Paper 149-2009

Ad Hoc Data Preparation for Analysis Using SAS® Enterprise Guide®

I-kong Fu, Wayne Thompson, Mike Porter, SAS Institute Inc., Cary, NC

ABSTRACT

SAS® Enterprise Guide® is a popular application for conducting ad hoc data manipulation and data preparation for data mining and other analytics. This paper will walk through a few examples of data preparation, putting features of SAS Enterprise Guide into action, and discuss scenarios where SAS Enterprise Guide can be used with other popular SAS® applications such as SAS® Enterprise Miner™. The examples are targeted to business analysts who would primarily like to use a point-and-click approach to these tasks.

INTRODUCTION

Data preparation is the most time consuming and important task of any analytical project. New business questions arise that require ad hoc analyses, but the data sources are often not in the right format for analysis, and thus require manipulation to prepare. Data preparation tasks include collecting appropriate data, sampling, and aggregating data attributes. Data sources are brought together at the customer or account level from many disparate sources. These sources may include billing and payment transactional data, demographic figures, financial data, and other kinds of data. Transformations are then applied to further enrich the data. In this paper we walk through a few examples of ad hoc data preparation using SAS Enterprise Guide. There are many examples that use SAS programming but fewer examples that use a point-and-click approach using SAS Enterprise Guide. This paper is targeted mainly toward business analysts and others who primarily want to use a non-programming approach to analysis but might feel comfortable with inserting a few lines of code on occasion. We will complete the examples using point-and-click functionality as much as possible, though in some cases, we point out where a few simple function expressions or lines of code would come in handy. For reference, we also point out some of the associated code for analysts or SAS programmers who want to view generated code as a learning aid or to associate it with a task.

SCENARIO 1: ASSOCIATIONS AND RECOMMENDATIONS

For our first examples, suppose that we work for a fictitious online organization. The Web site team wants to make improvements to the site by grouping and recommending items such as Frequently Asked Questions (FAQs), white papers, links to other Web pages, user-generated content such as ratings, and other resources to serve more relevant content to our users. Items have already been grouped based on human judgment, but as the site has grown, we have collected large amounts of data. We think we might be able to mine this data to group items based on users' actual behavior for a better user experience. At first we want to offer the groupings based on anonymous groups of users, but later, we might want to provide more personalized recommendations based on data from return visits from more loyal users who have registered and who log on to the site. Later, we also want to use our analysis of this behavioral data to support some of our decisions about other efforts such as whether to send a special offer to a particular group of users. First, we want to perform some exploratory data analysis to determine how feasible the project might be or what other requirements we might have, such as additional data collection. Before we can perform some of these analyses, we'll need to prepare some of the data we already have.

DATA ACCESS

In our first example, we have an access log from a basic Web server that contains some of the data we would like to analyze. It is a plain text file. It is already used along with other techniques for some operational reporting on metrics such as page views, browser types, referring sites or search engines, and other basic information. We would like to convert it to a data set so that we can more easily filter it, add new variables, and perform calculations and other manipulations so we can find more information to answer our new questions.

The text in our log file looks similar to this example, showing a couple of lines from our file:

```
111.222.333.444 - - [18/Jan/2009:10:01:08 -500] "GET /index.html HTTP/1.0" 200 6433
"/great.html" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.0.5)
Gecko/2008120122 Firefox/3.0.5" "Cookie 1" "Session 1"
111.222.444.333 - - [18/Jan/2009:10:01:08 -500] "GET / HTTP/1.0" 200 6433 "
http://www.google.com/search?hl=en&q=great+stuff&btnG=Search" "Googlebot/2.1"
"Cookie 2" "Session 1"
```

To correctly access, prepare, and analyze this data, we need a basic understanding of the raw data. Fortunately, our group already has this domain expertise. Each line corresponds to a request from a browser to the Web server for a particular object such as our home page, index.html. There are various pieces of information separated by a space character. We do not need all the information. For example, the second two fields are often missing.

For now, we are most interested in the date-and-time stamp, the object being requested, and the last two items, which are identifiers for users across browser sessions and for a particular session. The cookies are pieces of information that are stored with users' browsers that we will use as rough identifiers for users. Although there are problems associated with this (for example, different users visiting our site using different browsers, each of which stores its own cookies, or different users using the same browser to visit our site), the cookies are generally considered a better method to identify users than an IP address, which might not identify an individual computer or browser that makes requests through an intermediate server.

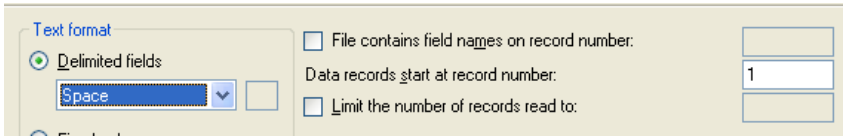
For simplification, we will not worry about these issues here, and we will use these cookies as approximate, anonymous identifiers for individual users so that later we can try to find patterns such as "users who browsed X also browsed Y." Our cookie could persist across visits to our sites so that we could look at patterns across visits, or it could represent an anonymous user for a particular visit or session. It could also correspond to previous login information so that we can personalize content for specific individuals.

For more sophisticated Web data preparation and analysis, we could also use SAS Web Analytics for sessionizing Web logs to overcome cookie-based session issues, and SAS Enterprise Miner's Path Analysis Node for analyzing paths to identify frequent paths, score paths, and more.

IMPORT DATA TASK

We use the Import Data Task in SAS Enterprise Guide to import this text file into a SAS data set, and we use the Query Builder to build some simple filters to create our initial data set. The Import Data Task can import many types of data, including Microsoft Excel files and all kinds of delimited text files. To convert our text log file to a data set, we select File>Import Data. We will import the file based on a space character as our delimiter. There are no variable names so we deselect the first box. This process is shown in Figure 1.

Figure 1. Set Delimiter and Record Start and End Positions



In the next step, we know we will not use a few columns, so we deselect them to avoid importing data we don't need. We are able to review or define attributes here as well. We have changed the output format of the first field, an IP address, to be treated as a string; we don't need to use it for calculations, only as an identifier for approximate geographical information later. We set datetime to DateTime. We also specify Names and Labels here. We could also complete these changes in later steps by editing properties in the data grid viewer.

Figure 2. Select Columns and Define Formats and Other Attributes

Select columns and define attributes:

Inc	Source Name	Name	Label	Type	Source Informat	Len.	Output Format	Output Informat
☑	F1	IP	IP Address	String	$CHAR15.	15	$CHAR15.	$CHAR15.
☐	F2	F2	F2	String	$CHAR1.	1	$CHAR1.	$CHAR1.
☐	F3	F3	F3	String	$CHAR1.	1	$CHAR1.	$CHAR1.
☑	F4	DateTime	Date Time Stamp	Date/...	ANYDTD...	8	DATETIM...	DATETIM...
☑	F5	Offset	Time Offset	String	$CHAR6.	6	$CHAR6.	$CHAR6.
☑	F6	Object	Object Requested	String	$CHAR28.	28	$CHAR28.	$CHAR28.
☑	F7	HTTPcode	HTTP status code	Number	BEST3.	8	BEST3.	BEST3.
☐	F8	F8	F8	String	$CHAR5.	5	$CHAR5.	$CHAR5.
☑	F9	Referrer	Referring Page	String	$CHAR111.	111	$CHAR111.	$CHAR111.
☑	F10	Browser	Browser	String	$CHAR137.	137	$CHAR137.	$CHAR137.
☑	F11	Cookie	Cookie	String	$CHAR8.	8	$CHAR8.	$CHAR8.

After importing, we have our SAS data set, a portion of which can be seen in Figure 3.

Figure 3. A Portion of Our New Data Set

	IP	DateTime	Offset	Object	HTTPcode
1	111.222.333.444	07JAN2009:10:0...	-500]	GET /index.html...	200
2	111.222.333.444	07JAN2009:10:0...	-500]	GET / HTTP/1.0	200
3	111.111.111.111	07JAN2009:11:5...	-0500]	GET /robots.txt...	404

EDIT, ADD, OR FILTER VARIABLES USING QUERY BUILDER

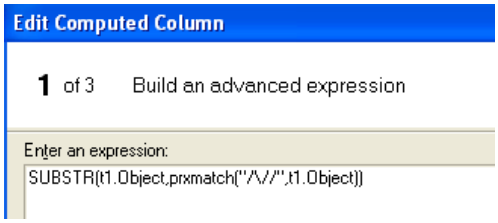
Once we have imported our data, we might need to edit, add, or filter variables. We will perform these tasks primarily using the Query Builder Task. The Query Builder makes these actions, along with other tasks such as joining data, easy. It uses the SQL procedure, but an analyst does not need to manually compose SQL or SAS code to use the Query Builder Task, which makes these manipulation tasks much easier. If we did not edit our variable names when importing, we could also create our own names by deselecting **Protect Data** on the Edit menu, then right-clicking each column to update its properties. Also, the time offset became separated from the date-and-time stamp, but that is fine because we likely need only the datetime value.

STRING PATTERN MATCHING USING PERL REGULAR EXPRESSION IN EXPRESSION EDITOR

The Object variable had the opposite problem. It was listed in quotation marks so the request method (usually “GET”) from the browser is still listed with the file or other object that is being requested. While we could keep it and accept the extra “noise” when grouping (because it should make no logical difference when grouping these items), we will separate it and keep only the object. We will do this by making a new computed column in the Query Builder. We then choose Advanced Expression Editor and use a combination of the SUBSTR and PRXMATCH functions. The SUBSTR function takes a string and returns a substring from a given position in a string. The PRXMATCH function matches a pattern within a string by using a Perl regular expression and returns its position. So we find the position for our actual object by using PRXMATCH, and then extract the actual substring by using this position as an argument for the SUBSTR function, as seen in this expression and in Figure 4:

```
SUBSTR(t1.Object,prxmatch("/\//",t1.Object))
```

Figure 4. Expression Editor



We have created a new computed column we will call Item. There are also a few other edits and filters we want to make while we're performing this task. We previously filtered some columns we did not need when we completed the import step. However, there are some values for some of the variables that we do not want to include. For example, because we want to look at actual user behavior, we want to filter requests from spiders that "crawl the Web" to acquire information for search engines. For the sake of simplicity, we will filter the googlebot only in the example. We will also filter only for objects that had the HTTP status code of 200, meaning that the objects were served from the Web server to the user's browser with no error on the server. See Figures 5 and 6:

Figure 5. Build a Basic Filter

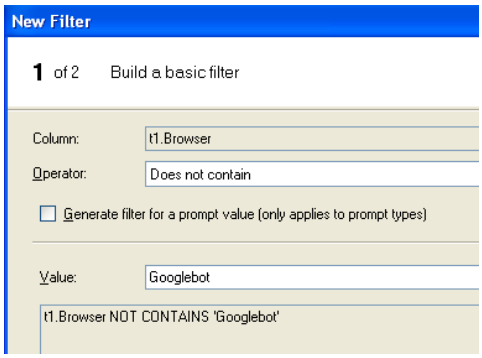
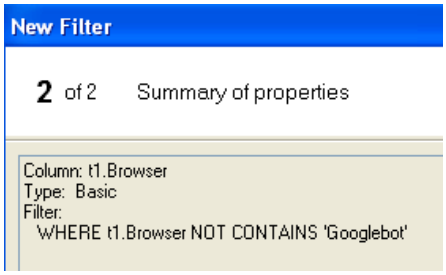


Figure 6. Summary



We have now created a new column called Item by removing the request method, and we have created a couple of filters. We are almost finished with our first variable editing steps! In fact, we already have an anonymous user identifier and an Item we want to associate, so now we can bring our resulting data set into SAS Enterprise Miner and analyze it with the Association Rules node. We have transformed our data from a "multiple-row-per-subject" data set to a "one-row-per-subject" data set required for this analysis. This "one-row-per-subject" data set is also the format required for other kinds of analytical methods such as regression, neural networks, decision trees, cluster analysis and factor analysis. For now, in our fictitious story line, we will assume we achieved a promising result from our association mining and will continue with our data preparation for some related goals by using this same user behavior Web data.

SCENARIO 2: PREDICTING SIGN-UP

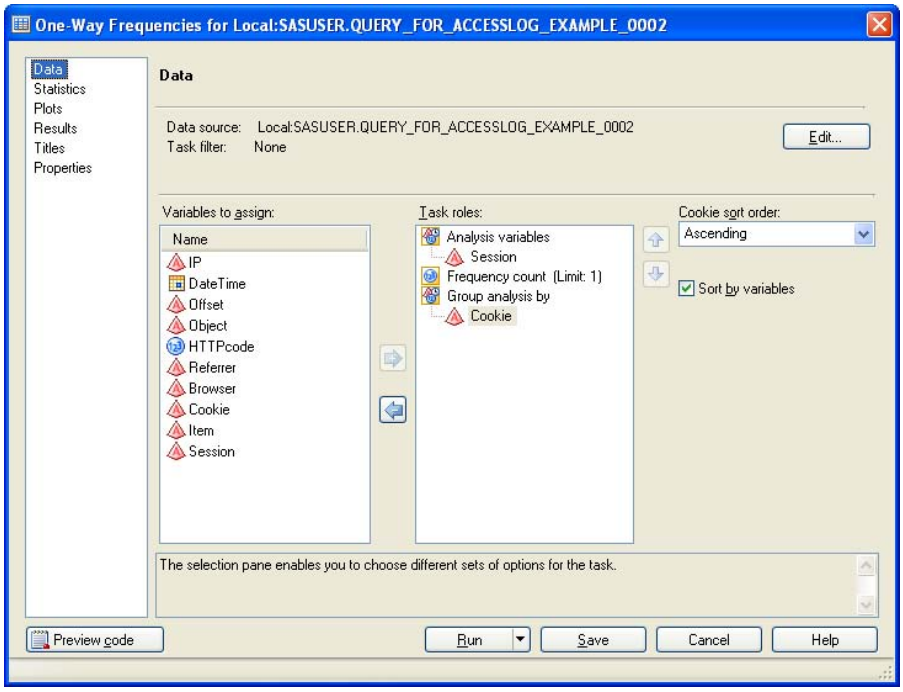
Looking ahead to other analyses we want to perform, we might also want to create some other analysis variables such as recency and frequency of our user visits. Our group wants to find out if we can better predict who might sign up for a particular service from a set of choices such as "Basic" or "Premium," based on our available behavioral data. These variables might help us with our model for this prediction. We might gain information to help design our marketing programs to better target our segment of users who will sign up for the premium service, or we might gain information to take back and run a/b or multivariate tests on the site to gather more data for additional analysis. In

some cases, we might already have these variables. In our example, our Web site does not yet have this amount of sophistication so we will create the variables as new ad hoc analysis variables.

CREATE A FREQUENCY VARIABLE

Calculating frequency can be done using the One-Way Frequency Task, which is listed in the Describe category and uses the FREQ procedure. Because we have a variable that we are using as our user ID across sessions as well as a session ID, we need to know how many sessions a particular ID had, given the period of our log (which might be a week, month, quarter, year, or some other period).

Figure 7. One-Way Frequencies



We select **Session** as our Analysis variable and **Cookie** as our Group By variable. We also specify to suppress output, opting for a data set to be output instead. Now, the problem is that we get a count of all the instances of a session cookie in the log (see Figure 8) instead of a count of sessions per user.

Figure 8. Frequencies Output

	Cookie	Session	COUNT
1	Cookie 1	111.222.333.444...	1
2	Cookie 1	111.222.333.444...	1
3	Cookie 1	111.222.333.444...	1
4	Cookie 1	111.222.333.444...	1
5	Cookie 1	111.222.333.444...	1
6	Cookie 1	111.222.333.444...	1

We could ask the Task to count cumulative frequencies but that count will give us only a cumulative count of session cookies, also not what we need. One simple way to handle this issue is to use the Summary Statistics Task, which uses PROC MEANS to give us a sum. We input the data set that our Frequencies Task output and are rewarded with our correct Frequency count.

Figure 9. Summary Statistics

	Cookie	_TYPE_	_FREQ_
1	Cookie 1	0	6
2	Cookie 10	0	6

We might also want to calculate recency. To do so, we could subtract the most recent date by a given user from today's date to calculate recency in days. We will leave this calculation as an exercise for later.

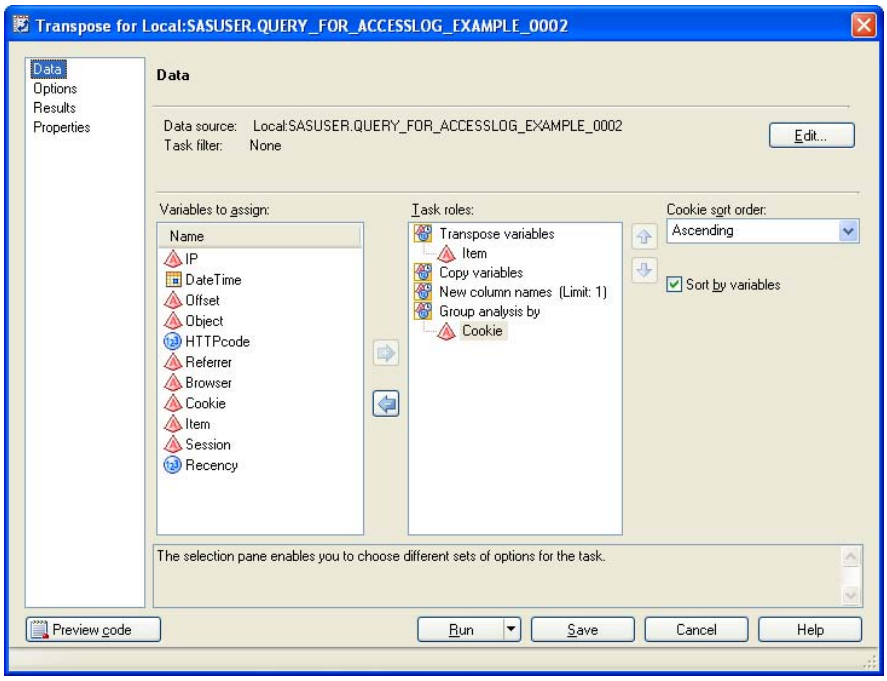
CREATING DUMMY VARIABLES

We also believe users who viewed a particular page are more likely to sign up. In this example, our FAQ page 3, faq3.html, provides some marketing information about the service. We want to create a dummy variable that reflects whether a user viewed this particular page, with a value of 1 to indicate yes, or a value of 0 to indicate no.

TRANSPOSE TASK

Because our data set is still listing each row by the particular browser request, we need to transpose it so that it organizes each row by unique cookie, our proxy for user ID. We can use a Transpose Task, based on PROC TRANSPOSE, to list our pages by our cookie. To do so, we assign Item as our Transpose variable and Cookie as our Group analysis by variable. See Figure 10.

Figure 10. Transpose Task



The Task generates PROC TRANSPOSE code that looks similar to the following:

```
PROC TRANSPOSE DATA=WORK.SORTTempTableSorted
    OUT=SASUSER.TRNSTransposedQUERY_FOR_ACCESSLO(LABEL="Transposed
SASUSER.QUERY_FOR_ACCESSLOG_EXAMPLE_0002")
    PREFIX=Col
    NAME=Source
```

```
LABEL=Label;
BY Cookie;
VAR Item;
```

We get output that looks like this:

Figure 11. Transpose Output

	Cookie	Source	Col1	Col2	Col3
1	Cookie 1	Item	/index.html HTT...	/index.html HTT...	/index.html HTT...
2	Cookie 10	Item	/faq3.html HTTP/...	/item4.html HTT...	/images/abc.jpg...
3	Cookie 5	Item	/ HTTP/1.0	/ HTTP/1.0	/ HTTP/1.0
4	Cookie 6	Item	/faq1.html HTTP/...	/faq2.html HTTP/...	/faq3.html HTTP/...
5	Cookie 8	Item	/faq3.html HTTP/...	/ HTTP/1.1	/ HTTP/1.1

We now have our data transposed to the right structure. However, there is a problem with this output. Our data has been transposed, but there is no logical name for our new variables and FAQ 3 might occur in any of these columns. We can deal with this issue in a couple of ways. First, we can simply use our handy Query Builder once again to build an expression to see if the page exists in any of our new columns, similar to this expression:

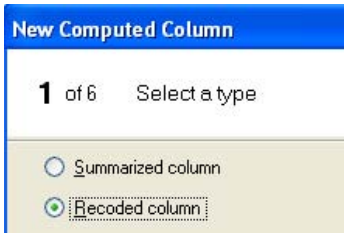
```
(t1.Col1 contains '/faq3.html HTTP/1.1') or (t1.Col3 contains '/faq3.html
HTTP/1.1')
```

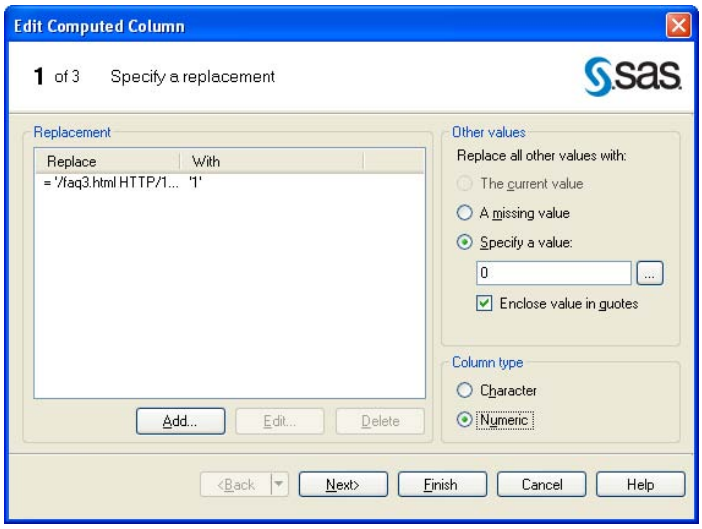
The expression returns a 1 or a 0, and we can put this value in a new variable. We can ignore having nice variable names because we no longer need them. We only need our dummy variable; otherwise, we would need to go back and create filters to ensure that any requested pages that are duplicated are listed only once (required by the Transpose Task to create an ID for each unique value). Also, our expression will become very long if our data set is very wide (which becomes more likely in this case if users view a lot of different pages on our site).

RECODE A COLUMN

Alternatively, we could create our computed column using a Recoded column from the Query Builder's Computed Column dialog:

Figure 12. Recoding a Column



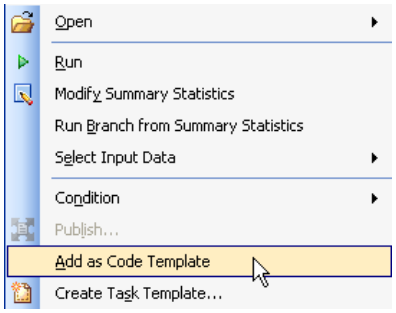


This selection will generate a CASE statement similar to this statement:

```
CASE
   WHEN t1.Col1 = '/faq3.html HTTP/1.1' THEN '1'
   ELSE '0'
END
```

The CASE statement is generated and not exposed for editing in this dialog box. However, as with the other Tasks, we could save the Task as a code template (see Figure 13) and edit the CASE statement.

Figure 13. Add as Code Template

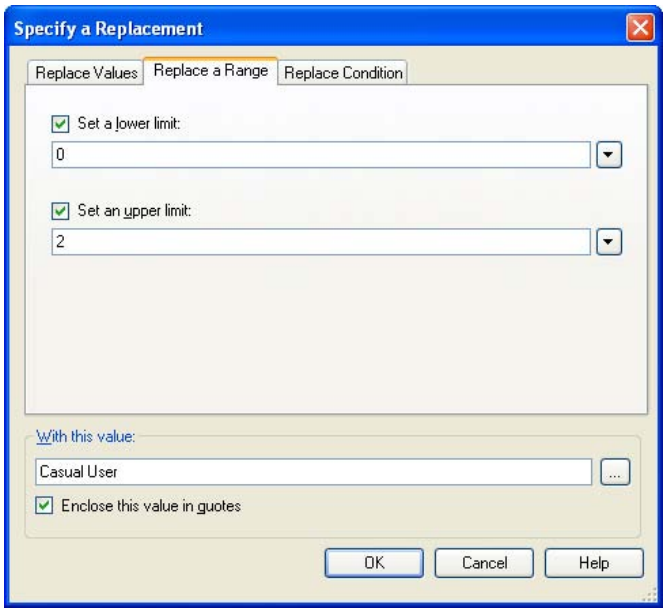


In this case it will be very handy to use an editor to edit either the CASE statement or our preceding expression. An advantage of editing the expression is that we can paste it back into our Expression Editor without creating a separate code template. On the other hand, the code template enables us to perform further customizations.

BINNING

Speaking of recoding, another useful task we can perform by recoding a column is binning to combine ranges into categories. For example, we might want to bin our new Frequency variable so that we have ranges of frequency: 0-2 visits per period might be defined as “Casual User;” 3-5 might be “Average User;” and 6 or more visits might be “Loyal User.” To perform this manual binning, we replace a range of values when we are recoding a column, as in Figure 14:

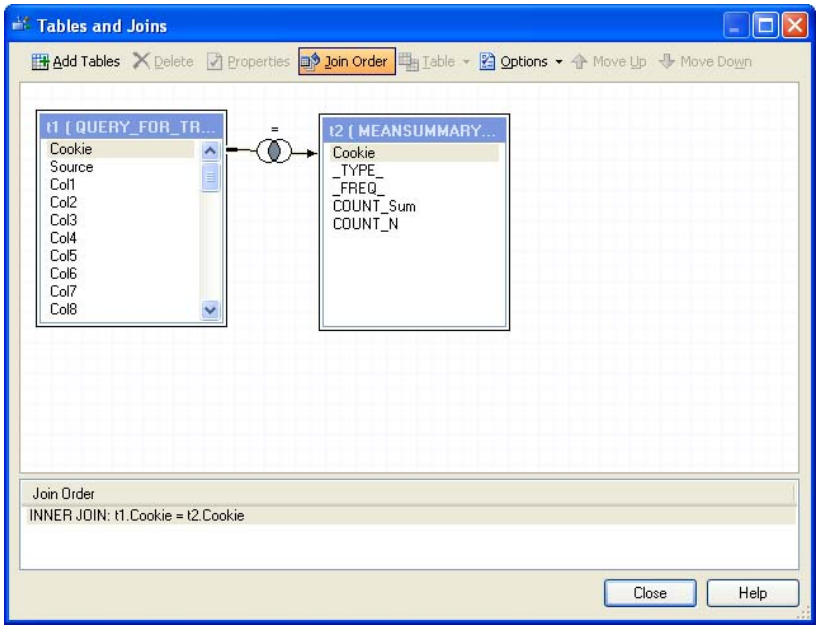
Figure 14. Binning: Replace a Range in Recoding a Column



JOINING OUR DATA

After creating these new variables, we are ready to bring them back together. We combine our frequency data, dummy variables, and other data that we might have created separately (for example, geographical data) into a new data set. We will input this data set into our prediction model. We use the join capability of the Query Builder to combine data sets with our favorite cookie as the common key. In this case we will use the default inner join type. There are several other join types built in. Figure 15 shows an example of an inner join.

Figure 15. Inner Join Using Query Builder



Our output looks similar to Figure 16:

Figure 16. Join Output

	Cookie	ViewFAQ	Frequency	Geography	Referrer
1	Cookie 1	0	Loyal User	South	Search Engine
2	Cookie 10	1	Loyal User	Central	Search Engine
3	Cookie 5	0	Loyal User	East	Banner Ad
4	Cookie 6	0	Loyal User	West	Email Campaign
5	Cookie 8	1	Average User	Asia Pac	Email Campaign

We will use this data as well as other data as input into our modeling of sign-up for our services. We are ready for our next analysis—we will use logistic regression, decision trees, or other methods by using SAS Enterprise Miner. Together with the association rules analysis, we should be able to use some interesting analytics to improve our site—serving better content to our users and increasing sign-up for our premium service.

CONCLUSION

We have walked through a few examples of ad hoc data preparation tasks using SAS Enterprise Guide, such as importing data, editing and creating new variables, creating string matching functions, recoding variables to create dummy variables and to bin variables, and transposing and joining data sets. There are of course many more possibilities that we could not cover in a short paper. We hope these examples will give you some ideas on how to use the productivity features of SAS Enterprise Guide at your site. We hope to update the paper with more examples in the future. We would love to hear from you about other topics to include or other suggestions.

REFERENCES

Cody, Ron. “An Introduction to Perl Regular Expressions in SAS 9.” Proceedings of the SAS Users Group International 2004 Conference. Montreal, Canada. Available http://www2.sas.com/proceedings/sugi29/265-29.pdf.

Darbha, Venkata R. 2007. “Validation Techniques in Clinical Trials Using Enterprise Guide.” *Proceedings of the Western Users of SAS Software 2007 Conference*. San Francisco, CA. Available http://www.lexjansen.com/wuss/2007/DataPresentationsBusIntell/DPBI_Darbha_ValidationTechniques.pdf.

McDaniel, Stephen, and Chris Hemedinger. 2006. *SAS For Dummies.* New York: Wiley.

Stuelpner, Janet. 2005. “Proc Transpose or How to Turn It Around.” *Proceedings of the Southeast SAS Users Group 2005 Conference.* Hilton Head, SC. Available http://analytics.ncsu.edu/sesug/2005/TU12_05.PDF.

Taylor, Matthew. 2007. “Taming the PROC TRANSPOSE.” *Proceedings of the Northeast SAS Users Group 2007 Conference*. Baltimore, MD. Available http://www.nesug.org/proceedings/nesug07/ff/ff14.pdf.

Svolba, Gerhard. “Efficient ‘One-Row-per-Subject’ Data Mart Construction for Data Mining.” *Proceedings of the SAS Users Group International 2006 Conference.* San Francisco, California. Available http://www2.sas.com/proceedings/sugi31/078-31.pdf.

Wikipedia. “HTTP cookies.” Available http://en.wikipedia.org/wiki/HTTP_cookie.

RECOMMENDED READING

Hemedinger, Chris. 2009. “What’s New in SAS Enterprise Guide 4.2.” *Proceedings of the SAS Global Forum 2009 Conference*. Cary, NC: SAS Institute Inc.

McDaniel, Stephen, and Chris Hemedinger. 2007. *SAS For Dummies.* New York: Wiley.

Refaat, Mamdouh. 2006. *Data Preparation for Data Mining Using SAS.* San Francisco: Morgan Kaufmann.

Slaughter, Susan J., and Lora D. Delwiche. 2006. *The Little SAS Book for Enterprise Guide 4.1.* Cary, NC: SAS Institute Inc.

Svolba, Gerhard. 2007. *Data Preparation for Analytics Using SAS*. Cary, NC: SAS Institute Inc.

CONTACT INFORMATION

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

I-kong Fu
SAS Institute Inc.
Work Phone: 919-677-8000
E-mail: I-kong.Fu@sas.com

Wayne Thompson
SAS Institute Inc.
Work Phone: 919-677-8000
E-mail: Wayne.Thompson@sas.com

Mike Porter
SAS Institute Inc.
Work Phone: 919-677-8000
E-mail: Mike.Porter@sas.com