Paper 2103-2015

Preparing Students for the Real World with SAS® Studio

AnnMaria De Mars, National University

ABSTRACT

A common complaint of employers is that educational institutions do not prepare students for the types of messy data and multi-faceted requirements that occur on the job. No organization has data that resembles the perfectly scrubbed data sets in the back of a statistics textbook. The objective of the Annual Report Project is to quickly bring new SAS® users to a level of competence where they can use real data to meet real business requirements. Many organizations need annual reports for stockholders, funding agencies, or donors. Or, they need annual reports at the department or division level for an internal audience. Being tapped as part of the team creating an annual report used to mean weeks of tedium, poring over columns of numbers in 8-point font in (shudder) Excel spreadsheets, but no more. No longer painful, using a few SAS procedures and functions, reporting can be easy and, dare I say, fun. All analyses are done using SAS® Studio (formerly SAS® Web Editor) of SAS OnDemand for Academics. This paper uses an example with actual data for a report prepared to comply with federal grant funding requirements as proof that, yes, it really is that simple.

INTRODUCTION

My first few years in business, I refused to hire anyone right out of a masters or Ph.D. program. The only experience they had was with perfect data from the back of statistics textbooks while our companies, in contrast, deal with nothing but messy data collected and entered by people who sometimes seem to be on a mission to make our lives difficult. In the past 15 years teaching as an adjunct professor, I am determined not to be part of the problem. By the second week of class, all of my students will be exposed to real data, and they are required to complete projects that are identical to what they will be required to do on the job.

The software formerly known as SAS Web Editor

This example uses SAS Studio, which is available free to students and faculty using SAS On-Demand. There are a few limitations of SAS Studio that professors (and their students) should keep in mind when deciding whether to use it:

- Only the professor can upload data to the course directory,
- Certain procedures are not available, most noticeably, PROC IMPORT for Excel files, because the web editor runs on Unix server
- Saving permanent SAS datasets to the course directory is not an option for students.

These drawbacks are outweighed by the significant advantage of being able to have data made available to students from off-campus and the fact that all statements in SAS Studio will run exactly as written on any machine with a regular SAS license. The primary objective of this assignment was to give students experience using real data to meet real requirements of doing business. That is, when students finish the course projects, they have experience that is directly transferable to the workplace. Gaining a bit of familiarity with Unix is also a plus. A second objective is to learn some of the basic statements, procedures and functions that can enable a student or new employee to become productive with SAS in a very short period of time.

THE PROBLEM

Many federally-funded programs require an annual progress report. This course assignment uses data from the first year of a Small Business Innovation Research Award from the U.S. Department of Agriculture. The purpose of this grant was to develop and test educational games that teach mathematics for grades three through six. Two games were developed in the first grant year. The assignment was designed to mirror the typical steps in reporting; taking an inventory of the data sets available, reading data sets into SAS when necessary, a first pass at cleaning the data, automating the data cleaning, scoring measures, descriptive statistics, tables and graphs. It should be noted that while students all have some basic knowledge of statistics, it is assumed that this is their first experience using SAS.

STEP 1: PROC DATASETS, OR, WHAT HAVE WE HERE?

PROC DATASETS is a very versatile procedure and we're going to use it twice in this project. The first pass is just to see what datasets we have available.

```
libname mydata  "/courses/abc123x456/sgf15/";

proc datasets library= mydata ;
```

This produces a listing of all of the datasets in the directory specified in the LIBNAME statement. Partial output is shown below.

	Name	Member Type	File Size	Last Modified
1	SLPOST_SCORED	DATA	208896	03/20/2015 07:59:51
2	SLPRE_SCORED	DATA	487424	03/20/2015 07:59:51
3	SL_ANSWERS	DATA	619520	03/20/2015 07:59:52
4	SL_PRE_POST	DATA	196608	03/20/2015 07:59:52

Output 1. Output from a PROC DATASETS Statement

The file size seems reasonable and the date last modified fits with the cut-off for data collection for the year, so it is clear these are the most current data. However, two games were developed, and the only data sets are for one game, Spirit Lake.

STEP 2: PROC IMPORT

Whenever possible, I try reading in the data using the IMPORT procedure, because, as can be seen below, it is very simple. There is no need to declare variable lengths, type or names. Only three statements are required.

```
proc import out= work.studentsf datafile=
        "/courses/abc123x456/sgf15/Fish_students.csv" dbms=csv replace;
        getnames=yes;
        datarow=2;
```

This PROC IMPORT statement gives the location of the data file, specifies that its format is csv (comma separate values), the output file name is studentsf, in the work directory and that if the file specified in the OUT= option already exists, I want it to be replaced.

➡ NOTE: You're working with Unix now. That means when SAS checks for your data file name and path, it is *case-sensitive*. If your dataset is named fish_students.csv the statement above will return an error.

The next statement in the procedure will cause SAS to get the variable names from the first row in the file. Since the variable names are in the first row of the file, the data begins in row 2.

LIMITATIONS OF PROC IMPORT

As handy as it can be, PROC IMPORT has its limitations. Three we ran into in this project are:

- Excel files cannot be uploaded via FTP to the SAS server, so , no PROC IMPORT with Excel if you are using the SAS Studio,

- If the data that you want to import is a type that SAS does not support, PROC IMPORT *attempts* to convert the data, but that does not always work.
- For delimited files, the first 20 rows are used to determine the variable attributes. You can give a higher value for the number of rows scanned using the GUESSINGROWS statement, but you may have no idea what that higher value should be. For example, the first 300 rows may all have numbers and then the class that was records 301-324 has entered their grade as “4th” instead of the number 4.

STEP 3: WHEN PROC IMPORT FAILS, USE A FILENAME AND DATA STEP

In the third step, we use a FILENAME, DATA, INFILE and INPUT statement

```
filename inf3 "/courses/abc123x456/sgf15/Fish_pretest.csv" ;

data pretestf ;
  infile inf3 firstobs = 2 delimiter = "," missover ;
  attrib username length = $16. ans18 length = $20.  ;
  input date_test $ username $ age      school  $ grade  ans18 $  ;
```

Note that the FILENAME statement includes the name of the file, unlike the LIBNAME statement, which only includes the complete path for the directory. This makes sense if you think about it for a minute, because in the LIBNAME statement you are referencing a directory where many SAS files may be stored, while the FILENAME statement refers to a specific file. I only mention this because a common error for new SAS users is leaving off the name of the file.

The DATA statement creates a temporary dataset named pretestf. With the SAS Web Editor, students do not have permission to write permanent datasets to the course directory.

The INFILE statement links back to the FILENAME statement with the fileref – in this case, inf3. Again, the data are in the second line of the file, so the first data observation read will be in line 2. The delimiter between fields is a comma. When SAS runs out of data, say, a student did not answer question number 18, I want the field to be set to missing. Without that option, SAS would go on to a new line when it ran out of data.

The ATTRIB statement is used to assign variable attributes – length, format, informat and labels. It can also be used to reorder variables in an existing data set. In this case, we want to set the length of selected variables. The maximum username assigned to students was 12 characters, but because they occasionally made errors typing in their names and added extra characters, we are going to assign the username a length of 16 in each data set and fix the data entry errors in a later step.

With variable lengths already assigned, the INPUT statement is simply a list of the variable names, in the order they occur in the data set, remembering to put a $ after each character variable.

STEP 4: TAKE A FIRST LOOK AT THE DATA

If there was only one lesson I could drill into anyone who will be doing data analysis, it would be this – *Before you do anything else with a data set, look for data problems.* Your problem could be the same username showing up many times when it should be unique, out of range variables like the 99-year-old in fourth grade, usernames that should not show up but do, e.g. the answer keys used for scoring and our interns who acted as game testers.

What you find will vary widely from one project to another, but you are almost guaranteed to find something that will skew your results if you don’t fix it. This is a reality check. The particular PROCs you use may also vary but I’d recommend always using PROC MEANS to examine all of your numeric variables. I used an option to set the maximum decimals equal to two because that seemed easier to read and more than sufficient with variables like age and grade.

```
proc means data= studentsf maxdec=2;
```

Variable	N	Mean	Std Dev	Minimum	Maximum
Age	494	12.65	9.65	0.00	100.00
Grade	494	4.64	0.96	1.00	7.00

Output 2. Output from a PROC MEANS Statement

I can tell at a glance that the number of subjects in this data set is about what I expected. The mean and standard deviation for age are a little high and it is apparent why when I look to the right and see that the minimum age entered was 0 and the maximum was 100. The grade distribution could be correct. Our games are designed for fourth through sixth graders, but we do have a few gifted children in grades 1-3 that use the games, as well as students in middle school who are below grade level. Still, I'd better do a PROC FREQ just to be sure.

```
proc freq data = studentsf ;
   tables grade username ;
```

grade	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	7	1.42	7	1.42
2	1	0.20	8	1.62
3	21	4.25	29	5.87
4	190	38.46	219	44.33
5	208	42.11	427	86.44
6	49	9.92	476	96.36
7	18	3.64	494	100.00

Output 3. Output from a PROC FREQ

The frequency distribution by grade looks exactly as I had expected, with one classroom each of advanced third-graders and not so advanced seventh- graders. There is one first-grader who is an outlier. It's all good and nothing further needs to be done with this variable.

I also included the username in the TABLES statement because I had good reason to be concerned there were problems there, plus I knew I had less than 500 records.

After users are familiar with some standard procedures, there are many good methods for automating the data quality checks. I've written about it previously myself (De Mars, 2012).

NOTE: You would often not want to do a frequency distribution with usernames because a large dataset might have hundreds of thousands, or even millions, of unique names. Know your data!

STEP 5: LOVE THOSE CHARACTER FUNCTIONS

Our first pass at fixing the data makes ample use of functions. Beta testers – teachers, staff, interns – played the game but their data should be deleted from the datasets for the annual report. There is also a problem with data entry errors. The subjects in this study were children in grades three through six and they frequently mistyped their usernames. SAS has a wealth of character functions and this is a first opportunity to get to know and love three of them.

username = **compress(upcase**(username),'. ') ;

The UPCASE function, not surprisingly, changes the value of a variable to upper case. The COMPRESS function, if given only the variable as an argument, will remove blanks from a value. You can, however, include additional characters to remove. Since many of the students entered their names on some days as JohnDoe and others as John.Doe , we are removing both blanks and periods.

Here is a general tip. Any time you find yourself thinking, “Gee it would be nice if SAS did thing X”, it is a pretty good bet that someone else thought the same idea and there is a function for it. The INDEX function is a perfect example of that. Our testers played the games many, many times and used usernames like “tester1”, “this.test”, “skippy the tester” or “intern7”.

“Wouldn’t it be nice if there was way to find out whether a given string appeared anywhere in a value?”

Enter the INDEX function, which does exactly that. This function is case-sensitive, but since we already converted the username to upper case above, that is no problem for us. The statement below will do exactly what we want.

```
        if index(username, “TEST”) > 0 or index(username,”INTERN”) > 0 then delete ;
```

The INDEX function returns a number that is the starting position in the string of the substring we are trying to find. So, in “skippy the tester”, the value is 12, in “tester1” it is 1. If the string is not found, the value is 0.

STEP 6: %INCLUDE IS YOUR FRIEND

This report uses eight datasets that all have the same problems with the usernames. In addition to needing to remove every username that contained the word “test” or “intern” we also needed to delete specific names of the classroom teachers who had played the game. We needed to correct names that were misspelled. Here are a few examples of a very long list of statements:

```
    if username in("MSDMARIA","1ANSWERKEY","MSDELAPAZ","MSCARRINGTON") then
    delete ;
     if username = "GRETBUFFALO" then username = "GREYBUFFALO" ;
      else if username = "HALFHORES" then username = "HALFHORSE" ;
      else if username ="TTCARLSON18TTCARLSON18" then username =
"TTCARLSON18" ;
```

These problems occurred in every dataset. A second problem found when looking at the contents of each of the 8 datasets was that the username variable was not the same length in all of them, which could cause problems later when they were to be merged together or concatenated. Also, now that all of the usernames have been cleaned up, none should be over 12 characters in length. Wouldn’t it be nice if there was a way to just get the first n characters of a string? Enter our fourth character function, substr, which returns a substring of a variable beginning at any position and for as many characters as you like. Problem solved.

```
        newid = substr(username, 1, 12) ;
```

It seems pretty inefficient to write this set of statements eight times in eight different data sets. Also, next year we will have another eight data sets, and some will have these same students’ usernames and same problems. Wouldn’t it be a lot easier to have these statements in one place and add to the “fixnames.sas” file whenever we find a new problem? So, now we have the write once, use anywhere solution of %INCLUDE.

What %INCLUDE does

The %INCLUDE statement references lines from an external file and processes them immediately. It has *almost* the exact same effect as if you had copied and pasted those lines write into your program. The exception that makes it “almost” is that a %INCLUDE statement must begin at a statement boundary. That is, it has to be either the first statement in your program or occur after a semi-colon ending a statement as in this example.

```
        data studentsf ;
```

```
            infile inf delimiter = "," missover ;
            attrib teacher length = $12. username length = $ 16. ;
            input username $ age sex $ grade school $ teacher $ ;
            %include "/courses/abc123x456/sgf15/fixnames.sas" ;
```

Also, you need to think about it as if you had copied and pasted those lines into your program. Is it still valid code? Whenever using %INCLUDE, you should make sure the code runs in your program as expected, with no errors, before cutting it out and making it an external file.

To source or not to source

The default is not to show the statements that were in the included file. Generally, this is desirable. This is code you have already debugged and if you are using it multiple times (otherwise, why bother with the %INCLUDE), having the same 20 lines repeated 8 times in your log just makes it harder to debug.

Professors might want to use real data but hide all of the messy data handling from the students initially in fear they would run screaming for the door. I meant, professors might want to gradually introduce SAS statements and functions for data handling. In either case, students could use the %INCLUDE statement as shown in the example above.

To see the code include in your log is quite simple, just add a source2 option as shown.

```
      %include "/courses/abc123x456/sgf15/fixnames.sas" /source2 ;
```

It will be in your log as follows

```
NOTE: %INCLUDE (level 1) file "/courses/abc123x456/sgf15/fixnames.sas is file
       "/courses/u_mine.edu1/wuss14/fixnames.sas.
 419       +username = compress(upcase(username),". ") ;
 420       +if (index(username,"TEST") > 0 or index(username,"INTERN") > 0
```

The + signs next to each statement denote it was in the included file.

WHAT HAVE WE HERE- AGAIN

Now that we have cleaned up the data in every data set, we are not quite ready to start merging them together. A common problem is that data sets have different names, lengths or types for the same variable. You'd be wise to check the variable names, types and lengths of all the variables.

```
      proc datasets library= work ;
      contents data = _all_ ;
```

You may recognize the code above as what we did in Step 1. This time, we added another statement. The "contents data = _all_ " will print the contents of all of the data sets. In perusing the contents, I see that grade is entered as character data in one – 5th, 4th and so on, while it is numeric data in another. This is the sort of thing you never run into in "back of the textbook" data, but that shows up often in real life.

NOTE: If you had hundreds of data sets in your library, or even a few data sets with thousands of variables, this is going to give you more output than you want to wade through. This should seldom be the case in a working directory, and almost never in a working directory in a classroom setting.

STILL LOVE THOSE CHARACTER FUNCTIONS

If only there was a way to eliminate every alphabetic character – and what did we just discuss about whenever you caught yourself saying that?

```
      nugrade=compress(upcase(grade),'ABCDEFGHIJKLMNOPQRSTUVWXYZ ') + 0 ;
        drop grade ;
```

This use of the COMPRESS function does exactly that, removes every alphabetic character. Since we used the UPCASE function first, we only need to worry about the upper case letters. Adding a zero as the variable is created will make it numeric.

Maybe counting Steps 1 and 5 again is cheating, but it also highlights two important points. First, data cleaning and analysis is an iterative process. You often don't catch all of the problems on the first pass through the data. Second, you can be a pretty effective programming by applying a small number of steps over and over. This is not to say you shouldn't branch out and learn new techniques, you should, but it is encouraging to know that you can accomplish quite a lot with a handful of trusty statements and functions.

STEP 7: IT'S ALWAYS GOOD TO KEEP YOUR OPTIONS OPEN

When merging data sets, it can be useful to keep a record of the source of the data. The IN = option does just that. It creates a variable that is 1 if you the record came from a particular data set and 0 otherwise. A fact that may throw off the new programmer is that values of IN= variables are available to program statements during the DATA step, but the variables are not included in the SAS data set that is being created. Once your data step is ended, the s and f variables created in the example below are gone. If you want to keep that information, you need to assign those values to a new variable you create, as shown below.

```
Data allstudents ;
  set sl_students (in = s rename =(nugrade= grade)) studentsf (in=f) ;
  if s then game = "SPIRIT" ;
         else if f then game = "FISH  " ;
```

It is also helpful to rename variables "on the fly" so that the variable names will match in the output dataset. Note the TWO sets of parentheses dataset_name (rename = (oldname = newname)) . That is not a typing error. Yes, you need two sets of parentheses. Because SAS sets variable length from the first occurrence in the data set, I add two spaces after the word FISH so that the variable length would be set as six.

I don't need a procedure at all to find the first number for my report, "How many total student records do we have? " All I need is to look in my SAS log.

```
NOTE: There were 1464 observations read from the data set WORK.SL_STUDENTS.
 NOTE: There were 503 observations read from the data set WORK.FISH_STUDENTS.
 NOTE: The data set WORK.ALLSTUDENTS has 1967 observations and 9 variables.
```

PROC FREQ AND FINALLY SOME RESULTS

The FREQ procedure is my favorite and its versatility is often overlooked. It can do so much more than just produce frequency tables. Let's start with answering a few questions. We know that some students are in our database multiple times. These aren't necessarily errors. They may have finished the game and then started over. Perhaps they were in our beta group and then played the same game again when it was released commercially and their school obtained a license. They may have played two of our games, Fish Lake and Spirit Lake. I'd like to know:

- How many unique student records do we have?
- How many unique students do we have per game? That is, if you played Spirit Lake and Fish Lake, it would be acceptable to count you as a unique id when we are doing analyses just of that game.

There is a bit of touchy subject here, too, because these are records from minors, I want to be extra careful and not show you the id values, just the number of unique IDs.

```
proc freq data = allstudents  nlevels;
  tables newid / noprint  ;
```

Produces the following output:

Number of Variable Levels			
Variable	Levels	Missing Levels	Nonmissing Levels
newid	1159	1	1158

Output 5. Output from PROC FREQ with NLEVELS and NOPRINT options

The NLEVELS option on the PROC FREQ statement requests the number of levels of the variable given in the TABLES statement. The NOPRINT option suppresses printing the frequency distribution table. Even if there is nothing confidential about your data, NOPRINT is still a handy option to keep in your back pocket for those times when you are really only interested in summary statistics and don't want to wade through twenty pages of tables to find the one number you need. Note in here that we have reduced the number of observations from 1,967 unique records to 1,158. If you didn't think data cleaning before analysis was important, I hope you see the error of your ways.

How NOT to do things

You might think the next question, unique student records per game, would be answered like this:

```
proc freq data = allstudents  nlevels;
  tables newid*game / noprint  ;
```

That would be logical. Wrong, but logical. The code above will actually give you these results, which are not exactly what we had in mind.

Number of Variable Levels	
Variable	Levels
newid	1158
game	2

Output 6. Output from PROC FREQ with Cross-tabulation, NLEVELS and NOPRINT options

STEP 8: SORT, THEN FREQ

To obtain the number of observations for each of the two levels separately, you need to sort the data, then do a PROC FREQ with a BY statement as shown.

```
proc sort data = allstudents ;
  by  game ;
proc freq data = allstudents  nlevels;
  tables newid / noprint  ;
```

Game = FISH

Number of Variable Levels	
Variable	Levels
newid	497

Game = SPIRIT

Number of Variable Levels			
Variable	**Levels**	**Missing Levels**	**Nonmissing Levels**
newid	746	1	745

Output 7. Output from PROC FREQ with BY statement, Cross-tabulation, NLEVELS and NOPRINT options

Note that the number of levels per game added up to 1,242– which is more than the total obtained in our previous analysis. If these should have been non-overlapping groups, say, the number of separate ids for males and females, then this would represent a problem. In our case, it is not a problem. This just points out a fact that should be become a mantra for the rest of your career – you need to have some substantive knowledge of what your data represents, or work closely with someone who does. When I'm working on a study on what causes tooth decay, I'm talking to the dentists who collected that data every day.

PROC FREQ AND THE RESULTS ARE POURING IN

This next simple analysis answered some important questions.

```
proc freq data=answersf order= freq;
        tables problem_num*prob_correct /nocol ;
```

At the next step, for each of the games, I need to answer a pretty substantial question. While the low mathematics achievement of students attending our target schools has been documented, we now have data on a large number of students at a finely-grained level of detail. The teachers were under the impression that students were not doing too badly as the students seem to be getting most of the answers correct. The first statistic we want to inspect, at the end of the table and labeled A, is the percentage of students who answered a problem correctly, 82%. This would seem to indicate the students are doing well.

Here is the first of two times where the ORDER=FREQ option comes in useful. Problems in the games are in order of difficulty. The first is two years below grade level. The next statistic we want to inspect, labeled B, is the percentage of the total answers that were to the first question and we can see that 25% of the time students were answering a question it was two years below grade level.

One of the objectives of this project was to develop valid and reliable measures for assessment within a computer game. The second use of the ORDER = FREQ option is that it shows the proposed order of difficulty on the test is generally appropriate. If you look at the row labeled C, Question 8 seems to be out of order and examining the wording and content of that question is warranted. (In case you are dying to know, it is a question on adding mixed fractions and converting the result from an integer to an improper fraction.) The ORDER = FREQ is the total number of times a category appears, not the number of times students answered correctly.

The last statistics we'll look at for now, labeled D, relate to the most difficult question in this particular game. Even though if you look at the percentage correct for that question, it seems pretty high, 9 out of 17 students answered correctly, that is 53%. However, inspection of the last column shows that less than 1% of the time were students attempting this question. (A careful reader would note that the total number of students attempting the first problem, N=539, exceeds the number of students, because many students played the game more than once. Re-analyzing the data with only including the first attempt at the question is not reproduced here because it showed essentially the same pattern of results. Those results showed a similar pattern, and also revealed that nearly 1 in 4 students gave up before attempting the first question.)

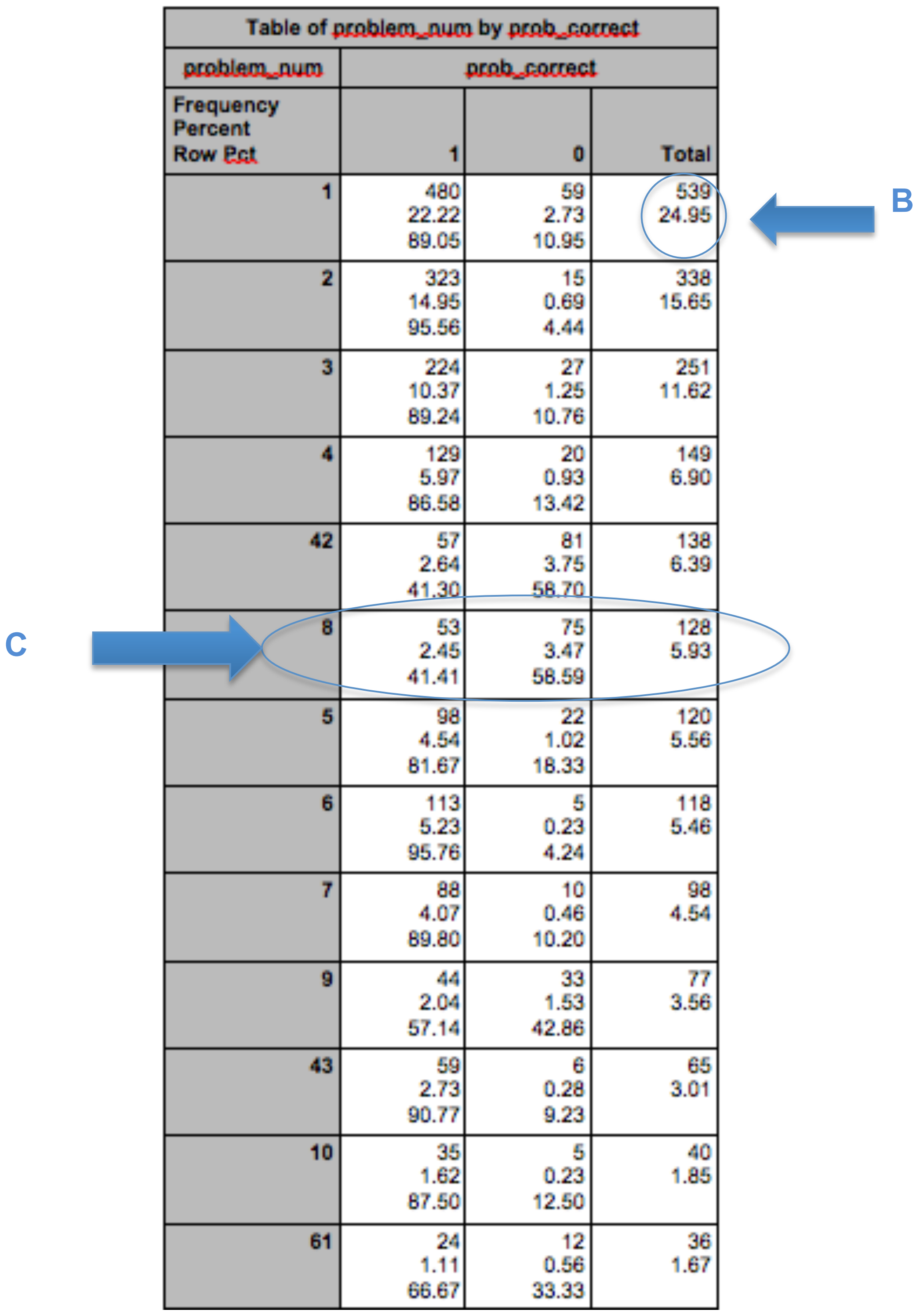
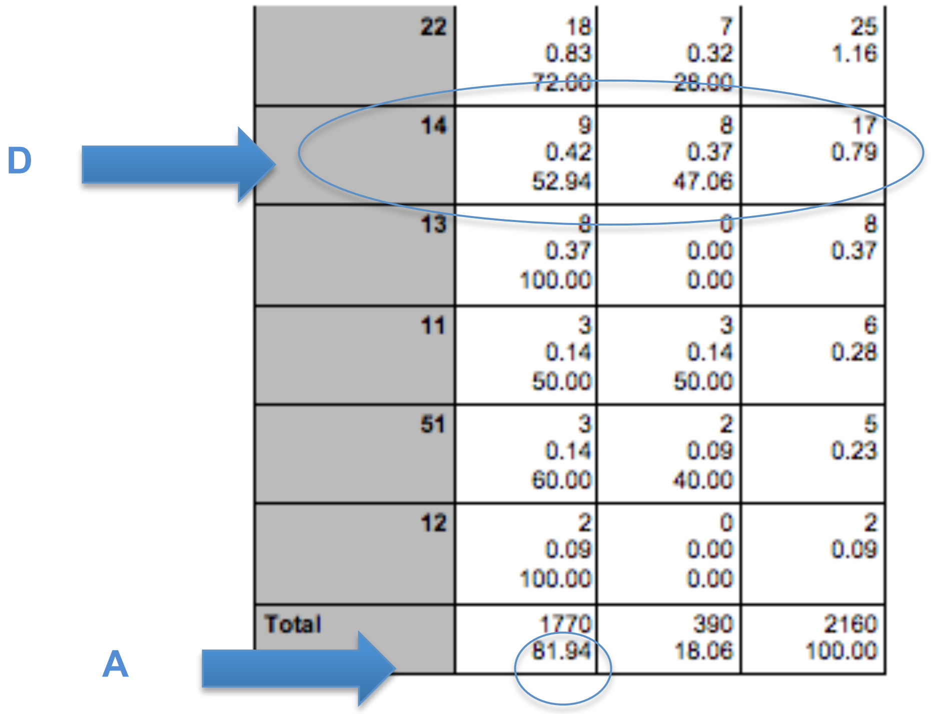


Table of problem_num by prob_correct

problem_num	prob_correct		
Frequency Percent Row Pct	1	0	Total
1	480 22.22 89.05	59 2.73 10.95	539 24.95
2	323 14.95 95.56	15 0.69 4.44	338 15.65
3	224 10.37 89.24	27 1.25 10.76	251 11.62
4	129 5.97 86.58	20 0.93 13.42	149 6.90
42	57 2.64 41.30	81 3.75 58.70	138 6.39
8	53 2.45 41.41	75 3.47 58.59	128 5.93
5	98 4.54 81.67	22 1.02 18.33	120 5.56
6	113 5.23 95.76	5 0.23 4.24	118 5.46
7	88 4.07 89.80	10 0.46 10.20	98 4.54
9	44 2.04 57.14	33 1.53 42.86	77 3.56
43	59 2.73 90.77	6 0.28 9.23	65 3.01
10	35 1.62 87.50	5 0.23 12.50	40 1.85
61	24 1.11 66.67	12 0.56 33.33	36 1.67

Output 7. Output from PROC FREQ Cross-tabulation with ORDER=FREQ option



22	18 0.83 72.00	7 0.32 28.00	25 1.16
14	9 0.42 52.94	8 0.37 47.06	17 0.79
13	8 0.37 100.00	0 0.00 0.00	8 0.37
11	3 0.14 50.00	3 0.14 50.00	6 0.28
51	3 0.14 60.00	2 0.09 40.00	5 0.23
12	2 0.09 100.00	0 0.00 0.00	2 0.09
Total	1770 81.94	390 18.06	2160 100.00

Output 7 (continued). Output from PROC FREQ Cross-tabulation with ORDER=FREQ option

Wait, why do questions 2.2, 4.2, 4.3, 5.1 and 6.2 show up out of order? Because this is an on-going research project and after the first year, cooperating teachers recommended that more math problems be added to the games. Since these problems were added in the second year of data collection, fewer students had the opportunity to try them. This, yet again, emphasizes the importance of knowing your data.

LOVE THOSE FUNCTIONS EVEN MORE

Because the Fish Lake questions are supposed to be more difficult than the Spirit Lake items, it would be of interest to combine the two data sets and compare the number of students who answered each item correctly. There are just two small problems. First, the variable prob_correct was originally entered as a character variable. Because it's impossible to obtain the mean of a character variable, this must be converted to numeric. Further complicating the situation, the problem has the name "probname" and a character type in one data set and the name problem_num and numeric type in the other.

How NOT to do things

A common problem in combining data from different sources is that the same variable will be different types in different data sets – which causes an error. If only there was a way to convert variable types …

You cannot actually change the type of a variable in SAS. What you can do is read the data into a new variable of the type you want. The INPUT function converts character data to numeric and the PUT function converts numeric data to character. Each function takes the arguments (variable,format) as shown below. The new variable, "correctly", will be a numeric variable with the value of prob_correct read into it with the 8.0 format, that is a width of 8 and 0 decimals.

```
Data all_ans2 ;
  set all_ans ;
  correctly = input(prob_correct,8.0) ;
  if game = "SPIRIT" then probname = put(problem_num, $11.);
```

There is already a variable named probname which is character type with a width of 11, so we read the value of the problem_num variable into that using the PUT function. It puts into the probname variable, the value of problem_num.

BEFORE

Game	Probname	Problem_num
FISH	F1	
FiSH	F2	
SPIRIT		8

AFTER

Game	Probname	Problem_num
FISH	F1	1
FISH	F2	2
SPIRIT	8	

Table 1: Data structure before and after applying functions

NOTE: Because you are using a character function, PUT, with a numeric variable, you will get a warning in your SAS log. You should always read your log and decide whether you need to worry about any warnings or not. In this case, the answer is "not". You already knew the variable was numeric.

```
WARNING: Variable problem_num has already been defined as numeric.
```

When is a FISH not a FISH?

Random SAS fact: Recall that in Step 7, instead of using an ATTRIB statement we just assigned the value "FISH " with two blank spaces added at the end. Won't this cause problems? Does "FISH" = "FISH "? As it turns out, yes, it does. SAS will ignore trailing blanks but not leading ones, so:

"FISH" = "FISH " ≠ " FISH"

STEP 9: A CHART!

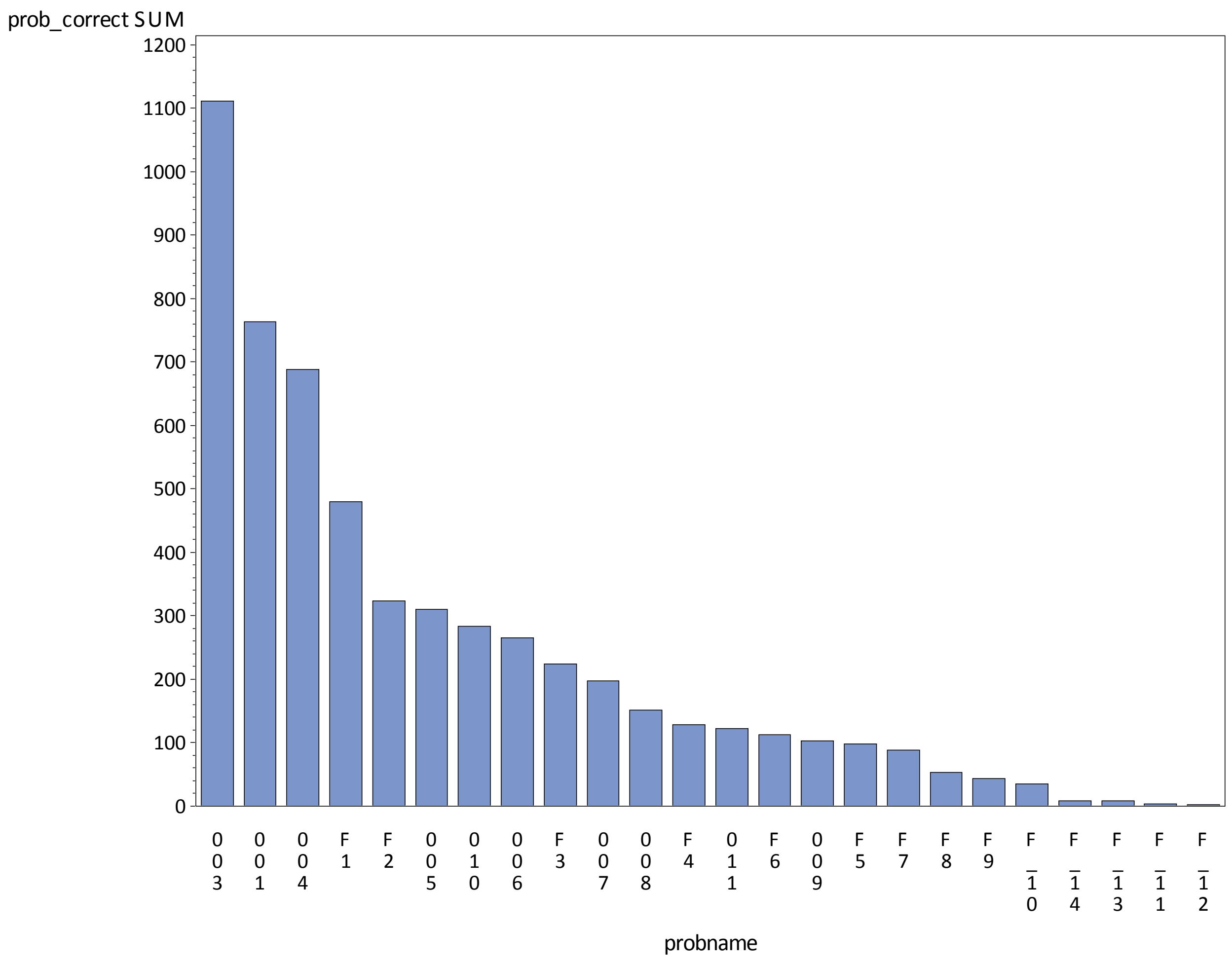
Speaking of appropriate level of difficulty, students had two games available. The easier game, Spirit Lake, began at the second grade level, with subtraction. The first few questions were multiplication items at the third grade level, before moving to the fourth and fifth grade items, which were at the ostensible grade level for the students. If our assumptions about the difficulty of the items are correct, we should see that fewer students answered the items for the second game correctly, and the further along each game they went, the fewer correct answers occurred. The code below produces the output we want.

```
proc gchart data=all_ans2 ;
 vbar probname / sumvar=correctly descending;
```

The first statement simply begins PROC GCHART. If you try to follow these steps and you get a really ugly chart that looks like it came from the year 1973 when charts were still produced on green and white lined paper, you made a mistake and left off the "G" in GCHART.

The VBAR statement requests a vertical bar chart for the variable named, in this case, the "probname" variable. After the / are the options. SUMVAR gives a variable that is going to be used to create statistics for each bar. The default variable type is sum, which is what we want, so there is no need to specify a type. The DESCENDING option requests the bars be shown on the chart in descending order of the statistic.

Problem names in the second game begin with a letter, to easily distinguish between the games. This chart paints a pretty clear picture. We can see that the easiest problems from the first game were answered by most students, followed by the easiest problems from the second game. There is a pretty clear separation, with the eight items with the least number of correct answers all from the second game. This was the last question from the first game and it strongly suggests we need to take a look at the appropriateness of that question.



Output 8. Output from PROC GCHART

Recall from Step 8 saying that we should take another look at the item that was out of order of expected difficulty, question number 8 from the first game, Spirit Lake. Now it isn't out of order. What happened? One point made by this step is the importance of looking at your data in multiple ways.

The frequency distribution was ordered by how frequently the questions appeared, that is, how often students got to that point in the game. The chart shows how often students got the item correct. Students may have actually missed that item at a higher frequency or it may represent a technical problem. An earlier version of the game had a problem when an internet connection was dropped that it kept repeatedly sending the data to the server until it found a connection. (It was fixed. We checked.)

It's not always easy to track down the option you need in SAS/GRAPH procedures, or even to know that those options exist. (Did you know how to get the bars ordered by descending height of a second variable? Be honest, now.) A gentle introduction is offered by Cochran (2004).

PROC FREQ, ONE LAST TIME

Because the Spirit Lake game teaches and tests information taught in earlier grades than Fish Lake, we should see, then, that there is a significant difference between the two tests with students scoring more questions correct on the easier game. PROC FREQ has a variety of options for inferential statistics. Here we'll just take the simplest one, a chi-square, which is requested with a simple CHISQ option added to the TABLES statement.

```
proc freq data=all_ans ;
  tables game*prob_correct / nocol nopercent chisq;
```

Table of game by prob_correct			
game	prob_correct		
Frequency Row Pct	0	1	Total
FISH	390 18.06	1770 81.94	2160
SL	2091 32.70	4303 67.30	6394
Total	2481	6073	8554

Statistic	DF	Value	Prob
Chi-Square	1	168.2138	<.0001
Likelihood Ratio Chi-Square	1	179.4845	<.0001
Continuity Adj. Chi-Square	1	167.5033	<.0001
Mantel-Haenszel Chi-Square	1	168.1941	<.0001
Phi Coefficient		-0.1402	
Contingency Coefficient		0.1389	
Cramer's V		-0.1402	

Output 9. Partial Output from PROC FREQ with CHISQ Option

This is barely scratching the surface of what you can do with PROC FREQ, in fact, the above only includes part of the output from chi-square, since I left off the Fisher's Exact Test table. One basic paper I recommend to give you an introduction to its many uses is "PROC FREQ: It's more than counts" , Severino (2000).

This is also a really mean thing to do to students since of the nine tests produced by this chi-square analysis (including those not shown), not one is appropriate for answering the question of whether the second game truly is harder. The reason is that all of the statistics produced by default assume independent samples and what we have is far from independent observations. Not even the McNemar test for repeated measures is appropriate (and that is also an option for PROC FREQ), because that is only appropriate for 2 x 2 tables and students may have completed 5 or 10 problems in each game.

The point of this step is to know your data (I cannot say that enough), including how it was collected and who it represents. Often, teachers began all students with the easier game, so only students who had successfully completed the first game ever had the opportunity to begin the second game. Other teachers assigned students to play games based on their assessment of the student's proficiency level. For both of these reasons, the players were non-randomly assigned to play the games.

What should students do with the highly significant results from the chi-square analysis? The answer is, "*They should do nothing with it. They should throw it out.*" This is another part of data analysis in the real world. You cannot assume that every analysis you are provided is appropriate and useful.

This step leads into a class discussion of what would be appropriate analyses for determining if games were of increasing difficulty, which leads into the next step of trying to create a pretest measure of mathematical proficiency.

STEP 10: MORE THAN CORRELATIONS WITH PROC CORR

While one might be excused for thinking that PROC CORR only produces correlations. It actually has a few other nifty uses. Of interest for our purposes are that it produces both descriptive statistics and reliability statistics. All it takes is two statements

```
      proc corr data= sl_pre alpha nocorr ;
         var sc1- sc24 ;
```

Descriptive statistics are produced by default. The NOCORR option suppresses printing the correlation matrix, since I'm not in the mood to look at 576 correlations and their associated probabilities. The ALPHA option will cause SAS to print a Cronbach alpha coefficient, which is a measure of internal consistency reliability.

Simple Statistics						
Variable	**N**	**Mean**	**Std Dev**	**Sum**	**Minimum**	**Maximum**
sc1	841	0.78597	0.41039	661.00000	0	1.00000
sc2	841	0.87515	0.33075	736.00000	0	1.00000
sc3	841	0.86207	0.34503	725.00000	0	1.00000
sc4	841	0.85493	0.35238	719.00000	0	1.00000
sc5	841	0.51249	0.50014	431.00000	0	1.00000
sc6	841	0.45184	0.49797	380.00000	0	1.00000
sc7	841	0.56599	0.49592	476.00000	0	1.00000
sc8	841	0.66350	0.47279	558.00000	0	1.00000
sc9	841	0.54459	0.49830	458.00000	0	1.00000
sc10	841	0.68728	0.46388	578.00000	0	1.00000
sc11	841	0.32342	0.46806	272.00000	0	1.00000
sc12	841	0.32699	0.46939	275.00000	0	1.00000
sc13	841	0.39001	0.48804	328.00000	0	1.00000
sc14	841	0.37693	0.48491	317.00000	0	1.00000
sc15	841	0.41260	0.49260	347.00000	0	1.00000
sc16	841	0.36623	0.48206	308.00000	0	1.00000
sc17	841	0.52556	0.49964	442.00000	0	1.00000
sc18	841	0.57907	0.49400	487.00000	0	1.00000
sc19	841	0.23068	0.42152	194.00000	0	1.00000
sc20	841	0.25565	0.43648	215.00000	0	1.00000
sc21	841	0.31153	0.46340	262.00000	0	1.00000
sc22	841	0.31629	0.46530	266.00000	0	1.00000
sc23	841	0.36029	0.48037	303.00000	0	1.00000
sc24	841	0.43876	0.49653	369.00000	0	1.00000

Output 10. Descriptive Statistics Output from PROC CORR

Because items are scored 1 if correct and 0 if incorrect (always a good practice), inspecting the means shows the percentage of students who answered each item incorrectly. The minimum and maximum values show at a glance that there were no items out of range, all were between 0 and 1.

It isn't a good measure just because you said so

It's rather amazing how often conclusions rest on the assumption that a set of questions the researcher made up are a valid and reliable measure of X. A complete discussion of what makes up a good test is beyond this paper (for that, check Nunnally & Bernstein, 1994).

The thinking behind internal consistency reliability is that there should be correlations among questions that supposedly measure the same thing, in this case, knowledge of topics in third through fifth-grade number and operations standards. One could inspect how well every one of the 24 items correlates with the other 23 items, or, do a single statistic which gives an estimate of how well the items, on average, relate to one another. The next table gives exactly such a statistic, Cronbach's coefficient alpha. Valid values for alpha range from 0 to 1.0. What is a "good" number depends on the context. If similar measures exist with a reliability of .94 then a .84 would be on the low end.

Cronbach Coefficient Alpha	
Variables	**Alpha**
Raw	0.869111
Standardized	0.867399

Output 11. Output from ALPHA OPTION of PROC CORR

CONCLUSION

If you want to build a ship, don't drum up people together to collect wood and don't assign them tasks and work, but rather teach them to long for the endless immensity of the sea."
--Antoine de Sainte Exupery

"If you want to be like most people, start with what you will have your students do. If you want to be more sensible, start with what you want them to learn."

Why I do what I do

Contrary to what my students might think, I don't just make this all up as I go along. I have some very definite reasons for the specific procedures, statements and data sets used in this assignment. While the specific requirements for each report will vary, there are some common goals to meet for one to be a competent in preparing these reports.

My number one goal is for students to understand the process of research and data analysis. It is not a simple linear "pure scientific" process like they learned in middle school, but a messy iterative one. Throughout this paper, I've given a lot of details on the games we created, level of difficulty of each game, individual items. Too often, masters and Ph.D. students (and their professors!) dismissively wave away these details,

"I'm not interested in that. The statistical procedures we teach apply to any data."

Yes, they do, but you're not working with <u>any</u> data, you're working with the data right in front of you and your capability to make sense of the output, to ask questions and give answers with SAS, or any other technology, is going to be <u>far</u> more valuable than the mere ability to email output. The type of data may vary, from health to marketing to educational applications, but the need to dig deeper into the implications of the results is always there. The fact is that all analyses, no matter how complex, are based on an assumption that the data are valid. Ironically, the "grunt work" of insuring that validity, of interim reports, evaluations, is often left in the hands of more junior staff.

My second goal is to develop students to be the most competent SAS programmers possible in a very short period of time.

When presented with two alternatives, learn the more general one first

In SAS, there is almost always more than one way to do things. You can set the variable length using an ATTRIB statement or a LENGTH statement. You can get the contents of a dataset using PROC DATASETS or PROC CONTENTS. On principle, I choose the more general PROC when teaching new SAS users, reasoning if you are only going to know a dozen procedures at first, it's better to make these procedures that can fulfill multiple functions.

There will always be charts and tables (everything else is optional)

There have always been tables – tables of sales figures, tables of sample descriptive statistics. Ever since they started printing graphs in the newspaper, apparently every document over two pages is required to have a chart. I call it the Law of USA Today. Other than those two stalwarts, every report is different and it is impossible in one presentation, or a single course assignment, to cover all the variation of charts, tables and statistics. Almost every SAS procedure offers many, many options. One lesson SAS programmers should learn early is to look for these and find to what new uses they can put old procedures.

Deliberately use SAS to learn basic programming

Debugging is a key concept in programming. All programmers spend more time debugging than they do actually writing code. Thus, early on, I want to introduce new users to the information they can find in their SAS log.

I try to introduce the %INCLUDE statement early on because once users are comfortable with the idea of referencing statements in an external file, it is easy to introduce the concept of macros which are crucial to moving beyond basic SAS programming. Functions, macros, also known as user-written functions, references to code in external files and defining variable attributes are basic features in many other programming languages.

SAS is a component of courses required in many majors not traditionally considered STEM – social work, nursing, education, psychology – and these programming concepts are frequently new to them.

Make the most of what you know

I might be accused of cheating since clearly this project entailed more than ten steps – or did it? If we define a step as something a student must learn, then it did not, and that is a key point I want to emphasize to beginning SAS users, that learning just a few steps can take you a long way.

REFERENCES

Cochran, B. (2004). A gentle introduction to SAS/GRAPH® Software. Paper presented at the Southeast SAS Users Group. Available at http://analytics.ncsu.edu/sesug/2004/TU02-Cochran.pdf

De Mars, AnnMaria, (2012). Data quality macro explained. AnnMaria's Blog Available at http://www.thejuliagroup.com/blog/?p=1364

Nunnally, J. C. & Bernstein, I. H. (1994). Psychometric theory. New York: McGraw-Hill.

Severino, R. (2000) PROC FREQ: It's more than counts. Proceedings of SAS Users Group International. Available at http://www2.sas.com/proceedings/sugi25/25/btu/25p069.pdf

ACKNOWLEDGMENTS

Data collection was funded from a Small Business Innovation Research Award 2013-33610-21047, U. S. Department of Agriculture.

We gratefully thank the staff of Tate Topa Tribal School, Warwick Public School District, Ojibwa Indian School and Turtle Mountain Elementary School for their assistance in data collection and research design.

Thanks to Tricia Anderude for comments on an earlier draft of this paper.

RECOMMENDED READING

- Cody, R. (2010) SAS functions by example, 2nd ed. Cary, NC: SAS Press.
- Murphy, W.C. (2006). Squeezing Information out of Data. *Proceedings of SAS Users Group International* Available at http://www2.sas.com/proceedings/sugi31/028-31.pdf
- Slaughter, S. J. & Delwiche, L.D. (2004). SAS ® Macro Programming for beginners. http://www2.sas.com/proceedings/sugi29/243-29.pdf

CONTACT INFORMATION

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

Name: AnnMaria De Mars
Enterprise: 7 Generation Games
Address: 2425 Olympic Blvd, Ste. 4000-W
City, State ZIP: Santa Monica, CA, 90404
Work Phone: (310) 717-9089
E-mail: annmaria@7generationgames.com
Web: www.7generationgames.com