Paper 253-2011

WHAT WERE WE TALKING ABOUT AT THOSE SAS® CONFERENCES, OR LET'S MAKE SOME TAG CLOUDS

Chang Y. Chung, Princeton University, Princeton, NJ
John King, Ouachita Clinical Data Services, Mount Ida, AR

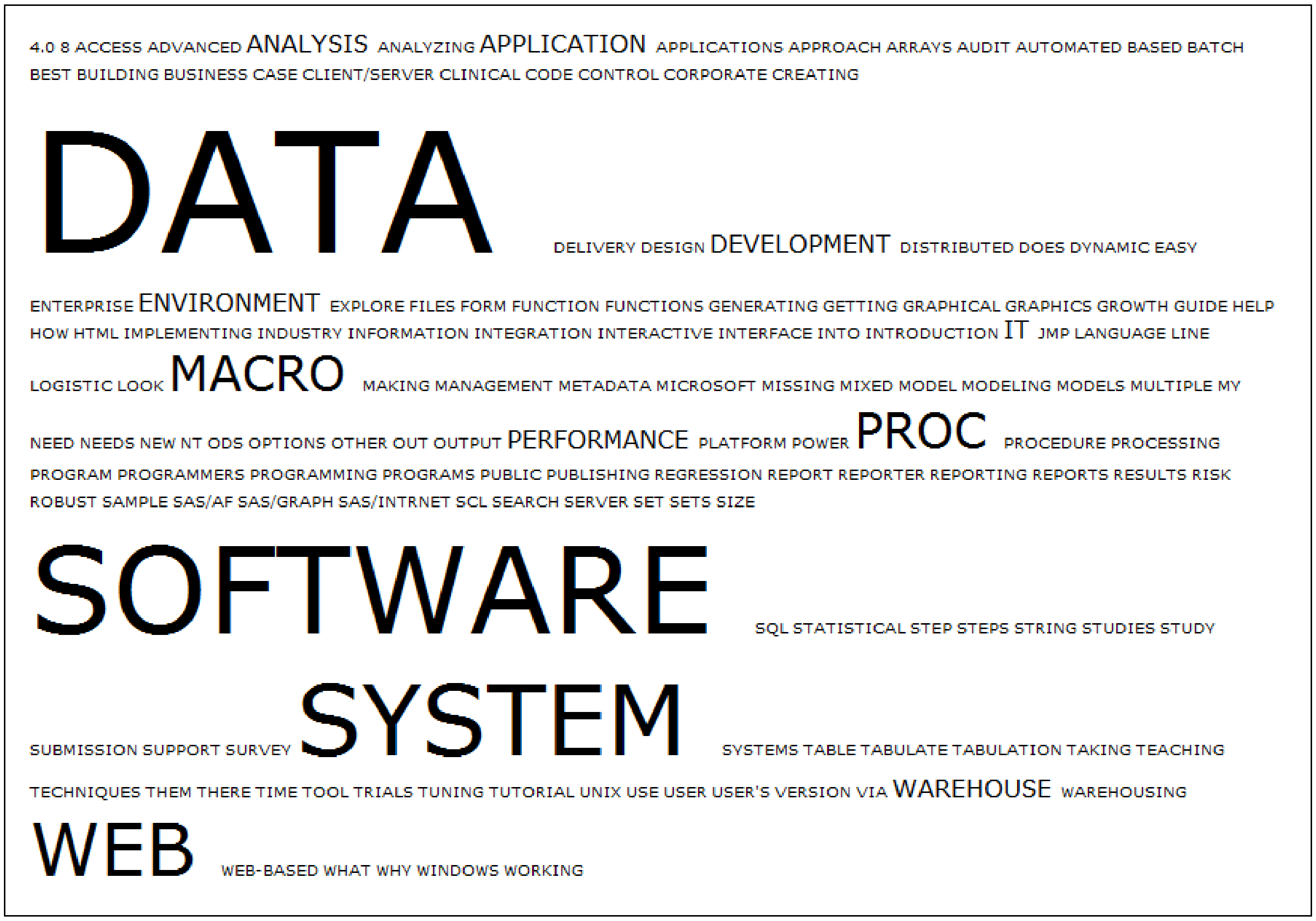
ABSTRACT

A tag cloud is a bunch of tags or words displayed nicely. More popular or important words are highlighted using different colors or font sizes. Popularized by Web 2.0 web sites, tag clouds can be informative and useful. This paper demonstrates making tag clouds using nothing but Base SAS®. The keywords come from the titles of all papers presented at two international SAS conferences: the 25th SAS Users Group International Conference in 2000 (SUGI 25) and the SAS Global Forum 2010.

A PICTURE IS WORTH A THOUSAND WORDS

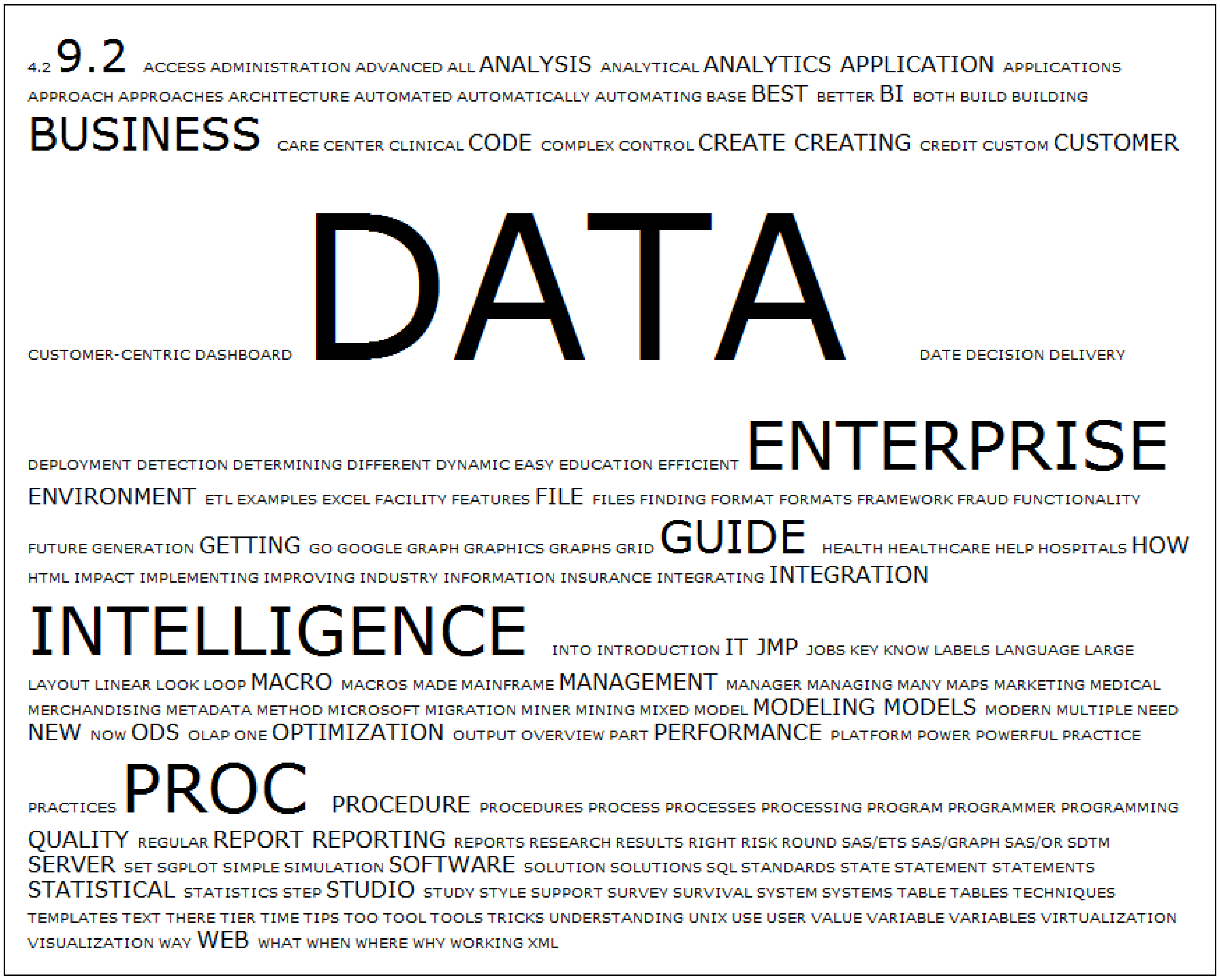
Then are two pictures worth two thousand words? A deep question! In any case, here they are. First from the 25th SAS Users Group International Conference (SUGI 25) held in Indianapolis, IN, April 9-12, 2000.

Figure 1 Popular Title Keywords at SUGI 25



And here is another one for the recent SAS Global Forum 2010 held in Seattle, WA, April 11-14, 2010.

Figure 2 Popular Title Keywords at SGF 2010



We have been talking about DATA for sure, but it seems that our conversation has not always been about the same things.

The full Base SAS source code that produced Figure 2 above is found in Appendix A. The code represents an end-to-end process of: web scraping the conference titles; creating a lookup table of ignored words (also known as "stop words"); chopping down the title into keywords; getting raw frequencies and mapping them into a number of levels; and writing out an HTML page with a tag cloud of conference paper title keywords.

In the rest of this paper, we explain the SAS code section by section. The code is written and submitted in an interactive Display Manager (DM) session on locally installed Windows (32bit) SAS 9.2 (TS1M0).

WEB SCRAPING

The titles of the conference papers are “harvested” from Lex Jansen’s wonderful SAS conference paper site (http://lexjansen.com/). In Lex’s web pages, each paper title is written in a single line of HTML with distinct id properties, which makes extracting the title easy.[1]

```
/*-- read sgf2010 page from lex^s web site ---------------------------------*/
%let amp = %nrstr(&);
%let param = s=sugi&amp.c=sugi&amp.x=;
%let lex = http://lexjansen.com/cgi-bin/xsl_transform.php?&param;

filename sgf2010 url "&lex.sgf2010" encoding="utf-8";                /* ① */
data titles;
   infile sgf2010 lrecl=10000 truncover;
   input line $10000.;

   sgf = prxmatch("|sgf2010\.|", line);                              /* ② */
   hilite = prxmatch("|hilite|", line);
   if sgf and not hilite;

   id = substr(substr(line,sgf), 9, 8);

   title = prxchange("s|.*>([^>]+)</a><br />.*$|$1|", 1, line);      /* ③ */
   keep id title;
run;

filename sgf2010 clear;                                              /* ④ */
```

The above data step generates a data set, TITLES, by reading Lex’s web page using FILENAME URL (①). If your computer is located behind a company firewall, then you may have to go through a proxy server. In this case, check the FILENAME URL PROXY= and related options. The ENCODING= option is necessary because the site is encoded in UTF-8. The ENCODING= option does a reasonable job translating the input characters to the local encoding.

We select input lines that: (1) have the string “sgf2010.”, including the dot (.); but (2) does not have the word “hilite”. This results in selecting only the input lines with paper titles. (②)

Once we know that a given input line contains a title, we extract it with a powerful Perl regular expression (③).[2] The regular expression matches certain patterns of html tags and the end of line ($). Once we have all the titles, we let go of the file handle (④).

In the recent version of SAS BASE, PROC HTTP makes inputting over Hypertext Transfer Protocol (HTTP) easier. The procedure provides various options that enable us to manipulate the header information, for instance, thus it opens up the possibility of making HTTP requests with the POST method, in addition to GET method.

It is possible to write a data step to scrape multiple web pages one after another. One way is to take advantage of INFILE statement’s FILEVAR= option as demonstrated in the code in the Appendix B.

Please be cautious not to impose too much on the web site’s owner and servers, though, since rapid multiple requests generated by running a data step like this can be a severe burden to a web server. We recommend downloading html files only once and then process them locally.

---

[1] We considered two alternative data sources. The SAS Conference Proceedings in Adobe Portable Document Format (PDF) are readily available. But in general, these PDF files are for printing and not for further processing. SAS Institute user support site also has conference proceeding pages. Unfortunately, these are coded with irregular HTML, which makes web scraping difficult, if not impossible.

[2] We also extract the paper id as well.

INFORMAT FOR TABLE LOOKUP

Not all the words in the title are equally worthy. Some words like “A” and “AN” convey little information. The word, “SAS”, appears so many times that if we allow it, the frequency distribution becomes severely skewed. To check and drop these “ignore” words, we need a convenient way of looking them upon a list. SAS offers many different ways to implement table lookups. When you have a moderate amount of items in a lookup table (say less than three thousand), using (IN)FORMAT is a fast and easy solution.

```
/*-- for table lookup -------------------------------------------------------*/
proc format;

  invalue ignored                                                       /* ① */
   "SAS", "THE", "AND", "TO", "A", "IN", "USING",
   "FOR", "OF", "WITH", "YOUR", "AN", "ON",
   "BY", "YOU", "FROM", "IS", "AT", "OR", "DO", "GET",
   "NOT", "MORE", "AS", "I", "BE", "US", "VERY", "WE",
   "YES", "YET", "YOU'RE", "THAT", "SO", "WHAT'S",
   "WORKSHOP", "OUR", "IT'S", ".", "IT’S", "-",
   "THAT'S", "THEIR", "SUCH", "CAN", "BETWEEN",
   "WHICH", "I’VE", "ITSELF" = 1
   other = 0;
run;
```

We write a numeric INFORMAT, ignored, which generates 1 if an input word is one of the ignore words or 0 otherwise (①). There are many alternative table lookup techniques available, including hash tables and data step MERGE or PROC SQL. Since we have a relatively small number of words to look up, creating an INFORMAT using PROC FORMAT provides a quick and convenient table lookup mechanism.

FROM TITLE INTO KEYWORDS

The next DATA step, WORDS, takes titles and splits each word out into its own observation.

```
/*-- split titles into words skipping ignore words --------------------------*/
data words;
   set titles;
   length word $20;
   i = 1;

   link getWord;                                                        /* ① */
   do while (not missing(word));

      if not input(word, ignored.) then output;                         /* ② */
      i + 1;

      link getWord;                                                     /* ③ */
   end;
   keep id word;
   return;

   getWord:

      word = upcase(scan(title, i, " :,()?®™-!=”“;""", "r"));           /* ④ */
   return;
run;
```

This is done using the powerful SCAN function (④). This function allows custom delimiters, which makes it easy for us to handle punctuation marks like colons (:), commas (,), and others. Notice that we also do UPCASE’ing to *normalize* words, so that same words can be recognized same, regardless of their casing differences.

When we first wrote the data step, the SCAN function was called in two different places in the code (① and ③), making it difficult to maintain. Later, we re-factored them into a single LINK destination. Refactoring repeated code into LINK’s shortens the length of the data step, and often makes it easier to read.

Each word in the title is checked to see if it is among the ignore words by calling the INPUT function with our INFORMAT (②). If the input function returns 1 (i.e., should be ignored) then we skip OUTPUT’ing the word.

COUNTING AND CATEGORIZING

Once we have words data set, then we use PROC FREQ to count the words (①).

```
/*-- get frequencies and scale -----------------------------------------------*/

proc freq data=words;                                               /* ① */

  tables word/out=freqs(keep=word count where=(count>=3));          /* ② */
run;


%let maxSize = 10;                                                  /* ③ */
data sizes;
  retain maxCount 0;
  do until (end1);
    set freqs(keep=count) end=end1;

    maxCount = max(count, maxCount);                                /* ④ */
  end;
  do until (end2);
    set freqs end=end2;

    size =  ceil(&maxSize * count / maxCount);                      /* ⑤ */
    output;
  end;
run;
```

Since there are so many words with very low frequencies, we arbitrarily decide to process only those words with three or more occurrences. This is easily done by applying the WHERE= DATASET OPTIONS to the OUT= dataset in the TABLES statement.

The *raw* frequency then may range from three to some maximum number. To size the keywords, however, we want to have a limited number of frequency levels. We arbitrarily decide that ten different levels should look good, and in order to highlight our decision (and also to make it easy to change it later, just in case), we store this "magic number" in a global macro variable (③).

Mapping the raw frequency to a level is known as *scaling*. There can be many different ways to scale. DeVenezia (2008) utilizes a linear scaling of a raw percentage into the range of font sizes from 8 point to 32 point. He also mentions three other possibilities: weighted, logarithmic, and exponential scaling.

The data step above implements scaling[3]. This manipulation requires two passes over the input data. The first pass finds the maximum frequency; the second scales using the found maximum. In the data step, we use two DO UNTIL loops with a SET statement inside each. This coding pattern is sometimes called "Double DoW[4]."

In the first DO UNTIL loop, we find the maximum by comparing the count value against the retained[5] maxCount so far (④). In the second loop, we linearly scale the proportion of the count (of the maximum) into the desired number of levels (10), rounding up to a whole number using the CEIL function (⑤). The end-result is that each word now being assigned into one of the 10 levels (from 1 to 10).

There are many alternatives to the Double DoW. We could have used PROC SUMMARY or PROC FREQ to find out the maximum frequency and then MERGE it to every observation of the input data, before carrying out the calculation. We could also have taken advantage of PROC SQL's automatic merging of the group-level aggregate statistics, as shown in the Appendix C.

[3] Due to the discrete nature of the end-result (i.e., each word being assigned into one of the 10 levels or *bins*), this process may better be called *binning*, instead of scaling.

[4] Another way of doing the same thing is to "interleave a dataset itself." See Schreier (2003) for explanation.

[5] The RETAIN statement here is not necessary because the data step as a whole loops just once and thus its value is never reset. We use the statement here mainly due to enhance code legibility and to emphasize that it is initialized to 0 at the beginning.

FINALLY, HTML OUTPUT

We are now ready to write out an html file with a tag cloud.

```
/*-- generate a web page with the tag cloud ----------------------------------*/
data _null_;

  infile cards;
  file "csgf2010.htm";


  input;                                                                    /* ① */

  if index(_infile_, "#spans#") then do until (end);
    set sizes end=end;
    length span $200;

    span = transtrn(transtrn(                                               /* ② */
      "<span class='s#size#'>#word# </span>",
      "#size#", trim(put(size,best.-l))),
      "#word#", trim(word));
    len = length(span);

    put span $varying. len @@;                                              /* ③ */
  end;


  else if index(_infile_, "#pageTitle#") then do;                           /* ④ */
    length line $200;

    line = transtrn(_infile_, "#pageTitle#",                                /* ⑤ */
       "What We Talked Most About in sgf2010");
    len = length(line);
    put line $varying. len;
  end;

  else do;

    put _infile_;                                                           /* ⑥ */

  end;

cards4;                                                                     /* ⑦ */
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">

  <title>#pageTitle#</title>                                                /* ⑧ */
  <style>
    body { margin: 2em; font-family: verdana; }
    h1 { font-weight: normal; font-size: 1.5em; }
    .cloud { padding: 1em; border: solid 1px #000000}
    .s1 { font-size:0.5em; }
    .s2 { font-size:1em; }
    .s3 { font-size:2em; }
    .s4 { font-size:3em; }
    .s5 { font-size:4em; }
    .s6 { font-size:5em; }
    .s7 { font-size:6em; }
    .s8 { font-size:7em; }
    .s9 { font-size:8em; }
    .s10 { font-size:9em; }
  </style>
</head>
<body>

  <h1>#pageTitle#</h1>                                                      /* ⑨ */
  <div class="cloud">

    #spans#                                                                 /* ⑩ */
  </div>
</body>
</html>
;;;;
run;
```

The DATA _NULL_ step reads in the HTML5 source in-lined under the CARDS4 statement (⑦). The template consists of plane html tags and several CSS style definitions, including 10 different classes (s1 to s10) with different font-sizes. The step reads each line in the CARDS4 with the INPUT statement (①), and writes out the line as it is, with the PUT _INFILE_ (⑥). Unless the input line contains one of the two placeholders imbedded in the template (#pageTitle# or #spans#), that is.

When the input line has a placeholder, however, the placeholder is replaced with the real content before the line is output. The first placeholder (#pageTitle#) is replaced with a title that describes the web page. The check for the existence of the #pageTitle# is in line ④, and the replacement operation is done by the TRANSTRN function in line ⑤. In other words, the template line that reads (⑧):

```
<title>#pageTitle#</title>
```

becomes

```
<title>What We Talked Most About in sgf2010</title>
```

when output. Similarly the line ⑨ is output as:

```
<h1>What We Talked Most About in sgf2010</h1>
```

When the input line has the second place holder, #span#, then we loop over a dataset, SIZES, whose observation has the keyword to display and its size. For each word, we then output a HTML SPAN tag with a CSS CLASS attribute. For instance, let's say the word is "DATA" and its size 10, then we write out:

```
<span class=”s10”>DATA</span>
```

As before, the TRANSTRN function shoulders the actual replacement. Here it is called twice: once for replacing the size and another for the word.

We prefer in-lining the output template in the CARDS4 block to dynamically generating the output using variables and character concatenations because the latter usually results in a mixture of quoted strings and programming statements, which is harder to read and maintain. An obvious alternative to in-lining the template within a data step is to store the template in a separate text file to be INFILED.

CHECK

Voilà! A tag cloud is made.

```
/*-- check -----------------------------------------------------------------*/
x csgf2010.htm;
```

You can see it in any modern browser. In the SAS Display Manager (DM) interactive environment, you can launch your default browser by using the X statement.

LIMITATIONS AND EXTENSIONS

A tag cloud is more useful when each word serves as a clickable link to the related items. It is straightforward to implement, but we omitted it in order to keep the code (and the discussion) short and simple.

Extracting keywords given text (also known as, term extraction or keyword extraction) is an actively developed and commercially promising research area of data mining. Various innovative algorithms exist and some demonstrations are available free of charge on the web, at the time of writing this paper.

The display of keywords can be enhanced in many interesting ways as well. Different (shades of) colors can either accentuates the font-size difference, or add an extra dimension to the picture. Orientation of words can be rotate so that some tags arranged vertically. The overall shape of the tag cloud does not have to be rectangular, either – a heart shaped tag cloud, anyone? At the time of writing this paper, there are web sites that can do just that. We recommend searching the web with a keyword, "tag clouds generator."

Given the powerful features of PROC TEMPLATE language, we believe that it may be possible to output a tag cloud using a custom-made tagset destination directly from a one-way frequency output of PROC FREQ. We have not tried it, yet.

SUMMARY

This paper shows how to create a simple tag cloud using nothing but Base SAS. SAS's powerful language features like Perl Regular Expressions and PROCs make it easy to implement the end-to-end process: from scraping web pages, to handling complicated strings, to generating fun and informative graphics like tag clouds.

REFERENCES

DeVenezia, Richard A (2008). "Tag Clouds – A list of tokens, sized by relative frequency" Paper SIB-096 in the *Proceedings of South East SAS Users Group Conference* (SESUG 2008). <http://analytics.ncsu.edu/sesug/2008/SIB-096.pdf>.

Schreier, Howard (2003). "Interleaving a Dataset with Itself: How and Why" Paper CC002 in the *Proceedings of North East SAS Users Group Conference* (NESUG 16). <http://www.howles.com/saspapers/cc002.pdf>.

ACKNOWLEDGEMENTS

Many thanks to Lex Jansen for providing the comprehensive SAS Conference Paper contents and links page through his web site http://lexjansen.com. Without his effort, scraping paper titles would have been much harder and more time-consuming.

Thanks also to Howard Schreier, Dawn Koffman, Cynthia Zender, Richard DeVenezia, and Perry Watts for their reading of the earlier versions of this paper and for their thoughtful comments and suggestions!

CONTACT INFORMATION

Chang Y. Chung
Princeton University
#216 Wallace Hall
Princeton, NJ 08540
cchung@princeton.edu
http://changchung.com/

John King
Ouachita Clinical Data Services, Inc.
24 Loblolly Circle
Mount Ida, AR 71957

APPENDIX A

```
/*-- making a tag cloud of keywords in the SAS Global Forum 2010 paper titles
     by chang y chung  and John King on 2010-04-30 ---------------------------*/

x cd "c:\tagCloud";

/*-- read sgf2010 page from lex^s web site ----------------------------------*/
%let amp = %nrstr(&);
%let param = s=sugi&amp.c=sugi&amp.x=;
%let lex = http://lexjansen.com/cgi-bin/xsl_transform.php?&param;

filename sgf2010 url "&lex.sgf2010" encoding="utf-8";
data titles;
   infile sgf2010 lrecl=10000 truncover;
   input line $10000.;

   sgf = prxmatch("|sgf2010\.|", line);
   hilite = prxmatch("|hilite|", line);
   if sgf and not hilite;

   id = substr(substr(line,sgf), 9, 8);
   title = prxchange("s|.*>([^>]+)</a><br />.*$|$1|", 1, line);
   keep id title;
run;
filename sgf2010 clear;


/*-- for table lookup -------------------------------------------------------*/
proc format;
  invalue ignored
   "SAS", "THE", "AND", "TO", "A", "IN", "USING",
   "FOR", "OF", "WITH", "YOUR", "AN", "ON",
   "BY", "YOU", "FROM", "IS", "AT", "OR", "DO", "GET",
   "NOT", "MORE", "AS", "I", "BE", "US", "VERY", "WE",
   "YES", "YET", "YOU'RE", "THAT", "SO", "WHAT'S",
   "WORKSHOP", "OUR", "IT'S", ".", "IT’S", "-",
   "THAT'S", "THEIR", "SUCH", "CAN", "BETWEEN",
   "WHICH", "I’VE", "ITSELF" = 1
   other = 0;
run;

/*-- split titles into words skipping ignore words ---------------------------*/
data words;
   set titles;
   length word $20;
   i = 1;
   link getWord;
   do while (not missing(word));
      if not input(word, ignored.) then output;
      i + 1;
      link getWord;
   end;
   keep id word;
   return;
   getWord:
      word = upcase(scan(title, i, " :,()?®™-!=”“;""", "r"));
   return;
run;

/*-- get frequencies and scale ----------------------------------------------*/
proc freq data=words;
  tables word/out=freqs(keep=word count where=(count>=3));
run;

%let maxSize = 10;
data sizes;
  retain maxCount 0;
  do until (end1);
    set freqs(keep=count) end=end1;
    maxCount = max(count, maxCount);
  end;
  do until (end2);
    set freqs end=end2;
    size =  ceil(&maxSize * count / maxCount);
    output;
  end;
run;


/*-- generate a web page with the tag cloud ---------------------------------*/
data _null_;

  infile cards;
  file "csgf2010.htm";

  input;

  if index(_infile_, "#spans#") then do until (end);
    set sizes end=end;
    length span $200;
    span = transtrn(transtrn(
      "<span class='s#size#'>#word# </span>",
      "#size#", trim(put(size,best.-l))),
      "#word#", trim(word));
    len = length(span);
    put span $varying. len @@;
  end;

  else if index(_infile_, "#pageTitle#") then do;
    length line $200;
    line = transtrn(_infile_, "#pageTitle#",
       "What We Talked Most About in sgf2010");
    len = length(line);
    put line $varying. len;
  end;

  else do;
    put _infile_;
  end;
cards4;
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>#pageTitle#</title>
  <style>
    body { margin: 2em; font-family: verdana; }
    h1 { font-weight: normal; font-size: 1.5em; }
    .cloud { padding: 1em; border: solid 1px #000000}
    .s1 { font-size:0.5em; }
    .s2 { font-size:1em; }
    .s3 { font-size:2em; }
    .s4 { font-size:3em; }
    .s5 { font-size:4em; }
    .s6 { font-size:5em; }
    .s7 { font-size:6em; }
    .s8 { font-size:7em; }
    .s9 { font-size:8em; }
    .s10 { font-size:9em; }
  </style>
</head>
<body>
  <h1>#pageTitle#</h1>
  <div class="cloud">
    #spans#
  </div>
</body>
</html>
;;;;
run;

/*-- check ------------------------------------------------------------------*/
x csgf2010.htm;
```

APPENDIX B

```
/* web scraping multiple pages using INFILE FILEVAR= option
     by chang y chung  and John King on 2011-02-25 ---------------------------*/


data titles(keep=order id title conference);
   length url $256;
   rx1 = prxparse("s|.*>([^>]+)</a><br />.*$|$1|");
   length order 8 conference $8 id $32 title $2048;
   do while(1);
      infile cards eof=eof firstobs=1 obs=max;
      input conference @@;
      order + 1;
      rx2 = prxparse(cats('|',conference,'\.|'));
      url=cats('http://lexjansen.com/cgi-bin/xsl_transform.php?s=sugi&c=sugi&x=',conference);
      infile dummy url filevar=url end=eof lrecl=5132;
      do while(not eof);
         input;
         sgf    = prxmatch(rx2,_infile_);
         hilite = index(_infile_,"|hilite|");
         if sgf and not hilite then do;
            id    = scan(substr(_infile_,sgf),1,'"');
            title = prxchange(rx1,1,_infile_);
            output;
         end;
      end;
      return;
   end;
   eof:
   stop;
cards;
sgf2010
sugi25
;
run;
```

APPENDIX C

```
/* get frequencies and scale using PROC SQL
     by chang y chung  and John King on 2011-02-25 ---------------------------*/


proc freq data=words;
   tables word/out=freqs(keep=word count where=(count>=3));
run;

%let maxSize = 10;
proc sql;
   create table sizes as
   select *, max(count) as maxCount
        , ceil(&maxSize * count / calculated maxCount) as size
   from   freqs;
quit;
```