Using SAS® to Build Customer Level Datasets for Predictive Modeling

Scott Shockley, Cox Communications, New Orleans, Louisiana

ABSTRACT

If you are using operational data to build datasets at the customer level, you're faced with the challenge of condensing a plethora of raw transactions into a data set that summarizes each customer, one row per customer. You will probably have to use multiple tables with different levels of granularity. Some of the data will change over time, and some of it won't. If the focus of your research is to predict events like customer defection, then changes over time will be a major consideration and make this process even more difficult.

The goal of this paper is to guide readers through the process of transforming raw data into a data set for predictive modeling that accurately represents a customer and the factors that could possibly impact the outcome being predicted. The paper will use specific examples like how to calculate derived variables based on complex conditions related to time. For example, it will show how to calculate an average billing amount over time, but only during the most recent uninterrupted period of customer tenure. The discussion mainly concerns the technical details, but also the business and mathematical logic behind the decisions being made.

INTRODUCTION

For the purposes of this paper, a customer level data set is one in which each customer is represented on its own row in a table so that customers can be the unit of analysis for a predictive model. To demonstrate the process of building this data set, examples will be based on the following business scenario: we want to build a logistic regression model that predicts customer defection. If you don't know what some of these terms mean, don't worry, the concepts are presented throughout the paper, and the primary focus is the SAS code.

MATHEMATICAL & BUSINESS CONTEXT

Logistic regression models are created to predict an outcome that has two mutually exclusive categories associated with it—a binary target (dependent variable). For example, is an insurance claim fraud, or not fraud? If we collect data leading up to the event in question and have a way to verify the true outcome, we can build a logistic regression model.

Customer defection, also known as churn, is a metric that describes the rate at which customers are defecting from a company. Measuring churn makes the most sense for subscription based businesses like internet service providers. Not only does a subscription require a lot of data collection, but there are discrete units of time in which to measure churn due to the monthly billing cycle, so customer churn is easy to define.

BUSINESS SCENARIO

Our fictional client provides web-based software to help businesses manage internet marketing campaigns. As you can see in Table 1 below, there is a row for every customer transaction in a given month. Our client has told us that the presence of a bill_date in this file means that the customer paid their bill that month, and we should use that to define whether or not an individual was a subscriber in a given month. Since the model we're developing predicts customer level churn, not product level churn, subscribing to one product is enough to classify an individual as a customer in a given month. Price is the retail price of the product, bill_amount is what the customer paid reflecting the promotional rate, and promotion is a binary variable where 1 means a promotion was in effect, and 0 means it wasn't.

customer_id	bill_date	customer_start_date	product	price	bill_amount	promotion
6	01/31/2012	03/11/2008	site-track-1	$45.00	$45.00	0
6	02/29/2012	03/11/2008	site-track-1	$45.00	$45.00	0
6	03/31/2012	03/11/2008	site-track-1	$45.00	$45.00	0
6	04/30/2012	03/11/2008	site-track-1	$45.00	$45.00	0
6	05/31/2012	03/11/2008	site-track-1	$45.00	$45.00	0
6	06/30/2012	03/11/2008	site-track-1	$45.00	$45.00	0
6	07/31/2012	03/11/2008	ad-manager-1	$30.00	$15.00	1
6	07/31/2012	03/11/2008	site-track-1	$45.00	$45.00	0
…	…	…	…	…	…	…

Table 1. Sample of Raw Data File.

The data represents all billing for every customer in the year 2012. The customer_start_date field represents the first time the customer signed up for service; we're not given any indication as to whether there are customer defections between that time and now. Table 1 does not contain every 2012 transaction for customer 6, but we can see that the customer's relationship with the company started on 03/11/2008, and that by July of 2012 the customer was subscribing to all services, ad-manager and site-track. The number at the end of each product label is the tier of service for that type of product, 1 being the lowest and 3 being the highest.

PREPARE THE TRANSACTION FILE

To make customer patterns easier to visually interpret, code and compress, we're going to transform the data from a long list of transactions, to a wide data set in which every month has its own column of data, but first we need to prepare the data for that transformation. First, execute the data step that provides the sample data set (see the first data step in Appendix). Since month is the unit of time that we are using to define churn, it makes sense to have a month variable that represents every month in our data set. The following data step creates a numeric month-year variable that is sortable and represents each month uniquely from year to year. The YEAR function returns a 4 digit numeric value describing the year of a date, the MONTH function performs similarly. The arithmetic is easy to understand: 2012*100+1=201201, a sortable description of every month in our data set.

```
data transactions;
       set transactions;
       YM=year(bill_date)*100+month(bill_date);
       one=1; /*We can ignore this variable for now, we'll use it later.*/
run;
```

Next we want to label each month in our observation period in order, 1 through n, with a variable called month. This will come in handy whenever we want to know a specific transaction's location in time relative to the observation period. First sort the data according to the year-month variable YM.

_N_ is a system variable that represents the Nth iteration of the data step below; the first IF-THEN statement creates a counter that starts off at 1. The counter is assigned to start at 1, but could be any value. As the data step reads through the sorted observations, we want the counter to accumulate another +1 every time YM changes. In the second IF-THEN statement the lag function identifies and subtracts the value of the previous YM record from the current YM record. If the difference is greater than 0, the counter equals 1 in that iteration. In the last statement month accumulates a 1 every time the counter variable identifies a difference between two consecutive YM records. In Figure 1 you can see that the month column only changes when YM does.

```
proc sort data=transactions;
       by  YM;
run;

data transactions (drop=counter);
       set transactions;
       if _N_ = 1 then counter=1;
       if YM-lag1(YM) > 0 then counter=1;
       month+counter;
run;
```

customer_id	bill_date	YM	month
1	01/31/2012	201201	1
1	01/31/2012	201201	1
2	01/31/2012	201201	1
4	01/31/2012	201201	1
4	01/31/2012	201201	1
4	01/31/2012	201201	1
5	01/31/2012	201201	1
5	01/31/2012	201201	1
6	01/31/2012	201201	1
1	02/29/2012	201202	2
1	02/29/2012	201202	2

Figure 1. Partial View of Transactions Data Set.

Knowing which products each customer subscribes to at a given time will help us describe each customer later on. Sometimes it's best to keep things simple and use product categories rather than individual products. In the following data step, the =: operator means "starts with." Any product name that starts with "site-track" will be added to the product group "site track", etc.

```
data transactions;
       set transactions;
       if product =:'site-track' then product_category='site track';
       else if product =:'manager' then product_category='ad manager';
run;
```

CREATE THE BASE CUSTOMER LEVEL DATA SET

Now our transactions are ready to be transformed into the base customer level data file—a data set with one record per row for each individual customer. It will be the foundation that we use to calculate metrics for our final customer level data set. The first step is to create a file that retains all of the static information in the transactions file—information that only varies by customere. Afterwards we can merge this file with data sets formed from other data steps until we have all the information that we need to calculate variables for model input.

CREATE BASE

Obviously we want to retain the customer_id and customer_start_date, but there's more static information that we can get from this file. Eventually we need to know what each customer's last (most recent) subscription month is in the observation period—we can get that information by using PROC SORT twice. In the first PROC SORT, use the KEEP statement to keep only the variables you want, use the OUT to name the new file. By sorting this way, if we were to eliminate every duplicate record based on customer_id, we would be left with the bill_date with the highest value, the most recent bill_date. The second PROC SORT uses the NODUPKEY option to ensure that after SAS reads a customer_id, it will delete any that follow it.

```
proc sort data=transactions out=base
       (keep=customer_id customer_start_date month bill_date);
       by customer_id descending bill_date;
run;

proc sort data=base out=base
       (rename=(bill_date=bill_date_last month=bill_month_last)) nodupkey;
       by customer_id;
run;
```

Obs	customer_id	bill_date_last	customer_start_date	bill_month_last
1	1	11/30/2012	10/10/2010	11
2	2	12/31/2012	03/03/2005	12
3	3	09/30/2012	02/29/2012	9
4	4	12/31/2012	08/13/2007	12
5	5	12/31/2012	10/01/2011	12
6	6	12/31/2012	03/11/2008	12
7	7	10/31/2012	03/07/2012	10

Figure 2. Data Set Base.

AGGREGATE, TRANSPOSE & MERGE DATA

Now we can perform a series of procedures that will transform elements of the transactions file into data sets that describe events over time horizontally, as opposed to the current vertical orientation of data.. Bill_date, for example, is an indication that an individual is a subscriber in that month. Not to say that you should look at every record of a million row file, but by looking at Table 2, it's easy to miss an interesting pattern. Customer 5 actually churns in April after the promotion ends, then resubscribes and gets another promotion.

customer_id	bill_date	customer_start_date	product	Price	bill_price	promotion
5	01/31/2012	10/01/2011	ad-manager-1	30	25	1
5	01/31/2012	10/01/2011	site-track-2	80	70	1
5	02/29/2012	10/01/2011	ad-manager-1	30	25	1
5	02/29/2012	10/01/2011	site-track-2	80	70	1
5	03/31/2012	10/01/2011	ad-manager-1	30	25	1
5	03/31/2012	10/01/2011	site-track-2	80	70	1
5	04/30/2012	10/01/2011	ad-manager-1	30	30	0
5	04/30/2012	10/01/2011	site-track-2	80	80	0
5	09/30/2012	10/01/2011	ad-manager-1	30	30	0
5	09/30/2012	10/01/2011	site-track-1	45	30	1

Table 2. Displaying Records Vertically.

In Table 3, however, you can easily see the customer's subscription pattern. It's easy to see when the customer defected, and how long it took for them to come back. Imagine you had millions of rows of data, customers in the hundred thousands, and your observation period was 2 years. Converting the data to this form is a trick that will not only help you wrap your mind around the analytical problem you're trying to solve, it's convenient for calculating metrics, and as we'll see later on.

customer_id	Month1	Month2	Month3	Month4	Month5	Month6	Month7	Month8	Month9
5	1	1	1	1	0	0	0	0	1

Table 3. Displaying Records Horizontally.

To make a data set like Table 3, we need to transform the month field into a series of 12 fields, with a binary indicator of whether or not a customer subscribed that month. Before we use PROC TRANSPOSE to perform this transformation, however, we need to remove duplicates so that the procedure knows which row to use in a column. By using the NODUPKEY option, PROC SORT will only keep one copy of a row with each unique customer_id/month combination. The one field that we created earlier will become the binary number in each horizontal month field. In PROC TRANSPOSE, PREFIX indicates the prefix for the names in the series of columns being created. BY is where we put the unique identifier for each row. ID becomes the suffix of the newly formed variables. The data step merges the base file with the newly created horizontal fields. _NAME_ is dropped—it's a variable created by PROC TRANSPOSE that we don't need.

```
proc sort data=transactions out=transactions_small
       (keep=customer_id month one) nodupkey;
       by customer_id month;
run;

proc transpose data=transactions_small out=month_status prefix=month;
       by customer_id;
       id month;
       var one;
run;

data base;
       merge base month_status (drop=_NAME_);
       by customer_id;
run;
```

Next we can apply the same concept to promotions—we want a series of fields that use a binary indicator to describe whether or not the customer had a promotion at a given point. We can repeat the same series of procedures except this time we need to do PROC SORT twice. The first ensures that the instances where promotion=1 are sorted higher than promotion=0, so that when we remove duplicates in the second PROC SORT, we'll retain the correct records.

```
proc sort data=transactions out=promotions (keep=customer_id month promotion);
       by customer_id month descending promotion;
run;

proc sort data=promotions nodupkey;
       by customer_id month;
run;
```

We can obtain product level churn patterns—just like month1-month12, except instead of a 1 or 0, we will indicate a quantity of products for each product group, which will provide two types of information stored in one group of variables: a number higher than 0 indicates the subscription to that product category in a given month, and the actual number indicates the quantity. The last two steps are just like the others, PROC TRANSPOSE and MERGE, but instead of a PROC SORT, we'll get the data we want using PROC SUMMARY. Think of this procedure as a generic pivot table in which we slice data by certain categories to get the type of information we want.

```
proc summary data=transactions
       (where=(product_category in ('ad manager'))) nway;
       var one;
       class customer_id month;
       output out=am sum=am_quantity;
run;
```

PROC SUMMARY will provide a data set with one row per customer per month subscribed to an ad manager product, with a sum that contains a quantity of ad manager products in a given month. In the first line of code a WHERE clause ensures that only ad manager products are being included here. The NWAY option forces the procedure only generate summary statistics for the specific combination of class variables that is specified, rather than all combinations which is the default. One is the VAR being used to calculate the sum. CLASS indicates that we want data at the customer_id/month level and SUM= tells the procedure a) that we want a sum and 2) it should be called am_quantity. We can do the same steps for any other product categories that exist as long as the new fields are named differently.

The exact same set of procedures can be executed to get bill totals over time—other than the variable names, the only change required is the VAR in PROC SUMMARY. Using bill_amount as the VAR will get calculate the total bill that a customer paid in each month, which will be helpful information to have down the line.

AGGREGATE & CLEAN DATA WITH DO LOOPS AND ARRAYS

Since there are still more 12-member variable groups to create, it would be premature to clean the base file at this point, but for the purposes of showing the easiest DO LOOP and ARRAY combination in this paper, we'll do it now.

At this point in our process we have a lot of variables that are blank, but should be 0; we can use a DO LOOP/ARRAY combination that replaces all the blanks with zeros, with very minimal code. For the purpose of this paper, we can think of arrays as a way to affect groups of variables at the same time with minimal code. In the following data step, the ARRAY statement is followed by the array name (which is month), the quantity of elements (variables in this case) is in braces, and at the end is a list of variables you want to include. If we wanted to include only month1, we would change {12} to {1} and instead of listing the range of variables, we would just put month1. On the next line the DO LOOP starts. i can be any letter you want it to be, it's just a way to refer to the iterations that the loop will do—we selected 1 to 12, 1 for each of the 12 variables in our array. Now, instead of writing 12 IF-THEN statements, one for each month variable, we can write one. Imagine the computer moving through the DO LOOP and fill in the blanks: the first iteration is telling SAS "if month1=. Then month1=0".

```
data base;
       set base;
       array month{12} month1-month12;
       do i = 1 to 12;
              if month{i}=. then month{i}=0;
       end;
       drop i;
run;
```

But we have many more than 12 variables to clean up! There's a solution to cleaning up all the numeric variables with almost no additional code. The data step below is a little different than the one above—this time we're replacing . with 0 in *all* numeric variables. Instead of explicitly naming the number of elements in the braces, * can be used to mean "any number." Rather than listing every range of variables we want to affect, writing _NUMERIC_ tells SAS we want this array to apply to all numeric variables. The only thing technically different in the DO LOOP is the range: we don't know the number of variables that will be affected but need to give the DO LOOP a range. The DIM function can be used—it returns the number of elements in your array.

```
data base;
       set base;
       array allnum{*} _NUMERIC_ ;
       do i=1 to dim(allnum);
              if allnum{i}=. then allnum{i}=0;
       end;
       drop i;
run;
```

Earlier we used PROC SUMMARY to get product quantities. Now we can get total product quantities using a DO LOOP/ARRAY combination, and create a binary variable for each product that signifies whether a customer has multiple products or not. In the first data step below, you can see that you can use a term like month, instead of i, to for the range of the do loop. In the loop, ad manager and site track quantities are added up to get a variable called prod_total. In this data step we're not only writing to the base file, but we're writing to the file productquants, because next we're going to change the values of the product quantities temporarily.

```
data productquants base;
	set base;
	array am_quant{12};
	array st_quant{12};
	array prod_total{12};
	do month = 1 to 12;
		prod_total{month}=am_quant{month}+st_quant{month};
	end;
	drop month;
run;
```

In the following data step, we're getting the multi-product variables. The logic of the IF-THEN statements is that we want a unique identifier for the presence of an am_quant{month}, and the presence of a st_quant{month}, then multi_prod{month} will be the sum, and will signify a bundle combination. 1 = ad manager only, 2 = site track only, 3 = both products. This may not seem very useful, but if there were 3 products it would be necessary, because there are 7 different ways to combine 3 products. If you assigned the values 1, 2, and 5 to each respective product, any of the 7 combinations would be accounted for as a unique positive integer. Below that we merge the multi_prod information with the base file.

```
data productquants;
	set productquants;
	array am_quant{12};
	array st_quant{12};
	array Multi_prod{12};
	do month = 1 to 12;
		if am_quant{month} ne 0 then am_quant{month} = 1;
		if st_quant{month} ne 0 then st_quant{month} = 2;
		multi_prod{month}=am_quant{month}+st_quant{month};
	end;
	drop month;
run;

data base;
	merge  base productquants (keep=customer_id multi_prod1-multi_prod12);
	by customer_id;
run;
```

We created the Month1-Month12 variables to represent an individual's status as subscriber or not a subscriber—but later we'll need to use those variables to signify more than subscriber status for that month. We want to know at what point in the customer relationship is the customer at any given time: beginning, middle, or end. A way to do that is replace the 1s and 0s in the month variables with specific numbers that describe each month better. This new variable will be called Status{i}, where 1 = "start month", 2 = "month after start", 3 = "month after defection" and 0 means nothing, it's an absence of 1, 2 or 3.

customer_id	Month1	Month2	Month3	Month4	Month5	Month6	Month7	Month8	Month9
5	1	1	1	1	0	0	0	0	1
customer_id	**Status1**	**Status2**	**Status3**	**Status4**	**Status5**	**Status6**	**Status7**	**Status8**	**Status9**
5	1	2	2	2	3	0	0	0	1

Table 4. Comparing Month{i} to Status{i}.

You can see in table 4 that in months 1 to 4, customer 4 was an active subscriber. In status 1 – 4 we give special labels to certain months—since Month1 is a start month, its status is 1. The 2s are just months where the customer is still active, but it's in the middle of their tenure. Status5 is a 3 because that's the month after defection.

The code to accomplish this is pretty simple: the two IF conditions describe the month, and the THEN labels it. Translating the first IF-THEN into English: "if the individual is a subscriber *this month*, but the customer wasn't a subscriber *last month*, then Status = 1." Another thing to note about this data step is that it uses *i – 1*, which means that it goes out of the ARRAY boundary and won't work unless you set the range at 2 – 12. It turns out that in every case, Status1 will be equal to Month1, so that is hard coded before the DO LOOP begins.

```
data base;
	set base;
	array Status{12};
	array month{12};
	status1=month1;
	do i = 2 to 12;
		if month{i} = 1 and month{i-1}= 0 then status{i} = 1;
		if month{i} = 1 and month{i-1} = 1 then status{i} = 2;
		if month{i} = 0 and month{i-1}= 1 then status{i} = 3;
		if month{i} = 0 and month{i-1} = 0 then status{i} = 0;
	end;
	drop i;
run;
```

COMBINE VARIABLE GROUPS INTO ONE SMALL VARIABLE

If any of our groups of 12 variables contain only positive integers, the numbers can be concatenated into a text string, and that one field will replace the 12 fields it was based off of. For the sake of brevity, the only code displayed below applies to the Month1-Month12 variables, but the full code is in the Appendix. The CATS function removes leading and trailing blanks, and concatenates fields. Notice that we can use OF to list the range of variables instead of specifically listing every variable.

```
data base (drop=month1-month12);
	length month_cat $12.;
	set base;
	month_cat=cats(of month1-month12);
run;
```

CREATE THE FINAL CUSTOMER LEVEL DATA SET

Now we begin to calculate the variables that will actually be used to build the logistic regression model. The first variable is churn, which is the dependent variable that we want to predict. The code is simple, anyone that's not a customer is month 12 is a defector. The SUBSTR function returns the 12th character in the month_cat string. If it equals 0 then churn = 1. Also note that the base data file has been renamed customer_level_dataset.

```
data customer_level_dataset;
	set base;
	if SUBSTR(month_cat, 12)=0 then churn=1;
	else churn=0;
run;
```

The next variable is called lifetime_tenure. The customer_start_date variable represents the first time the customer opened an account with the company. We have no way of knowing if there have been any defections between the customer_start_date and the 12 month observation period. So this variable doesn't represent the length of the customer's tenure leading up to defection, it just signifies the date that the company and its client started a relationship. For example, maybe customers that first experienced the company long ago have a negative perception of the company because its products were very low quality when the company started. For those types of reasons it's worth calculating this variable and investigating its potential to predict churn.

Since the last day in our observation period is 12/31/2012, we measure the lifetime_tenure by subtracting the customer_start_date from that. 19359 is the SAS date for 12/31/2012.

```
data customer_level_dataset;
	set customer_level_dataset;
	lifetime_tenure=19359-customer_start_date;
run;
```

CONSIDERATION OF TIME TO EVENT & VARIABLE ACCURACY

Next we'll calculate short_term_tenure. This refers to the most recent period of uninterrupted customer tenure, meaning that it's the last set of consecutive months as a subscriber that a customer has had. There are two very important reasons to calculate this variable. This first reason is that it could be a significant predictor of customer churn for the model—generally speaking, the longer a customer stays with you, the chances of defection decrease. Perhaps more importantly, most of the input variables that we calculate will depend on the customer's most recent period of tenure because events that happened in that period have the strongest likelihood of affecting the actual outcome. For example: imagine a customer subscribes to site-track, but the customer defects because it's a product that has a lot of competition and it's inferior to the alternatives. Perhaps a few months later that customer signs up for ad manager and likes the product—this is an innovative product, without much competition, and the customer thinks it can help them save money by automating some of their online marketing processes. In this case, if we made no distinction between the two time periods, we could produce variables that indicate the customer subscribed to both products, and that the customer is not a defector—but in reality the customer churned when he had site track. This misinformation would limit our model's ability to predict.

Even though the code isn't very difficult to understand, short_term_tenure is a very tricky variable to calculate. At first we might think that all we need to do is look at the status_cat string, find the *position* of most recent instance of 1 (the 1 furthest to the right), and subtract that from bill_month_last. For example, in Figure 3, short_term_tenure is equal to the 4 months highlighted in green, but if we subtract the position of the 1, that would be 8, and bill_month_last=12. 12-8=3, so we would have to add 1 to get to 4. *But this is still problematic*.

customer_id	bill_month_last	status_cat	
5	12	122230001222	
The 1 is in the 8th position ⟷			⟷ *The 2 is in position 12*

Table 5.

If we used the previously mentioned formula to calculate short_term_tenure there would be one major problem with the variable: for every single customer with short_term_tenure = 12, churn would always be equal to 0; the customer would never be classified as a defector. That type of information would help make the predictive model dysfunctional in practice. The model might think that if short_term_tenure = 12, the customer never defects, but this won't hold true in real life. It's extremely important to not feed your model inputs that contain the target that you're trying to predict. Once you realize this, it's pretty easy to avoid: any customer that was active every month in our observation period will automatically be assigned the number of months active minus 1, which is 11 for a 12 month observation period. See Table 6. That way, all customers with short_term_tenure of 11 months will be considered very loyal customers, but like in real life, some of them will be defectors and some of them will not.

customer_id	bill_month_last	status_cat	short_term_tenure
5	12	122230001222	4
6	12	122222222222	11

Table 6.

Customers where status_cat indicates 12 month tenure will be assigned 11 months, as you can see in the IF-THEN below. Behind the ELSE, the formula subtracts the position of the most recent start date from bill_month_last. We get the position of the start date using the FIND function which returns the position of any character that you request in a string. The last element in the function is -12, which tells find to start looking for '1' backwards, by starting from the last character in the status_cat string.

```
data customer_level_dataset;
       set customer_level_dataset;
       if status_cat='122222222222' then short_term_tenure=11;
       else short_term_tenure=bill_month_last-find(status_cat, '1',-12)+1;
run;
```

Next we want to get a series of binary variables that describe which product the customer had either right before defection or in the last observation period. There are 3 choices: ad manager, site track, or both, which we'll call bundle. We will use multi_prod_cat to calculate this since it tells us which products a customer had in each month.

This first component of this data step is a series of variables that use the FIND function—it finds the recent month that a customer subscribed to each respective product combination. For example, am is the month that ad manager was last subscribed to by a customer. Below that a series of IF-THEN-ELSE statements assign a value to each binary variable depending on which variable month is the greatest.

```
data customer_level_dataset (drop= am st bu);
       set customer_level_dataset;
       am=find(multi_prod_cat, '1',-12);
       st=find(multi_prod_cat, '2',-12);
       bu=find(multi_prod_cat, '3',-12);
       if bu > am and bu > st then do;
              am_have=1;
              st_have=1;
              bundle_have=1;
       end;
       else if am > st and am > bu then am_have=1;
       else if st > am and st > bu then st_have=1;
run; /*Don't forget to replace the . with 0 in the results.*/
```

We're trying to predict what led to the most recent defection, but that doesn't mean we should ignore the past. Having a count of prior defections can be an extremely powerful indicator of whether someone is likely to defect again, and it's easy to calculate. The COUNT function will return a count of '3' in status_cat. 3 is the first month after a customer drops service, so that will indicate prior defections. The IF-THEN logic says that if an individual isn't currently a defector, than prior_defections= the number of '3' found in status_cat. For those with churn = 1, we want to subtract 1 from that count, so that we're only counting defections that happened before the one we're trying to predict.

```
data customer_level_dataset;
       set customer_level_dataset;
       if churn=0 then prior_defections=count(status_cat,'3');
       else if churn=1 then prior_defections=(count(status_cat,'3')-1);
run;
```

We've seen various ways that we can use the concatenated strings to form variables that reflect the right period of time, but up to this point we have no way to deal with variables that we can't easily concatenate—things like bill amounts. Luckily, we can use DO LOOPS and ARRAYS to calculate measures based on these big groups of time-varying fields.

Our next calculation is Average Bill Amount, or ABA. We want to know how much each customer's bill was on average, but only during the period leading up to the outcome. Since there are 12 months in our observation period, the DO LOOP range will be 1 to 12, one for each month. The dummy array is there because in order for the IF-THEN statements to work, we need a variable whose value is equal to i at any given time. Since the values for dummy are 1-12, when the do loop is on iteration 3, for example, dummy{i} will be equal to 3 at that moment.

Please see the green numbers throughout the data step, and explained below, to walk through the DO LOOP.

```
data customer_level_dataset;
       set customer_level_dataset;
       format ABA dollar10.2;
       array dummy{12} (1 2 3 4 5 6 7 8 9 10 11 12);
       array BA{12};
         do i=1 to 12; /*1*/
       /*2*/ retain TBA 0;
       /*3*/ if dummy{i} >= find(status_cat, '1',-12)and dummy{i} <= bill_month_last
       /*4*/ then TBA=TBA+BA{i};
       /*5*/ ABA=(TBA/(bill_month_last-find(status_cat, '1',-12)+1));
         end;
       output;
       TBA=0;
drop i dummy1-dummy12;
run;
```

1. The DO LOOP just started for the first time—we're on iteration 1 of the loop, record 1 of the data file.
2. TBA, total billing amount is set to zero, so that we count only the billing amounts that we want to.
3. If the current month (determined by dummy) is within customer's most recent period of tenure…
4. Then TBA is equal to itself, plus whatever billing amount if found in BA{i}
5. Average Billing Amount is equal to TBA, divided by the number of months in the customer's most recent period of tenure.

MORE IDEAS

Typically when building a churn model like this, more data is given to you and you can adapt the code you learned here to those scenarios. If for example we had a call log file that described calls to customer service in the period leading up to the outcome being predicted, we could calculate counts of those calls, just like we calculated Average Billing Amount. Even if the information is in a different table, as long as it has a date and ID variable associated with it, the data can be aggregated then merged to the base file.

Also, you may find that you need to calculate a variable similar to Average Billing Amount, but need to specify different conditional programming for different members of your sample—to do so you can use SELECT-WHEN-DO to control the conditions in which certain DO LOOPS are used.

CONCLUSION

Knowing how to build models or write code is really important for developing predictive analytics, but what should not be understated is the value of understanding how your variables should be created and what they actually mean. Instead of gathering as many variables as possible, many of which are redundant, not interpretable or worthless, why not try first to squeeze all the possible explanatory power out of the most valuable data first? Not only will predictive models perform better, but the analyst will better understand the model, and the results will be more explainable, and therefore actionable. Hopefully this paper has given you the tools, both the SAS code and the logic, to apply this methodology to your own analytical problem.

REFERENCES

Cody, Ron. 2007. *Learning SAS® by Example*. Cary, NC: SAS Institute Inc.

UCLA: Statistical Consulting Group. *Introduction to SAS: How to reshape data long to wide using proc transpose.* From http://www.ats.ucla.edu/stat/sas/modules/ltow_transpose.htm. Accessed April 2, 2013.

UCLA: Statistical Consulting Group. *Introduction to SAS: Arrays in SAS.* From http://www.ats.ucla.edu/stat/sas/modules/ltow_transpose.htm. Accessed April 2, 2013.

Pattnaik, Susmita. 2012. *PROC SUMMARY Options Beyond the Basics.* Proceedings of the 2012 SESUG Conference.

ACKNOWLEDGMENTS

Thanks to the following: the faculty and staff of the Louisiana State University Master of Science in Analytics program, the 2013 MSA Class including Mhel Lazo and David Maradiaga, and my wife Heather Shockley.

CONTACT INFORMATION

scottmshockley {at} gmail . com
http://www.linkedin.com/in/shockley

APPENDIX

COMPLETE SAS PROGRAM

```
/*Read in transaction data*/
data transactions;
          retain customer_id  bill_date customer_start_date product price bill_amount promotion;
          format bill_date MMDDYY10. customer_start_date MMDDYY10. price dollar10.2 bill_amount dollar10.2;
          input customer_id $ bill_date MMDDYY10. customer_start_date :MMDDYY10. product :$12. price bill_amount promotion 1. ;
          datalines;
1 01/31/2012 10/10/2010 ad-manager-1 30 30 0
1 01/31/2012 10/10/2010       site-track-1 45     45 0
1 02/29/2012 10/10/2010       ad-manager-1 30     30 0
1 02/29/2012 10/10/2010       site-track-1 45     45 0
1 04/30/2012 10/10/2010       ad-manager-1 30     30 0
1 04/30/2012 10/10/2010       site-track-1 45     45 0
1 05/31/2012 10/10/2010       ad-manager-1 30     30 0
1 05/31/2012 10/10/2010       site-track-1 45     45 0
1 08/31/2012 10/10/2010       ad-manager-1 30     30 0
1 08/31/2012 10/10/2010       site-track-1 45     45 0
1 09/30/2012 10/10/2010       ad-manager-1 30     30 0
1 10/31/2012 10/10/2010       ad-manager-1 30     30 0
1 11/30/2012 10/10/2010       ad-manager-1 30     30 0
1 11/30/2012 10/10/2010       site-track-1 45     45 0
2 01/31/2012 03/03/2005       ad-manager-1 30     30 0
2 02/29/2012 03/03/2005       ad-manager-1 30     30 0
2 03/31/2012 03/03/2005       ad-manager-1 30     30 0
2 04/30/2012 03/03/2005       ad-manager-1 30     30 0
2 05/31/2012 03/03/2005       ad-manager-1 30     30 0
2 06/30/2012 03/03/2005       ad-manager-1 30     30 0
2 07/31/2012 03/03/2005       ad-manager-1 30     30 0
2 08/31/2012 03/03/2005       ad-manager-1 30     30 0
2 09/30/2012 03/03/2005       ad-manager-1 30     30 0
2 10/31/2012 03/03/2005       ad-manager-1 30     30 0
2 11/30/2012 03/03/2005       ad-manager-1 30     30 0
2 12/31/2012 03/03/2005       ad-manager-1 30     30 0
3 02/29/2012 02/29/2012       ad-manager-2 35     28 1
3 02/29/2012 02/29/2012       site-track-3 99 99 0
3 03/31/2012 02/29/2012       ad-manager-2 35     28 1
3 03/31/2012 02/29/2012       site-track-3 99 99 0
3 04/30/2012 02/29/2012       ad-manager-2 35     28 1
3 04/30/2012 02/29/2012       site-track-3 99 99 0
3 05/31/2012 02/29/2012       ad-manager-2 35     35 0
3 05/31/2012 02/29/2012       site-track-3 99 99 0
3 06/30/2012 02/29/2012       ad-manager-2 35     35 0
3 06/30/2012 02/29/2012       site-track-3 99 99 0
3 07/31/2012 02/29/2012       ad-manager-2 35     35 0
3 07/31/2012 02/29/2012       site-track-3 99 99 0
3 08/31/2012 02/29/2012       ad-manager-2 35     35 0
3 08/31/2012 02/29/2012       site-track-3 99 99 0
3 09/30/2012 02/29/2012       ad-manager-2 35     35 0
3 09/30/2012 02/29/2012       site-track-3 99 99 0
4 01/31/2012 08/13/2007       ad-manager-1 30     30 0
4 01/31/2012 08/13/2007       ad-manager-2 35     35 0
4 01/31/2012 08/13/2007       site-track-2 80     80 0
4 02/29/2012 08/13/2007       ad-manager-1 30     30 0
4 02/29/2012 08/13/2007       ad-manager-2 35     35 0
4 02/29/2012 08/13/2007       site-track-2 80     80 0
4 03/31/2012 08/13/2007       ad-manager-1 30     30 0
4 03/31/2012 08/13/2007       ad-manager-2 35     35 0
4 03/31/2012 08/13/2007       site-track-2 80     80 0
4 04/30/2012 08/13/2007       ad-manager-1 30     30 0
4 04/30/2012 08/13/2007       ad-manager-1 30     30 0
4 04/30/2012 08/13/2007       site-track-2 80     80 0
4 05/31/2012 08/13/2007       ad-manager-1 30     30 0
4 05/31/2012 08/13/2007       ad-manager-1 30     30 0
4 05/31/2012 08/13/2007       site-track-2 80     80 0
4 06/30/2012 08/13/2007       ad-manager-1 30     30 0
4 06/30/2012 08/13/2007       ad-manager-1 30     30 0
4 06/30/2012 08/13/2007       site-track-2 80     80 0
4 07/31/2012 08/13/2007       ad-manager-1 30     30 0
4 07/31/2012 08/13/2007       ad-manager-1 30     30 0
4 07/31/2012 08/13/2007       site-track-2 80     80 0
4 08/31/2012 08/13/2007       ad-manager-1 30     30 0
4 08/31/2012 08/13/2007       ad-manager-1 30     30 0
4 08/31/2012 08/13/2007       site-track-2 80     80 0
4 09/30/2012 08/13/2007       ad-manager-1 30     30 0
4 09/30/2012 08/13/2007       ad-manager-1 30     30 0
4 09/30/2012 08/13/2007       site-track-2 80     80 0
4 10/31/2012 08/13/2007       ad-manager-1 30     30 0
4 10/31/2012 08/13/2007       ad-manager-1 30     30 0
4 10/31/2012 08/13/2007       site-track-2 80     80 0
4 11/30/2012 08/13/2007       ad-manager-1 30     30 0
4 11/30/2012 08/13/2007       ad-manager-1 30     30 0
4 11/30/2012 08/13/2007       site-track-2 80     80 0
4 12/31/2012 08/13/2007       ad-manager-1 30     30 0
4 12/31/2012 08/13/2007       ad-manager-1 30     30 0
4 12/31/2012 08/13/2007       site-track-2 80     80 0
5 01/31/2012 10/01/2011       ad-manager-1 30     25 1
5 01/31/2012 10/01/2011       site-track-2 80     70 1
5 02/29/2012 10/01/2011       ad-manager-1 30     25 1
5 02/29/2012 10/01/2011       site-track-2 80     70 1
5 03/31/2012 10/01/2011       ad-manager-1 30     25 1
```

```
5 03/31/2012 10/01/2011       site-track-2 80     70 1
5 04/30/2012 10/01/2011       ad-manager-1 30     30 0
5 04/30/2012 10/01/2011       site-track-2 80     80 0
5 09/30/2012 10/01/2011       ad-manager-1 30     30 0
5 09/30/2012 10/01/2011       site-track-1 45     30 1
5 10/31/2012 10/01/2011       ad-manager-1 30     30 0
5 10/31/2012 10/01/2011       site-track-1 45     30 1
5 11/30/2012 10/01/2011       ad-manager-1 30     30 0
5 11/30/2012 10/01/2011       site-track-1 45     30 1
5 12/31/2012 10/01/2011       ad-manager-1 30     30 0
5 12/31/2012 10/01/2011       site-track-1 45     45 0
6 01/31/2012 03/11/2008       site-track-1 45     45 0
6 02/29/2012 03/11/2008       site-track-1 45     45 0
6 03/31/2012 03/11/2008       site-track-1 45     45 0
6 04/30/2012 03/11/2008       site-track-1 45     45 0
6 05/31/2012 03/11/2008       site-track-1 45     45 0
6 06/30/2012 03/11/2008       site-track-1 45     45 0
6 07/31/2012 03/11/2008       ad-manager-1 30     15 1
6 07/31/2012 03/11/2008       site-track-1 45     45 0
6 08/31/2012 03/11/2008       ad-manager-1 30     15 1
6 08/31/2012 03/11/2008       site-track-1 45     45 0
6 09/30/2012 03/11/2008       ad-manager-1 30     15 1
6 09/30/2012 03/11/2008       site-track-1 45     45 0
6 10/31/2012 03/11/2008       ad-manager-1 30     30 0
6 10/31/2012 03/11/2008       site-track-1 45     45 0
6 11/30/2012 03/11/2008       site-track-1 45     45 0
6 12/31/2012 03/11/2008       site-track-1 45     45 0
7 03/31/2012 03/07/2012       ad-manager-1 30     12 1
7 04/30/2012 03/07/2012       ad-manager-1 30     12 1
7 05/31/2012 03/07/2012       ad-manager-1 30     12 1
7 06/30/2012 03/07/2012       ad-manager-1 30     15 0
7 08/31/2012 03/07/2012       ad-manager-1 30     15 1
7 09/30/2012 03/07/2012       ad-manager-1 30     15 1
7 10/31/2012 03/07/2012       ad-manager-1 30     15 1
;run;
/*---PREPARE THE TRANSACTION FILE---*/
/*create YM*/
data transactions;
          set transactions;
          YM=year(bill_date)*100+month(bill_date);
          one=1; /*We can ignore the variable One for now, we'll use it later.*/run;
/*give each transaction a month label*/
proc sort data=transactions; By YM; run;
data transactions (drop=counter);
          set transactions;
          if _N_ = 1 then counter=1;
          if YM-lag1(YM) > 0 then counter=1;
          month+counter;run;
/*create product category*/
data transactions;
          set transactions;
          if product =:'site-track' then product_category='site track';
          else if product =:'ad-manager' then product_category='ad manager';run;
/*---CREATE THE BASE CUSTOMER LEVEL DATA SET---*/
/*CREATE BASE*/
proc sort data=transactions out=base
          (keep=customer_id customer_start_date month bill_date);
          by customer_id descending bill_date;run;
proc sort data=base out=base
          (rename=(bill_date=bill_date_last month=bill_month_last)) nodupkey; by customer_id;run;
/*AGGREGATE, TRANSPOSE & MERGE DATA*/
/*monthly binary status*/
proc sort data=transactions out=transactions_small (keep=customer_id month one) nodupkey;
          by customer_id month;run;
proc transpose data=transactions_small out=month_status prefix=month;
          by customer_id;
          id month;
          var one;run;
data base;
          merge base month_status (drop=_NAME_);
          by customer_id;run;
/*promotion status*/
proc sort data=transactions out=promotions (keep=customer_id month promotion);
          by customer_id month descending promotion;run;
proc sort data=promotions nodupkey;
          by customer_id month;run;
proc transpose data=promotions out=promo_wide prefix=Promotion;
          by customer_id;
          id month;
          var promotion;run;
data base;
          merge base promo_wide (drop=_NAME_);
          by customer_id;run;
/*ad manager -quantity*/
proc summary data=transactions (where=(product_category in ('ad manager'))) nway;
          var one;
          class customer_id month;
          output out=am sum=am_quantity;run;
proc transpose data=am out=am_wide prefix=am_Quant;
          by customer_id;
          id month;
          var am_quantity;run;
data base;
          merge base am_wide (drop=_NAME_);
          by customer_id;run;
```

```
/*site track -quantity*/
proc summary data=transactions (where=(product_category in ('site track'))) nway;
          var one;
          class customer_id month;
          output out=st sum=st_quantity;run;
proc transpose data=st out=st_wide prefix=st_Quant;
          by customer_id;
          id month;
          var st_quantity;run;
data base;
          merge base st_wide (drop=_NAME_);
          by customer_id;run;
/*total bill amount*/
proc summary data=transactions nway;
          var bill_amount;
          class customer_id month;
          output out=BA sum=Total_Bill_Amount;run;
proc transpose data=BA out=BA_wide prefix=BA;
          by customer_id;
          id month;
          var Total_Bill_Amount;run;
data base;
          merge     base BA_wide (drop=_NAME_);
          by customer_id;run;
/*Total Price*/
proc summary data=transactions nway;
          var price;
          class customer_id month;
          output out=P sum=Total_Price;run;
proc transpose data=P out=P_wide prefix=P;
          by customer_id;
          id month;
          var Total_Price;run;
data base;
          merge     base P_wide (drop=_NAME_);
          by customer_id;run;
/*AGGREGATE & CLEAN DATA WITH DO LOOPS AND ARRAYS*/
/*replace missing with 0*/
data base;
          set base;
          array month{12} month1-month12;
          do i = 1 to 12;
                    if month{i}=. then month{i}=0;
          end;
          drop i;run;
data base;
          set base;
          array allnum{*} _numeric_ ;
          do i=1 to dim(allnum);
                    if allnum{i}=. then allnum{i}=0;
          end;
          drop i;run;
/*get product quantities*/
data productquants base;
          set base;
          array am_quant{12};
          array st_quant{12};
          array prod_total{12};
          do month = 1 to 12;
                    prod_total{month}=am_quant{month}+st_quant{month};
          end;
          drop month;run;
data productquants;
          set productquants;
          array am_quant{12};
          array st_quant{12};
          array Multi_prod{12};
          do month = 1 to 12;
                    if am_quant{month} ne 0 then am_quant{month} = 1;
                    if st_quant{month} ne 0 then st_quant{month} = 2;
                    multi_prod{month}=am_quant{month}+st_quant{month};
          end;
          drop month;run;
data base;
          merge     base productquants (keep=customer_id multi_prod1-multi_prod12);
          by customer_id;run;
/*create status_cat*/
data base ;
          set  base;
          array Status{12};
          array month{12};
  status1=month1;
          do i = 2 to 12;
                    if month{i} = 1 and month{i-1}= 0 then status{i} = 1;
                    if month{i} = 1 and month{i-1} = 1 then status{i} = 2;
                    if month{i} = 0 and month{i-1}= 1 then status{i} = 3;
                    if month{i} = 0 and month{i-1} = 0 then status{i} = 0;
          end;
          drop i;run;
/*COMBINE VARIABLE GROUPS INTO ONE SMALL VARIABLE*/
/*concatenate variables*/
data base (drop=multi_prod1-multi_prod12 promotion1-promotion12 status1-status12 am_quant1-am_quant12 month1-month12 prod_total1-
prod_total12 st_quant1-st_quant12);
          length month_cat $12. multi_prod_cat $12. prod_total_cat $12. am_quant_cat $12. st_quant_cat $12. month_cat $12.
promotion_cat  $12. status_cat $12.;
```

```
          set base;
          multi_prod_cat=cats(of multi_prod1-multi_prod12);
          prod_total_cat=cats(of prod_total1-prod_total12);
          am_quant_cat=cats(of am_quant1-am_quant12);
          st_quant_cat=cats(of st_quant1-st_quant12);
          month_cat=cats(of month1-month12);
          promotion_cat=cats(of promotion1-promotion12);
          status_cat=cats(of status1-status12);run;
/*---CREATE THE FINAL CUSTOMER LEVEL DATA SET---*/
/*create churn*/
data customer_level_dataset;
          set base;
          if SUBSTR(month_cat, 12)=0 then churn=1;
          else churn=0;run;
/*life time tenure*/
data customer_level_dataset;
          set customer_level_dataset;
          lifetime_tenure=19359-customer_start_date; run;
/*short term tenure*/
data customer_level_dataset;
          set customer_level_dataset;
          if status_cat='122222222222' then short_term_tenure=11;
          else short_term_tenure=bill_month_last-find(status_cat, '1',-12)+1;run;
/*product binary variables*/
data customer_level_dataset (drop= am st bu);
          set customer_level_dataset;
          am=find(multi_prod_cat, '1',-12);
          st=find(multi_prod_cat, '2',-12);
          bu=find(multi_prod_cat, '3',-12);
          if bu > am and bu > st then do;
                    am_have=1;
                    st_have=1;
                    bundle_have=1;
          end;
          else if am > st and am > bu then am_have=1;
          else if st > am and st > bu then st_have=1;run;/*Don't forget to replace the . with 0 in the results.*/
/*prior_defections*/
data customer_level_dataset;
          set customer_level_dataset;
          if churn=0 then prior_defections=count(status_cat,'3');
          else if churn=1 then prior_defections=(count(status_cat,'3')-1); /*if the customer is a churner in month 12, then their
defection count is one less than the amount of 3s we count in status_cat */run;
/*calculate average bill amount, ABA*/
data customer_level_dataset;
          set customer_level_dataset;
          format ABA dollar10.2;
          array dummy{12} (1 2 3 4 5 6 7 8 9 10 11 12);
          array BA{12};
       do i=1 to 12; /*1*/
          /*2*/ retain TBA 0;
          /*3*/ if dummy{i} >= find(status_cat, '1',-12)and dummy{i} <= bill_month_last
          /*4*/ then TBA=TBA+BA{i};
          /*5*/ ABA=(TBA/(bill_month_last-find(status_cat, '1',-12)+1));
            end;
          output;
          TBA=0;
drop i dummy1-dummy12;run;
data customer_level_dataset;
          set customer_level_dataset;
          array allnum{*} _numeric_ ;
          do i=1 to dim(allnum);
                    if allnum{i}=. then allnum{i}=0;
          end;
          drop i;run;
```