Introduction to basic Text Mining in R.

As published in Benchmarks RSS Matters, January 2014

`http://web3.unt.edu/benchmarks/issues/2014/01/rss-matters`

Jon Starkweather, PhD

Jon Starkweather, PhD
`jonathan.starkweather@unt.edu`
Consultant
Research and **S**tatistical **S**upport

`http://www.unt.edu`

`http://www.unt.edu/rss`

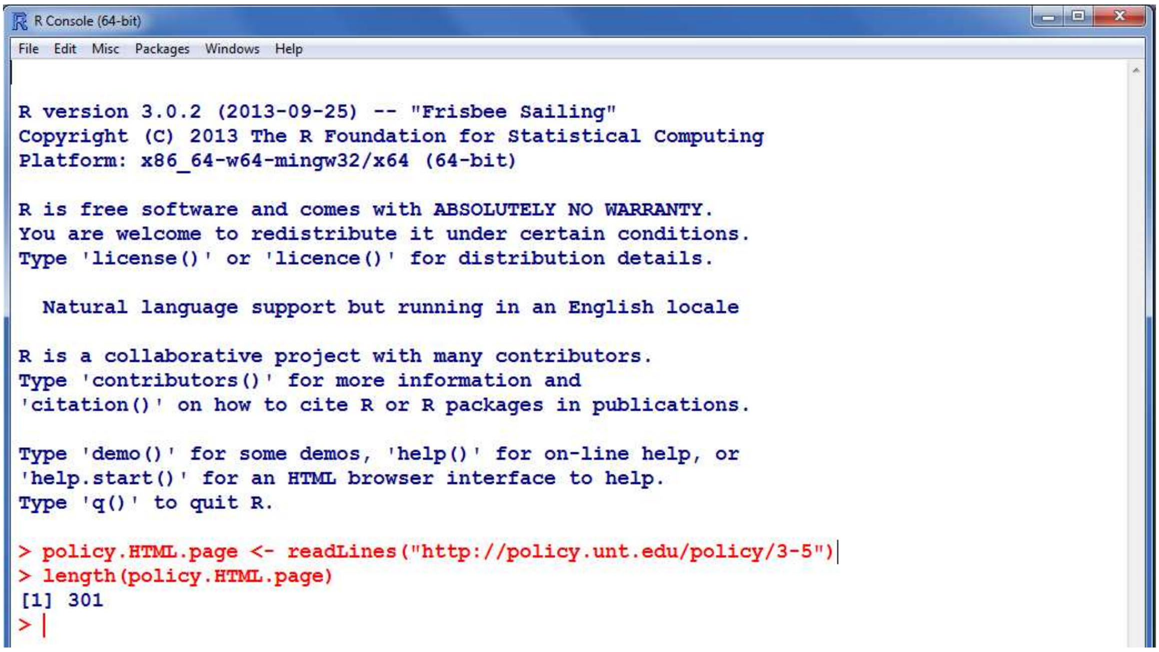
RSS hosts a number of “Short Courses”.
A list of them is available at:
`http://www.unt.edu/rss/Instructional.htm`

Those interested in learning more about R, or how to use it, can find information here:
`http://www.unt.edu/rss/class/Jon/R_SC`

Introduction to basic Text Mining in R.

This month, we turn our attention to *text mining*. Text mining refers to the process of parsing a selection or corpus of text in order to identify certain aspects, such as the most frequently occurring word or phrase. In this simple example, we will (of course) be using R[1] to collect a sample of text and conduct some rudimentary analysis of it. Keep in mind, this article simply provides a cursory introduction to some text mining functions.

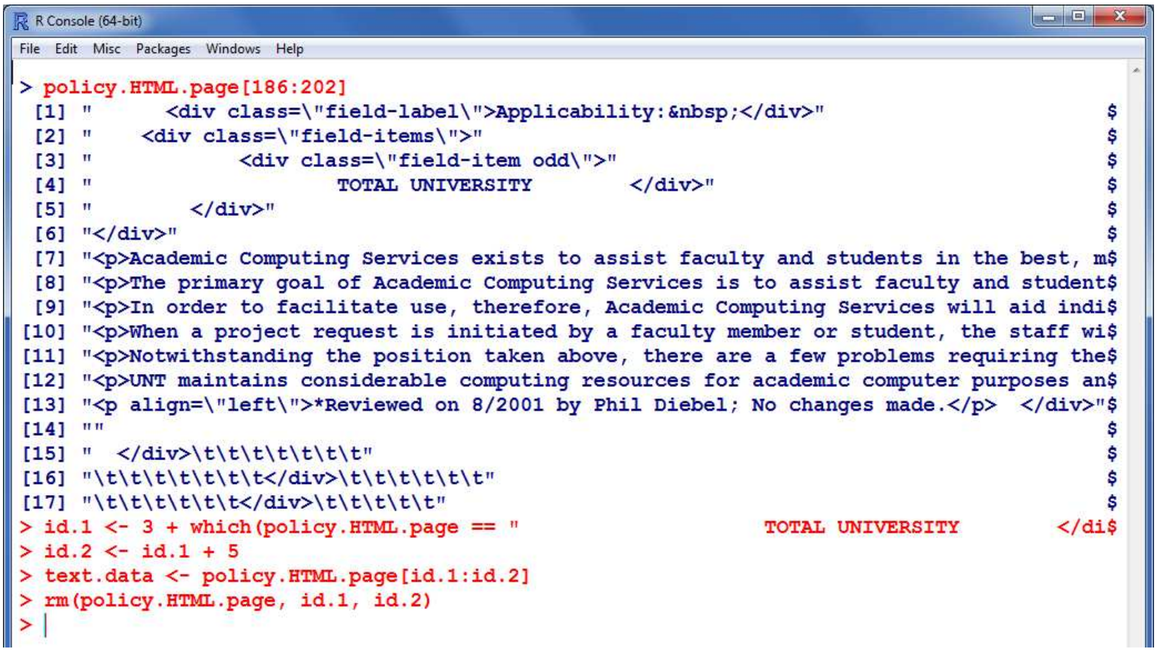
First, we need to retrieve or import some text. We will use the University of North Texas (UNT) policy which governs Research and Statistical Support (RSS) services; UNT policy 3 - 5 for this example. We can use the 'readLines' function available in the 'base' package to retrieve the policy from the UNT Policy web site. Notice this policy's HTML page is 305 lines long, which includes all the HTML formatting; not just the text of the policy.



```
R version 3.0.2 (2013-09-25) -- "Frisbee Sailing"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> policy.HTML.page <- readLines("http://policy.unt.edu/policy/3-5")
> length(policy.HTML.page)
[1] 301
>
```

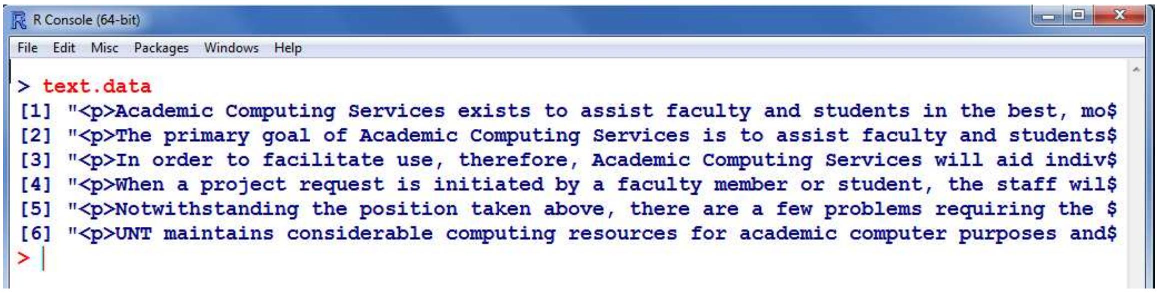
Next, we need to isolate the actual text of the policy's HTML page. This can take some investigating – using the head and tail functions or simply pasting the HTML page into a text editor will allow us to identify the line number(s) which contain the actual text of interest. Once identified, we can use a 'which' function to isolate or extract the lines we are interested in parsing. We notice below the actual text of the policy exists on lines 192 through 197, prefaced by the "Total University" header on line 189. We use the 'which' function to identify the line (189) with the header statement, then add 3 to it to arrive at line 192 (id.1; which identifies the first line of the policy). Then, we add a further 5 to that (192 + 5 = 197) to identify the last line of the policy (id.2). Then, we create a new object ('text.data') which contains only those lines which contain the text of the policy.

[1] http://cran.r-project.org/



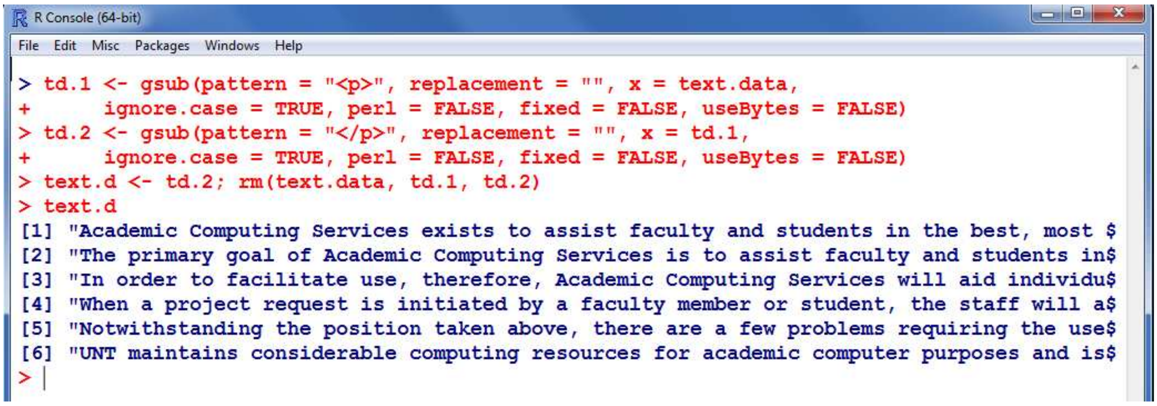
```
> policy.HTML.page[186:202]
 [1] "      <div class=\"field-label\">Applicability:&nbsp;</div>"                       $
 [2] "    <div class=\"field-items\">"                                                   $
 [3] "            <div class=\"field-item odd\">"                                        $
 [4] "                    TOTAL UNIVERSITY        </div>"                                $
 [5] "        </div>"                                                                    $
 [6] "</div>"                                                                            $
 [7] "<p>Academic Computing Services exists to assist faculty and students in the best, m$
 [8] "<p>The primary goal of Academic Computing Services is to assist faculty and student$
 [9] "<p>In order to facilitate use, therefore, Academic Computing Services will aid indi$
[10] "<p>When a project request is initiated by a faculty member or student, the staff wi$
[11] "<p>Notwithstanding the position taken above, there are a few problems requiring the$
[12] "<p>UNT maintains considerable computing resources for academic computer purposes an$
[13] "<p align=\"left\">*Reviewed on 8/2001 by Phil Diebel; No changes made.</p>  </div>"$
[14] ""                                                                                  $
[15] "  </div>\t\t\t\t\t\t\t\t"                                                          $
[16] "\t\t\t\t\t\t\t\t</div>\t\t\t\t\t\t"                                                $
[17] "\t\t\t\t\t\t\t</div>\t\t\t\t\t"                                                    $
> id.1 <- 3 + which(policy.HTML.page == "                    TOTAL UNIVERSITY        </di$
> id.2 <- id.1 + 5
> text.data <- policy.HTML.page[id.1:id.2]
> rm(policy.HTML.page, id.1, id.2)
> |
```

Now we are left with a vector object (text.data), which contains only the 6 lines of text of the policy (i.e. each paragraph of the policy has become a character string line of the vector).



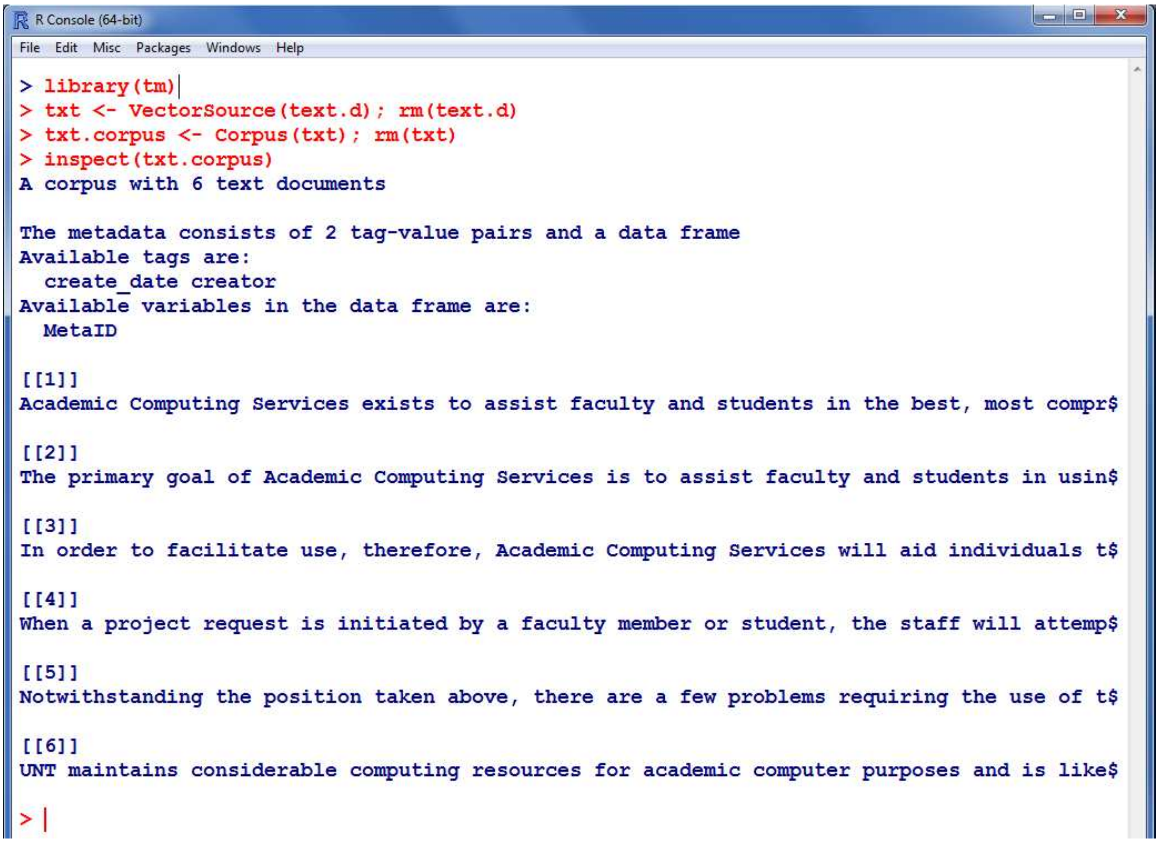
```
> text.data
[1] "<p>Academic Computing Services exists to assist faculty and students in the best, mo$
[2] "<p>The primary goal of Academic Computing Services is to assist faculty and students$
[3] "<p>In order to facilitate use, therefore, Academic Computing Services will aid indiv$
[4] "<p>When a project request is initiated by a faculty member or student, the staff wil$
[5] "<p>Notwithstanding the position taken above, there are a few problems requiring the $
[6] "<p>UNT maintains considerable computing resources for academic computer purposes and$
> |
```

Next, we need to remove the HTML tags (e.g. <p>) from each line of text. Generally multiple characters can be given in the 'pattern' argument within one implementation of 'gsub' function; but here we are using two implementations so that we can specifically remove each of the HTML (paragraph) tags while leaving in place all other instances of the letter 'p'. Notice, we are using the 'replacement' argument to eliminate any instance of the 'pattern' argument – using nothing between the quotation marks of the 'replacement' argument.



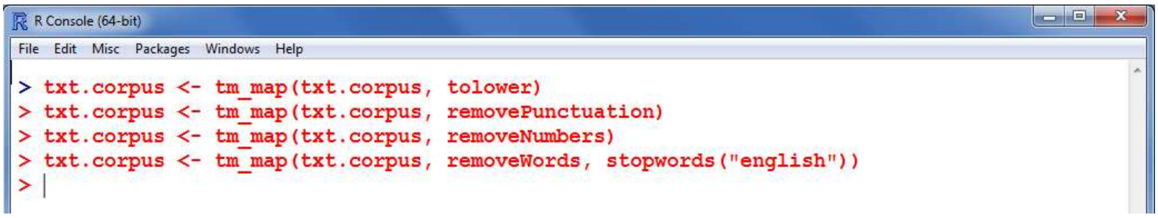
```
> td.1 <- gsub(pattern = "<p>", replacement = "", x = text.data,
+      ignore.case = TRUE, perl = FALSE, fixed = FALSE, useBytes = FALSE)
> td.2 <- gsub(pattern = "</p>", replacement = "", x = td.1,
+      ignore.case = TRUE, perl = FALSE, fixed = FALSE, useBytes = FALSE)
> text.d <- td.2; rm(text.data, td.1, td.2)
> text.d
[1] "Academic Computing Services exists to assist faculty and students in the best, most $
[2] "The primary goal of Academic Computing Services is to assist faculty and students in$
[3] "In order to facilitate use, therefore, Academic Computing Services will aid individu$
[4] "When a project request is initiated by a faculty member or student, the staff will a$
[5] "Notwithstanding the position taken above, there are a few problems requiring the use$
[6] "UNT maintains considerable computing resources for academic computer purposes and is$
> |
```

Now we can load the 'tm' package (Feinerer & Hornik, 2013) and convert our vector of character strings into a recognizable corpus of text using the 'VectorSource' function and the 'Corpus' function.



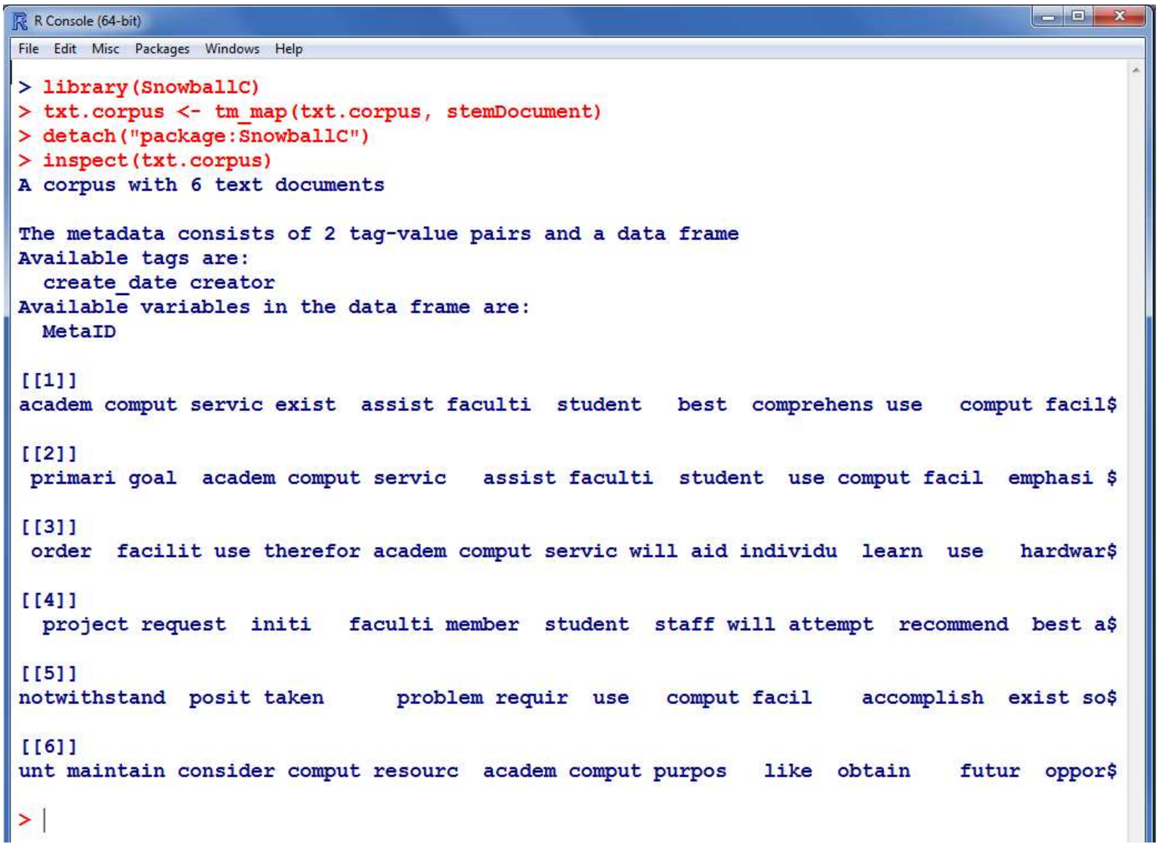
```
> library(tm)
> txt <- VectorSource(text.d); rm(text.d)
> txt.corpus <- Corpus(txt); rm(txt)
> inspect(txt.corpus)
A corpus with 6 text documents

The metadata consists of 2 tag-value pairs and a data frame
Available tags are:
  create_date creator
Available variables in the data frame are:
  MetaID

[[1]]
Academic Computing Services exists to assist faculty and students in the best, most compr$

[[2]]
The primary goal of Academic Computing Services is to assist faculty and students in usin$

[[3]]
In order to facilitate use, therefore, Academic Computing Services will aid individuals t$

[[4]]
When a project request is initiated by a faculty member or student, the staff will attemp$

[[5]]
Notwithstanding the position taken above, there are a few problems requiring the use of t$

[[6]]
UNT maintains considerable computing resources for academic computer purposes and is like$

> 
```

Next, we make some adjustments to the text; making everything lower case, removing punctuation, removing numbers, and removing common English stop words. The ‘tm_map’ function allows us to apply transformation functions to a corpus.



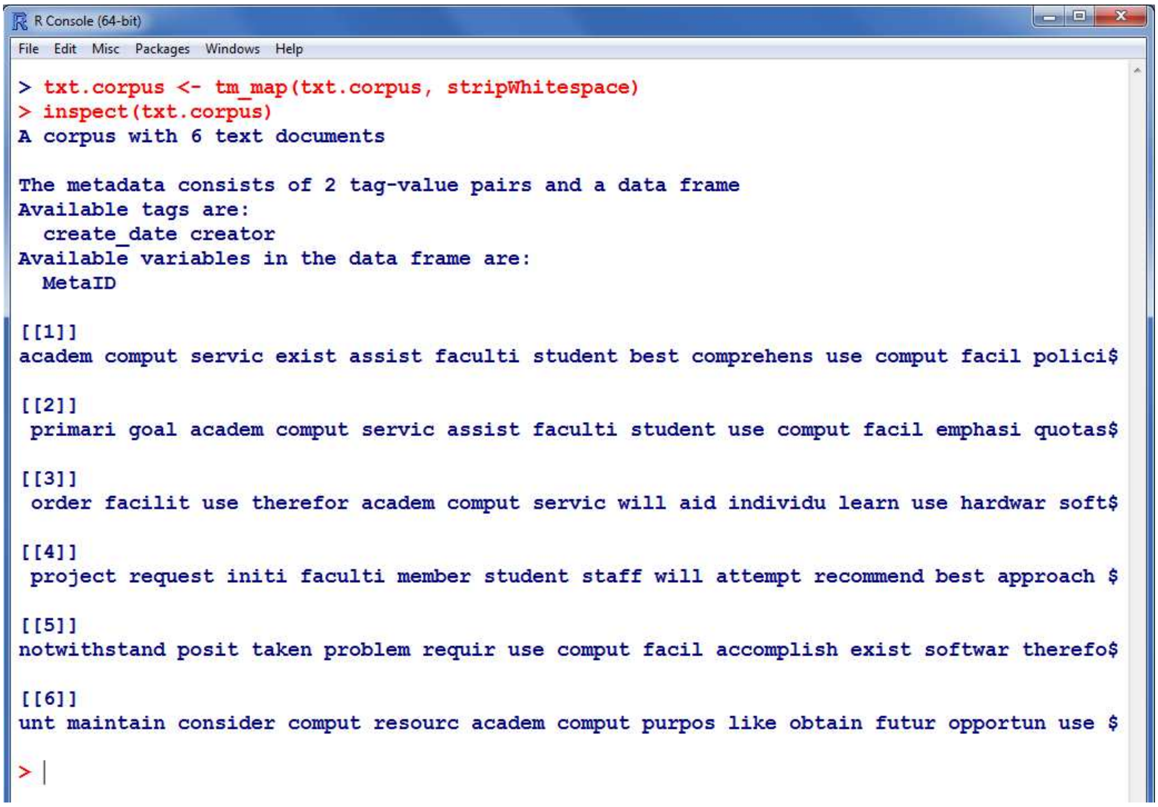
```
> txt.corpus <- tm_map(txt.corpus, tolower)
> txt.corpus <- tm_map(txt.corpus, removePunctuation)
> txt.corpus <- tm_map(txt.corpus, removeNumbers)
> txt.corpus <- tm_map(txt.corpus, removeWords, stopwords("english"))
> 
```

Next we perform stemming, which truncates words (e.g., “compute”, “computes” & “computing” all become “comput”). However, we need to load the ‘SnowballC’ package (Bouchet-Valat, 2013) which allows us to identify specific stem elements using the ‘tm_map’ function of the ‘tm’ package.



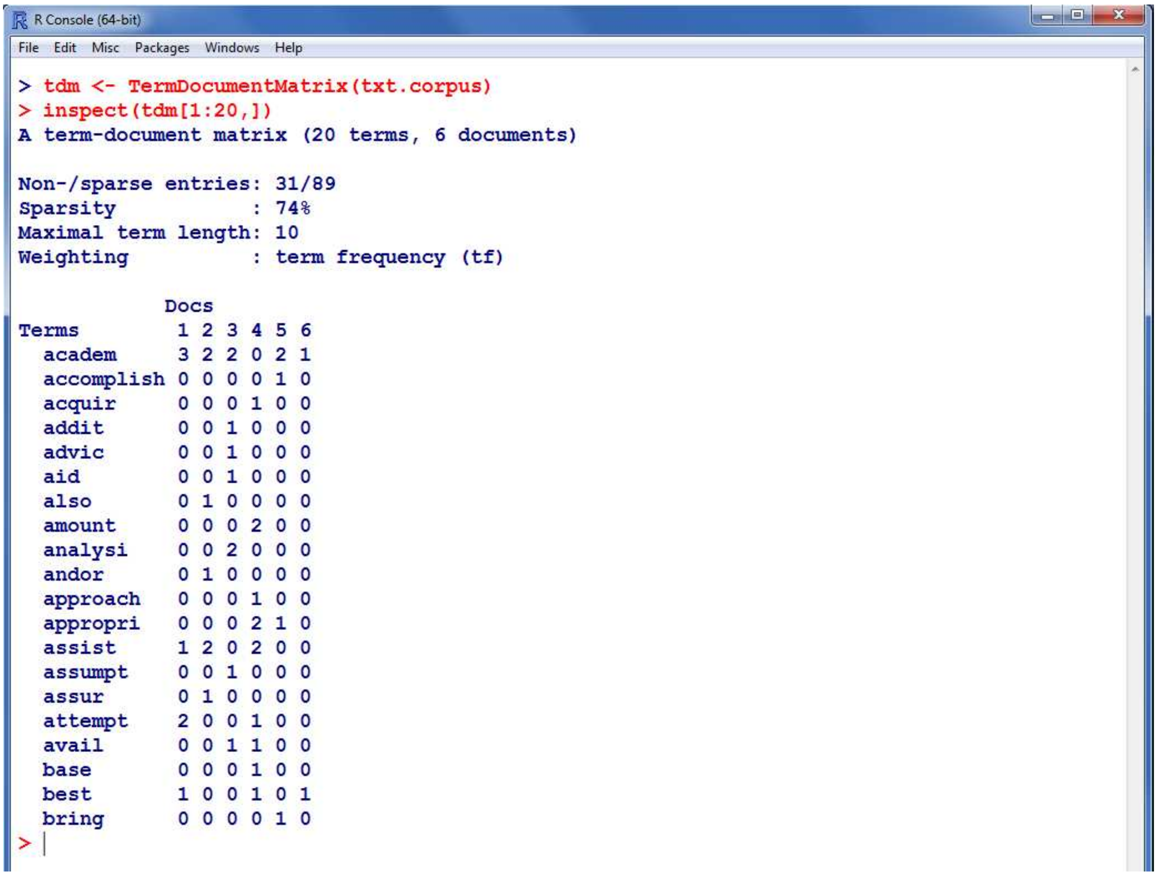
```
> library(SnowballC)
> txt.corpus <- tm_map(txt.corpus, stemDocument)
> detach("package:SnowballC")
> inspect(txt.corpus)
A corpus with 6 text documents

The metadata consists of 2 tag-value pairs and a data frame
Available tags are:
  create_date creator
Available variables in the data frame are:
  MetaID

[[1]]
academ comput servic exist  assist faculti  student   best  comprehens use   comput facil$

[[2]]
 primari goal  academ comput servic   assist faculti  student  use comput facil  emphasi $

[[3]]
 order  facilit use therefor academ comput servic will aid individu  learn  use   hardwar$

[[4]]
  project request  initi   faculti member  student  staff will attempt  recommend  best a$

[[5]]
notwithstand  posit taken      problem requir  use   comput facil    accomplish  exist so$

[[6]]
unt maintain consider comput resourc  academ comput purpos   like  obtain    futur  oppor$

> |
```

Next, we remove all the empty spaces generated by isolating the word stems in the previous step. We use the ‘stripWhitespace’ argument of the ‘tm_map’ function to accomplish this task.



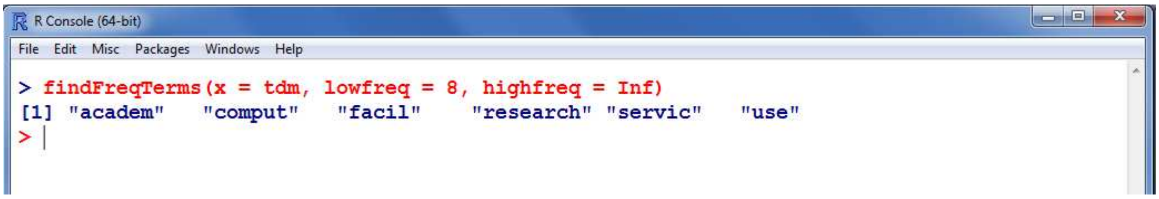
```
> txt.corpus <- tm_map(txt.corpus, stripWhitespace)
> inspect(txt.corpus)
A corpus with 6 text documents

The metadata consists of 2 tag-value pairs and a data frame
Available tags are:
  create_date creator
Available variables in the data frame are:
  MetaID

[[1]]
academ comput servic exist assist faculti student best comprehens use comput facil polici$

[[2]]
 primari goal academ comput servic assist faculti student use comput facil emphasi quotas$

[[3]]
 order facilit use therefor academ comput servic will aid individu learn use hardwar soft$

[[4]]
 project request initi faculti member student staff will attempt recommend best approach $

[[5]]
notwithstand posit taken problem requir use comput facil accomplish exist softwar therefo$

[[6]]
unt maintain consider comput resourc academ comput purpos like obtain futur opportun use $

> |
```

Now we can actually begin to analyze the text. First, we create something called a Term Document Matrix (TDM) which is a matrix of frequency counts for each word used in the corpus. Below we only show the first 20 words and their frequencies in each document (i.e. for us, each ‘document’ is a paragraph in the original policy).



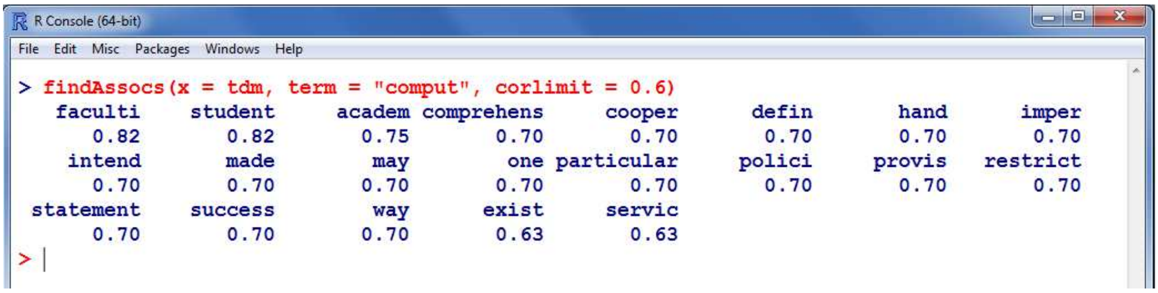
```
> tdm <- TermDocumentMatrix(txt.corpus)
> inspect(tdm[1:20,])
A term-document matrix (20 terms, 6 documents)

Non-/sparse entries: 31/89
Sparsity           : 74%
Maximal term length: 10
Weighting          : term frequency (tf)

            Docs
Terms         1 2 3 4 5 6
  academ      3 2 2 0 2 1
  accomplish  0 0 0 0 1 0
  acquir      0 0 0 1 0 0
  addit       0 0 1 0 0 0
  advic       0 0 1 0 0 0
  aid         0 0 1 0 0 0
  also        0 1 0 0 0 0
  amount      0 0 0 2 0 0
  analysi     0 0 2 0 0 0
  andor       0 1 0 0 0 0
  approach    0 0 0 1 0 0
  appropri    0 0 0 2 1 0
  assist      1 2 0 2 0 0
  assumpt     0 0 1 0 0 0
  assur       0 1 0 0 0 0
  attempt     2 0 0 1 0 0
  avail       0 0 1 1 0 0
  base        0 0 0 1 0 0
  best        1 0 0 1 0 1
  bring       0 0 0 0 1 0
> |
```

Next, we can begin to explore the TDM, using the ‘findFreqTerms’ function, to find which words were used most. Below we specify that we want term / word stems which were used 8 or more times (in all documents / paragraphs).



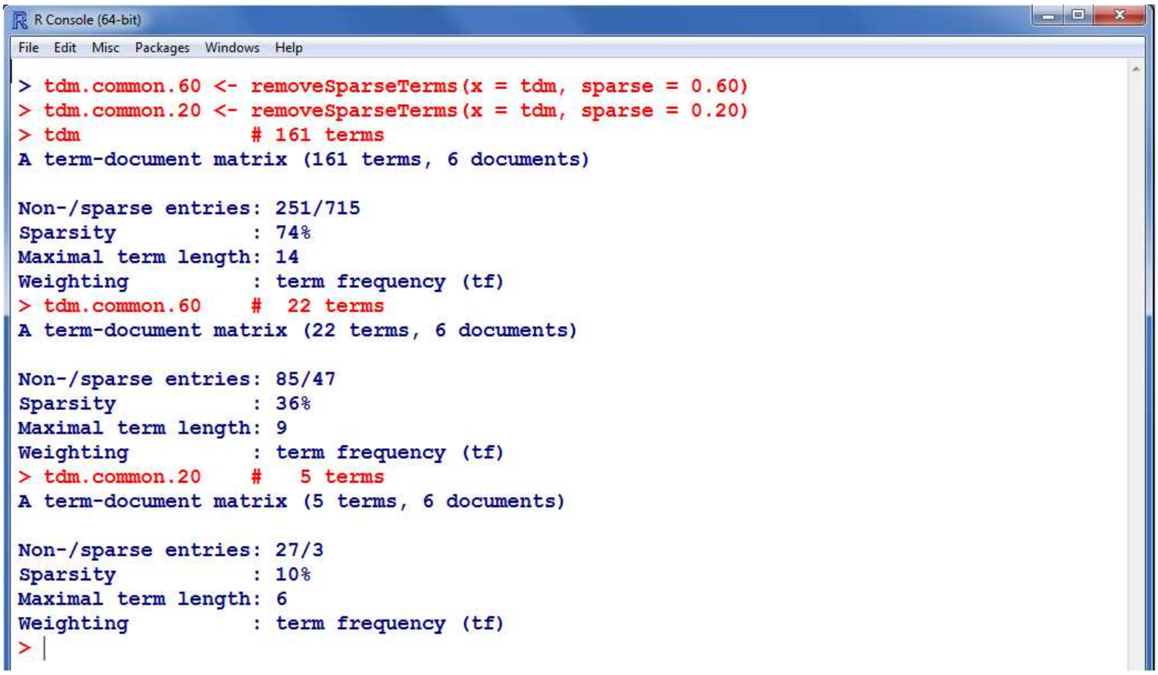
```
> findFreqTerms(x = tdm, lowfreq = 8, highfreq = Inf)
[1] "academ"   "comput"   "facil"    "research" "servic"   "use"
> |
```

Next, we can use the ‘findAssocs’ function to find words which associate together. Here, we are specifying the TDM to use, the term we want to find associates for, and the lowest acceptable correlation limit with that term. This returns a vector of terms which are associated with ‘comput’ at r = 0.60 or more (correlation) – and reports each association in descending order.



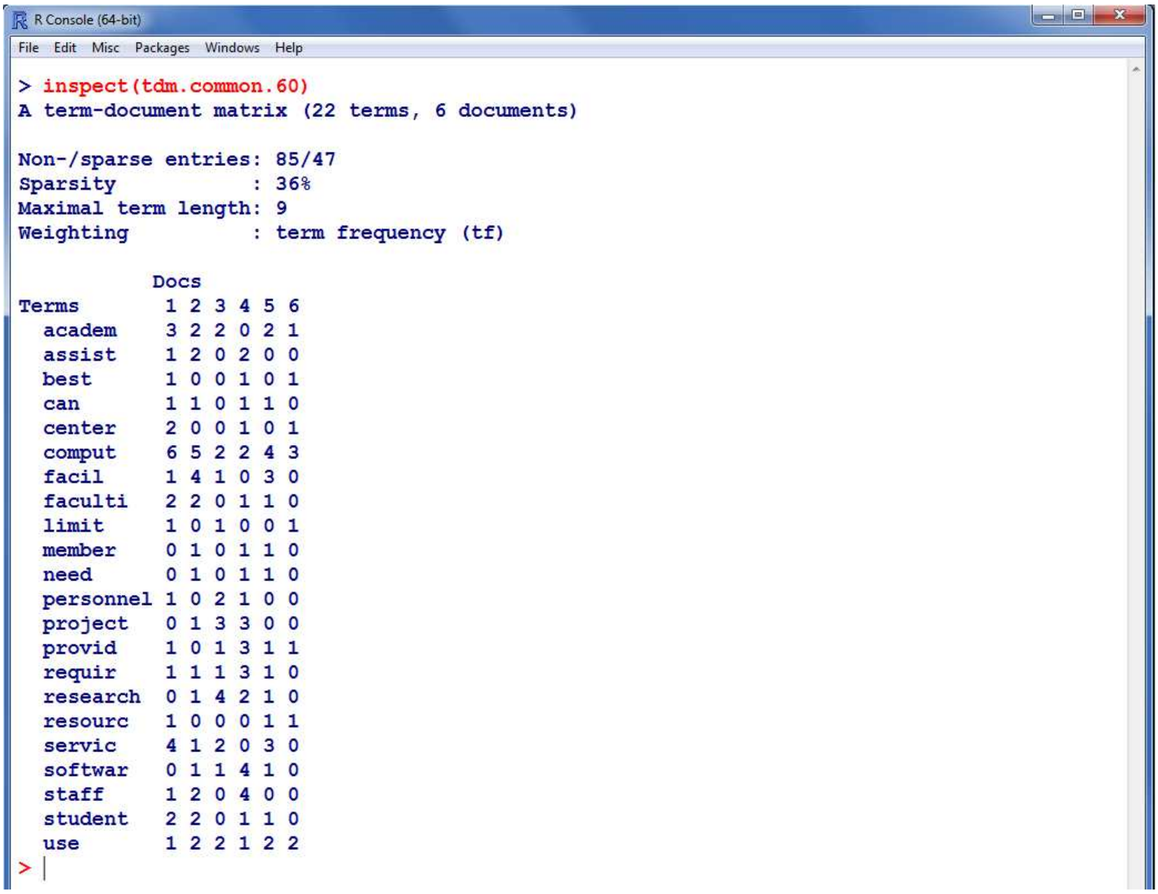
```
> findAssocs(x = tdm, term = "comput", corlimit = 0.6)
   faculti     student      academ comprehens      cooper       defin        hand       imper
      0.82        0.82        0.75       0.70        0.70        0.70        0.70        0.70
    intend        made         may        one particular      polici      provis    restrict
      0.70        0.70        0.70       0.70        0.70        0.70        0.70        0.70
 statement     success         way      exist      servic
      0.70        0.70        0.70       0.63        0.63
> |
```

If desired, terms which occur very infrequently (i.e. sparse terms) can be removed; leaving only the 'common' terms. Below, the 'sparse' argument refers to the MAXIMUM sparse-ness allowed for a term to be in the returned matrix; in other words, the larger the percentage, the more terms will be retained (the smaller the percentage, the fewer [but more common] terms will be retained).



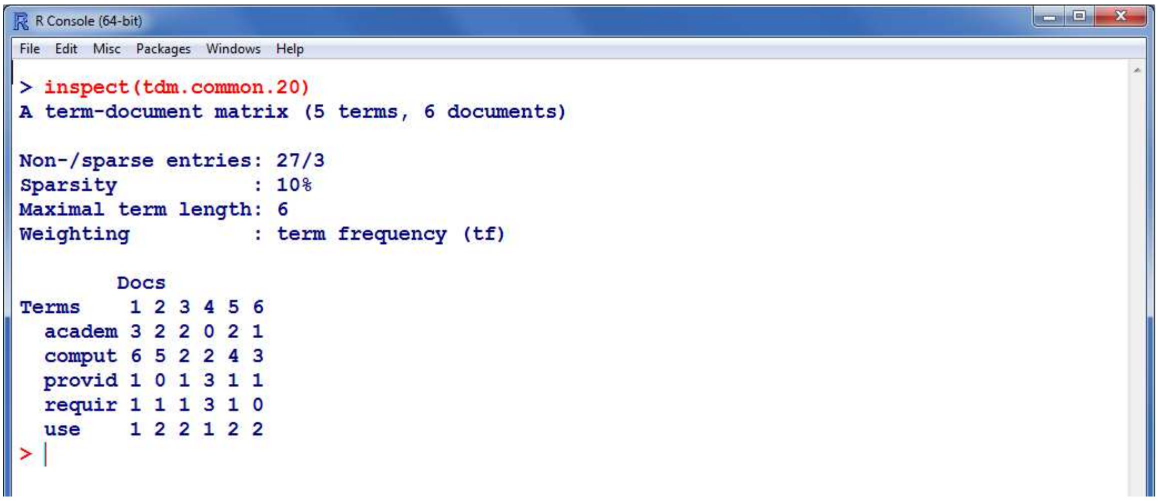
```
> tdm.common.60 <- removeSparseTerms(x = tdm, sparse = 0.60)
> tdm.common.20 <- removeSparseTerms(x = tdm, sparse = 0.20)
> tdm             # 161 terms
A term-document matrix (161 terms, 6 documents)

Non-/sparse entries: 251/715
Sparsity           : 74%
Maximal term length: 14
Weighting          : term frequency (tf)
> tdm.common.60    #  22 terms
A term-document matrix (22 terms, 6 documents)

Non-/sparse entries: 85/47
Sparsity           : 36%
Maximal term length: 9
Weighting          : term frequency (tf)
> tdm.common.20    #   5 terms
A term-document matrix (5 terms, 6 documents)

Non-/sparse entries: 27/3
Sparsity           : 10%
Maximal term length: 6
Weighting          : term frequency (tf)
> |
```

We can review the terms returned from a specific sparse-ness by using the 'inspect' function with the TDMs containing those specific sparse-ness rates (i.e. the terms retained at specific spare-ness levels). Below we see the 22 terms returned when sparse-ness is set to 0.60.



```
> inspect(tdm.common.60)
A term-document matrix (22 terms, 6 documents)

Non-/sparse entries: 85/47
Sparsity           : 36%
Maximal term length: 9
Weighting          : term frequency (tf)

           Docs
Terms        1 2 3 4 5 6
  academ     3 2 2 0 2 1
  assist     1 2 0 2 0 0
  best       1 0 0 1 0 1
  can        1 1 0 1 1 0
  center     2 0 0 1 0 1
  comput     6 5 2 2 4 3
  facil      1 4 1 0 3 0
  faculti    2 2 0 1 1 0
  limit      1 0 1 0 0 1
  member     0 1 0 1 1 0
  need       0 1 0 1 1 0
  personnel 1 0 2 1 0 0
  project    0 1 3 3 0 0
  provid     1 0 1 3 1 1
  requir     1 1 1 3 1 0
  research   0 1 4 2 1 0
  resourc    1 0 0 0 1 1
  servic     4 1 2 0 3 0
  softwar    0 1 1 4 1 0
  staff      1 2 0 4 0 0
  student    2 2 0 1 1 0
  use        1 2 2 1 2 2
> |
```

Next, we see the 5 terms returned when sparse-ness is set to 0.20 – fewer terms which occur more frequently (than above).



```
> inspect(tdm.common.20)
A term-document matrix (5 terms, 6 documents)

Non-/sparse entries: 27/3
Sparsity           : 10%
Maximal term length: 6
Weighting          : term frequency (tf)

        Docs
Terms    1 2 3 4 5 6
  academ 3 2 2 0 2 1
  comput 6 5 2 2 4 3
  provid 1 0 1 3 1 1
  requir 1 1 1 3 1 0
  use    1 2 2 1 2 2
> |
```

Conclusions

As stated in the introduction to this article, the above functions provide only a cursory introduction to importing some text and parsing it. Those seeking more information may want to consider taking a look at the ‘Natural Language Processing’ Task View at CRAN (Fridolin, 2013; link provided below). The task view provides information on a number of packages and functions available for processing textual data, including an R-Commander plugin which new R users are likely to find easier to use (at first).

For more information on what R can do, please visit the Research and Statistical Support Do-It-Yourself Introduction to R[2] course website. An Adobe.pdf version of this article can be found here[3].

Until next time; *"no twerking while working!"*

References & Resources

Bouchet-Valat, M. (2013). Package SnowballC. Documentation available at:
http://cran.r-project.org/web/packages/SnowballC/index.html

Feinerer, I., & Hornik, K. (2013). Package tm. Documentation available at:
http://cran.r-project.org/web/packages/tm/index.html

Fridolin Wild, F. (2013). Natural Language Processing. A CRAN Task View, located at:
http://cran.r-project.org/web/views/NaturalLanguageProcessing.html

Tutorial on text mining (author name not stated on page):
http://www.exegetic.biz/blog/2013/09/text-mining-the-complete-works-of-will

This article was last updated on January 6, 2014.

This document was created using LaTeX

[2] http://www.unt.edu/rss/class/Jon/R_SC/

[3] http://www.unt.edu/rss/rssmattersindex.htm