Paper PO-23

Using Dictionary Tables to Explore SAS® Datasets

Phillip Julian, Bank of America, Charlotte, NC

ABSTRACT

To create a report, you need requirements and data. Your hardest task may be finding the data and learning how to use it properly. Two programs are presented here to aid your quest. One finds all of your SAS datasets, and the other shows you what your datasets look like on the inside. SAS dictionary tables provide basic information about the structure and contents of your data. The MEANS and SQL procedures profile the dataset, and show values and statistics for any reporting column.

INTRODUCTION

I was writing a report on a specific column that belonged to several SAS datasets. To find the reporting datasets, I searched the columns dictionary table in SAS. Then I generalized that solution because we may need to answer similar requests for finding data and producing reports. We had no useful metadata documents, and no profiling tools like SAS® Data Integration, Dataflux® dfPower® software, or JMP® software. So I created a profiling tool that used only the Base SAS® language.

There are many good papers that cover all the basic facts about SAS dictionary tables. But few papers describe sophisticated uses of SAS meta-information to solve tough problems like data discovery. This paper describes a practical SAS solution that may guide your deeper dive into the data repository.

USES OF DATASET PROFILE INFORMATION

If you have any doubts about the correctness of your data, then you need to profile the data, learn its quirks, and decide how you should use it. You may want to save that knowledge to help others understand how to use the data. Knowledge about data is imperfect under these conditions:

- Datasets may exist outside of the data warehouse.
- Metadata documents may be missing, invalid, or incomplete.
- Dataset owners may not be available to provide advice on proper usage of the data.
- Keys may be unknown, and dataset joins may require guesswork and experimentation.
- Internal reorganizations, mergers, and acquisitions may create gaps in your data knowledge.

PRACTICAL APPLICATIONS OF DATASET PROFILE INFORMATION

These are actual applications of dataset profile information in my work:

- You want to analyze all the datasets that contain a particular column. Use the Columns spreadsheet from the first program, *Dataset_Explorer.sas*, and find all SAS datasets that have that column. See ***Figure 2***.
- You have an address column, but you want additional information from a foreign key in the dataset. Use the Columns spreadsheet to find datasets with column names similar to any column in the dataset. See ***Figure 3***.
- You want to update the data repository and add a column to all campaign datasets. Use the Libnames file, created by *Dataset_Explorer.sas*, and make SAS libnames for all your datasets. Then use SAS dictionary tables to create the SAS statements that will add a new column to the datasets.
- You need to plan some data analysis, but you don't know which columns have a a sufficient number of non-missing analysis variables. Create a dataset profile using the second program, *Dataset_Profiler.sas*, which counts non-missing and unique values for every column. See ***Figure 4***.

Here are some other possible uses of dataset profiles:

- Profile datasets over time, and compare profiles to find added, deleted, or changed columns.
- Compare detailed column profiles to see whether variables in different datasets may contain similar data.
- Use dataset ownership from *Dataset_Explorer.sas* to assign responsibility, and ask dataset owners to document the proper usage of the data.
- Use owner information and LDAP to assign responsibility to the department of the dataset owner. Then create and publish departmental measures for active ownership and proper dataset documentation.
- Use the dataset documentation to create meaningful labels for all variables. In that way, the SAS dataset documents itself and describes how to use its data. Good examples of this practice are the datasets supplied by SAS Institute. You can learn how to use the data by reading the column labels.
- With good SAS labels, you can use Proc Print to create an excellent data dictionary. This was done at GlaxoSmithKline for their PRx data warehouse, and a documentation specialist maintained the labels. Note that label maintenance is quick and easy to do using PROC DATASETS with the MODIFY statement.

THE SAS DICTIONARY DEFINES THE ARCHITECTURE

The SAS dictionary contains a wealth of information about SAS itself and about every customization or definition that you have created. The SAS System has a self-defining data architecture like a relational database. One table defines all of the dictionary tables, and each of those tables defines all of the SAS objects and their current instances. The master table in SAS is the *Dictionary.Dictionaries* table.

This master dictionary defines the dictionary tables for each SAS object, such as options, formats, ODS styles, macros, catalogs, libnames, and datasets. All such information is metadata, or data about the data.

To view the definitions of all of your SAS dictionary tables, you can submit this code:

```
proc sql noprint;
  %*-- Create Describe commands for each table in the Master Dictionary. --;
  select distinct 'describe table ' || compress('DICTIONARY.' || memname)
      into :Desc_Dictionary_Tables
      separated by ";  "
      from Dictionary.Dictionaries
      order by memname;
  %*-- Print the description for each table in the Master Dictionary. --;
  &Desc_Dictionary_Tables;
quit;
```

For example, these are some well-labeled dictionary tables in my local SAS System:

Member Name	Dataset Label
CATALOGS	Catalogs and catalog-specific information
CHECK_CONSTRAINTS	Check constraints
COLUMNS	Columns from every table
CONSTRAINT_COLUMN_USAGE	Constraint column usage
CONSTRAINT_TABLE_USAGE	Constraint table usage
DICTIONARIES	DICTIONARY tables and their columns
ENGINES	Available engines
EXTFILES	Files defined in FILENAME statements, or implicitly
FORMATS	Available formats
GOPTIONS	SAS/Graph options

Figure 1. *Some Dictionary Tables in the SAS System*

EXAMPLE USES OF THE COLUMNS SPREADSHEET

The Columns spreadsheet looks like a stripped down PROC CONTENTS listing. It contains information on all columns of all SAS datasets in your system. After you open the CSV file, and save the data as an Excel 2007+ workbook, you can filter the columns to find your analysis datasets. This example shows the results of Excel filtering on ***name*** to show any column that contains the string, ***curr_rate***. The ***Directory*** column shows the location of the dataset, and ***memname*** shows the SAS dataset name. See ***Figure 2***.

The next example demonstrates looking for a foreign key to get more information about your data. The ***name*** column was filtered for any string that contained ***LPO***. The SAS dataset, ***WFF_75MR***, was eventually used for the analysis, after looking up the dataset owner and confirming proper usage of the data. See ***Figure 3***.

Directory	memname	name	type	length	modate
/cart/dart/contact_hist/dev/master_file	REFI07A	curr_rate	char	200	27OCT09:14:29:53
/cart/dart/contact_hist/dev/master_file	REFI07B	curr_rate	char	200	27OCT09:14:29:57
/cart/dart/contact_hist/dev/master_file	REFI07C	curr_rate	char	200	27OCT09:14:30:03
/cart/dart/contact_hist/dev/master_file	REFI08A	curr_rate	char	200	27OCT09:14:30:08
/cart/dart/contact_hist/dev/master_file	REFI08B	curr_rate	char	200	27OCT09:14:30:12
/cart/dart/contact_hist/dev/master_file	REFI08C	curr_rate	char	200	27OCT09:14:30:17
/cart/dart/contact_hist/dev/master_file	REFI09A	curr_rate	char	200	27OCT09:14:30:22
/cart/dart/contact_hist/dev/master_file	REFI09B	curr_rate	char	200	27OCT09:14:30:23
/cart/dart/contact_hist/dev/master_file	REFI10A	curr_rate	char	200	27OCT09:14:30:29
/cart/dart/contact_hist/dev/master_file	REFI10B	curr_rate	char	200	27OCT09:14:30:39
/cart/dart/refi	GROUP2_CONV	curr_rate	char	200	06OCT09:12:51:50
/cart/dart/refi	GROUP2_CONV_REFI09B	curr_rate	char	200	18SEP09:09:36:18
/cart/dart/refi	GROUP2_GOVT	curr_rate	char	200	06OCT09:12:51:50
/cart/dart/refi	GROUP2_GOVT_REFI09B	curr_rate	char	200	18SEP09:09:36:19
/cart/dart/refi	NEWREFI10A_NOCONTROL_OCT6	curr_rate	char	200	08OCT09:18:35:26
/cart/dart/refi	ORIGREFI10A_NOCONTROL_OCT6	curr_rate	char	200	08OCT09:20:53:59
/cart/dart/refi	ORIGREFI10B_NOCONTROL_OCT13	curr_rate	char	200	13OCT09:11:12:04
/cart/dart/refi	ORIGREFI10B_NOCONTROL_OCT19	curr_rate	char	200	21OCT09:14:31:11
/cart/dart/refi	ORIGREFI11B_NOCONTROL_NOV02	curr_rate	char	200	04NOV09:14:12:46
/cart/dart/refi	ORIGREFI12A_NOCONTROL_NOV23	curr_rate	num	8	30NOV09:08:22:05
/cart/dart/refi	ORIGREFI12B_NOCONTROL_DEC07	curr_rate	num	8	09DEC09:16:54:48
/cart/dart/refi	REFI01A_FINAL_W_CTRL	curr_rate	num	8	22DEC09:19:53:07

Figure 2. *The Columns Spreadsheet – Filtered on* ***name*** *contains* ***curr_rate***

Directory	memname	name	type	length	varnum	modate
/cart/dart/data_repository	ALL_WFF_50MR_1006_EXCLUSIONS	LPO	num	8	4	07OCT09:12:37:59
/cart/dart/data_repository	ALL_WFF_50MR_1006_EXCLUSIONS	New_LPO	num	8	66	07OCT09:12:37:59
/cart/dart/data_repository	ALL_WFF_50MR_1006_EXCLUSIONS	Old_LPO_No	num	8	12	07OCT09:12:37:59
/cart/dart/data_repository	ALL_WFF_75MR_1007_INCLUSION	new_LPO	num	8	4	08OCT09:11:00:53
/cart/dart/data_repository	WFF_75MR	New_LPO	num	8	2	04NOV09:16:21:01
/cart/dart/MAP_Maint/op10	HPC_BRANCHES	LPO	num	8	1	28JAN10:15:06:33
/cart/dart/MAP_Maint/op10	MAIL_OP10_W_SEEDS	LPO	num	8	11	02FEB10:16:24:53
/cart/dart/non_resp/nr10	NONR1002A_SEEDS_FINAL_HPCX	LPO	num	8	30	02FEB10:14:04:38
/cart/dart/non_resp/nr8	MAIL_NR8_W_SEEDS	LPO	num	8	92	15DEC09:14:40:49
/cart/dart/non_resp/nr8	MAIL_NR8_W_SEEDS	Old_LPO_	char	4	97	15DEC09:14:40:49
/cart/dart/non_resp/nr8	NR8_FINAL_MAILFILE_W_SEEDS_SUPP1	LPO	num	8	92	02DEC09:17:02:41
/cart/dart/non_resp/nr8	NR8_FINAL_MAILFILE_W_SEEDS_SUPP1	Old_LPO_	char	4	97	02DEC09:17:02:41
/cart/dart/non_resp/nr8	NR8_FINAL_MAILFILE_W_SEEDS_SUPP2	LPO	num	8	92	09DEC09:18:28:16
/cart/dart/non_resp/nr8	NR8_FINAL_MAILFILE_W_SEEDS_SUPP2	Old_LPO_	char	4	97	09DEC09:18:28:16
/cart/dart/non_resp/nr8	PDQ_FINAL_MAILFILE_W_SEEDS_SUPP1	LPO	num	8	92	16DEC09:12:01:34
/cart/dart/non_resp/nr8	PDQ_FINAL_MAILFILE_W_SEEDS_SUPP1	Old_LPO_	char	4	97	16DEC09:12:01:34

Figure 3. *The Columns Spreadsheet – Filtered on* ***name*** *contains* ***LPO***

THE DATASET EXPLORER PROGRAM

Dataset_Explorer.sas provides information about all SAS datasets in a directory tree. The SAS program performs these tasks:

1. Find all SAS datasets in the directory tree, and also get the dataset ownership and permissions.
2. Create SAS libnames for every directory that contains any SAS datasets.
3. Query the SAS dictionary tables for information on all tables and all columns, using these tables:
    a. *Dictionary.Tables*, which contains information on all SAS datasets.
    b. *Dictionary.Columns*, which contains information on all SAS variables.
4. Create Excel and CVS reports that can be filtered and searched for relevant SAS datasets and variables.

The reports are automatically date tagged. If today is June 30, 2010, then the two reports would be:

1. *Dataset_Explorer_20100630.xls* – an Excel workbook with information on all SAS datasets and their file permissions.
2. *Dataset_Explorer Columns_20100630.xls* – a CSV file with information on all SAS variables. ***Figures 2*** and ***3*** show what the CSV file looks like after being converted to Excel.

CAVEATS ABOUT THE SAS PROGRAM ENVIRONMENT

The SAS program was written for PC SAS. The SAS data repository is on UNIX, and remote access is by SAS/CONNECT®. The code for remote UNIX is between the RSUBMIT and ENDRSUBMIT statements.

If you are not using SAS/CONNECT, then you would omit the RSUBMIT and ENDRSUBMIT statements, as well as the %syslput macros. If your SAS repository is not on UNIX, then you would need an equivalent command to search a directory for all SAS datasets.

HOW THE PROGRAM WORKS

At the top of the program, two definitions control how the program works. Set these to appropriate values for your system:

```
/*-- Start at this UNIX tree --*/
%let UNIX_Tree = /cart/dart;

filename DSNs PIPE
    "cd &UNIX_Tree; /usr/bin/find &My_Dirs -name ""*.sas7bdat"" -exec ls -l {} \;" ;
```

The %let defines the directory tree to search. The *DSNs* filename opens a UNIX pipe that searches for any SAS dataseet (**.sas7bdat*) in the directories (*&My_Dirs*, which is a subset of all directories). For every SAS dataset that is found, a full listing (ls –l) is produced, so that ownership and permissions can be captured. The result of those commands go to a PIPE, which is read by a SAS data step. If you need further details, the complete program is well commented and available for download.

THE DATASET PROFILER PROGRAM

Dataset_Profiler.sas is a much more complex program than the previous one. It was developed for the same SAS environment, and uses more tricks to process the data. The Excel spreadsheet is tagged by a constant inside the program, and its name is *Dataset Profiler_CH.xls*.

The following explanations may be easier to follow if you see the final results of a heavily filtered excerpt in ***Figure 4***.

This program profiles any SAS dataset, and shows values and statistics for all the report columns in the dataset. A report column is defined as any column that has fewer than 300 unique values, but you may want to adjust that number for your own work. The profile analysis is performed using only 2 passes of the dataset:

1. PROC SQL counts the number of rows, the number of missing values per column, and the number of unique values per column.
2. PROC MEANS summarizes the reporting columns, and produces a wide range of statistics for each unique value of each reporting column.
3. ***Figure 4*** shows an excerpt from a dataset profile report. Column A has the name of the column, column B has the values for that column, and the other columns are statistics from the PROC MEANS output dataset. Note that all the possible column values are shown for the ***offer_type*** and ***offer_pct*** columns, along with statistics on each value of those columns. The ***autolabel*** option names the columns based upon the statistic, which is N, a count of non-missing values. For example, ***refi_payment_ben_flag*** has non-missing values only when ***offer_type*** equals " " or "P", in which case it has 55,876 and 55,295 respectively.

	A	B	G	J	AE	AQ	BC	BI	BO	BU	CA	CV
1	Variable	Values	contact date N	Count	First Prin Bal N	Last changed date N	ltv N	New payment N	refi actual payment N	refi new payment N	refi payment ben flag N	Column Number
235	refi_offered_rate	N/A	557	557	557	557	532	0	0	0	0	12
236	offer_type		150683	150683	147421	150591	92109	22240	55871	55876	55876	11
237	offer_type	D	9766	9766	7370	9766	7238	0	0	0	0	11
238	offer_type	P	291499	291499	224120	291499	106248	0	60934	60938	55295	11
239	offer_pct		15287	15287	15287	15287	14227	0	0	0	0	10
240	offer_pct	0%	36012	36012	25328	36012	3882	0	0	0	0	10
241	offer_pct	4%	14642	14642	11779	14642	6295	0	0	0	0	10
242	offer_pct	8%	12576	12576	10473	12576	4994	0	0	0	0	10
243	offer_pct	0%	5386	5386	5386	5386	2737	0	0	0	0	10
244	offer_pct	10%	14347	14347	3200	14347	1405	0	0	0	0	10
245	offer_pct	18%	65445	65445	58831	65445	26472	0	0	0	0	10
246	offer_pct	20%	33180	33180	4613	33180	1760	0	0	0	0	10
247	offer_pct	4%	2735	2735	2735	2735	1760	0	0	0	0	10
248	offer_pct	8%	1925	1925	1925	1925	1211	0	0	0	0	10
249	offer_pct	Gen	972	972	270	972	183	0	8	8	8	10
250	offer_pct	na	249441	249441	239084	249349	140669	22240	116797	116806	111163	10

Figure 4. *The Dataset Profiler Spreadsheet – a Heavily Filtered excerpt*

HOW THE PROGRAM WORKS

Several programming tricks are used in this program, and they will be covered from top to bottom. Please ignore the references to REFI; the program was originally developed for that type of analysis.

The first pass of the data uses SQL in a two-step process. In the first step, the count variable query is created from the dictionary table for the analysis dataset:

```
/*-------------------------------------------------------------------------*/
/*-- Look for various possible key columns ----------------------------------*/
/*-------------------------------------------------------------------------*/

%let Count_All_Vars =;
proc sql noprint stimer;
    %*-- Count non-missing and unique rows in the dataset. --------------------;
    %*-- Prepare the SQL statements to perform the counts. --------------------;
    %*-- Handle issue where SAS variable name would be longer than 32 chars ---;
    select "count(" || strip(Name)  || ") as N_"  ||
                substr(left(Name),1,min(29, length(strip(Name)))) ||
         ", count(distinct " || strip(Name) || ") as ND_" ||
                substr(left(Name),1,min(29, length(strip(Name))))
        into :Count_All_Vars separated by ", "
        from dictionary.columns
        where libname = upcase("&Refi_Libname") and memname = upcase("&Refi_DSN");
    %put Count_All_Vars has &sqlobs rows;
quit;
```

In the second step, the query selection is executed by PROC SQL:

```
/*-------------------------------------------------------------------------*
 * Run this on the remote because lots of network traffic occurs here.
 *-------------------------------------------------------------------------*/
%syslput Check_Keys = &Check_Keys;
%syslput Count_All_Vars = &Count_All_Vars;
%syslput Means_DSN = &Refi_Libname2..&Refi_DSN;
rsubmit; /*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$*/
proc sql noprint stimer;
    %*-- Perform the dataset counts prepared above. --;
    create table key_values_0 as
        select count(*) as All_Rows, &Count_All_Vars
        from &Means_DSN;
    %put Dataset has &sqlobs rows;
quit;
endrsubmit; /*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$*/
```

You should know one SAS paradigm when reading my code. I use CARDS4 as a SAS comment statement because it skips over everything but 4 semi-columns in a row. In the code below, *&Check_Keys* skips SAS code up to the semi-colon line when it equals CARDS4, but executes the SAS code when *&Check_Keys* is blank:

```
/*-- If &Check_Keys ^= cards4, then the next SQL query runs --*/
data _null_;
    &Check_Keys;
run;
/* … … … whatever code you like to be here … … … */
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

The previous SQL query returns a single row of data about missing and unique counts. That data is transposed and used to create all the other basic counts for each column. Then a determination is made whether a column is a reporting column by this code:

```
        /*-----------------------------------------------------------------*
         * This criteria decides which variables get detailed statistics.
         * The threshholds of 0.1 and 300 may need tweaking, depending on
         * your data.
         *-----------------------------------------------------------------*/
        if (Unique_Pct_All <= 0.1 and Unique < 300 and Count ^= 0) then
            Stats = "Y";
        output;
```

Further manipulation is performed on the list of reporting variables. The only important criteria for selection should be the code above, since both alpha and non-continuous numeric variables can be used as class variables for PROC MEANS. The variables lists for MEANS are produced by SQL, and these are the PROC statements:

```
rsubmit; /*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$*/
/*-- Date formats cause some issues with some stats:  TBF later --*/
proc means data=&Means_DSN &Mean_Drop_Num_Stmt missing noprint;
    class &CH_Vars;
    types &CH_Vars;
    output out=&Means_Output N= NMiss= max= min= mean= std= / autoname;
run;
proc download data=&Means_Output; run;
endrsubmit;  /*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$*/
```

The output data from the MEANS procedure is a sparse matrix because of the *types* clause, which limits the output to summary statistics on the variables by themselves. The statistic columns are to the right of the *_type_* column, and all the columns before *_type_* have only one column with non-missing data.

We want something like ***Figure 4*****,** where the column name is on the left, then a specific value for that column, and finally all the statistics for that column value. Lots of computations are done to process the dataset so we can use the coalesce SQL function to get the desired results:

```
/*-- Process the Means dataset, and produce the final statistics add ------*/
proc sql noprint;
    %*-- Squish the leading columns that occur before the _type_ column --;
    create table Refi_DSN_Anly3 as
        select coalesce(&Coalesce_List) as Values format=$62.,
            &Coalesce_After, Column_Number
        from Refi_DSN_Anly2;
    %put Dataset has &sqlobs rows;
quit;
```

The %put in the SQL code is a useful diagnostic to count the rows you select and make sure that you get correct results. There is a bit more processing and merging before the Excel reports are produced.

You will need some creativity when using the Excel reports. You should hide and filter columns to display what you looks interesting. You may want to print the results to guide your report development. I found these methods useful in creating or enhancing reports. Time was saved because I could see the data and the range of values for each column. I used the statistics to evaluate whether I had good enough data to create a report. Instead of spending my time running custom queries, I just browsed and selected what I liked.

CONCLUSION

These two SAS programs provided a great starting point for analytical reporting. They have been used in a real work environment to solve real problems. I believe this tool has many other uses in data quality, data audits, and data governance. The results give you a window into your datasets, and use minimal system resources.

Like any new program, a lot will be learned as these programs get into the field and solve real business problems. I am interested in your results and your suggestions. Please let me know what you think.

ACKNOWLEDGMENTS

Thanks to Brent and Jessy for letting me spend a little time on this project.

CONTACT INFORMATION

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

Phillip Julian
Bank of America
101 S. Tryon St.
Charlotte, NC 28255
(980) 387-0507
julianp@acm.org
www.acm.org/~julianp

```
/*-------------------------------------------------------------------------*
 * Find all SAS table names on the UNIX remote host
 *-------------------------------------------------------------------------*
 *
 * See my paper, "Using Dictionary Tables to Explore SAS Datasets", given at
 * the Charlotte SAS Users Group on February 26, 2010.
 *
 * All rights reserved by the author, Phillip Julian.
 *
 * Phillip Julian
 * Certified SAS Advanced Programmer for SAS 9
 * February 26, 2010
 * julianp@acm.org
 *-------------------------------------------------------------------------*/

/*-- Define the program and its location ----------------------------------*/
%let Pgm_Path = c:\Documents and Settings\x820700\My Documents\SAS - TMT;
%let Pgm = Dataset_Explorer;

/*-- Create a date tag for the data and output files --*/
data _null_;
    call symput ('Date_Tag', put(today(), yymmddn8.));
run;
%put Date Tag is &Date_Tag;
%let XLS_File = &Pgm_Path\&Pgm._&Date_Tag..xls;
%syslput Date_Tag = &Date_Tag;

options noxwait;  /*-- For DOS to move on!! --*/

/*-------------------------------------------------------------------------*/
rsubmit; /*-------------------------------------------------------------------*/
/*-------------------------------------------------------------------------*/

/*-- Various listings of directories under /cart/dart ---------------------*/

%let All_Dirs=2008 2009 FDM IXI MAP_Maint PA PD PortSummary SJB adhoc analysis
    bank_refi_lead bova campaigns2010 contact_hist cosi cpidata data_repository
    deb gm hamp ldg_tw_home loan_mod lost+found mod_time_analysis
    modeling_files mortxs non_resp non_responder pension pge refi report
    reporting ricktemp short_sale targ_mkt_files temp waterfall;

/*-- These 2 defines add up to the defininition above. --------------------*/
/*-- These directories were LEFT OUT of the analysis list. ----------------*/
%let All_Dirs_Left_Out = 2008 2009 DIRT FDM PortSummary adhoc bova bpowell deb
    lost+found ricktemp temp;
/*-- These directories were INCLUDED in the analysis list. ----------------*/
%let My_Dirs=FDM IXI MAP_Maint PA PD SJB analysis bank_refi_lead campaigns2009
    campaigns2010 contact_hist cosi cpidata data_repository gm hamp ldg_tw_home
    loan_mod mod_time_analysis modeling_files mortxs non_resp non_responder
    pension pge phase refi report reporting short_sale targ_mkt_files tmt
    waterfall;

/*-------------------------------------------------------------------------*
 * Note that you Should check whether new directories were added to this list,
 * because directory contents change over time.  This task can be accomplished
 * easily:
 *
 *     1. Use UNIX pipes that create a SAS dataset of the current directories
 *     2. Use Proc SQL to find the set difference between the current
 *        directories and SAS_Directories, matching on this:
 *            compress(&UNIX_Tree/Current_Directory) = Directory
 *
 *-------------------------------------------------------------------------*/

/*-- Start at this UNIX tree --*/
%let UNIX_Tree = /cart/dart;

/*-- Check the directory list for any updates --*/
x "cd &UNIX_Tree; ls -l | grep ^d";

/*-- PIPE to find all SAS datasets in the directory trees in the list --------*/
```

```
/*-- Note that the long character string "&My_Dirs" may cause a SAS Warning --*/
/*-- A better pipe would be this:
     find . -name "*.sas7bdat" -exec ls -l {} \;

     Output looks like this:

-rwxrwxrwx    1 a506570  cart     1011294208 Oct 21 2008  ./cpidata/cpi_082008.sas7bdat
-rwxrwxrwx    1 a506570  cart     1010442240 Oct 21 2008  ./cpidata/cpi_072008.sas7bdat
 *--*/
filename DSNs PIPE
    "cd &UNIX_Tree; /usr/bin/find &My_Dirs -name ""*.sas7bdat"" -exec ls -l {} \;" ;
/*  "cd &UNIX_Tree; /usr/bin/find &My_Dirs -name ""*.sas7bdat"" -print"; */

/*---------------------------------------------------------------------------*/
/*-- Read the raw file ------------------------------------------------------*/
/*---------------------------------------------------------------------------*/


/*---------------------------------------------------------------------------*/
/*-- Read and disect the list of SAS datasets -------------------------------*/
/*---------------------------------------------------------------------------*/
data SAS_Datasets;
    /*-- Input the UNIX pipe command, defined above --*/
    infile DSNs end=done;
    /*-- At first, use an arbitrarily long Path name to get ALL the data --*/
    length Path_Name $ 1500 Permissions $ 10 Owner Group $ 12;
    /*-- This next variable determines the maximum length of Path_Name --*/
    retain maxlen 0;
    drop maxlen Overall_Len;

    /*-- Read the PIPE --*/
    input;
    Path_Name = _infile_;

    /*-- Delete names with errors ... error cause is unknown --*/
    if ((scan(Path_Name, 1, ' ') = "find:") or
        (scan(Path_Name, 1, ' ') = ":"))
    then delete;
    else do;
        /*-- Get the other variables from the "ls -1" command --*/
        Permissions = scan(_infile_, 1, ' ');
        Owner       = scan(_infile_, 3, ' ');
        Group       = scan(_infile_, 4, ' ');
        Path_Name   = scan(_infile_, 9, ' ');
    end;

    /*-- Check whether the current Path_Name is the longest one --*/
    if (maxlen < length(strip(Path_Name))) then
        maxlen = length(strip(Path_Name));
    /*-- When the PIPE is finished ... --*/
    if done then do;
        /*-- Add  the length of the TREE name to the Path length --*/
        Overall_Len = maxlen + length("&UNIX_tree");
        /*-- Save the maximum length in a global MACRO variable --*/
        call symput('MAX_Path_Len', put(Overall_Len,4.));
    end;
run;
/*-- SAS trick to left justify the macro variable --*/
%let MAX_Path_Len = &MAX_Path_Len;
%put Maximum Path Length is &MAX_Path_Len;

/*---------------------------------------------------*/
/*-- Process the results of the UNIX find command --*/
/*---------------------------------------------------*/
data SAS_Datasets;
    /*-- Shorten the Path length BEFORE doing the SET --*/
    attrib Path_Name length=$&MAX_Path_Len format=$&MAX_Path_Len..
        informat=$&MAX_Path_Len..;
    set SAS_Datasets end=done;
    /*-- For now, use an arbitrarily long SAS dataset name for MemName --*/
    length Directory $ &MAX_Path_Len MemName $ 500;
```

```
    /*-- Determine the maximum length of MemName --*/
    retain maxlen 0;
    drop i j maxlen;

    /*-- Process each word of the pathname until you reach the SAS dataset --*/
    i = 1;
    MemName = scan(Path_Name, i, '/');
    do while(index(lowcase(MemName), 'sas7bdat') = 0);
        i = i + 1;
        MemName = scan(Path_Name, i, '/');
        /*-- Some names are blank for some reason:  exit if that occurs. --*/
        if (MemName = " ") then leave;
        /*-- Some names have other problems ... print them to the LOG. --*/
        if i > 10 then do;
            put "Loop > 10:   " Path_Name= MemName= ;
            leave;
        end;
    end;

    /*-------------------------------------------------------------------------*
     * At this point, MemName should contain the SAS dataset name.
     *
     * The Directory name should be everything before MemName, in this form:
     *     Directory/MemName.sas7bdat
     * The UNIX pathname -- used in 'find' -- completes the actual pathname:
     *     UNIX_Tree/Directory/MemName.sas7bdat
     *
     * Note that MemName must be STRIPPED for Index to get the proper Length.
     *-------------------------------------------------------------------------*/
    j = index(Path_Name, strip(MemName)) - 2;
    /*-- Some names have problems, like internal spaces. Just print to LOG. --*/
    if (j < 0) then do;
        put "Tricky name: " Path_Name= MemName= _N_= i= j= ;
        delete;
    end;
    /*-- The other names are OK --*/
    else do;
        /*-- Inline DEBUG code -->  put _N_= i= j= MemName=; --*/
        Directory = "&UNIX_Tree./" || substr(Path_Name, 1, j);
    end;

    /*-- Check whether the current Path_Name is the longest one --*/
    if (maxlen < length(strip(MemName))) then
        maxlen = length(strip(MemName));
    /*-- When finished reading the SAS dataset ... --*/
    if done then do;
        /*-- Save the maximum length in a global MACRO variable --*/
        call symput('MAX_Path_Len2', put(maxlen,4.));
        /*-- Write the current observation to the SAS dataset --*/
        if not (MemName = " " and Path_Name = " ") then output;
        /*-- Add a new observation to get SAS datasets in the UNIX Tree --*/
        Directory = "&UNIX_Tree.";
        Path_Name = " ";
        MemName = " ";
    end;
    /*-- "output" is required here, because of the above "output" statement --*/
    if not (MemName = " " and Path_Name = " ") then output;
run;
/*-- SAS trick to left justify the macro variable --*/
%let MAX_Path_Len2 = &MAX_Path_Len2;
%put Max Length2 is &MAX_Path_Len2;

/*-----------------------------------------------------*/
/*-- Set the path length, and order the variables --*/
/*-- This finishes processing of SAS_Datasets ------*/
/*-----------------------------------------------------*/
data SAS_Datasets;
    /*-- Order the variables --*/
    label MemName= Directory= Path_Name=;
    /*-- Shorten the SAS dataset length BEFORE doing the SET --*/
    attrib MemName length=$&MAX_Path_Len2 format=$&MAX_Path_Len2..
```

```
        informat=$&MAX_Path_Len2..;
    set SAS_Datasets;
run;

proc sort data=SAS_Datasets;
    by Directory MemName;
run;

/*-------------------------------------------------------------------------*/
/*-- Unique directories become libnames to search in the SAS Dictionary ------*/
/*-------------------------------------------------------------------------*/

/*-- Get the unique list of Directories --*/
proc sort data=SAS_Datasets (drop=MemName Path_Name Owner Group Permissions)
          out=SAS_Directories nodupkey;
    by Directory;
run;
/*-- Get the unique list of SAS Dataset information --*/
proc sort data=SAS_Datasets (drop=Path_Name)
          out=SAS_Directories_Mem_1 nodupkey;
    by Directory Memname;
run;

/*-- Use the list of directories to create SAS libname's --*/
data SAS_Directories;
    set SAS_Directories;
    length Libref $ 200 Library $ 20;
    file "All_Libnames_&Date_Tag..sas";

    Library = compress("Dir" || put(_N_, 5.));
    Libref = "libname " || trim(Library) || ' "' || strip(Directory) || '";';
    put libref ;
    output SAS_Directories;

    file "All_Libnames_Clear_&Date_Tag..sas";
    Libref = "libname " || trim(Library) || ' clear;';
    put libref ;
run;

/*-- Get the libname for each directory, for use later in a join --*/
proc sql noprint;
    create table SAS_Directories_Mem as
        select b.Library, a.*
        from SAS_Directories_Mem_1 a, SAS_Directories b
        where a.Directory = b.Directory;
quit;

/*-------------------------------------------------------------------------*
 * Assign Libnames to all directories that have SAS datasets.
 *
 * Once SAS libnames are assigned, the SAS Dictionary tables can be queried
 * for information about the SAS datasets and SAS libraries.
 *-------------------------------------------------------------------------*/
%include "All_Libnames_&Date_Tag..sas";

/*-------------------------------------------------------------------------*
 * The following SQL queries may cause I/O errors from damaged SAS datasets.
 * Such errors will not affect the results, but you may want to notify your
 * system administrator about the problem.
 *
 * Dictionary.Columns contains information about variables in SAS datasets.
 * Dictionary.Tables contains information about the SAS datasets themselves.
 *
 * Note that all SAS datasets are scanned, and not just those with type=DATA.
 *-------------------------------------------------------------------------*/

proc sql stimer noprint;
    %*-- Get information about the variables in each SAS dataset --------------;
    %*-- For true alpha sorting, the character case must be specified ---------;
    %*-- 'Order by Name' gives different results than 'propcase(Name)' --------;
    create table All_UNIX_Contents as
```

```
    select a.libname, b.Directory, a.MemName,
           Name, Type, Length, Format, Informat, Label, VarNum
        from dictionary.columns a, SAS_Directories b
        where libname = upcase(b.Library)
        order by a.libname, a.MemName, propcase(Name);

    %*-- Needed to create two SQL steps to make this work, since --------------;
    %*-- system errors occured, -----------------------------------------------;

    %*-- Get date stamps and sizes, etc., for each SAS dataset ----------------;
    create table All_UNIX_Libnames_1 as
    select libname, memname, CrDate, MoDate, nobs, nvar, filesize
        from dictionary.tables
        order by libname, memname;

    %*-- Get date stamps and sizes, etc., for each SAS dataset ----------------;
    create table All_UNIX_Libnames as
    select a.libname, b.Directory, a.memname,
           a.CrDate, a.MoDate, a.nobs, a.nvar, a.filesize,
           b.Owner, b.Group, b.Permissions
        from All_UNIX_Libnames_1 a, SAS_Directories_Mem b
        where upcase(a.libname) = upcase(b.Library) and
              upcase(a.memname) = upcase(scan(b.memname, 1, '.'))
        order by a.libname, a.memname;
quit;

/*-- In this case, the SAS merge is much easier to write than SQL code --*/
data All_UNIX_Contents2;
    merge All_UNIX_Contents (in=OK)
          All_UNIX_Libnames (keep=LibName MemName MoDate);
    by LibName MemName;
    if OK;
run;

/*-- Reorder datasets for greater usability --*/
proc sort data=All_UNIX_Contents;
    by Directory MemName Name;
run;
proc sort data=All_UNIX_Libnames;
    by LibName MemName;
run;

/*-- Clear the Libnames, since we have the information we need --*/
%include "All_Libnames_Clear_&Date_Tag..sas";

/*-- Save some datasets on the client host --*/
proc download data=SAS_Datasets; run;
proc download data=SAS_Directories; run;
proc download data=All_UNIX_Contents; run;
proc download data=All_UNIX_Libnames; run;
proc download data=All_UNIX_Contents2; run;

/*-------------------------------------------------------------------------*/
endrsubmit; /*-----------------------------------------------------------*/
/*-------------------------------------------------------------------------*/


/*-- Unique directories, which will become libnames to search ----------------*/

/*-- Get the SAS?CONNECT server name to setup remote libnames ----------------*/
proc sql noprint;
    %*-- SETTING has a length of 1024, so we need to set it shorter in SAS ----;
    select trim(setting) length=80 into :Conn_Server
        from dictionary.options
        where optname = "CONNECTREMOTE";
quit;
%let Conn_Server = &Conn_Server;
%put SAS/CONNECT server = &Conn_Server;

/*-- Create a list of remote and local libnames ... for easier browsing ------*/
data SAS_Directories_Rem;
```

```
    set SAS_Directories;
    drop Libref;
    length Rem_Libref $ 200;

    file "&Pgm_Path\All_Libnames_&Date_Tag..sas";
    put Libref;

    file "&Pgm_Path\All_&Conn_Server._Libnames_&Date_Tag..sas";
    Rem_Libref = "libname " || compress("RDir" || put(_N_, 5.)) || ' "' ||
        strip(Directory) || """ server=&Conn_Server;" ;
    put Rem_Libref;
    output SAS_Directories_Rem;

    file "&Pgm_Path\All_&Conn_Server._Libnames_Clear_&Date_Tag..sas";
    Rem_Libref = "libname " || compress("RDir" || put(_N_, 5.)) || ' clear;' ;
    put Rem_Libref;
run;

/*-- Save datasets for offline analysis ------------------------------------*/
PROC EXPORT DATA=All_UNIX_Libnames
    OUTFILE= "&XLS_File"
    DBMS=EXCEL2000 REPLACE;
    SHEET="Datasets";
RUN;
PROC EXPORT DATA=SAS_Directories
    OUTFILE= "&XLS_File"
    DBMS=EXCEL2000 REPLACE;
    SHEET="Libname Paths";
RUN;

/*-- All of the data is output to CSV, which can be read directy with  --*/
/*-- by MS Access and MS Excel 2007 ----------------------------------*/
PROC EXPORT DATA=All_UNIX_Contents2
    OUTFILE= "&Pgm_Path\&Pgm. Columns &Date_Tag..csv"
    DBMS=CSV REPLACE;
RUN;
```

```
/*---------------------------------------------------------------------------*/
/*-- I N I T I A L I Z E ----------------------------------------------------*/
/*---------------------------------------------------------------------------*/

/*-- I N I T I A L I Z E --------- The CHOSEN option from the list above -----*/
/*-- Contact History -------------------------------------------------------*/
%let Refi_Libname = R_ch;
%let Refi_Libname2 = ch;
%let Lib2_Path = /cart/dart/contact_hist;
%let Refi_DSN = Map_Chist;
%let Tag = _CH;

%syslput Refi_Libname2 = &Refi_Libname2;
%syslput Lib2_Path = %bquote(&Lib2_Path);
rsubmit; /*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$*/
libname ds_Prd "/cart/dart/mod_time_analysis";
libname ds     "/cart/dart/mod_time_analysis/dev";
libname &Refi_Libname2 "&Lib2_Path";
endrsubmit; /*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$*/
libname &Refi_Libname slibref=&Refi_Libname2 server=sigma;
libname R_ds_Prd slibref=ds_Prd server=sigma;

/*-- Special check for keys Loan_No and Contact_Date --*/
%let Check_Keys = cards4;  /*-- No check --*/
%let Check_Keys = ;        /*-- Check    --*/

/*---------------------------------------------------------------------------*/
/*-- Define where the program is located ------------------------------------*/
/*---------------------------------------------------------------------------*/
%let Pgm_Dir = v:\TMT\Phillip\REFI Analysis;
/*-- TAG is used to define output files --*/
%let Pgm = Dataset_Profiler&Tag;
%let RTF_File = &Pgm_Dir\&Pgm..rtf;
%let XLS_File = &Pgm_Dir\&Pgm..xls;

options fullstimer;

ods rtf file="&RTF_File";
ods listing close;

/*---------------------------------------------------------------------------*/
/*-- Contents of the REFI metadata and dataset ------------------------------*/
/*---------------------------------------------------------------------------*/
proc sql noprint;
    %*-- List the SAS Metadata for this dataset --;
    create table Contents_Listing as
        select Name, Type, Length, strip(Format) as Format,
            strip(Informat) as InFormat, strip(Label) as Label, varnum
        from dictionary.columns
        where libname = upcase("&Refi_Libname") and memname = upcase("&Refi_DSN")
        order by upcase(Name);
    %put Dataset has &sqlobs rows;
quit;

/*-- Custom dataset contents --*/
proc print data=Contents_Listing label uniform width=minimum noobs;
    title "Contents of &Refi_DSN";
run;

/*---------------------------------------------------------------------------*/
/*-- Look for various possible key columns ----------------------------------*/
/*---------------------------------------------------------------------------*/

%let Count_All_Vars =;
proc sql noprint stimer;
    %*-- Count non-missing and unique rows in the dataset. --------------------;
    %*-- Prepare the SQL statements to perform the counts. --------------------;
    %*-- Handle issue where SAS variable name would be longer than 32 chars ---;
    select "count(" || strip(Name)  || ") as N_"  ||
              substr(left(Name),1,min(29, length(strip(Name)))) ||
         ", count(distinct " || strip(Name) || ") as ND_" ||
```

```
                substr(left(Name),1,min(29, length(strip(Name))))
         into :Count_All_Vars separated by ", "
         from dictionary.columns
         where libname = upcase("&Refi_Libname") and memname = upcase("&Refi_DSN");
     %put Count_All_Vars has &sqlobs rows;
quit;

/*-----------------------------------------------------------------------------*
 * Run this on the remote because lots of network traffic occurs here.
 *-----------------------------------------------------------------------------*/
%syslput Check_Keys = &Check_Keys;
%syslput Count_All_Vars = &Count_All_Vars;
%syslput Means_DSN = &Refi_Libname2..&Refi_DSN;
rsubmit; /*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$*/
proc sql noprint stimer;
    %*-- Perform the dataset counts prepared above. --;
    create table key_values_0 as
        select count(*) as All_Rows, &Count_All_Vars
        from &Means_DSN;
    %put Dataset has &sqlobs rows;
quit;

/*-- If &Check_Keys ^= cards4, then the next SQL query runs --*/
data _null_;
    &Check_Keys;
run;
proc sql noprint stimer;
    %*-- The following two columns should be a compound key ... check it. --;
    create table key_values2 as
        select a.*
        from &Means_DSN a,
           (select Loan_No, Contact_Date, count(*) as Count
            from &Means_DSN
            group by Loan_No, Contact_Date
            having count(*) > 1) b
        where a.Loan_No = b.Loan_No and a.Contact_Date = b.Contact_Date
        order by Loan_No, Contact_Date;
    %put Dataset has &sqlobs rows;
quit;
proc download data=key_values2; run;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

proc download data=key_values_0; run;
endrsubmit;  /*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$*/


/*-----------------------------------------------------------------------------*/
/*-- Create a more useable dataset for Key Column analysis --------------------*/
/*-----------------------------------------------------------------------------*/

/*-- Exchange rows and columns --*/
proc transpose data=key_values_0 out=key_values_0_T;
run;

/*-- Derive other column counts from the transposed SQL results --*/
data key_values;
    /*-- Order the variables --*/
    label
        Variable=
        Count="Rows"
        Filled="Percent of Rows Filled"
        NMiss="Number of Missing Values"
        Miss_Pct="Percent of Rows Missing"
        Unique="Number of Distinct Values"
        Unique_Pct="Percent Unique for Filled Rows"
        Unique_Pct_All="Percent Unique for All Rows"
        Stats="Stats"
        ;
    set key_values_0_T (rename=(_name_=Varname col1=Num));
    length Variable $ 40 Stats $ 1;
    retain Row_Count Variable Count NMiss;
```

```
    drop Row_Count Varname Num;
    format Filled Unique_Pct Miss_Pct Unique_Pct_All percent9.2;

    /*-- Default value is N = No Statistics --*/
    Stats = "N";

    /*-- The first OBS contains the overall totals --*/
    if _N_=1 then do;
        Variable = Varname;
        Count = Num;
        Filled = 1;
        Unique = Num;
        Unique_Pct = 1;
        Unique_Pct_All = 1;
        NMiss = 0;
        Miss_Pct = 0;
        Row_Count = Num;
        output;
    end;

    /*-------------------------------------------------------------------------*
     * Besides the first row, other rows contain, in order, records with the
     * non-missing count, followed by another with the unique count.
     * Other measures are derived from this data.
     *-------------------------------------------------------------------------*/

    /*-- Got a new Variable ... Set the non-missing counts --*/
    if (substr(Varname, 1, 2) = "N_") then do;
        Variable = substr(Varname, 3);
        Count = Num;
        NMiss  = Row_Count - Count;
    end;
    /*-- This row has unique counts, and most measures are computed here --*/
    if (substr(Varname, 1, 3) = "ND_") then do;
        Filled = Count / Row_Count;;
        Unique = Num;
        /*-- This data is NOT all Missing values --*/
        if (Count ^= 0) then do;
            Unique_Pct = Num / Count;
            Unique_Pct_All = Num / Row_Count;
            Miss_Pct = NMiss / Row_Count;
        end;
        /*-- This data IS all Missing values, and cannot be summarized --*/
        else do;
            Unique_Pct = 0;
            Unique_Pct_All = 0;
            Miss_Pct = 1;
        end;
        /*---------------------------------------------------------------------*
         * This criteria decides which variables get detailed statistics.
         * The threshholds of 0.1 and 300 may need tweaking, depending on
         * your data.
         *---------------------------------------------------------------------*/
        if (Unique_Pct_All <= 0.1 and Unique < 300 and Count ^= 0) then
            Stats = "Y";
        output;
    end;
run;
proc sort data=key_values;
    by descending Count descending NMiss descending Unique Variable;
run;

/*-- The analysis so far, before the merge with the Contents dataset --*/
proc print data=key_values label uniform width=minimum noobs;
    title "Counts and Uniqueness for Variables in &Refi_DSN";
run;

/*-- Combine the Contents dataset with the Key Values analysis above ---------*/
%let Mean_Drop_Num =;
%let N_Mean_Drop_Num = 0;
%let Mean_Drop_Num_Stmt =;
```

```
proc sql noprint;
    create table Key_Values_Contents as
        select b.*, a.type, a.length, a.Format, a.InFormat, a.Label,
            a.varnum
        from Contents_Listing a, key_values b
        where upcase(a.name) = upcase(b.Variable);
    %put Dataset has &sqlobs rows;

    %*-- WARNINGS will occur when all-missing numeric variables do NOT EXIST --;
    %*-- Create the list of numeric variables whose values are all missing ----;
    %*-- These variables will be removed from Proc Means analysis -------------;
    select Variable into :Mean_Drop_Num
        separated by " "
        from Key_Values_Contents
        where Count = 0 and type = "num";
    %put Mean_Drop_Num has &sqlobs rows;
    %*-- Need a count of these variables to decide whether to run Proc Means --;
    select count(*) into :N_Mean_Drop_Num
        from Key_Values_Contents
        where Count = 0 and type = "num";
    %put N_Mean_Drop_Num has &sqlobs rows;
quit;

/*-- Create the drop statment for any all-missing numeric values --*/
data _null_;
    if (&N_Mean_Drop_Num > 0) then
        call symput('Mean_Drop_Num_Stmt', "(drop=" || strip("&Mean_Drop_Num")
                    || ")");
run;

proc print data=Key_Values_Contents
        (drop=Unique Unique_Pct length Format InFormat Label)
        label uniform width=minimum noobs;
    title "All-missing numeric variables will not be analyzed by Proc Means";
    where Count = 0 and type = "num";
run;

/*---------------------------------------------------------------------------*/
/*-- Create a list of variables to remove from the categorical analysis ------*/
/*---------------------------------------------------------------------------*/
data _null_;
    set key_values end=done;
    length line OK_List OK_List2 Remove_List $ 2000;
    retain line OK_List OK_List2 Remove_List;

    if (Stats = "N" or Variable = "All_Rows") then do;
        /*-- Used to remove columns (in SQL) that should not be in the Proc --*/
        /*-- Means CLASS and TYPES statements. -------------------------------*/
        line        = strip(line) || ", '" || strip(upcase(Variable)) || "'";
        /*-- Not used --*/
        Remove_List = strip(Remove_List) || " " || strip(Variable);
    end;
    else do;
        /*-- Not used: replaced by CH_Vars below --*/
        OK_List  = strip(OK_List) || " " || strip(Variable);
        /*-- Not used --*/
        OK_List2 = strip(OK_List2) || ", " || quote(strip(upcase(Variable)));
    end;

    if done then do;
        /*-- Fix the leading comma --*/
        if (index(line, ',') > 0) then
            line = "and upcase(name) not in (" || trim(substr(line, 3)) || ")";
        if (index(OK_List2, ',') > 0) then
            OK_List2 = "and upcase(name) in (" || trim(substr(OK_List2, 3)) || ")";
        call symput('CH_Vars_Missing', strip(line));
        call symput('CH_VarsList', strip(Remove_List));  /*-- Not used --*/
        call symput('Class_List', strip(OK_List));       /*-- Not used --*/
        call symput('OK_SQL_List', strip(OK_List2));
    end;
run;
```

```
%put Missing Character variables are:  &CH_VarsList;

%let CH_Vars =;
%let N_Num_vars = 0;
proc sql noprint;
    %*-- For Means, create a list of variables for the Class statement --;
    select Name into :CH_Vars
        separated by " "
        from dictionary.columns
        where libname = upcase("&Refi_Libname") and
            memname = upcase("&Refi_DSN") and type = "char"
            &CH_Vars_Missing;

    %*-- If a dataset only has character values, then skip the Proc Means ----;
    select (count(name) - &N_Mean_Drop_Num) into :N_Num_vars
        from dictionary.columns
        where libname = upcase("&Refi_Libname") and
            memname = upcase("&Refi_DSN") and type ^= "char";
quit;

/*-----------------------------------------------------------------------------*/
/*-- Perform the categorical analysis -----------------------------------------*/
/*-----------------------------------------------------------------------------*/

/*-- See whether to run or skip this analysis --*/
data _null_;
    /*-- The criteria to perform the analysis is determined here. --*/
    if (&N_Num_vars = 0) then
         call symput('Skip_Means', "cards4");
    else call symput('Skip_Means', " ");
run;

/*-- RUN it if Skip_Means = " ", or SKIP it until the line of ;'s ------------*/
data _null_;
    &Skip_Means;
run;

/*----------------------------------------------------------------------------*
 * Run this on the remote because lots of network traffic occurs here.
 * It also runs much faster on the server.
 *
 * Several problems occurred when this job was run locally on the PC: (1) many
 * SAS warnings occured in Proc Means, and (2) the Means output dataset
 * produced unexpected results where the column order did not match the CLASS
 * statement.  I was surprised when this issue was fixed by running the job on
 * the server.  There is possibly a difference in physical memory and in SAS
 * options that affect Proc Means.  Memory Max options should be compared
 * between the PC and the server.
 *-----------------------------------------------------------------------------*/

%put OK Character variables are:  &CH_Vars;
%syslput CH_Vars = &CH_Vars;
%syslput Means_Output = Refi_DSN_Anly;
%syslput Mean_Drop_Num_Stmt = &Mean_Drop_Num_Stmt;
rsubmit; /*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$*/
/*-- Date formats cause some issues with some stats:  TBF later --*/
proc means data=&Means_DSN &Mean_Drop_Num_Stmt missing noprint;
    class &CH_Vars;
    types &CH_Vars;
    output out=&Means_Output N= NMiss= max= min= mean= std= / autoname;
    /* output out=Refi_DSN_Anly sum= qrange= std= / autoname */;
run;
proc download data=&Means_Output; run;
endrsubmit;  /*$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$*/
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

/*-----------------------------------------------------------------------------*/
/*-- Squish and process the data to create a better summary ------------------*/
/*-----------------------------------------------------------------------------*/
proc sql noprint;
    %*-- Get the POSITION of the _type_ variable in the Means output dataset --;
```

```
    select varnum into :Type_Column
        from dictionary.columns
        where libname = "WORK" and memname = "REFI_DSN_ANLY" and
            upcase(name) = "_TYPE_";
    %*-- Get the LENGTH for the _type_ variable in the Means output dataset --;
    select Length into :Type_Column_Len
        from dictionary.columns
        where libname = "WORK" and memname = "REFI_DSN_ANLY" and
            upcase(name) = "_TYPE_";
    %*-- Get the TYPE for the _type_ variable in the Means output dataset --;
    %*-- See next datastep for usage ... value is Char or Num --------------;
    select type into :Type_Column_Type
        from dictionary.columns
        where libname = "WORK" and memname = "REFI_DSN_ANLY" and
            upcase(name) = "_TYPE_";

    %*-- Tag the dataset with column numbers, so we can look up names later --;
    create table Columns_Number as
        select name, varnum
        from dictionary.columns
        where libname = "WORK" and memname = "REFI_DSN_ANLY" and
            varnum < &Type_Column;
    %put Dataset has &sqlobs rows;

    %*-- Create the DROP list for columns before _type_ --;
    select name into :Drop_List
        separated by " "
        from dictionary.columns
        where libname = "WORK" and memname = "REFI_DSN_ANLY" and
            varnum < &Type_Column
        order by name;
    %put Drop_List has &sqlobs rows;

    %*-- Create the COALESCE list for columns before _type_ --;
    select name as Coal_Name into :Coalesce_List
        separated by ", "
        from dictionary.columns
        where libname = "WORK" and memname = "REFI_DSN_ANLY" and
            varnum < &Type_Column
        order by name;
    %put Coalesce_List has &sqlobs rows;

    %*-- Create the list of columns AFTER _type_ --;
    select name as Coal_Name_After into :Coalesce_After
        separated by ", "
        from dictionary.columns
        where libname = "WORK" and memname = "REFI_DSN_ANLY" and
            varnum >= &Type_Column
        order by name;
    %put Coalesce_After has &sqlobs rows;
quit;
%put Start processing at varnum &Type_Column;
%put Drop List = &Drop_List;

/*-- Define the statements to setup Last_Type, depending on the _type_ var ---*/
%let Last_Type_DCL    = ;  /*-- Declare Last_Type as a character --*/
%let Last_Type_Retain = ;  /*-- Retain and initialize Last_Type --*/
data _null_;
    /*-- _type_ is a CHARACTER variable --*/
    if ("&Type_Column_Type" = "char") then do;
        /*-- length Last_Type $ &Type_Column_Len; --*/
        call symput('Last_Type_DCL', "length Last_Type $ &Type_Column_Len");
        /*-- retain Last_Type " "; --*/
        call symput('Last_Type_Retain', 'retain Last_Type " "');
    end;
    /*-- _type_ is a NUMERIC variable --*/
    else do;
        /*-- retain Last_Type 0; --*/
        call symput('Last_Type_Retain', 'retain Last_Type 0');
    end;
run;
```

```
%put Last_Type_DCL = &Last_Type_DCL;
%put Last_Type_Retain = &Last_Type_Retain;

/*-- Process the Means dataset, and add the column number that has data ------*/
data Refi_DSN_Anly2;
    set Refi_DSN_Anly;
    /*-- The character length is got form the Means output dataset (above) --*/
    &Last_Type_DCL;     /*-- Set the length of Last_Type, if necessary ------*/
    &Last_Type_Retain;  /*-- Retain and initialize Last_Type ----------------*/
    retain Column_Number;
    drop Last_Type;
    ** drop &Drop_List;

    /*-- Initialize the counters --*/
    if _N_ = 1 then do;
        Last_Type = _type_;
        Column_Number = &Type_Column - 1;
        end;

    /*-- The relevant column number is based upon _type_, which is ordered --*/
    if (Last_Type ^= _type_) then do;
        Last_Type = _type_;
        Column_Number = Column_Number - 1;
        end;
run;

/*-- Process the Means dataset, and produce the final statistics add ------*/
proc sql noprint;
    %*-- Squish the leading columns that occur before the _type_ column --;
    create table Refi_DSN_Anly3 as
        select coalesce(&Coalesce_List) as Values format=$62.,
            &Coalesce_After, Column_Number
        from Refi_DSN_Anly2;
    %put Dataset has &sqlobs rows;

    %*-- Add back the column name from Refi_DSN_Anly2, using Col# as key --;
    create table Refi_DSN_Anly4 as
        select b.name as Variable, a.*
        from Refi_DSN_Anly3 (drop=_type_ rename=(_freq_=Count)) a,
           (select name, varnum
            from dictionary.columns
            where libname = "WORK" and memname = "REFI_DSN_ANLY2" and
            varnum < &Type_Column) b
        where a.Column_Number = b.varnum;
    %put Dataset has &sqlobs rows;

    %*-- Create LABEL list for the first 3 variables --;
    select compress(name || "=") into :Label_1
        separated by " "
        from dictionary.columns
        where libname = "WORK" and memname = "REFI_DSN_ANLY4" and
            varnum <= 3 and upcase(name) ^= "COLUMN_NUMBER";
    %put Label_1 has &sqlobs rows;

    %*-- Create LABEL list to order columns by varname and statistic --;
    select compress(name || "=") into :Label_2
        separated by " "
        from dictionary.columns
        where libname = "WORK" and memname = "REFI_DSN_ANLY4" and
            varnum > 3 and upcase(name) ^= "COLUMN_NUMBER"
        order by upcase(name);
    %put Label_2 has &sqlobs rows;
quit;

/*-- Create the new dataset with re-ordered columns --*/
data Refi_DSN_Anly5;
    label &Label_1 &Label_2 Column_Number=;
    set Refi_DSN_Anly4;
run;

/*----------------------------------------------------------------------------*/
```

```
/*-- R E P O R T S ------------------------------------------------------------*/
/*-----------------------------------------------------------------------------*/

PROC EXPORT DATA=Key_Values_Contents
    OUTFILE= "&XLS_File"
    DBMS=EXCEL REPLACE;
    sheet="Contents of &Refi_DSN";
RUN;
PROC EXPORT DATA=Refi_DSN_Anly5
    OUTFILE= "&XLS_File"
    DBMS=EXCEL REPLACE;
    sheet="Stats on &Refi_DSN";
RUN;

/*-- If &Check_Keys ^= cards4, then the next PROC EXPORT runs --*/
data _null_;
    &Check_Keys;
run;
PROC EXPORT DATA=key_values2
    OUTFILE= "&XLS_File"
    DBMS=EXCEL REPLACE;
    sheet="Dupe Keys on &Refi_DSN";
RUN;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

ods rtf close;
ods listing;

/*-----------------------------------------------------------------------------*/
/*-- E N D --------------------------------------------------------------------*/
/*-----------------------------------------------------------------------------*/
```