaineh,Yohannes T EPIDEMIOLOG… A65026   HLTH-… 07/23/2… $62000.00     $58654
5 Abbene,Anthony M    POLICE OFFI… A99416   Polic… 07/24/2… $43999.00     $39686
6 Abbey,Emmanuel      CONTRACT SE… A40001   M-R I… 05/01/2… $52000.00     $47019
```

'Find' functions: finding values, `stringr` and `dplyr`

```
str_subset(Sal$Name, "Rawlings")
```

```
[1] "Rawlings,Kellye A"            "Rawlings,MarqWell D"
[3] "Rawlings,Paula M"             "Rawlings-Blake,Stephanie C"
```

```
Sal %>% filter(str_detect(Name, "Rawlings"))
```

```
# A tibble: 4 × 7
  Name                     JobTitle AgencyID Agency HireDate AnnualSalary GrossP
  <chr>                    <chr>    <chr>    <chr>  <chr>    <chr>        <chr>
1 Rawlings,Kellye A        EMERGEN… A40302   M-R I… 01/06/2… $47980.00    $68426
2 Rawlings,MarqWell D      AIDE BL… W02384   Youth… 06/15/2… $11310.00    $507.5
3 Rawlings,Paula M         COMMUNI… A04015   R&P-R… 12/10/2… $19802.00    $8195.
4 Rawlings-Blake,Stepha… MAYOR    A01001   Mayor… 12/07/1… $163365.00   $16121
```

Replacing and subbing: `stringr`

We can do the same thing (with 2 piping operations!) in dplyr

```
dplyr_sal = Sal
dplyr_sal = dplyr_sal %>% mutate(
  AnnualSalary = AnnualSalary %>%
    str_replace(fixed("$"), "") %>%
    as.numeric) %>%
  arrange(desc(AnnualSalary))
```

Showing difference in `str_extract` and `str_extract_all`

`str_extract_all` extracts all the matched strings - `\\d` searches for DIGITS/numbers

```
head(str_extract(Sal$AgencyID, "\\d"))
```

```
[1] "0" "0" "2" "6" "9" "4"
```

```
head(str_extract_all(Sal$AgencyID, "\\d"), 2)
```

```
[[1]]
[1] "0" "2" "2" "0" "0"

[[2]]
[1] "0" "3" "0" "3" "1"
```

'Find' functions: base R

`grep`: `grep`, `grepl`, `regexpr` and `gregexpr` search for matches to argument pattern within each element of a character vector: they differ in the format of and amount of detail in the results.

`grep(pattern, x, fixed=FALSE)`, where:

- pattern = character string containing a regular expression to be matched in the given character vector.
- x = a character vector where matches are sought, or an object which can be coerced by as.character to a character vector.
- If fixed=TRUE, it will do exact matching for the phrase anywhere in the vector (regular find)

‘Find’ functions: stringr compared to base R

Base R does not use these functions. Here is a “translator” of the `stringr` function to base R functions

- `str_detect` - similar to `grepl` (return logical)
- `grep(value = FALSE)` is similar to `which(str_detect())`
- `str_subset` - similar to `grep(value = TRUE)` - return value of matched
- `str_replace` - similar to `sub` - replace one time
- `str_replace_all` - similar to `gsub` - replace many times

Important Comparisons

Base R:

- Argument order is `(pattern, x)`
- Uses option `(fixed = TRUE)`

`stringr`

- Argument order is `(string, pattern)` aka `(x, pattern)`
- Uses function `fixed(pattern)`

'Find' functions: Finding Indices

These are the indices where the pattern match occurs:

```
grep("Rawlings",Sal$Name)
```

```
[1] 13832 13833 13834 13835
```

```
which(grepl("Rawlings", Sal$Name))
```

```
[1] 13832 13833 13834 13835
```

```
which(str_detect(Sal$Name, "Rawlings"))
```

```
[1] 13832 13833 13834 13835
```

‘Find’ functions: Finding Logicals

These are the indices where the pattern match occurs:

```
head(grepl("Rawlings",Sal$Name))
```

```
[1] FALSE FALSE FALSE FALSE FALSE FALSE
```

```
head(str_detect(Sal$Name, "Rawlings"))
```

```
[1] FALSE FALSE FALSE FALSE FALSE FALSE
```

'Find' functions: finding values, base R

```r
grep("Rawlings", Sal$Name, value=TRUE)
```

```
[1] "Rawlings,Kellye A"          "Rawlings,MarqWell D"
[3] "Rawlings,Paula M"           "Rawlings-Blake,Stephanie C"
```

```r
Sal[grep("Rawlings", Sal$Name), ]
```

```
# A tibble: 4 × 7
  Name                       JobTitle AgencyID Agency HireDate AnnualSalary GrossP
  <chr>                      <chr>    <chr>    <chr>  <chr>    <chr>        <chr>
1 Rawlings,Kellye A          EMERGEN… A40302   M-R I… 01/06/2… $47980.00    $68426
2 Rawlings,MarqWell D        AIDE BL… W02384   Youth… 06/15/2… $11310.00    $507.5
3 Rawlings,Paula M           COMMUNI… A04015   R&P-R… 12/10/2… $19802.00    $8195.
4 Rawlings-Blake,Stepha… MAYOR    A01001   Mayor… 12/07/1… $163365.00   $16121
```

Showing differnce in `str_extract`

`str_extract` extracts just the matched string

```
ss = str_extract(Sal$Name, "Rawling")
head(ss)
```

```
[1] NA NA NA NA NA NA
```

```
ss[!is.na(ss)]
```

```
[1] "Rawling" "Rawling" "Rawling" "Rawling"
```

Showing differnce in `str_extract` and `str_extract_all`

`str_extract_all` extracts all the matched strings

```
head(str_extract(Sal$AgencyID, "\\d"))
```

```
[1] "0" "0" "2" "6" "9" "4"
```

```
head(str_extract_all(Sal$AgencyID, "\\d"), 2)
```

```
[[1]]
[1] "0" "2" "2" "0" "0"

[[2]]
[1] "0" "3" "0" "3" "1"
```

Using Regular Expressions

- Look for any name that starts with:
- Payne at the beginning,
- Leonard and then an S
- Spence then capital C

```
head(grep("^Payne.*", x = Sal$Name, value = TRUE), 3)
```

```
[1] "Payne El,Jackie"          "Payne Johnson,Nickole A"
[3] "Payne,Chanel"
```

```
head(grep("Leonard.?S", x = Sal$Name, value = TRUE))
```

```
[1] "Payne,Leonard S"      "Szumlanski,Leonard S"
```

```
head(grep("Spence.*C.*", x = Sal$Name, value = TRUE))
```

```
[1] "Greene,Spencer C"    "Spencer,Charles A"   "Spencer,Christian O"
[4] "Spencer,Clarence W"  "Spencer,Michael C"
```

Using Regular Expressions: **stringr**

```
head(str_subset( Sal$Name, "^Payne.*"), 3)
```

```
[1] "Payne El,Jackie"          "Payne Johnson,Nickole A"
[3] "Payne,Chanel"
```

```
head(str_subset( Sal$Name, "Leonard.?S"))
```

```
[1] "Payne,Leonard S"      "Szumlanski,Leonard S"
```

```
head(str_subset( Sal$Name, "Spence.*C.*"))
```

```
[1] "Greene,Spencer C"    "Spencer,Charles A"   "Spencer,Christian O"
[4] "Spencer,Clarence W"  "Spencer,Michael C"
```

Replace

Let’s say we wanted to sort the data set by Annual Salary:

```
class(Sal$AnnualSalary)
```

```
[1] "character"
```

```
sort(c("1", "2", "10")) #  not sort correctly (order simply ranks the data)
```

```
[1] "1"  "10" "2"
```

```
order(c("1", "2", "10"))
```

```
[1] 1 3 2
```

Replace

So we must change the annual pay into a numeric:

```r
head(Sal$AnnualSalary, 4)
```

```
[1] "$11310.00" "$53428.00" "$68300.00" "$62000.00"
```

```r
head(as.numeric(Sal$AnnualSalary), 4)
```

```
Warning in head(as.numeric(Sal$AnnualSalary), 4): NAs introduced by coercion
```

```
[1] NA NA NA NA
```

R didn't like the `$` so it thought turned them all to `NA`.

`sub()` and `gsub()` can do the replacing part in base R.

Replacing and subbing

Now we can replace the `$` with nothing (used `fixed=TRUE` because `$` means ending):

```
Sal$AnnualSalary <- as.numeric(gsub(pattern = "$", replacement="",
                                    Sal$AnnualSalary, fixed=TRUE))
Sal <- Sal[order(Sal$AnnualSalary, decreasing=TRUE), ]
Sal[1:5, c("Name", "AnnualSalary", "JobTitle")]
```

```
# A tibble: 5 × 3
  Name                  AnnualSalary JobTitle
  <chr>                        <dbl> <chr>
1 Bernstein,Gregg L           238772 STATE'S ATTORNEY
2 Charles,Ronnie E            200000 EXECUTIVE LEVEL III
3 Batts,Anthony W             193800 EXECUTIVE LEVEL III
4 Black,Harry E               190000 EXECUTIVE LEVEL III
5 Swift,Michael               187200 CONTRACT SERV SPEC II
```

Replacing and subbing: `stringr`

We can do the same thing (with 2 piping operations!) in dplyr

```
dplyr_sal = Sal
dplyr_sal = dplyr_sal %>% mutate(
  AnnualSalary = AnnualSalary %>%
    str_replace(
      fixed("$"),
      "") %>%
    as.numeric) %>%
  arrange(desc(AnnualSalary))
check_Sal = Sal
rownames(check_Sal) = NULL
all.equal(check_Sal, dplyr_sal)
```

```
[1] TRUE
```

Removing `$` and `,` in Practice

`readr::parse_*` is a number of useful helper functions for parsing columns

```
head(readr::parse_number(raw_salary_data$AnnualSalary))
```

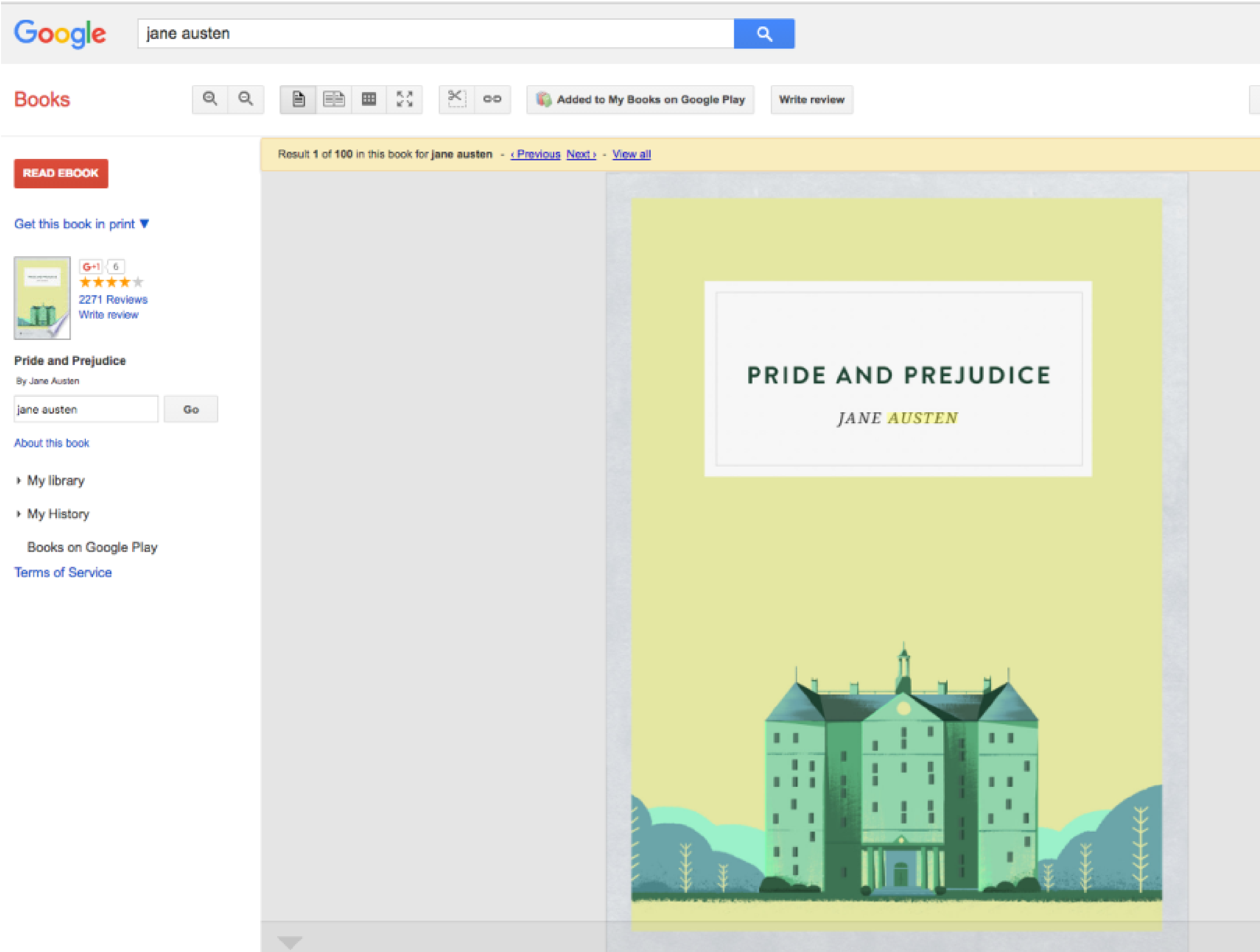
```
[1] 11310 53428 68300 62000 43999 52000
```

```
raw_salary_data %>%
  mutate(across(matches("Salary|Pay"), readr::parse_number)) %>%
  select(matches("Salary|Pay"))
```

```
# A tibble: 18,981 × 2
   AnnualSalary GrossPay
          <dbl>    <dbl>
 1        11310     874.
 2        53428   52868.
 3        68300   67439.
 4        62000   58655.
 5        43999   39687.
 6        52000   47020.
 7        62175   61452.
 8        70918   87900.
 9        42438   53668.
10        11310      NA
# ℹ 18,971 more rows
```

Tidying Text - What about all of Jane Austin’s Novels?

Jane Austin



Google
jane austen
Books
Added to My Books on Google Play
Write review
Result 1 of 100 in this book for jane austen - ‹ Previous Next › - View all
READ EBOOK
Get this book in print ▼
2271 Reviews
Write review
Pride and Prejudice
By Jane Austen
jane austen
Go
About this book
My library
My History
Books on Google Play
Terms of Service
PRIDE AND PREJUDICE
JANE AUSTEN

Data Available via: `janeaustenr`

Attached with row numbers (by book).

```
library(janeaustenr)
original_books <- austen_books() %>%
  group_by(book) %>%
  mutate(linenumber = row_number()) %>%
  ungroup()
head(original_books)
```

```
# A tibble: 6 × 3
  text                        book                linenumber
  <chr>                       <fct>                    <int>
1 "SENSE AND SENSIBILITY"     Sense & Sensibility          1
2 ""                          Sense & Sensibility          2
3 "by Jane Austen"            Sense & Sensibility          3
4 ""                          Sense & Sensibility          4
5 "(1811)"                    Sense & Sensibility          5
6 ""                          Sense & Sensibility          6
```

TidyText

tidytext: Text Mining and Analysis Using Tidy Data Principles in R

Authors
Julia Silge / David Robinson

Repository: Repository link »

Paper: PDF link »

Review: View review issue »

DOI: http://dx.doi.org/10.21105/joss.00037

Status badge: JOSS 10.21105/joss.00037

Cite this paper: doi2bib

Summary

The tidytext package (Silge, Robinson, and Hester 2016) is an R package (R Core Team 2016) for text mining using tidy data principles. As described by Hadley Wickham (Wickham 2014), tidy data has a specific structure:

- each variable is a column
- each observation is a row
- each type of observational unit is a table

Tidy data sets allow manipulation with a standard set of "tidy" tools, including popular packages such as dplyr (Wickham, Francois, and RStudio 2015), ggplot2 (Wickham, Chang, and RStudio 2016), and broom (Robinson et al. 2015). These tools do not yet, however, have the infrastructure to work fluently with text data and natural language processing tools. In developing this package, we provide functions and supporting data sets to allow conversion of text to and from tidy formats, and to switch seamlessly between tidy tools and existing text mining packages.

http://joss.theoj.org/papers/89fd1099620268fe0342ffdcdf66776f

A nice tutorial


STAT 545

Home FAQ Syllabus Topics People

Regular Expression in R

Gloria Li and Jenny Bryan

October 19, 2014

- String functions related to regular expression
- Regular expression syntax
  - Escape sequences
  - Quantifiers
  - Position of pattern within the string
  - Operators
  - Character classes
- General modes for patterns
  - Exercise
- Examples
  - Some more advanced string functions
  - Exercise
- Regular expression vs shell globbing
- Resources

In this tutorial, we will use the Gapminder data and file names in our class repository as examples to demonstrate using regular expression in R. First, let's start off by cloning the class repository, getting the list of file names with `list.files()`, and load the Gapminder dataset into R.

We will also need to use some functions from the stringr package. It provids a clean, modern alternative to common string operations, and is sometimes easier to remember and use than R basic string functions. If you have not done so yet, install the package.

```
install.packages("stringr")
```

http://stat545-ubc.github.io/block022_regular-expression.html

Large workhorse function: `unnest_tokens`

```
library(tidytext)
txt = c("These are words", "so are these", "this is running on")
sentence = c(1, 2, 3)
dat = tibble(txt, sentence)
unnest_tokens(dat, tok, txt)
```

```
# A tibble: 10 × 2
   sentence tok
      <dbl> <chr>
 1        1 these
 2        1 are
 3        1 words
 4        2 so
 5        2 are
 6        2 these
 7        3 this
 8        3 is
 9        3 running
10        3 on
```

What is tokenization?

“The process of segmenting running text into words and sentences.”

- Split on white space/punctuation
- Make lower case
- Keep contractions together
- Maybe put quoted words together (not in unnest_tokens)

One token per row

```
tidy_books <- original_books %>% unnest_tokens(word, text)
head(tidy_books)
```

```
# A tibble: 6 × 3
  book                linenumber word
  <fct>                    <int> <chr>
1 Sense & Sensibility          1 sense
2 Sense & Sensibility          1 and
3 Sense & Sensibility          1 sensibility
4 Sense & Sensibility          3 by
5 Sense & Sensibility          3 jane
6 Sense & Sensibility          3 austen
```

Stop words/words to filter

http://xpo6.com/list-of-english-stop-words/

Stop words/words to filter

```
tidy_books %>%
  group_by(word) %>%
  tally() %>%
  arrange(desc(n))
```

```
# A tibble: 14,520 × 2
   word      n
   <chr> <int>
 1 the   26351
 2 to    24044
 3 and   22515
 4 of    21178
 5 a     13408
 6 her   13055
 7 i     12006
 8 in    11217
 9 was   11204
10 it    10234
# ℹ 14,510 more rows
```

Stemming

Can use `wordStem` to reduce certain words to their primary stem (e.g. remove gerunds/tense):

```
library(SnowballC)
wordStem(c("running","fasted"))
```

```
[1] "run"  "fast"
```

Filtering with joins

```
head(stop_words)
```

```
# A tibble: 6 × 2
  word      lexicon
  <chr>     <chr>  
1 a         SMART  
2 a's       SMART  
3 able      SMART  
4 about     SMART  
5 above     SMART  
6 according SMART  
```

```
tidy_books = tidy_books %>% anti_join(stop_words, by = "word")
head(tidy_books)
```

```
# A tibble: 6 × 3
  book                linenumber word       
  <fct>                    <int> <chr>      
1 Sense & Sensibility          1 sense      
2 Sense & Sensibility          1 sensibility
3 Sense & Sensibility          3 jane       
4 Sense & Sensibility          3 austen     
5 Sense & Sensibility          5 1811       
6 Sense & Sensibility         10 chapter    
```

Example classification

```
library(tm);
```

```
Loading required package: NLP
```

```
data("AssociatedPress", package = "topicmodels")
AssociatedPress
```

```
<<DocumentTermMatrix (documents: 2246, terms: 10473)>>
Non-/sparse entries: 302031/23220327
Sparsity           : 99%
Maximal term length: 18
Weighting          : term frequency (tf)
```

```
class(AssociatedPress)
```

```
[1] "DocumentTermMatrix"    "simple_triplet_matrix"
```

```
head(tidy(AssociatedPress)) # generics::tidy
```

```
# A tibble: 6 × 3
  document term       count
     <int> <chr>      <dbl>
1        1 adding         1
2        1 adult          2
3        1 ago            1
4        1 alcohol        1
5        1 allegedly      1
6        1 allen          1
```

Compare frequencies: Jane Austin vs. the AP

```
comparison <- tidy(AssociatedPress) %>%
  count(word = term, name = "AP") %>%
  inner_join(count(tidy_books, word, name = "Austen")) %>%
  mutate(AP = AP / sum(AP),
         Austen = Austen / sum(Austen),
         diff = AP - Austen) %>%
  arrange(diff)
```

```
Joining with `by = join_by(word)`
```

```
head(comparison)
```

```
# A tibble: 6 × 4
  word            AP  Austen     diff
  <chr>        <dbl>   <dbl>    <dbl>
1 lady      0.000102 0.00580 -0.00569
2 time      0.00382  0.00948 -0.00566
3 sir       0.000120 0.00572 -0.00560
4 sister    0.000216 0.00516 -0.00494
5 elizabeth 0.000162 0.00487 -0.00471
6 friend    0.000288 0.00421 -0.00392
```

Bag of words

```
tidy_freq = tidy_books %>%
  dplyr::ungroup() %>%
  count(book, word, name = "count")
head(tidy_freq)
```

```
# A tibble: 6 × 3
  book                  word  count
  <fct>                 <chr> <int>
1 Sense & Sensibility 1         2
2 Sense & Sensibility 10        1
3 Sense & Sensibility 11        1
4 Sense & Sensibility 12        1
5 Sense & Sensibility 13        1
6 Sense & Sensibility 14        1
```

Bag of words

`nonum` removes any words that are all numeric (many ways of doing this):

```
nonum = tidy_freq %>%
  filter(is.na(as.numeric(word)))
```

```
Warning: There was 1 warning in `filter()`.
ℹ In argument: `is.na(as.numeric(word))`.
Caused by warning:
! NAs introduced by coercion
```

```
head(nonum)
```

```
# A tibble: 6 × 3
  book                word      count
  <fct>               <chr>     <int>
1 Sense & Sensibility 7000l         1
2 Sense & Sensibility abandoned     1
3 Sense & Sensibility abatement     1
4 Sense & Sensibility abbeyland     1
5 Sense & Sensibility abhor         1
6 Sense & Sensibility abhorred      2
```

Combine "bags"

```
tidy_ap = tidy(AssociatedPress) %>%
  rename(book = document,
         word = term,
         count = count)
dat = rbind(tidy_ap, tidy_freq)
head(dat)
```

```
# A tibble: 6 × 3
  book  word      count
  <chr> <chr>     <dbl>
1 1     adding        1
2 1     adult         2
3 1     ago           1
4 1     alcohol       1
5 1     allegedly     1
6 1     allen         1
```

Term-document matrices

Make a `DocuemntTermMatrix`/reshape the data:

```
dtm = dat %>% cast_dtm(document = book, term = word, value = count)
inspect(dtm[1:6,1:10])
```

```
<<DocumentTermMatrix (documents: 6, terms: 10)>>
Non-/sparse entries: 15/45
Sparsity           : 75%
Maximal term length: 10
Weighting          : term frequency (tf)
Sample             :
    Terms
Docs adding adult ago alcohol allegedly allen apparently appeared arrested
   1      1     2   1       1         1     1          2        1        1
   2      0     0   0       0         0     0          0        1        0
   3      0     0   1       0         0     0          0        1        0
   4      0     0   3       0         0     0          0        0        0
   5      0     0   0       0         0     0          0        0        0
   6      0     0   2       0         0     0          0        0        0
    Terms
Docs assault
   1       1
   2       0
   3       0
   4       0
   5       0
   6       0
```

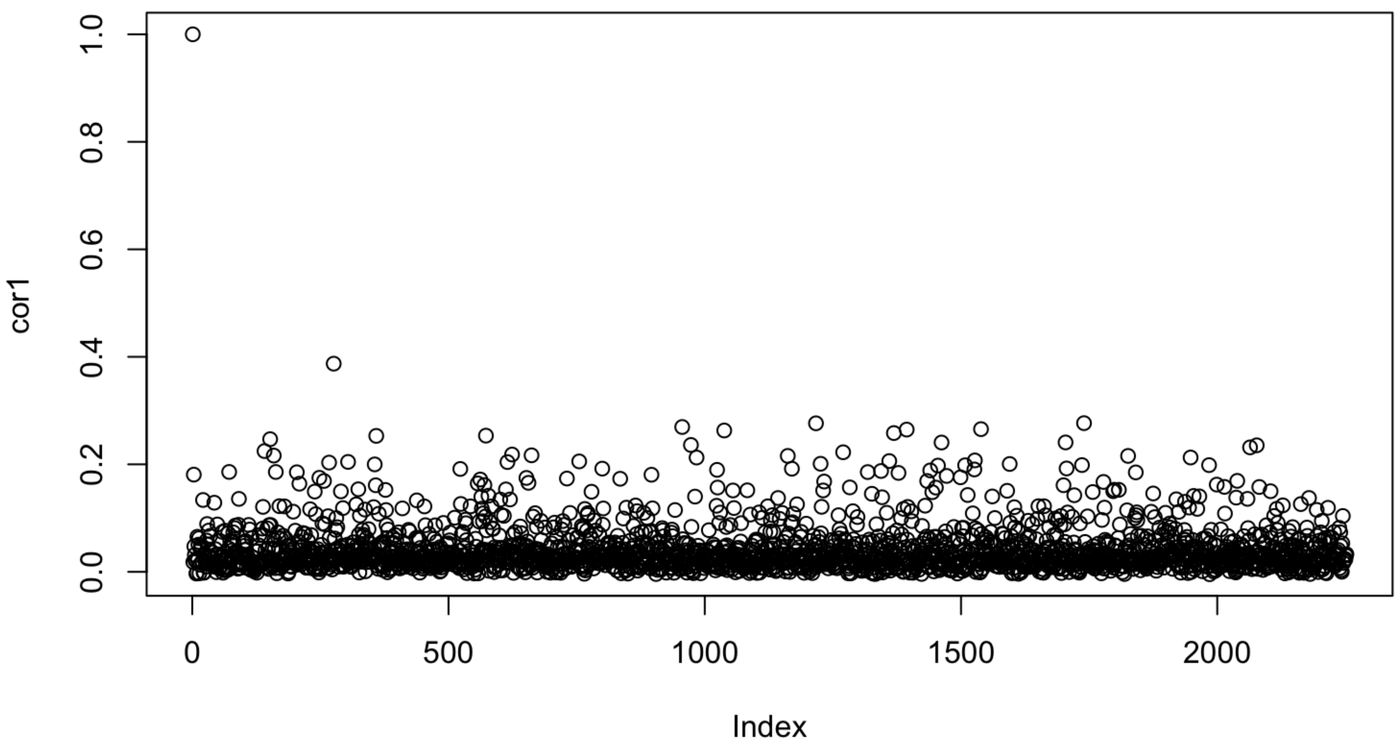
```
dtm = as.matrix(dtm)
dtm = dtm/rowSums(dtm)
```

Classify

Show the similarity (based on count correlations with first document):

```
cor1 = cor(dtm[1,], t(dtm))[1,]; print(cor1[1:5]);
```

```
         1          2          3          4          5 
1.00000000 0.01817799 0.18085240 0.04745425 0.03157564 
```

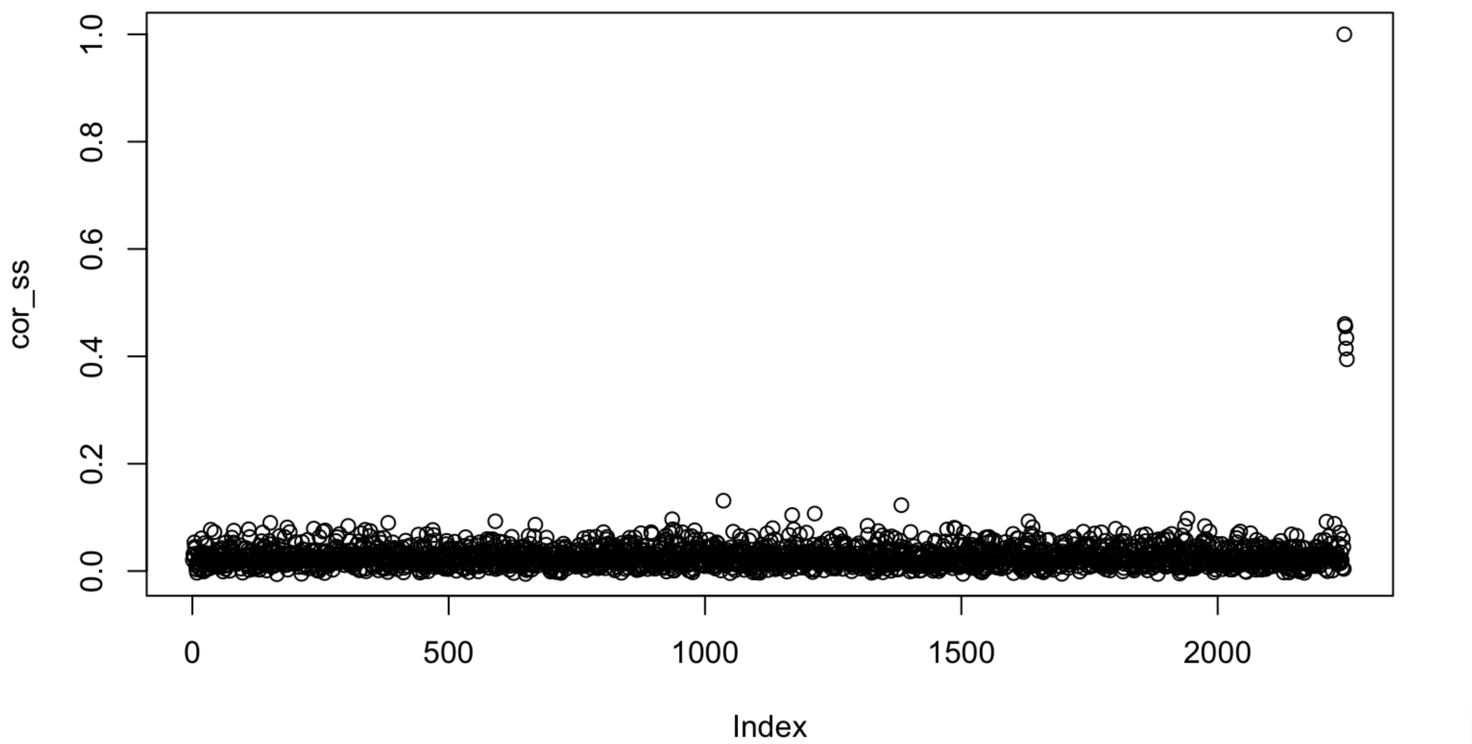


Classify

We see that there is a large clustering of Austen compared to AP:

```
cor_ss = cor(dtm["Sense & Sensibility",], t(dtm))[1,]; print(cor_ss[1:5]);
```

```
         1          2          3          4          5 
0.02056637 0.03157958 0.02608218 0.05342795 0.04504179 
```



Classify

The max similarity is not symmetrical (closest document/book to document 1 does not have document 1 as its closest document/book):

```
(index <- which.max(cor1[-1]))
```

```
276
275
```

```
cor_ss = cor(dtm[index,],t(dtm))[1,]
which.max(cor_ss[-index]) # not 1!
```

```
1126
1125
```

Sentiment analysis

"I hate this stupid class. But I love the instructor"

Sentiment analysis

"I hate this stupid class. But I love the instructor"

Sentiment analysis

"I hate this stupid class. But I love the instructor"

"Oh yeah, I totally love doing coding sessions"

Sentiments

```
bing <- tidytext::sentiments
head(bing)
```

```
# A tibble: 6 × 2
  word       sentiment
  <chr>      <chr>
1 2-faces    negative
2 abnormal   negative
3 abolish    negative
4 abominable negative
5 abominably negative
6 abominate  negative
```

```
(dupes <- bing %>% janitor::get_dupes(word))
```

```
# A tibble: 6 × 3
  word        dupe_count sentiment
  <chr>            <int> <chr>
1 envious              2 positive
2 envious              2 negative
3 enviously            2 positive
4 enviously            2 negative
5 enviousness          2 positive
6 enviousness          2 negative
```

Sentiments: A little Tidying

Let's remove those cases that it says these duplicates were positive

```
bing = bing %>%
  anti_join(dupes %>% filter(sentiment == "positive"))
```

```
Joining with `by = join_by(word, sentiment)`
```

```
anyDuplicated(bing$word)
```

```
[1] 0
```

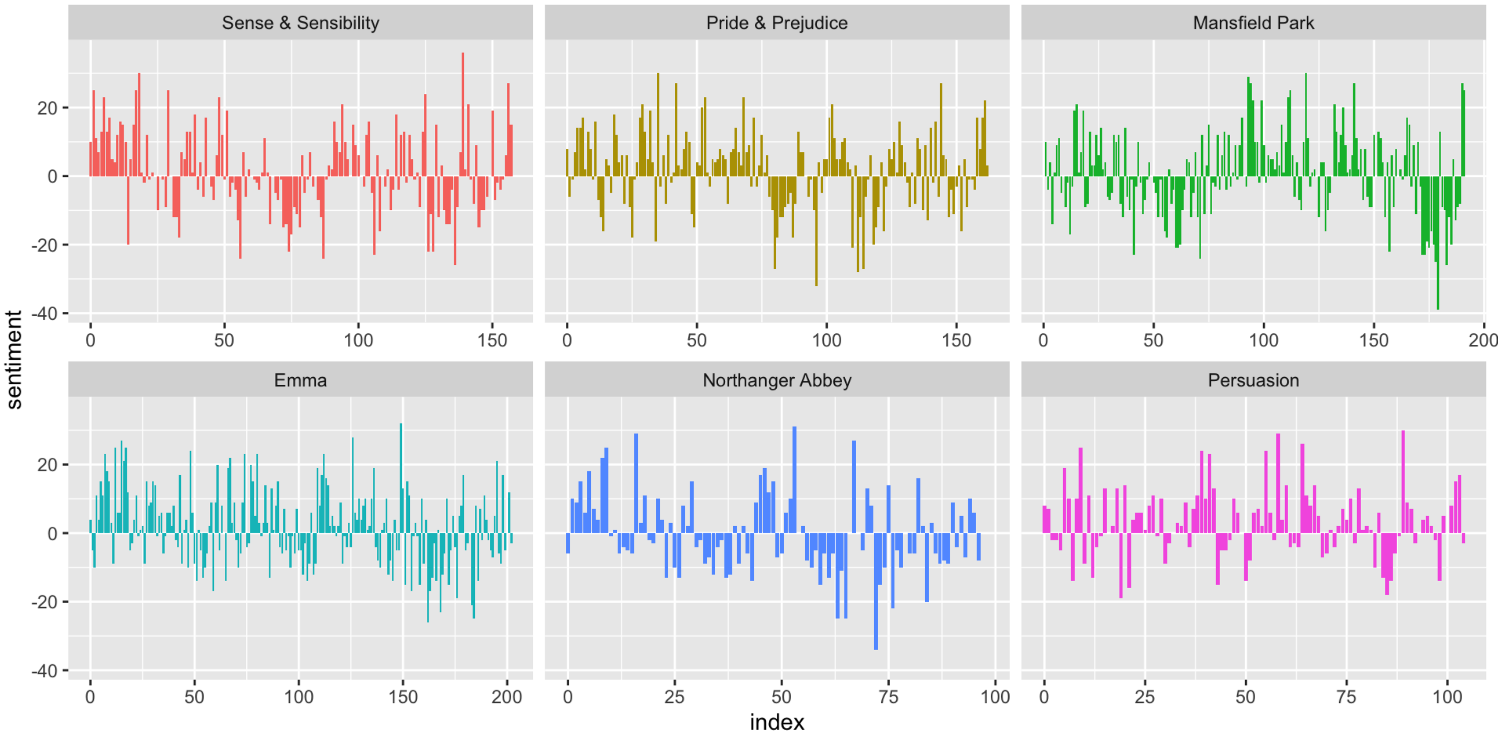
Assigning sentiments to words

```
janeaustensentiment <- tidy_books %>%
  inner_join(bing, by = join_by(word)) %>%
  count(book, index = linenumber %/% 80, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative)
head(janeaustensentiment)
```

```
# A tibble: 6 × 5
  book                index negative positive sentiment
  <fct>               <dbl>    <dbl>    <dbl>     <dbl>
1 Sense & Sensibility     0       16       26        10
2 Sense & Sensibility     1       19       44        25
3 Sense & Sensibility     2       12       23        11
4 Sense & Sensibility     3       15       22         7
5 Sense & Sensibility     4       16       29        13
6 Sense & Sensibility     5       16       39        23
```

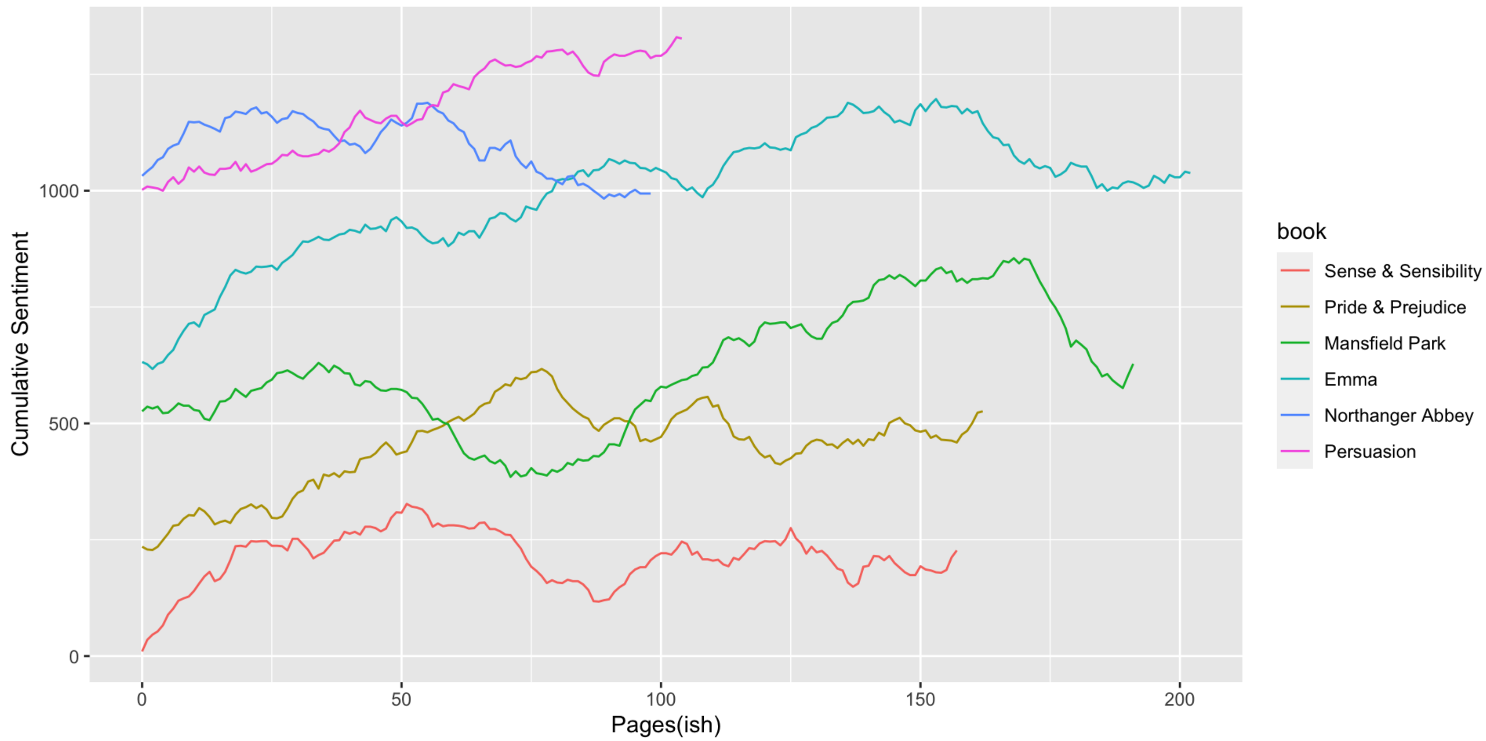
Plotting the sentiment trajectory (ggplot2 loaded)

```
ggplot(janeaustensentiment, aes(index, sentiment, fill = book)) +
  geom_bar(stat = "identity", show.legend = FALSE) +
  facet_wrap(~book, ncol = 3, scales = "free_x")
```



Plotting the cumulative sentiment

```
ggplot(janeaustensentiment, aes(index, cumsum(sentiment), colour = book)) +
  geom_line() + ylab("Cumulative Sentiment") + xlab("Pages(ish)")
```



Plotting the cumulative sentiment (normalized book length)

```
janeaustensentiment %>%
  group_by(book) %>%
  mutate(index = index/max(index)) %>%
  ggplot(aes(index, cumsum(sentiment), colour = book)) +
  geom_line()  + ylab("Cumulative Sentiment") + xlab("Percent Pages")
```