

Decision Tree Classification in Direct Marketing

Robert Mengis, Bear Creek Corp. IS, Medford, OR

ABSTRACT

The SAS Institute's %TREEDISC macro is implemented in a client-server, Windows 95-UNIX, SAS/AF® application context. This paper discusses a Direct Marketing promotion response model application of the macro with regard to variable selection and formatting, performance optimization, tree generation, tree display, classification of test data, analysis of leaf nodes, and the application logic to manage and control the modeling process.

BACKGROUND

I was asked late last year to develop a CHAID (Chi-squared Automatic Interaction Detection) adjunct component to the current scoring development methodology at Bear Creek (BC) for exploratory data purposes, i.e., the marketing analysts wanted to know more about their predictor variables than their regression models were telling them. So having heard about the %TREEDISC macro at PNWSUG'97, and because we already had a SAS/AF modeling interface running on Windows 95 and AIX®, I decided to integrate a SAS CHAID analysis into the present methodology. The marketing users had been using another Windows based CHAID product, that was apparently easy to use for small data set analyses. So I knew that the user interface would have to be fairly simple and mostly automatic.

Requirements

The user documentation is downloaded from the SAS web site as commentary in the %TREEDISC macro, which is in the SAS® Sample Library. Some of the highlights are: you must have SAS® 6.08 or later, SAS/IML® is required, you need the %XMACRO macros from SAS/STAT®, and to graphically draw the tree you need SAS/OR®, although in this paper I will show a way of graphical display using a SAS/AF object.

Method

The problem is to predict promotion response, in our case, from various marketing indicators and variables. The data is formed into subsets based on the categories of the most influential variable. Then these subsets are subsequently split along another predictor variable's discrete classes, and the

partitioning continues until some significance criterion is reached. What is formed as a result is a hierarchical tree structure with the subsets being the nodes of the tree. The basis for partitioning the subsets at each stage is the variable that is most significantly associated with the response through a chi-squared test in a contingency table. The stopping criterion is the p-value of the chi-squared test. See table 1 below.

Table 1. Sample Contingency Table.

Ordered Previously X Response	0	1	Total
0	57821	1269	59090
1	4058	618	4676
Total	61879	1887	63766

The chi-squared statistic for this table is 1848.7, p-value of 0.001. One can conclude that ordering previously has a very significant influence on responding to this promotion, hence two subsets would be formed, one with 59,090 members who did not order previously versus one with 4676 population who did order previously. Then each of these subsets would be crossed independently with the next most significant variable, etc., until the p-value of the chi-squared test exceeded the specified tolerance at which point the node or subset would become a leaf node. For more information on the algorithm in %treedisc that is used to select the next most significant partition see the documentation in the macro on the SAS Sample Library at http://www.sas.com/techsup/download/sample/samplelib/imlsampSASIML_Sample_Programs.html.

Performance Considerations

The instructions in the macro mention that up to two billion observations could be modeled in theory, but that it had been tested with up to a million observations. Also the cap on variables is 4,000 and it is recommended that you do not format the predictors to categorize them. It is also recommended to keep the number of categories "fairly small" for predictors to reduce computer time and memory. In my personal experience on an IBM RS6000-58H with 256 MB of memory with data sets on the order of 100,000s observations and up to 600 variables, it was very easy to exhaust memory and die. I never exercised the "formatted" option because of the caution that it was less efficient, and because

most of our case data was not formatted anyway. I also saw a post on the SAS-L List-Server (sas-l@uga.cc.uga.edu) about a problem with the "formatted" option not working correctly. I realized early on that one could not just run wide open with all predictor variables—even after categorization—without running out of memory. I found a way to subset the data set with the variables I wanted, and at the same time, I tried to keep the categories of the predictors to ten or less through various transformations. Also you cannot run the %TREEDISC macro on "compressed" data sets because SAS/IML is doing direct access on the observations. In conclusion, memory is definitely the bottleneck. Even running with "memsize=0M" one must find transformations for continuous variables to keep the number of categories low, less than or equal to 10.

VARIABLE PREPARATION

Nominal vs. Ordinal

To use %TREEDISC a predictor variable must be identified as either "nominal" or "ordinal". There is also a type of variable called "ordinal with a float" to allow for some continuous chaff in the variable range. To determine variable type the Meta-data of the data set was used to add each variable to the nominal or ordinal list. By Meta-data, I mean any data that is available about the specific variables or columns in the data set or database. In our case, the Meta-data amounted to the variable name and label and optionally the length, i.e., all nominal variables had length of three. Hence a routine was constructed to examine the name structure and the label text of each variable in order to classify it as either nominal or ordinal. See Appendix 1 for code example.

Variable Subset

The marketing users wanted to know more about the predictor variables that were coming into their stepwise regression models or that were finally selected as comprising the logistic regression model. Hence these variable selections were offered up as two options, e.g., initial or final regression variables, for input to the tree creation process. These sets of variables were very manageable, usually being 30 variables or less. There is also a third option where the user selects the predictor variables of interest.

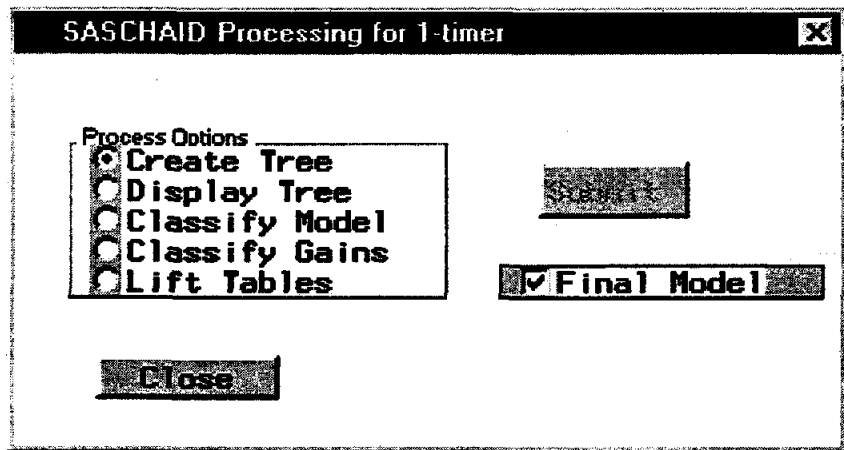
Variable Transformation

All the continuous ordinal variables in the variable subset had to be transformed into categorical variables with few (≤ 10) classes. Again the Meta-data was used to control what transformation occurred. Some variables simply needed rounding to some significant digits to become discrete. Some, particularly sales-related data, had to be categorized by in stream format generation. This was accomplished by building a decile format out of the "total sales" variable because it was largest in dollar magnitude, and then transforming all sales-related variables in the subset with this format. It is critical to be unit--consistent in variable transformation, e.g., transform continuous dollars to ordinal dollars, not some other arbitrary measure or level, because when the tree is used for classification there will be no need to transform the original unit to the new unit of measure before classification. Hence for tree construction a preparatory data set is generated with a subset of nominal and transformed ordinal predictor variables. See Appendix 1 for code example.

APPLICATION FLOW

The window in the SAS/AF application to manage the %TREEDISC implementation is in Figure 1.

Figure 1.



All process options are effected with a batch submit to the server except the "Display Tree" process which is a synchronous display of the tree created in the first process. The "Final Model" checkbox indicates take the final logistic-regression, model independent variables as the predictor-set input to the "Create Tree" process. Otherwise take the initial stepwise regression variables as the predictor-set. Each process option features a call to %TREEDISC except "Display Tree" which just displays the tree in graphical form. So in short, a tree is created, displayed graphically, used to classify either test or training data sets, and then reports are generated on the resulting leaf nodes or segmentation if you will. All data is stored on the server, but some is passed to the client, e.g., like to display the tree.

TREE CREATION

In the "Create Tree" option, the data are prepared as discussed above featuring designation of nominal and ordinal lists, automatic categorization of continuous variables, and a data set prepared for input to the %TREEDISC macro. A sample call is shown below.

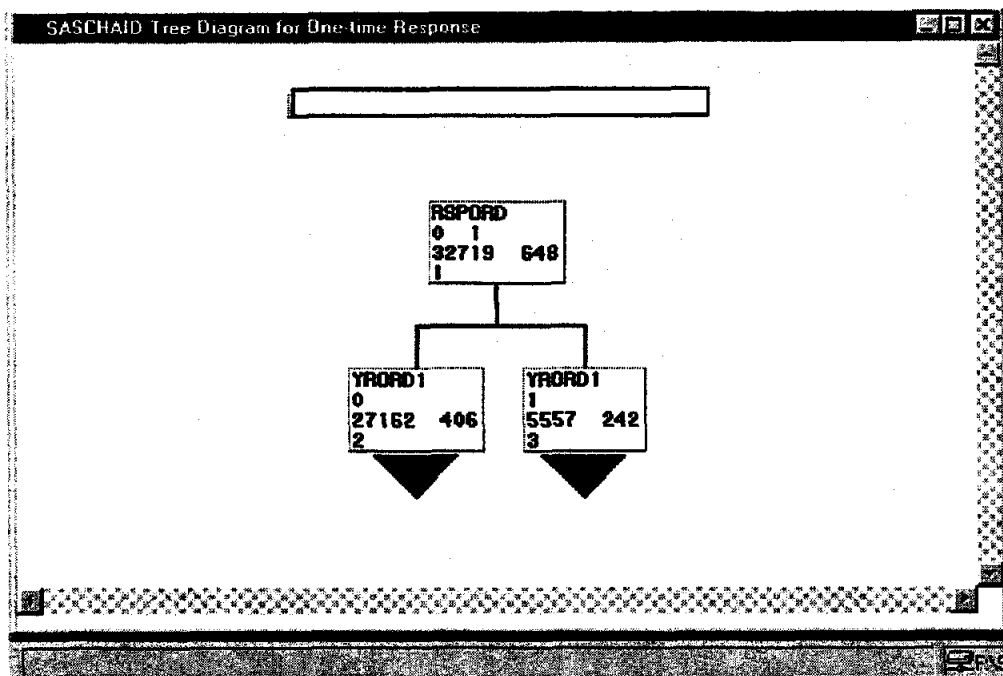
```
%treedisc(data=&chiname, depvar=rspond,
nominal=&nomlist,
ordinal=&ordlist, leaf=100, maxdepth=5,
outtree=c.&treename, options=noformat list);
```

The parameters are in order: the input SAS data set, the dependent variable (in our case response to a promotion=1, 0 otherwise), the nominal variable list, the ordinal variable list, a minimum population for leaf nodes(100), the maximum depth of tree excluding the root node(5), the output tree data set name, and the options list including "noformat" and "list"(meaning to print the tree). The output tree data set is the driver or input data set for all the subsequent steps. This data set has all the information that anyone could want about each of the nodes in the tree. The "leaf=" and "maxdepth=" options restrict the size of the tree as the marketing users wanted to keep things as simple as possible. A partial listing of a tree is seen in Appendix 2. This step is the step where the process is most likely to fail for memory shortage.

TREE DISPLAY

The "Display Tree" option drops the tree SAS data set into the Organizational Chart SAS/AF object in a frame. There is a great deal of information to choose from that can be displayed in the nodes. I settled on the sub-setting predictor variable name, the variable value, the response distribution, and the node number. See Figure 2.

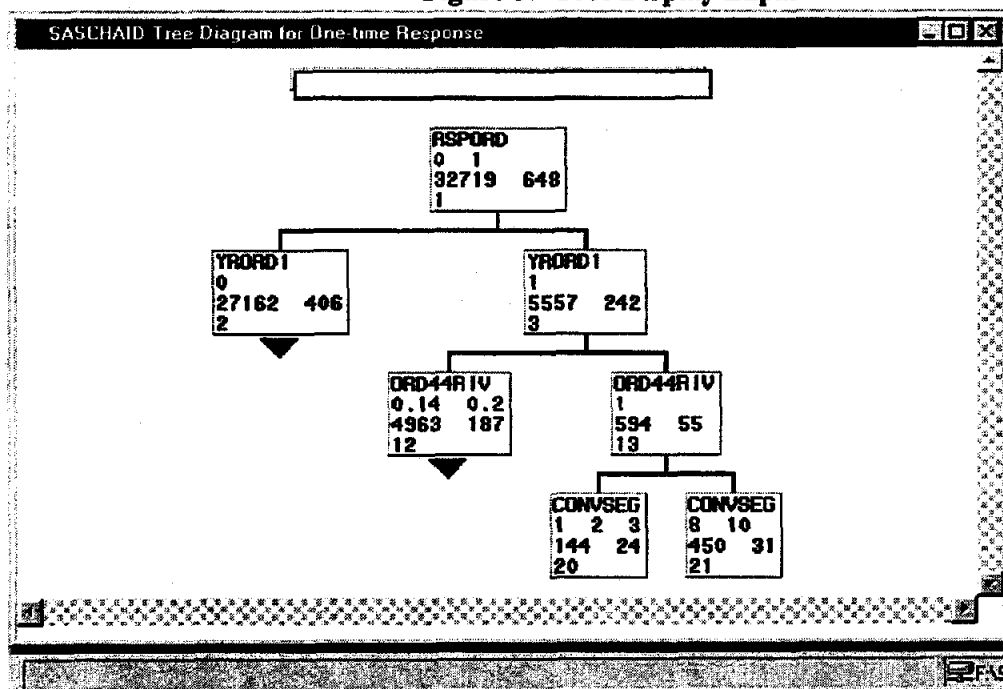
Figure 2. - Tree Display Frame



In Figure 2 above, node 1 is the root node, RSPORD is the dependant binary response variable, YRORD1 is the first splitting predictor variable, node 2 has 27,568 members and a YRORD1 value of "0", node 3 has a population of 5799 and a YRORD1 value of "1". The subsequent depth of the sub-trees can be

displayed by double clicking on the arrows. The overall response for this segment is 2%(node 1), while the customers in node 2 are expected to better respond, i.e., expected response=4%. So pursuing this line of the tree results in leaf nodes 20 and 21 as seen in Figure 3 below.

Figure 3. -Tree Display Expansion



So node 20 where customers have predictor attributes of {YRORD1=1, ORD44RIV=1, CONVSEG in (1,2,3)} and an expected response of 14%, is probably the best responding segment of the population, although the other leaf nodes have to be examined. From the graphical display one can get a good picture of what variables have key influence on promotion response. Clearly this tree is biased toward the data used to construct the model, so a classification of training data should be undertaken next. The code to prepare the tree data set for the Organizational Chart AF object is in Appendix 3. It is not overly complicated.

DATA CLASSIFICATION

In the regression modeling methodology at Bear Creek Corp. a test data set is set aside from the original sample for testing and classification. It is called the "Gains" data set, as in Figure 1. The %TREEDISC macro is called again, this time with the tree model as input, and code is generated to include in a classification data step. See below.

```
%treedisc(intree=c.&treenom, code=chcode)
data c.cls&k4box
(label="Validation-Data Classify of
%trim(&lablt)-timers");
set sel;
%include chcode;
run;
```

In this case, "chcode" is a file reference to a server directory where the classification code is written based on the "intree=" tree data set. A snippet of this code is provided below.

```
*** TREEDISC Decision Tree Code ***;
* DV counts: 32719 648 ;
node_ = 1;
into_ = 0;
tie_ = 0;
post_ = 0.9805796146;
if -1e38 <= YRORD1 <= 0.5 then do;
* DV counts: 27162 406 ;
node_ = 2;
into_ = 0;
tie_ = 0;
post_ = 0.98527278;

end;
else
if 0.5 <= YRORD1 then do;
* DV counts: 5557 242 ;
node_ = 3;
into_ = 0;
```

```
tie_ = 0;
post_ = 0.958268667;
if -1e38 <= ORD44RIV <= 0.6 then do;
* DV counts: 4963 187 ;
node_ = 12;
into_ = 0;
tie_ = 0;
post_ = 0.9636893204;

end;
else
if 0.6 <= ORD44RIV then do;
* DV counts: 594 55 ;
node_ = 13;
into_ = 0;
tie_ = 0;
post_ = 0.9152542373;
if -1e38 <= CONVSEG <= 5.5 then
do;
* DV counts: 144 24 ;
node_ = 20;
into_ = 0;
tie_ = 0;
post_ = 0.8571428571;

end;
else
if 5.5 <= CONVSEG then do;
* DV counts: 450 31 ;
node_ = 21;
into_ = 0;
tie_ = 0;
post_ = 0.9355509356;

end;
end;
```

Notice that the code is written so that none of the continuous to discrete transformations are needed for ordinal data. As long as no arbitrary ordinal levels were introduced by the transformations there will be no need to worry about the consistency of the generated code with the original data. Also notice that four classification variables are defined: node_ which is the node identifier, into_ which is the value of response (0,1) that the populace of this node is classified as, tie_ which is a tie indicator of classification, and post_ which is the posterior probability of correct classification. The response probability in our case if into_=0 is 1 minus post_(1-post_).

REPORTS AND ANALYSIS

As a result of the classification process, each observation or customer in our case, has come to rest in a leaf node of the tree. One should not be disappointed if the tree does not classify any of the customers as responders. From what I've gathered in

Analysis and Modeling

my limited experience with direct marketing models, because the overall response rate can be quite low, it is not unusual for the default tree classification to be no-response (RSPORD=0), and the response rate is viewed as the error rate. Also if the tree depth is capped, then there is less likelihood of response classification than if the tree was allowed to segment further. Hence we find less interest in the predicted classification, i.e., into $_1$ or 0, than in the predicted response rate or error rate (1-post $_$) of the specific leaf node or segment, especially as compared to the actual response rate for the segment. And also, of course, we want to know the actual variable subset

for the node expressed in a friendly way, called a Ruleset by some (<http://www.rulequest.com/see5-win.html>). Two reports are shown below with discussion.

Lift Table

The classic Lift Table is generated by ranking the customers by the score or in our case the predicted probability of response, and then generating statistics grouped by decile. In our sample case, the Lift Table for the classification of the test or validation data set would look like Figure 4.

Figure 4 – Lift Table for Test Data

Decile	N	Resps.	Resp/N	Cum N	Cum Resp	Cum Resp/N	Lift	Gap
1	9068	53	0.006	9,068	53	0.006	70	-3.6
2	7332	228	0.031	16,400	281	0.017	205	22.4
3	6657	57	0.009	23,057	338	0.015	175	22.6
5	18913	123	0.007	41,970	461	0.011	131	17.1
8	34766	181	0.005	76,736	642	0.008	100	0.0
Total	76736	642						

In this test case the Lift Table indicates that a %TREEDISC model as is, would not be a good model with which to score one's customers. The decision tree needs more granularity, i.e., more leaf nodes, which could be accomplished by increasing the depth, increasing the minimum leaf node population, increasing the number of available predictor variables, or decreasing the alpha significance for using a predictor in the tree. The above model only had 11 leaf nodes. In my brief experience, even with good granularity, decision tree models can provide fantastic lift in the upper 50%, but often fail to discriminate in the lower 50%. But

this is acceptable in our methodology because we have a combined score model that performs well in the lower deciles, and we can use the decision tree model to tell us more about our high response segments.

High Response Segments

So the ultimate value of decision tree models seemed to be in listing the high response leaf nodes along with the path description of the node, as below in Figure 5.

Figure 5 – Print of Top 20% Nodes

Analysis of Leafnode Path Responders 14:49 Wednesday, May 27, 1998 11

Where count=#of responses, and response=rate of node

DECILE LEAFNODE COUNT RESPONSE PREDRESP NODEDESC

1 6 7 0.000849 0.05714	2: -1e38<=YRORD1<=0.5
	4: -1e38<=ORD44RIV<=0.29
	6: -1e38<=TOTORD5<=0.5
1 18 6 0.042553 0.07299	3: 0.5<=YRORD1<=
	12: -1e38<=ORD44RIV<=0.6
	14: -1e38<=RECMONLG<=1.5
	17: 3.5<=CONVSEG<=
	18: CNVCAT9=1
1 20 13 0.067358 0.14286	3: 0.5<=YRORD1<=
	13: 0.6<=ORD44RIV<=
	20: -1e38<=CONVSEG<=5.5
1 21 27 0.054767 0.06445	3: 0.5<=YRORD1<=
	13: 0.6<=ORD44RIV<=

2 5 39 0.030636 0.03049	21: 5.5<=CONVSEG<=
	2: -1e38<=YRORD1<=0.5
	5: 0.29<=ORD44RIV<=
2 15 81 0.025015 0.02899	3: 0.5<=YRORD1<=
	12: -1e38<=ORD44RIV<=0.6
	15: 1.5<=RECMONLG<=
2 16 54 0.041699 0.05633	3: 0.5<=YRORD1<=
	12: -1e38<=ORD44RIV<=0.6
	14: -1e38<=RECMONLG<=1.5
	16: -1e38<=CONVSEG<=3.5
2 19 54 0.035387 0.03043	3: 0.5<=YRORD1<=
	12: -1e38<=ORD44RIV<=0.6
	14: -1e38<=RECMONLG<=1.5
	17: 3.5<=CONVSEG<=
	19: CNVCAT9=0

The second column is the leaf node identifier, the third is the number of actual responses in the node, the fourth is the actual response rate of the node, and the fifth is the predicted response rate of the node from the model. The node path is described by the 6th text field as node identifier followed by a colon, followed by the predictor variable range or value. There is also a subsequent field that is the predictor variable label but it is truncated in the figure. The node path description starts from the root and reads down ending with the leaf node itself. The predicted response rate for each node is greater than the actual response rate for most of the nodes due to the fact that the model was constructed on mailed customers, whereas the test data set has almost as many non-mailed customers as mailed. The highest predicted response nodes are 20 and 18, whilst the highest actual response nodes are 20 and 21. Indeed for more response one should take customers with YRORD1>=0.5 and ORD44RIV<=0.6 for a total of 195 as opposed to YRORD1>=0.5 and ORD44RIV>=0.6 (nodes 20 and 21) with a total responses of 40, or YRORD1<=0.5 (nodes 5 and 6) with a total of 46. Also the precision of the nodes (predicted-actual response) is better in this branch (YRORD1>=0.5 and ORD44RIV<=0.6) than in the other branches with node 6 being especially deviant from predicted (actual of 7 versus predicted of 471). One must keep in mind the overall response of the test data set is 0.008 from Figure 4. In summary one would surely target customers in the YRORD1>=0.5, ORD44RIV<=0.6 branch (nodes 15, 16, 18, and 19) and any of the other nodes with good precision (5, 21), and examine why node 6 is so aberrant, which in this case I believe has to do with so many non-mailed customers with these attributes in the test data set. Also it might be better to sort the report in Figure 5 by leaf node rather than decile, leaf node for clarity sake.

CONCLUSION

I believe the use of the %TREEDISC macro in Bear Creek Corporation Marketing Systems has allowed the analysts to determine the segments as defined by predictor variable interactions that are likely to be high in response. Future development will involve tweaking some of the other parameters, and finding a way to display and print the whole tree at once.

SAS, SAS/AF, SAS/IML, SAS/STAT, and SAS/OR are registered trademarks or trademarks of the SAS Institute, Inc. in the USA and other countries. IBM and AIX are registered trademarks or trademarks of International Business Machines Corporation. ® Indicates USA registration.

APPENDIX 1-DATA PREPARATION CODE

```
%macro makefmt(datax,varx); /*was for
totsale originally*/
proc univariate data=&datax ;var &varx;
  output out=wez mean=ave min=minv
  pctlpre=pctl pctlpts=10 to 100
  by 10;run;
data intots;
  length fmtname $8 type hlo $1 label $16 ;
  set wez;
  retain fmtname "&varx" type 'N' hlo ' ';
  array pctl{10} pctl10 pctl20 pctl30 pctl40
  pctl50 pctl60 pctl70 pctl80
  pctl90 pctl100;
  keep start end label sexcl eexcl
fmtname type hlo;
  lb=minv;
  if lb<0 then do;
    sexcl='N';start=-999999999;
    eexcl='N';end=0;label='O';
    output;
    lb=0;
  end;
  do i=1 to 10;
    start=lb;
    end=pctl{i};
    if i=10 then end=999999999;
    sexcl='N';
    eexcl='Y';
    if i=1 and start=0 then sexcl='Y';
    if i=10 then eexcl='N';
  *   label=put(((start>0)*start+end)/2,5.);
  label=put(pctl{i},5.);
    output;
    lb=pctl{i};
  end;
run;
proc format cntlin=intots;run;
%Mend;
%makefmt(sel,totsale)
%let ipds=sel;
data noms ords; /*generate variable types
for chaid*/
  set &ipds(obs=1 keep=&xlist);
/*generate discretization code*/
  length vnamex $8 vlab $40 lst1 $1 lst2 $2;
  array nums{*} _numeric_;
  keep vnamex vlab;
  call execute('data
  '||resolve('&chiname')||';set '||
  resolve('&ipds')||';');
  do i=1 to dim(nums);
    call vname(nums{i},vnamex);
    call label(nums{i},vlab);
    lst1=substr(vnamex,length(vnamex),1);
```

```
    lst2=substr(vnamex,length(vnamex)-1,2);
    if index(vlab,'Y/N')>0 then do;
      output noms;
      continue;
    end;
    if vnamex='PRORD' and lst1^='R' or
    vnamex='FZP' or vnamex='CVS' or
    vnamex='FSTC' or vnamex='CS' and
    length(vnamex)=3 or vnamex in
    ('FPHONE','BUS','SNEWCONV','NEWCONV')
    then do;
      output noms;
      continue;
    end;
    if (vnamex='SORD'|vnamex='ORD' &
    verify(substr(vnamex,4,2),'0123456789')=1)
    and lst1^='R' then do;
      output noms;
      continue;
    end;
    output ords;
    if vnamex='LSTY' and lst2^='IV' then
    continue;
    if vnamex='FSTY' then continue;
    if vnamex in ('SALCAT','SALFLG') then
    continue;
    select;
      when(vnamex='CNVDATE')
        call execute(
          'cnvdate=round(mod(cnvdate,1000),90)+int(cnv
          date/1000)*1000+1;');
      when(vnamex='TMONFILE')
        call
        execute('tmonfile=floor(round(tmonfile)/12);
        ');
    when((vnamex='SSAL'|vnamex='TOTSAL'|vnamex
    =:'AO'|vnamex='SAL')
    and lst1^='R' and lst2 not in
    ('LG','IV') or
    vnamex in
    ('CUSTAORD','CONVSALE','MAXDOL') )
      call
      execute(trim(vnamex)||'=input(put('||trim(vn
      amex)||
        ',totsale.),6.);');
      when(lst2='LG'|vnamex in
      ('SALMISCR','SALSROSR'))
        call
        execute(trim(vnamex)||'=round('||trim(vnamex
        )||');');
      when(index(vlab,'/ TMONFILE')>0 and
      lst1='R' or lst2='IV')
        call
        execute(trim(vnamex)||'=round('||trim(vnamex
        )||
```

```

        ',0.01));');
    otherwise ;
end;
end;
call execute('run;');
run;

```

APPENDIX 2-TREE LISTING

```

TREEDISC Analysis of Dependent Variable (DV)
RSPORD
RSPORD value(s): 0 1
DV counts: 32719 648 Best p-value(s):
0.0001 0.0001
    YRORD1 value(s): 0
    DV counts: 27162 406 Best p-value(s):
0.0001 0.0001
        ORD44RIV value(s): 0.14 0.2 0.25
        DV counts: 25922 367 Best p-
value(s): 0.0001 0.0001
            TOTORD5 value(s): 0
            DV counts: 132 8
            TOTORD5 value(s): 1
            DV counts: 25790 359 Best p-
value(s): 0.0001 0.0008
                CNVCAT9 value(s): 1
                DV counts: 3329 71 Best p-
value(s): 0.1230 0.9784
                    CNVCAT9 value(s): 0
                    DV counts: 22461 288 Best
p-value(s): 0.0013 0.0026
                        CONVSEG value(s): 1
                        DV counts: 8266 137
                        CONVSEG value(s): 2 3
4 5 6 8 9 10
                            DV counts: 14195 151
                            ORD44RIV value(s): 0.33 0.5
                            DV counts: 1240 39 Best p-
value(s): 0.3156 0.6897
                                YRORD1 value(s): 1
                                DV counts: 5557 242 Best p-value(s):
0.0001 0.0020
                                    ORD44RIV value(s): 0.14 0.2
                                    DV counts: 4963 187 Best p-
value(s): 0.0200 0.0371
                                        RECMONLG value(s): -5 0 1
                                        DV counts: 2551 115 Best p-
value(s): 0.0445 0.0849
                                            CONVSEG value(s): 1 2 3
                                            DV counts: 1022 61 Best p-
value(s): 0.3614 0.8938
                                                CONVSEG value(s): 4 5 6 8
9 10

```

```

DV counts: 1529 54 Best p-
value(s): 0.0087 0.9974
    CNVCAT9 value(s): 1
    DV counts: 127 10
    CNVCAT9 value(s): 0
    DV counts: 1402 44
        RECMONLG value(s): 2
        DV counts: 2412 72 Best p-
value(s): 0.3509 1.0000
            ORD44RIV value(s): 1
            DV counts: 594 55 Best p-value(s):
0.0067 0.2162
                CONVSEG value(s): 1 2 3
                DV counts: 144 24
                CONVSEG value(s): 8 10
                DV counts: 450 31 Best p-
value(s): 1.0001 1.0571

```

APPENDIX 3 - ORG CHART CODE

```

submit continue;
    rsubmit &rmte_nm;
libname c
"/mkdbseg/&system/model/prom&dsn";
data ko;set c.&treenom;where _to_ is not
null;
back=node_;
    rename _to_=node_;
keep back _to_;run;
proc sort;by node_;run;
proc sort data=c.&treenom out=tree;by
node_;run;
data tree;merge tree ko;by node_;run;
proc sort data=tree;by node_ back;run;
data htree;set tree;by node_ back;
    if first.node_;run;
proc download data=htree out=sasuser.treex
status=no;run;
endrsubmit;
endsubmit;
call display('chaid.frame')

```