Paper 342-2013

%GetReviews: A SAS® Macro to retrieve user reviews in JSON format from review websites and create SAS® Datasets

Ganesh Badisa, Siddhartha Mandati and Dr. Goutam Chakraborty
Oklahoma State University, Stillwater, OK

ABSTRACT:

The proliferation of social networking sites and consumers' desires to create and share content on such sites has continued to generate a huge amount of unstructured data. Analytics users often want to tap into such unstructured data and extract information. Many websites such as Twitter, Facebook, and Rotten Tomatoes offer APIs for external systems to interact and retrieve the data in JSON format. The API of Rotten Tomatoes returns data in a complex text pattern that has information about user reviews. Currently there is no designated code in SAS® to read the JSON response directly and fetch the needed data. This paper illustrates the development and application of a SAS® Macro **%GetReviews** to retrieve the reviews of any desired movie from Rotten Tomatoes' API.

INTRODUCTION:

Social Media has gained a lot of attention in the past decade as a valuable source for information for businesses, government and non-profit organizations across the world. The rate of growth in the numbers of users of social media sites is in the hundreds of thousands every day. Collecting quality textual data from social media websites has been a challenging part of Text Analytics because web crawlers often get stumped when collecting data from websites that use different structure and formats such as XML, JSON, etc. Applying text analysis on user reviews has become popular research for text categorization and to discover trends in the textual data. Rotten Tomatoes is one such website which has data about movie reviews by users of the website. Due to the complexity in the returned stream of data from the API of Rotten Tomatoes, analysts are often forced to collect user reviews manually from the website.

This paper illustrates the development and application of a SAS® Macro to retrieve the reviews of any desired movie from Rotten Tomatoes website's API. The macro looks for patterns along the input data in JSON format from Rotten Tomatoes API and retrieves the relevant information into SAS® Datasets.

Perl Regular Expressions (PRX) introduced in SAS® 9 offer powerful features for reading complex data streams such as extracting the position of a particular pattern, extracting data and storing or manipulating extracted data. In this paper, we have used PRXPARSE and PRXNEXT functions along with a set of string functions to create the SAS® macro, **%GetReviews**. Although our macro is customized to suit the Rotten Tomatoes site, experienced programmers can easily modify it to retrieve text and other data from any site using the JSON format.

GETTING THE API KEY FROM ROTTEN TOMATOES WEBSITE:

For this macro to interact with the Rotten Tomatoes website's API and download reviews into SAS® Dataset, it needs an API Key. This key can be obtained by registering with Rotten Tomatoes' developer website. The link for the developer website is: www.developer.rottentomatoes.com. Anyone can register with the website and provide an email address to which an API key will be sent.

UNDERSTANDING PERL REGULAR EXPRESSIONS:

A regular expression specifies the program to find an exact match to the characters in the expression within input data. If we need to search for the phrase "Text Analytics" in a text document or in strings of text, then a regular expression would be the phrase within a pair of forward slashes like this: /Text Analytics/. A PRX function called PRXPARSE is used to create a regular expression ID to search the phrase in the input document. This ID would be used in other PRX functions to perform data manipulations or data capturing.

Example:

```
data _null_;
  ExpressionID = prxparse('/James Gordon/'); /*Pattern to search for in the document*/
  text = "Commissioner James Gordon is a fictional character, an ally of Batman. Gordon made his debut in the first
panel of this comic"; /*Input document*/
  start = 1;/*Start position of the search*/
  stop = length(text);/*Stop position of the search*/
  call prxnext(ExpressionID, start, stop, text, position, length);
    do while (position > 0);
      found = substr(text, position, length);
      put found= position= length=;/*prints the position, length and search string to the log*/
      call prxnext(ExpressionID, start, stop, text, position, length);
    end;
```

```
run;
```

PRXPARSE creates an ID for the search pattern specified. The Position and length of the search string are the outputs returned by the PRXNEXT function. Notice that the PRXNEXT function finds only the exact matches to the phrase *James Gordon* but not a part of the phrase like *Gordon* or *James.*

OUTPUT:

```
found=James Gordon position=14 length=12
NOTE: DATA statement used (Total process time):
      real time           0.02 seconds
      cpu time            0.03 seconds
```

JSON FORMAT DESCRIPTION:

Java Script Object Notation (JSON) is a light wait data interchange format, which is based on a subset of Java script programing language conventions. The JSON is built on two structures.

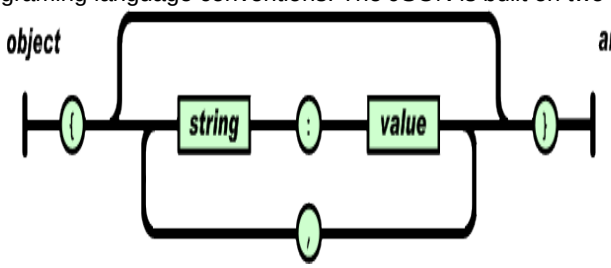

Figure 1: Structure of JSON Object

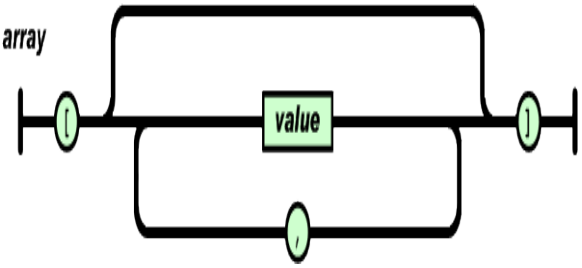

Figure 2: Structure of JSON Array

1. Object: It is the collection of name/value pairs, which can be realized in many programing languages as object, Struct, hash table and maps. In the JSON representation, each object is enclosed within a pair of curly braces "**{ }**". The variables in each object contain string (identifier) and value (value of the identifier) pairs separated by colons - "**:**", and if there is more than one variable in an object then each variable is separated by commas- "**,**". [1]

Examples:

1. Student record can be represented as: {"name":"Jack Kale", "age":"23"}

2. Array: In most of the programming languages, an ordered list of values can be realized as arrays. Array structure can contain multiple values or multiple objects. The array structure is enclosed in brackets- **"[ ]"** and values/objects in the array are separated by commas- **","**. [1]

Examples:

1. List of Names: {"Names":["Ricky Martin", "James Cook","Rob Tayler"]}
2. List of employees: {"employees":[{"Name":"David Boon","Age":"32"},{"Name":"Christy Jane","Age":"25"}]}

The Rotten Tomatoes' API returns data in JSON format with reviews or search results for a given movie name. Figure-3 shows a part of actual data returned by Rotten Tomatoes website's API with reviews of a movie. In this data **"total"** is the identifier of a variable and **"291"** is the value of that variable. Similarly, **"reviews"** is the identifier of a variable whose value is an array of review objects. Each review object contains variables like "**Critic,**" "**date,**" "**freshness,**" "**Publication,**" "**quote,**" and "**links**". To extract the value of each variable from the JSON string, the position of the variable should be identified and the value next to that variable should be extracted. The position of the variable is identified using SAS® regular expressions, and the value next to the variable is extracted using string operations.


```
{"total":291,"reviews":[{"critic":"Robert Denerstein","date":"2012-10-
20","freshness":"fresh","publication":"Movie Habit","quote":"Nolan keeps things dark and
dangerous","links":{"review":"http://www.moviehabit.com/review.php?story=dar_gs12"}},{"cri
tic":"AliGray","date":"2012-10-
19","original_score":"5/5","freshness":"fresh","publication":"TheShiznit.co.uk","quote":"Doubtl
ess flawed and arguably overlong, it's nonetheless a personal and heartfelt ode from the
director to the greatest comic-book character ever committed to paper -- and now
film.","links":{"review":"http://www.theshiznit.co.uk/review/the-dark-knight-
rises.php"}},{"critic":"Kofi Outlaw","date":"2012-10-
02","original_score":"4/5","freshness":"fresh"
```


Figure 3: A part of data returned by Rotten Tomatoes website's API

%GETREVIEWS MACRO IMPLEMENTATION:

The macro allows anyone to collect data from Rotten Tomatoes website's API. Anyone with a basic to intermediate knowledge in SAS/BASE and SAS/MACRO programming can understand and execute the macro to extract the data of a chosen movie from the website. %GetReviews requires a few attributes before executing the macro:

1. Name of the movie for which user reviews should be extracted
2. API key for the website's API

The step by step procedure of extracting the attributes is discussed below.

Step 1: Search for a given movie and extract the movie ID from the search result

The macro %GetMovieID takes the movie name and API key as parameters and interacts with the Rotten Tomatoes API to search for the given movie name. The search results are in JSON string format. From the stored JSON string, the movie ID is extracted and set to a global macro variable "movieID". It is assumed that if there is more than one movie returned in a search result string, the first movie in the result is considered as the desired movie and the ID of that movie is extracted.

The moviesearchjson dataset has the Movie ID that will be used in Step 2 and Step 3. Extracting the movie ID from the dataset is performed using PRXNEXT function. Using the position and length returned by the PRXNEXT function the actual length of the Movie ID is determined and a simple substr function is used to capture the Movie ID.

Step 2: Obtaining total number of pages and total number of reviews

The macro %GetTotalPages takes the movie ID and the API key and queries the API for the first page of reviews (with first 50 reviews). The returned result is in JSON format, which is stored into the movieReviewJSON dataset. The macro extracts the information about the total number of reviews available for the movie and assigns it to a global macro variable "totalrevs". As the API returns a maximum of 50 reviews at a time, this macro also calculates the number of pages to be retrieved from the API and assigns it to a macro variable "numofsets".

Step 3: Creating datasets for all pages returned from the API

The macro %GetAllJSONReviews takes the API key, the "movieID" obtained in step 1 and the total number of pages obtained and queries the API iteratively to retrieve one review page at a time. The macro creates a *movieRevieJSON<pageno>* dataset with data in JSON format for each page of reviews that API returns. Here <pageno> is the number of the page being retrieved in each iteration.

Step 4: Creating final dataset with Review date, Reviews and Freshness of the review

The macro %RetrieveReviews takes total number of pages as input parameter and reads all the datasets (**movieReviewJSON<pageno>**) to extract all the attribute values into a single dataset. Each PRXPARSE function creates an ID for each pattern specified and these IDs are used in PRXNEXT function to capture values of the attributes (date, user reviews and freshness of the review). SUBSTR function is used after each PRXNEXT function to capture and store the attribute values into **movieReview<pageno>** dataset. The CALL PRXNEXT is executed iteratively until no further pattern matches for date or review or freshness is found in all the datasets. This final dataset Reviews_For_<movieID> contains all the instances of review date, freshness of the review and review for the movie.

%GETREVIEWS:

The %GetReviews is the main macro which takes the movie name and API key as parameters and passes them to the macros discussed in steps 1 through 4. This macro contains the macro calls for all the macros discussed in the above steps to execute them in a sequential order. Before executing this macro all the macros discussed in steps 1 through 4 should be compiled.

FLOW CHART OF MACRO %GETREVIEWS:

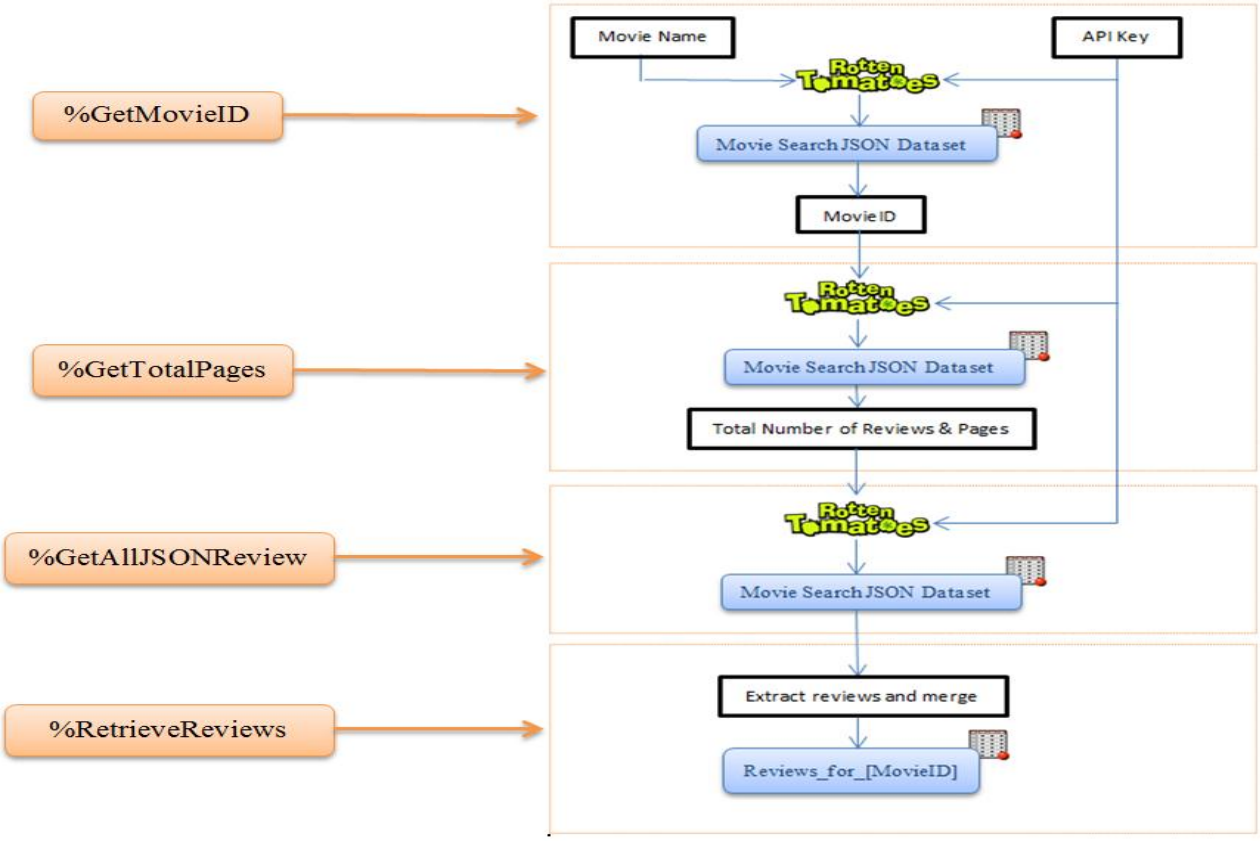

Figure4: Flowchart of %GetReviews macro

CONCLUSION:

Social networking sites are generating a huge amount of unstructured data. The difficulty lies in getting cleansed data as input to text analytics. The %GetReviews macro attempts to solve these problems by taking advantage of SAS® Perl Regular Expression functions and extract reviews from website that return data in JSON format, a growing alternative to XML for interchanging information between applications. The output of %GetReviews is a cleansed dataset that can be analyzed effectively via text and sentiment mining. The macro can be modified to retrieve text and other data from any site that returns data in the JSON format.

CONTACT INFORMATION:

Your comments and questions are valued and encouraged. Contact the authors at:

Ganesh Babu Badisa, Email: ganesh.badisa@okstate.edu

Ganesh Badisa is Master's student in Management Information Systems at Oklahoma State University. He is a BASE SAS® 9, JMP certified data explorer and a certified SAS® predictive modeler using Enterprise Miner 6. In May, 2012, he received his SAS® and OSU Data Mining Certificate and currently enrolled in Graduate Certificate in Business Data Mining.

Siddhartha Reddy Mandati, Email: siddhartha.mandati@okstate.edu

Siddartha Reddy Mandati is a Master's student in Telecommunications Management at Oklahoma State University. He is a SAS Certified BASE programmer for SAS® 9, JMP certified Data Explorer and SAS® Certified Predictive Modeler using SAS® Enterprise Miner 6. He is currently in the SAS® and OSU Data Mining Certificate program and Graduate Certificate in Business Data Mining program.

Dr. Goutam Chakraborty, Oklahoma State University, Stillwater OK, Email: goutam.chakraborty@okstate.edu

Dr. Goutam Chakraborty is a professor of marketing and founder of SAS® and OSU data mining certificate and SAS® and OSU business analytics certificate at Oklahoma State University. He has published in many journals such as Journal of Interactive Marketing, Journal of Advertising Research, Journal of Advertising, Journal of Business Research, etc. He has chaired the national conference for direct marketing educators for 2004 and 2005 and co-chaired M2007 data mining conference. He has over 25 years of experience in using SAS® for data analysis. He is also a Business Knowledge Series instructor for SAS®.

TRADEMARKS:

APPENDIX:

%GetReviews Macro:

```
dm "log; clear; output; clear;";
options mprint merror mlogic;
%macro GetReviews(Movie, Key);
/*the following macro will downloads the search results for the given movie  */
/*in JSON format and retrieves the movie ID of first result found         */
%GetMovieId(&Movie, &Key);
/*The following macro downloads the reviews data in JSON format and extracts *
 *the total number of pages needed to be downloaded from the API.          */
%GetTotalPages(&movieID, &Key);
/*The following macro call will retrieve all the review pages in JSON     *
 *format and stores in the SAS data sets                                  */
%GetAllJSONReviews(&movieID, &Key, &numofsets);
/*the following macro call will extract all the review variables from    *
 *different pages and stores them into a single dataset.                  */
%RetrieveReviews(&numofsets);
%mend;

%macro GetMovieId(Movie, Key);
%put executing getMovieId macro;
%let apikey = &Key;
%let MovieName = &Movie;
%let page_limit = 50;
%let MovieNameUrl = %sysfunc(translate(%sysfunc(strip(&MovieName)),"+"," "));
%let URLParams = %nrstr(apikey=)&apikey%nrstr(&q=)&MovieName;

%let MovieSearchUrl =
http://api.rottentomatoes.com/api/public/v1.0/movies.json?%nrstr(apikey=)&apikey.%nrstr(&q=)&MovieNameUrl;
%put "&MovieSearchUrl";


filename rturl url
"http://api.rottentomatoes.com/api/public/v1.0/movies.json?%nrstr(apikey=)&apikey.%nrstr(&q=)&MovieNameUrl"
;
%let MovieSearchDataFileName = MovieSearchJSON;
data work.&MovieSearchDataFileName;
     length line $30000;
     infile rturl lrecl=32767;
     input line & $30000;
run;

/*To find movieID which would be unique to each movie*/
data _null_;
```

```
set work.moviesearchjson;
  /*Creates ExpressionID that we want to search for, here the pattern */
  /*is ":" and this pattern is searched for in the text              */
 ExpressionID = prxparse('/":"/');
  /*we want to start searching the pattern from starting of the text file*/
  start = 1;
  /*point where the search stops*/
  stop = length(line);
    /* Use PRXNEXT to find the first instance of the pattern, */
    /* then use DO WHILE to find all further instances.       */
    /* PRXNEXT changes the start parameter so that searching  */
    /* begins again after the last match.                     */
    /* Here "line" is the variable that was created in        */
    /* moviesearchjson dataset, position and length attributes*/
    /* are empty and get populated as the prxnext function    */
    /* executes. It is the position where the desired         */
    /* pattern is found and its length                        */
  call prxnext(ExpressionID, start, stop, line, position, length);
                    strt=position+length;
    /*pos is the position where the movieID ends              */
                    pos=find(line,"",strt);
            /*leng is calculated to find the length of movieID        */
                    leng=pos-strt;
    /*variable to search is "line", start position is strt,   */
    /*length of search is leng                                */
                    var2=substr(line,strt,leng);
            /*value of intermediate variable var2 is assigned to macro variable movieID*/
  call symputx('movieID',var2,'G');
run;

%put The movie id of &MovieName is &movieID;

%mend;

/*Getting movie reviews JSON DATA for selected movie*/
%macro GetTotalPages(ID, Key);
%let apikey = &Key;
%let pageLimit = 50;
%let page = 1;
%let reviewType = all;

%let reviewsUrl =
http://api.rottentomatoes.com/api/public/v1.0/movies/&ID./reviews.json?apikey=&apikey.%nrstr(&review_type=)&revie
wType.%nrstr(&page_limit=)&pageLimit.%nrstr(&page=)&page.;
%put &reviewsUrl;
filename rtrevw url
"http://api.rottentomatoes.com/api/public/v1.0/movies/&ID./reviews.json?apikey=&apikey.%nrstr(&review_type=)&revi
ewType.%nrstr(&page_limit=)&pageLimit.%nrstr(&page=)&page."
;
%let MovieReviewDataFile = MovieReviewJSON;

data work.&MovieReviewDataFile;
    length line $30000;
    infile rtrevw lrecl=32767;
    input line & $30000;
run;

/*retrieving total number review from JSON data*/
data _null_;
set work.moviereviewjson;
    /*we want to start searching the pattern from starting of the text file*/
start = 1;
```

```
    /*This is the pattern we look for, so an ExpressionID is */
    /*created to be used in prxnext function                 */
ExpressionID = prxparse('/{"total":/');
    /*point where the search stops at the end of the text file*/
  stop = length(line);
  if stop gt 1 then do;
    /* Use PRXNEXT to find the first instance of the pattern, */
    /* then use DO WHILE to find all further instances.       */
    /* PRXNEXT changes the start parameter so that searching  */
    /* begins again after the last match.                     */
    /* Here "line" is the variable that was created in        */
    /* moviereviewjson dataset, position and length attributes*/
    /* are empty and get populated as the prxnext function    */
    /* executes. It is the position where the desired         */
    /* pattern is found and its length                        */
          call prxnext(ExpressionID, start, stop, line, position, length);
          end;
    /*strt is the local variable created to find the actual   */
    /*position where the integer(total number of reviews) begins in the code*/

          if position ge 1 then do;
          strt=position+length;
           /*pos is the position where the integer ends in this text file*/
                pos=find(line,',',strt);
           /*numb is the length of the integer calculated            */
                numb=pos-strt;
           /*var2 is a local variable created to capture the integer */
           /*using the substr function                               */
                var2=substr(line,strt,numb);
           /*Integer is assigned to the macro variable totrevs       */
          call symputx('totalrevs',var2,'G');
  end;
run;

/*calculating the Total number of pages that need to be retrieved to get all the reviews.*/
data _null_;
set work.moviereviewjson;
    /*Checking if the total reviews are greater than 50 as the   */
    /*pagelimit for each capture is 50 by default.               */
    /*If the page limit is greater than 50 then count the factor */
if &totalrevs>50 then do;
    /*the factor is assigned to numofsets if the reviews >50     */
          call symputx('numofsets',%sysevalf(&totalrevs/50,ceil),'G');
end;
    /*Else the totrevs macro variable value is assigned to numofsets*/
else do;
call symputx('numofsets',1,'G');
end;
run;

%mend;

%macro GetAllJSONReviews(ID, Key,totalPages);
%let apikey = &Key;
%let pageLimit = 50;
%let page = 1;
%let reviewType = all;
%do %while (&page <= &totalPages);
%let reviewsUrl =
http://api.rottentomatoes.com/api/public/v1.0/movies/&ID./reviews.json?apikey=&apikey.%nrstr(&review_type=)&revie
wType.%nrstr(&page_limit=)&pageLimit.%nrstr(&page=)&page.;
%put &reviewsUrl;
```

```
%put dowloading page &page;
filename rtrevw url
"http://api.rottentomatoes.com/api/public/v1.0/movies/&ID./reviews.json?apikey=&apikey.%nrstr(&review_type=)&revi
ewType.%nrstr(&page_limit=)&pageLimit.%nrstr(&page=)&page."
;
%let MovieReviewDataFile = MovieReviewJSON;
data work.&MovieReviewDataFile&page;
     length line $30000;
     infile rtrevw lrecl=32767;
     input line & $30000;
   run;
%let page = %eval(&page + 1);
%end;
%mend;

%macro RetrieveReviews(totalSets);
%let page = 1;
%do %while (&page <= &totalSets);
%let dset = work.MovieReviewJSON&page;
%put processing page No: &page dataset: &dset;
%if %sysfunc(exist(&dset))= 0 %then %do;
%put ERROR: Data set &dset does not exist.;
%put ERROR- Macro will terminate now.;
%return;
%end;
data work.MovieReviews&page(keep= Review freshness date);
set &dset;
dateExprID = prxparse('/"date":"/');
dateStart = 1;
dateStop = length(line);

freshExprID = prxparse('/"freshness":"/');
freshStart = 1;
freshStop = length(line);

quoteExprID = prxparse('/"quote":"/');
quoteStart = 1;
quoteStop = length(line);

call prxnext(dateExprID, dateStart, dateStop, line, datePos, dateLen);
call prxnext(freshExprID, freshStart, freshStop, line, freshPos, freshLen);
call prxnext(quoteExprID, quoteStart, quoteStop, line, quotePos, quoteLen);
strt =1;
        do while (freshPos > 0 | quotePos > 0 | datePos>0);
                %if datePos > 0 %then %do;
                strt = datePos+datelen;
                pos = find(line,'",',strt);
                numb = pos-strt;
                date = substr(line,strt,numb);
                *put  'current date' date=;
                %end;

                %if quotePos > 0 %then %do;
                strt = quotePos+quotelen;
                pos = find(line,'",',strt);
                numb = pos-strt;
                Review = substr(line,strt,numb);
                Review = COMPRESS(Review,'+\/}{][@!',' ');
                *put 'current Review' Review=;
                %end;

                %if freshPos > 0 %then %do;
```

```
                        strt = freshPos+freshlen;
                        pos = find(line,"',',strt);
                        numb = pos-strt;
                        freshness = substr(line,strt,numb);
                        *put 'current freshness' freshness=;
                        %end;
                        output;
call prxnext(dateExprID, dateStart, dateStop, line, datePos, dateLen);
call prxnext(freshExprID, freshStart, freshStop, line, freshPos, freshLen);
call prxnext(quoteExprID, quoteStart, quoteStop, line, quotePos, quoteLen);
end;
run;
%let page = %eval(&page + 1);
%end;

data RVWS&movieID;
   set MovieReviews1-MovieReviews&numofsets.;
 run;
/* clean up all temperory files*/
 /*comment the following section if you wish to see the actual JSON data*/
 proc datasets lib=work nolist;
           delete movieReviewJSON;
           delete movieSearchJSON;
   delete movieReviewJSON1-movieReviewJSON&numofsets.;
   delete movieReviews1-movieReviews&numofsets.;
 quit;
%mend;

%GetReviews(Name_of_the_movie, Api_Key );/*Name of the movie and Api Key have to be passed as input
parameters*/
```