Paper: SA02-2014

SAS® 101 For Newbies--Not Too Little, Not Too Much--Just Right

Ira Shapiro UnitedHealth Group, Minnetonka, MN

ABSTRACT

The intent of this paper is an overview of the important step-by-step basic features and functions of SAS® and then introduce more complex SAS® features.

INTRODUCTION

Although you may think this paper is long and overwhelming, the reader should approach this paper in baby steps taking one step at a time. The data file most used in this paper (Shoes) has been provided by SAS® and should be available to each reader in their SAS environment. I would suggest copying snippets of the provided code, running it and then experimenting by modifying the code where applicable.

Basic SAS Coding Techniques
SAS Programs have Job Steps and SAS Procs such as:

/*A Sample of a Job Step*/
```
Data out;  Set in;
Run;
```

/*A Sample of a SAS Proc*/
```
Proc print data=out;
run;
```

SAS handles alphanumeric and numeric data. Character strings are surrounded by single quotes such as 'Ira Shapiro'.
SAS dates are processed as numbers and printed with a date Format statement.

```
/*Temporary and 'Permanent' files defined in autoexec*/
/*An actual autoexec is much more complicated*/
libname temp    '/dss/ira temp files';
libname is      '/dss/ira perm files';
```

The code examples will be using a Shoes file found in the SASHELP library and coded as SASHELP.SHOES.
Part of SASHELP.SHOES Input File:

Region	Product	Subsidiary	Stores	Sales	Inventory	Returns
Africa	Boot	Addis Ababa	12	$29,761	$191,821	$769
Africa	Men's Casual	Addis Ababa	4	$67,242	$118,036	$2,284
Africa	Men's Dress	Addis Ababa	7	$76,793	$136,273	$2,433
Africa	Sandal	Addis Ababa	10	$62,819	$204,284	$1,861

Africa	Slipper	Addis Ababa	14	$68,641	$279,795	$1,771

Figure 1: Partial Sashelp.shoes contents

Ouput Delivery System is used to provide reports in a manner that can be provided to customers. This example provides the 'Presentation.xls' spreadsheet on a server.

```
/*ODS – Output Delivery System – Proc Print printed above*/
ods html body="/userid/Presentation03.xls";
proc print noobs data=sashelp.shoes;
run;
ods html close;
```

Mathematical operators such as < (less than), > (greater than) and = (equal to) are used in program logic to limit or allow rows of data to be processed. The operators can be combined such as >= (equal to or greater than).

```
options obs=5;/*Mathematical Operators and examples*/
data temp.issug_presentation;set SASHELP.SHOES;
   * Operators are < > = and can be combined;
   if stores =<4 then delete; /*Data/Row Exclusion*/
   if returns > 1000;        /*Data/Row Inclusion*/
Run;
```

Region	Product	Subsidiary	Stores	Sales	Inventory	Returns
Africa	Men's Dress	Addis Ababa	7	$76,793	$136,273	$2,433
Africa	Sandal	Addis Ababa	10	$62,819	$204,284	$1,861

Africa	Slipper	Addis Ababa	14	$68,641	$279,795	$1,771

Figure 2: Partial output of the use of mathematical operators

A null numeric field means that the field is blank. SAS assisgn a period (.) to this field which a user can test. A null character field is also a blank or blanks depending on the size of the field. A user can test a character field for blank(s).

For numeric fields, if a user detects a null field, the user can reset that field to any numeric value that would make logical sense for processing. Simarly, a user can change a null character field to any character string surrounded by single quotes.

```
/*Missing Data  - Null numeric and null character fields*/
/*  Input File for work.Missing_data
iId       First     Last        DOB         Money
 1        John                  1/2/1946     123.45
 2        Jane      Smith       7/12/1944   1234.89
          Tom       Jones       3/6/1950    12345.69*/

data work.Book03;set work.Missing_data;
        if ID=. then delete;/*Missing numeric data is represented by . (period)*/
        if LAST=' ' then delete;/*Missing character data is represented by ' ' (space or spaces)*/
Run;
proc print noobs data=work.Book03;run;
```

Id	First	Last	DOB	Money
2	Jane	Smith	12JUL1944	1234.89

Figure 3: Resultant output file when null ID or null LAST fields have been processed

Users can create new character (or numeric fields) as shown. For new character fields, a Format statement is strongly advised giving the maximum length of the newly defined field which, in this case, is 15 characters. The '$' in the format tells the program that the new field is a character field.

```
/*Setting new variables*/
data temp.issug_presentation;set SASHELP.SHOES;
        format Product_Other $15.;
        if Product='Boot' then Product_Other='Tall Boot';
Run;
```

Region	Product	Subsidiary	Stores	Sales	Inventory	Returns	Product_Other
Africa	Boot	Addis Ababa	12	$29,761	$191,821	$769	Tall Boot
Africa	Men's Casual	Addis Ababa	4	$67,242	$118,036	$2,284	
Africa	Men's Dress	Addis Ababa	7	$76,793	$136,273	$2,433	
Africa	Sandal	Addis Ababa	10	$62,819	$204,284	$1,861	
Africa	Slipper	Addis Ababa	14	$68,641	$279,795	$1,771	

Figure 4: Creating a new character field

If/then/else logic allows the user to first (**IF**) see if a field contains a given value with the given mathematical logic. If the field is true (meets that condition) the **THEN** condition is executed. If the field is not true (does not meets that condition) the ELSE condition is executed. Note that the ELSE portion of the code does not have exist. The ELSE portion is optional.

```
/* If/Then/Else  logic*/
data temp.issug_presentation;set SASHELP.SHOES;
        if Product='Sandal' then Sandal_Found=1;  else Sandal_Found=0;
        Run;
```

Region	Product	Subsidiary	Stores	Sales	Inventory	Returns	Sandal_Found
Africa	Boot	Addis Ababa	12	$29,761	$191,821	$769	0
Africa	Men's Casual	Addis Ababa	4	$67,242	$118,036	$2,284	0
Africa	Men's Dress	Addis Ababa	7	$76,793	$136,273	$2,433	0
Africa	Sandal	Addis Ababa	10	$62,819	$204,284	$1,861	1
Africa	Slipper	Addis Ababa	14	$68,641	$279,795	$1,771	0

Figure 5: Partial output of If/then/else logic

The IN logic allows for testing of multiple true conditions for a given field. Not shown- If the field is numeric, the user can test for a series of numbers i.e. if area_code in (201,202,203) then Area_code_found=1; else Aread_code_found=0;

```
/*Use of IN for qualification*/
data temp.issug_presentation;set SASHELP.SHOES;
        if Product in ('Boot','Sandal') then Boot_Sandal='Yes'; else Boot_Sandal='No ';
Run;
```

Region	Product	Subsidiary	Stores	Sales	Inventory	Returns	Boot_Sandal
Africa	Boot	Addis Ababa	12	$29,761	$191,821	$769	Yes
Africa	Men's Casual	Addis Ababa	4	$67,242	$118,036	$2,284	No
Africa	Men's Dress	Addis Ababa	7	$76,793	$136,273	$2,433	No
Africa	Sandal	Addis Ababa	10	$62,819	$204,284	$1,861	Yes

Africa	Slipper	Addis Ababa	14	$68,641	$279,795	$1,771	No

Figure 6: Partial output of IN test

The Delete statement is used to delete row(s) of data.

```
/*Excluding rows of data – Delete*/
data temp.issug_presentation;set SASHELP.SHOES;
        if Product in ('Boot','Sandal') then Delete;
Run;
```

Region	Product	Subsidiary	Stores	Sales	Inventory	Returns
Africa	Men's Casual	Addis Ababa	4	$67,242	$118,036	$2,284
Africa	Men's Dress	Addis Ababa	7	$76,793	$136,273	$2,433
Africa	Slipper	Addis Ababa	14	$68,641	$279,795	$1,771
Africa	Sport Shoe	Addis Ababa	4	$1,690	$16,634	$79
Africa	Women's Casual	Addis Ababa	2	$51,541	$98,641	$940

Figure 7: Partial sample output of use of the delete to omit rows of data

The **unqualified IF** statement only includes row(s) that meet the IF criteria.

```
/*Including rows of data –Unqualified IF*/
data temp.issug_presentation;set SASHELP.SHOES;
        if Product in ('Boot','Sandal');
Run;
```

Region	Product	Subsidiary	Stores	Sales	Inventory	Returns
Africa	Boot	Addis Ababa	12	$29,761	$191,821	$769
Africa	Sandal	Addis Ababa	10	$62,819	$204,284	$1,861
Africa	Boot	Algiers	21	$21,297	$73,737	$710
Africa	Sandal	Algiers	25	$29,198	$84,447	$1,530
Africa	Boot	Cairo	20	$4,846	$18,965	$229

Figure 8: Partial output sample of the unqualified if statement to include rows of data

Character input fields can be put together (concatenate) fields to create a new field. In this case two concatenated fields are created from concatenating region to subsidiary. In the first case, the new field Region_subsidiary is created from region and subsidiary with no intervening space. In the second case, the new field Region_subsidiary_1 is created from region and subsidiary with an intervening single space.

```
/*Simple concatenation of character fields*/
proc sort data=SASHELP.SHOES out=temp.issug_presentation_sort;by Region;run;
data temp.issug_presentation;set temp.issug_presentation_sort;by region;
        format Region_Subsidiary Region_Subsidiary_1  $50.;
        if last.Region;
```

```
        Region_Subsidiary= trim(region) || trim(Subsidiary);/*Trim function removes spaces from variable*/
        Region_Subsidiary_1= trim(region) || ' ' ||trim(Subsidiary);
        keep region subsidiary r_s Region_Subsidiary Region_Subsidiary_1;
Run;
```

Region	Subsidiary	Region_Subsidiary	Region_Subsidiary_1
Africa	Nairobi	AfricaNairobi	Africa Nairobi
Asia	Tokyo	AsiaTokyo	Asia Tokyo
Canada	Vancouver	CanadaVancouver	Canada Vancouver
Central America/Caribbean	San Juan	Central America/CaribbeanSan Juan	Central America/Caribbean San Juan
Eastern Europe	Warsaw	Eastern EuropeWarsaw	Eastern Europe Warsaw

Figure 9: Partial sample of output showing concatenation of two character fields with and without an intervening space

To minimize the size of output data files especially when they can be very large, a user can code a **Keep** which contains the name(s) of field(s) the user wishes to have on the output data file. In this exampe the issug_presentation datset only contains the region, subsidiary and product field.

```
/*Keep (Keeps one or more variables in the output file)*/
data temp.issug_presentation;set SASHELP.SHOES;
        format Region_Subsidiary $125.;
        Region_Subsidiary_product= trim(product) || ' ' || trim(region) || ' ' || trim(Subsidiary);
        keep  Region_Subsidiary_product;
Run;
```

Region_Subsidiary_product

Boot Africa Addis Ababa
Men's Casual Africa Addis Ababa
Men's Dress Africa Addis Ababa
Sandal Africa Addis Ababa
Slipper Africa Addis Ababa

Figure 10: Sample output of the use of a keep statement

The **Drop** statement is the reverse of a keep in which the coded fields in the Drop are not placed on the output data file. By using the drop, the user can also minimize the size of the output file. In the example, region and subsidiary are dropped from being placed on the output data file.

```
/*Drop (Reverse of Keep- Drops one or more variables from the output file)*/
data temp.issug_presentation;set SASHELP.SHOES;
        format Region_Subsidiary $25.;
        Region_Subsidiary= trim(region) || trim(Subsidiary);
        Drop Region Subsidiary;
Run;
```

Product	Stores	Sales	Inventory	Returns	Region_Subsidiary
Boot	12	$29,761	$191,821	$769	AfricaAddis Ababa
Men's Casual	4	$67,242	$118,036	$2,284	AfricaAddis Ababa
Men's Dress	7	$76,793	$136,273	$2,433	AfricaAddis Ababa

Sandal	10	$62,819	$204,284	$1,861	AfricaAddis Ababa
Slipper	14	$68,641	$279,795	$1,771	AfricaAddis Ababa

Figure 11 Sample output of the use of the drop statement

The Retain statement is quite useful if it is understood. A Retain field(s) maintains its value whether it is null or a value through multiple observations (or rows) of data until it is reset by the user. In this case the new character Sandal_Found is initialized to spaces. If (and only if) the product field is 'Sandal' does Sandal_Found becomes the character value *Yes*. When the next row of data for the product is *Slipper*, Sandal_Found remains *Yes* because it hasn't bee changed!

```
/*Retain (The vampire that lives forever)*/
data temp.issug_presentation;set SASHELP.SHOES;
        Format Sandal_Found $3.;
        Sandal_Found='  ';
        if product='Sandal' then Sandal_Found='Yes';
        retain Sandal_Found;
        keep product Sandal_Found;
Run;
```

Product	**Sandal_Found**
Boot	
Men's Casual	
Men's Dress	
Sandal	Yes

Slipper	Yes

Figure 12: Sample output use of the retain statement

Self-defined variables similarly retained their value unless changed. In the case below, the new numeric variable count has an implied initial value of 0 (zero). As Figure 13 shows, for the first row of data count is 1, for the second count is now 1+1 or 2. Notice that 'count+1' has no equal sign in the statement. If you wish to increment a self-defined variable, do NOT put an = in the code.

```
/* Self-defined retained variables */
data temp.issug_presentation;set SASHELP.SHOES;
        count+1;/*NOTE: NO use of = sign for a self-defined variable*/
        keep count product Region;
Run;
```

Region	**Product**	**count**
Africa	Boot	1
Africa	Men's Casual	2
Africa	Men's Dress	3
Africa	Sandal	4
Africa	Slipper	5

Figure 13: Same output of a self-defined retained variable

AND logic is used to check two (or more) variables to see if they are true. In this case, the code is an unqualified If statement which will allow the output file to have those rows of data which have fields Region being Africa and Stores greater than 4.

```
/*AND logic*/
data temp.issug_presentation;set SASHELP.SHOES;
        if Region='Africa' and Stores > 4;
Run;
```

Region	Product	Subsidiary	Stores	Sales	Inventory	Returns
Africa	Boot	Addis Ababa	12	$29,761	$191,821	$769
Africa	Men's Dress	Addis Ababa	7	$76,793	$136,273	$2,433
Africa	Sandal	Addis Ababa	10	$62,819	$204,284	$1,861
Africa	Slipper	Addis Ababa	14	$68,641	$279,795	$1,771
Africa	Women's Dress	Addis Ababa	12	$108,942	$311,017	$3,233

Figure 14: Sample and logic output

OR logic is used to check two (or more) variables to see if any of the variables being tested are true. In this example the output file will contain those rows where Product is *Sandal* OR Returns are less than 1000.

```
/*Or logic*/
data temp.issug_presentation;set SASHELP.SHOES;
        if Product='Sandal' or Returns < 1000;
Run;
```

Region	Product	Subsidiary	Stores	Sales	Inventory	Returns

Africa	Boot	Addis Ababa	12	$29,761	$191,821	$769
Africa	Sandal	Addis Ababa	10	$62,819	$204,284	$1,861
Africa	Sport Shoe	Addis Ababa	4	$1,690	$16,634	$79
Africa	Women's Casual	Addis Ababa	2	$51,541	$98,641	$940
Africa	Boot	Algiers	21	$21,297	$73,737	$710

Figure 15: Sample output using or logic

The simple unqualified **DO** loop is used to set more than one variable's value. In this example, if Product is *Boot* and Returns < 100 then the varialble Boot_Yes_No is set to *Yes* and variable Returns_lt_1000 is also set to *Yes*.

```
/*Simple unqualified do loop*/
data temp.issug_presentation;set SASHELP.SHOES; format Boot_Yes_No Returns_lt_1000 $3.;
    if Product='Boot' and Returns < 1000
                then    do;
                        Boot_Yes_No='Yes';
                        Returns_lt_1000='Yes';
                        end;
        keep Product Returns Boot_Yes_No Returns_lt_1000;
Run;
```

Product	Returns	Boot_Yes_No	Returns_lt_1000
Boot	$769	Yes	Yes
Men's Casual	$2,284		
Men's Dress	$2,433		
Sandal	$1,861		

Slipper	$1,771		

Figure 16: Sample unqualified do loop output

The user can create Multiple Output Files in a single job step depending on input data. In this example, if the Product is *Boot* the code creates a temp.Boot file. If the Product is *Sandal*, the code creates a temp.Sandal file. If the Product is neither *Boot* nor *Sandal*, the code creates a temp.presentation file.

```
/*Multiple Output files*/
data temp.Boot temp.Sandal temp.presentation;
        set SASHELP.SHOES;
        if Product='Boot'   then output temp.Boot;else
        if Product='Sandal' then output temp.sandal;else
                                 output temp.presentation;
        keep Product Returns;
Run;
```

presentation

Product	Returns
Men's Casual	$2,284
Men's Dress	$2,433
Slipper	$1,771

boot

Product	Returns
Boot	$769

sandal

Product	Returns
Sandal	$1,861

Figure 17: Sample multiple output files

Before you can use First. And Last. Code, the Set file will have to be sorted. In this example, the SASHELP.Shoes file is sorted by Subsidiary within region creating the temp.Shoes file by Proc Sort. The objective of the Data Step is to create summary totals of stores, inventory and returns by subsidiary within region. To accomplish this objective, the set statement for the input file has a by clause of region subsidiary because that is

what temp.shoes is sorted by. 'First.subsidiary' if the first occurrence of the value of subsidiary within a given region. 'Last.subsidiary' if the last occurrence of the value of subsidiary within a given region. The first occurrence of the value of subsidiary causes the code to initialize new self-defined variables called total_stores, total_inventory and total_returns all to a value of 0 (zero). For any row of data for any given value of subsidiary, the self-defined variables get totaled by their respective row variable. When the last occurance of subsidiary is reached, the last.subsidiary puts out via an output statement the summarized inventory, returns and stored value as shown.

```
/*Set BY clause with First. and Last.*/
proc sort data=SASHELP.SHOES out=temp.SHOES_sort;by region subsidiary;run;

data temp.presentation;set temp.SHOES_sort;by region subsidiary;
format total_inventory comma11.0 total_returns dollar11.0;
        if first.subsidiary then do; total_stores=0;total_inventory=0;total_returns=0;end;
            total_stores+stores;
            total_inventory+inventory;
            total_returns+returns;
         if last.subsidiary then output temp.presentation;
         keep region subsidiary  total_stores total_inventory total_returns;
    Run;
```

• Region	• Subsidiary	• total_inventory	• total_returns	• total_stores
• Africa	• Addis Ababa	• 1,356,501	• $13,370	• 65
• Africa	• Algiers	• 1,212,116	• $12,763	• 101
• Africa	• Cairo	• 2,245,536	• $22,477	• 88
• Africa	• Johannesburg	• 375,534	• $3,962	• 51
• Africa	• Khartoum	• 588,019	• $7,051	• 71

Figure 18: Sample First./Last. User summarized report

SAS has user debugging tools to help the user hone in on possible data concerns. These tools, particularly the Put, have to be used extremely carefully otherwise the user could easily fill up SASLOG and possibly terminating the SAS session. The _N_ variable is a SAS reserved variable. It contains the current input record number. In the example's code by checking the value of _N_ to < 6, it guarantees that the the ensuing Put statement will only put out five line of the variables and constants being requested.

```
/*Debugging use of _N_ and put*/
data temp.issug_presentation;set SASHELP.SHOES;
        if _n_ < 6 then put _N_ 'REGION=' Region 'Subsidiary=' subsidiary 'Product=' product;
Run;
```

Results in SASLOG:

```
1 REGION=Africa Subsidiary=Addis Ababa Product=Boot
2 REGION=Africa Subsidiary=Addis Ababa Product=Men's Casual
3 REGION=Africa Subsidiary=Addis Ababa Product=Men's Dress
4 REGION=Africa Subsidiary=Addis Ababa Product=Sandal
5 REGION=Africa Subsidiary=Addis Ababa Product=Slipper
```

Figure 19: Sample use of _N_ variable with a Put statement

The Format allows the user to report the data that is understandable to the customer. There are many formats available in SAS for numbers and especially for dates (and times). The Dollar format puts a leading dollar sign and possible commas in the reporting of the data. The two date formats put out the dates with normal month numbers/days/year in either 2 digit or 4 digit form. A general warning – If you use ODS to put out your report in Excel (xls,xlsx), Excel formats the data, particularly numeric, the way it wants to not necessarily the way the user wants to see it.

```
/*Intro to some Formats*/
data temp.issug_presentation;set SASHELP.SHOES;
        format returns dollar12.2 dt  mmddyy10.      dt2 mmddyy8.;
          if _N_=1 then dt='01Jan2011'd;else dt+1;
          dt2=dt+31;
          returns=returns+0.01;
          keep returns dt dt2;
Run;
```

*_Watch out for Excel!!_;

Returns	Dt	dt2
$769.01	1/1/2011	02/01/11

$2,284.01	1/2/2011	02/02/11
$2,433.01	1/3/2011	02/03/11
$1,861.01	1/4/2011	02/04/11
$1,771.01	1/5/2011	02/05/11

Figure 20: Sample dollar and date reporting formats

When running code, the user MUST review the SASLOG file to see errors, warnings and comments. The errors must be corrected. The warnings should be looked at with a determination by the user if remediation needs to take place. Comments also can be useful because they contain record (row) counts of input and output records for each job step. This information can be somewhat useful in determining potential logic errors.

```
/*SASLOG Error*/
data temp.issug_presentation;set SASHELP.SHOES;
        if Product='Boot' and Returns < 1000
                then do;Boot_Yes_No='Yes';Returns_lt_1000='Yes';
        keep Product Returns Boot_Yes_No Returns_lt_1000;
Run;
```

Partial SASLOG

```
326  data temp.issug_presentation;set SASHELP.SHOES;
327   if Product='Boot' and Returns < 1000
328    then do;Boot_Yes_No='Yes';Returns_lt_1000='Yes';
329   keep Product Returns Boot_Yes_No Returns_lt_1000;
330  Run;
          -
          117
ERROR 117-185: There was 1 unclosed DO block.
```

Figure 21: Sample SASLOG errors

Advanced SAS Techniques

It's almost a guarantee that you will be using Pro SQL whether it be strictly in SAS mode or extracting data from a database. This paper will not address using Proc SQL to extract database data (DB2, Oracle, SQL Server, Teradata). There are whole books written on that subject and will be dependent on your working environment.

In the following example, all bold items are the skeletal Proc SQL required words. The Create Table creates an output dataset with fields listed after the Select from the table designated with the From. Order by is optional and orders the output file by stated field(s).

```
/*Introduction to Proc SQL*/
proc sql;
    create table temp.sql01 as
    select    Product
            , Region
            , Subsidiary
            , Stores
            , Inventory
            , Sales
            , Returns
    FROM sashelp.shoes a
    ORDER BY product,region,subsidiary;
    quit;
```

Product	Region	Subsidiary	Stores	Inventory	Sales	Returns
Boot	Africa	Addis Ababa	12	$191,821	$29,761	$769
Boot	Africa	Algiers	21	$73,737	$21,297	$710
Boot	Africa	Cairo	20	$18,965	$4,846	$229
Boot	Africa	Johannesburg	14	$33,011	$8,365	$483
Boot	Africa	Khartoum	24	$105,370	$19,282	$700

Figure 22: Basic Proc SQL sample with output

Proc SQL has a lot of power including the capability to do calculations. The following example shows the power of creating a sum within Proc SQL in combination with Group By. The sums various fields including a count and formatting the summed field grouping by each different Product. The Order By, orders by the Product field when producing the temp.sql02 output file.

```
/*PROC SQL Group By and Order By*/
proc sql;
    create table temp.sql02 as
    select          Product
                  , sum(Stores) as stores_sum format comma11.
                  , sum(Inventory) as inventory_sum format dollar12.
                  , sum(Sales) as sales_sum format dollar12.
                  , sum(Returns) as returns_sum format dollar12.
                  , sum(1) as Count format comma11.
    FROM sashelp.shoes a  GROUP by product
    ORDER BY product;quit;
```

Product	stores_sum	inventory_sum	sales_sum	returns_sum	Count
Boot	864	$9,724,671	$2,350,543	$98,622	52
Men's Casual	399	$17,085,253	$7,933,707	$311,035	45
Men's Dress	480	$14,507,340	$5,507,243	$164,099	50
Sandal	564	$3,232,275	$868,436	$38,170	49
Slipper	794	$22,231,380	$6,175,834	$209,940	52
Sport Shoe	616	$3,322,702	$651,467	$25,179	51
Women's Casual	270	$9,696,651	$4,137,861	$131,394	45
Women's Dress	614	$19,304,779	$6,226,475	$193,653	51

Figure 23: Example of summing and grouping by in Proc SQL.

The Proc SQL does a Full Join of SASHELP.shoes with temp.sql02 only by product. Proc SQL can do many types of calculations. This example takes the temp.sql02 dataset created in the prior example and calculates a percent of stores.

```
/*PROC SQL with Computed Value*/
proc sql;
    create table temp.sql03
    as  select      a.Product
                  , Region
                  , Subsidiary
                  , Stores
                  , stores_sum
                  , stores/stores_sum as stores_pc format percent6.2
                  , Inventory
                  , inventory_sum
                  , Sales
                  , sales_sum
                  , Returns
                  , returns_sum

    FROM      temp.sql02 a
    full join sashelp.shoes b
          on a.product=b.product
    ORDER BY product;quit;
```

Product	Region	Subsidiary	Stores	stores_sum	stores_pc	Inventory	inventory_sum	Sales	sales_sum	Returns	returns_sum
Boot	Canada	Vancouver	31	864	3.60%	$882,080	$9,724,671	$286,497	$2,350,543	$9,160	$98,622
Boot	Middle East	Al-Khobar	10	864	1.20%	$44,658	$9,724,671	$15,062	$2,350,543	$765	$98,622
Boot	Eastern Europe	Warsaw	26	864	3.00%	$363,358	$9,724,671	$78,992	$2,350,543	$3,246	$98,622
Boot	Africa	Johannesburg	14	864	1.60%	$33,011	$9,724,671	$8,365	$2,350,543	$483	$98,622
Boot	Asia	Bangkok	1	864	0.12%	$9,576	$9,724,671	$1,996	$2,350,543	$80	$98,622

Figure 24: Full join with summed dataset calculating percent of stores

Proc SQL supports a where including clause. The example only puts *Boot* and *Sandal* Products into temp.sql04.

```
/*Proc SQL with WHERE clause*/
proc sql;
    create table temp.sql04 as
    select      Product
               , Region
               , Subsidiary
               , Stores
               , Inventory
               , Sales
               , Returns
FROM sashelp_shoes a
where product in ('Boot','Sandal')
ORDER BY region,subsidiary;quit;
```

• Product	• Region	• Subsidiary	• Stores	• Inventory	• Sales	• Returns
• Sandal	• Africa	• Addis Ababa	• 10	• $204,284	• $62,819	• $1,861
• Boot	• Africa	• Addis Ababa	• 12	• $191,821	• $29,761	• $769
• Boot	• Africa	• Algiers	• 21	• $73,737	• $21,297	• $710
• Sandal	• Africa	• Algiers	• 25	• $84,447	• $29,198	• $1,530
• Boot	• Africa	• Cairo	• 20	• $18,965	• $4,846	• $229

Figure 25: Proc SQL with where

Proc Sql with a Left Join example. The Join of temp.sql04 (from Figure 25) with temp.sql02 (from Figure 23) on Product. The file Sql012 has Product and summary values for stores, inventory, sales and returns. The file Sql04 has Product, stores_pc, and detail values for Product of *Boot* and *Sandal*. The Left Join joins the two files in the From clause. In the from clause, the coding show an 'a' after sql02 and aa 'lj' after sql04. These are used as shorthand identifiers for both the 'on' portion of the From clause and in the fields enumerated in the Select.

```
/*Proc SQL Left Join*/
proc sql; create table temp.sql05 as
        select    a.Product
                , lj.Region
                , lj.Subsidiary
                , lj.Stores
                , a.stores_sum
                , lj.stores
                ,a.stores_sum as stores_pc format percent6.2
                , lj.Inventory
                , a.inventory_sum
                , lj.Sales
                , a.sales_sum
                , lj.Returns
                , a.returns_sum
    FROM     temp.sql02 a left join temp.sql04 lj
        on a.product=lj.product
    ORDER BY region;quit;
```

Product	Region	Subsidiary	Stores	stores_sum	stores_pc	Inventory	inventory_sum	Sales	sales_sum	Returns	returns_sum
Slipper			.	794	.	.	$22,231,380	.	$6,175,834	.	$209,940
Men's Dress			.	480	.	.	$14,507,340	.	$5,507,243	.	$164,099
Sport Shoe			.	616	.	.	$3,322,702	.	$651,467	.	$25,179
Men's Casual			.	399	.	.	$17,085,253	.	$7,933,707	.	$311,035
Women's Casual			.	270	.	.	$9,696,651	.	$4,137,861	.	$131,394
Women's Dress			.	614	.	.	$19,304,779	.	$6,226,475	.	$193,653

Boot	Africa	Algiers	21	864	2.40%	$73,737	$9,724,671	$21,297	$2,350,543	$710	$98,622
Boot	Africa	Khartoum	24	864	2.80%	$105,370	$9,724,671	$19,282	$2,350,543	$700	$98,622
Sandal	Africa	Johannesburg	13	564	2.30%	$63,003	$3,232,275	$17,337	$868,436	$809	$38,170

Figure 26: Sample output of Proc SQL Left Join example

This summary of Proc SQL gives the user an outline of many, but not all Proc SQL key words. The Case statement is introduced here. You might consider the Case statement as an IF/THEN/ELSE alternative. It allows a usere to check for various true field values and setting a new variable with different values. In this code, the receiving variable is Checkit. If Product is *Boot*, then Checkit is set to *B*. If Product is Sandal, then Checkit is set to *S*. If neither is true, Checkit is set to a space. Also note that one useful feature of Proc SQL is that within the ON clause the fields being checked for equality can have different names. Note though within the ON clause, the fields being checked for equality both have to be character or both have to be numeric.

```
/* Summary of SQL key words*/
proc sql;
 create table temp.sql07 as
select      a.Product
      , lj.stores
                  , sum(1) as Count
                  , lj.stores/a.stores_sum as stores_pc format percent6.2
                  ,case when Product eq 'Boot'   then 'B'
                        when Product eq 'Sandal' then 'S' else ' '
                  end as CheckIt
FROM      temp.sql02 a
left join temp.sql04 lj
          on a.product=lj.product /*ON field names can be different*/
where a.product in ('Boot','Sandal')
group by a.product
ORDER BY a.product;
quit;
```

In preparation for a Merge, all files must be sorted in the same order on all fields to be merged. In addition a job step creates a new variable before doing the Merge.

```
/*Preparation of files for a Merge*/
proc sort data=TEMP.ISSUG_SQL04;by  product region subsidiary;run;
proc sort data=sashelp.shoe out=ISSUG_Shoess;by  product region subsidiary;run;
data TEMP.ISSUG_SQL04_;set TEMP.ISSUG_SQL04;format Boot_sandal $7.;
        if product='Boot' then Boot_sandal=  'BOOT   '; else
        if product='Sandal' then Boot_sandal='SANDAL '; else
                                 Boot_sandal='Neither'; run;
```

Merge provides the capability to combine the data from two or more files together. As you will see, Merge within a job step can have controls as to exactly how to combine the files. This simple Merge combines two files together by product, region and subsidiary.

```
/*Simple Unqualified Merge*/
data ISSUG_merge01;
        Merge ISSUG_Shoes(In=A) ISSUG_sql04_(IN=B);by product region subsidiary;
run;
```

```
NOTE: There were 395 observations read from the data set sashelp.shoes.
NOTE: There were 101 observations read from the data set TEMP.ISSUG_SQL04_.
NOTE: The data set TEMP.ISSUG_MERGE01 has 395 observations and 8 variables.
```

Region	Product	Subsidiary	Stores	Sales	Inventory	Returns	Boot_sandal
Africa	Boot	Addis Ababa	12	$29,761	$191,821	$769	BOOT
Africa	Men's Casual	Addis Ababa	4	$67,242	$118,036	$2,284	
Africa	Sandal	Addis Ababa	10	$62,819	$204,284	$1,861	SANDAL

Figure 27: Sample results of a simple merge

The logical operators of a Merge are created in an In clause as shown. In this example the In variable A is set to 1 if the row of data in ISSUG_Shoes exists for a combination of the By fields. In this example In variable A is set to 0 if the row of data in ISSUG_Shoes does not exist for a combination of the By fields. The In variable B is set to 1 if the row of data in ISSUG_sql04_ exists for a combination of the By fields.The In variable B is set to 0 if the row of data in ISSUG_sql04_ does not exist for a combination of the By fields.

In ths examp of 'If A and not B;' means if product region subsidiary values on ISSUG_Shoes exist and product region subsidiary values on ISSUG_sql04 do not exist, then output this record/row of data.

```
/*MERGE Negative Logic*/
proc sort data=TEMP.ISSUG_SQL04_;by  product region subsidiary;run;
proc sort data=SASHELP.SHOES;by  product region subsidiary;run;
data ISSUG_merge02;
        Merge ISSUG_Shoes(In=A) ISSUG_sql04_(IN=B); by  product region subsidiary;
        If A and not B;
run;
```

NOTE: There were 395 observations read from the data set Temp.ISSUG_SHOES.
NOTE: There were 101 observations read from the data set TEMP.ISSUG_SQL04_.
NOTE: The data set TEMP.ISSUG_MERGE02 has 294 observations and 8 variables.

• Region	• Product	• Subsidiary	• Stores	• Sales	• Inventory	• Returns	• Boot_sandal
• Africa	• Men's Casual	• Addis Ababa	• 4	• $67,242	• $118,036	• $2,284	•
• Africa	• Men's Casual	• Algiers	• 4	• $63,206	• $100,982	• $2,221	•

•

Figure 28: Sample output of Negative Merge

This Merge tests if the By fields exist on B (TEMP.SQL04_) but DON'T exist on A (Temp.ISSUG_SHOES). The result is that no data meets that criteria since ALL By combinations exist on both files.

```
data temp.merge03;
     Merge Temp.ISSUG_SHOES (In=A) temp.sql04_(IN=B); by  product region subsidiary;
   If B and not A;/*Same as If B; */
run;
```

NOTE: There were 395 observations read from the data set Temp.ISSUG_SHOES.
NOTE: There were 101 observations read from the data set TEMP.SQL04_.
NOTE: The data set TEMP.MERGE03 has 0 observations and 8 variables.
Temp.sql04_(IN=B) Does not have any unique keys which do not exist On Temp.ISSUG_SHOES (In=A)

The job step deletes 16 rows of data in preparation for the next Merge sample.

```
Data temp.Shoes_; set sashelp.Shoes;
     if Region='Africa' and Product in ('Boot','Sandal') then delete;
run;
```

NOTE: There were 395 observations read from the data set sashelp.shoes.
NOTE: The data set temp.shoes_ has 379 observations and 7 variables.
     16 rows were deleted by code

Since Temp.Shoes_ has 16 rows of missing combinations of the By fields, 'If B and not A;' now places 16 records/rows of data into temp.merge4.

```
data temp.merge04;
     Merge temp.Shoes_(In=A) temp.sql04_(IN=B); by  product region subsidiary;
     If B and not A;/*Same as If B; */
run;
```

NOTE: There were 379 observations read from the data set temp.shoes_.
NOTE: There were 101 observations read from the data set TEMP.SQL04_.
NOTE: The data set TEMP.MERGE04 has 16 observations and 8 variables.

Merge Review:
(In=A) (In=B) A and B are logical operators.
If A means that if keyed record for the file associated with A, then process it. If B means that if keyed record for the file associated with B, then process it.

The Merge is a job step like any other job step for those merged records which exist at that point.
Proc SQL is different from Merge in that the fields is stated on the ON can have different names. Also the Proc SQL file(s) do not have to be sorted. Proc SQL does handle more than one to more than one combinations but be careful, VERY careful. Merge only handles one to one and one to many.

Selected Common Procs

A Proc is a SAS procedure with options which perform a process on the input data file. Procs are well-tested and save the analysts time to develop the equivalent code in data steps.

As far a proc options are concerned, the options can be the same though certainly not always.

There will be maybe a dozen procs that you'll become expert at by using them over and over again.

The simplest way to learn about procs which you believe apply to a given situation but you're unfamiliar with is to Google SAS PROC *procname.* That will get you to SAS' excellent documentation. You could also Google SAS PROC *procname* proceedings to white paper proceedings usually in pdf format.

In preparation for an explaination of a number of Proc Sort option, a temp shoes file is created.

/*Create a temporary Shoes dataset*/
```
Data temp.shoes;set sashelp.shoes;
run;
```

The first Proc Sort example is a non-destructive sort by (ascending assumed) product within (ascending assumed) region. Region is the primary key and Product the secondary key. A Proc Sort has a single primary key and may contain multiple secondary keys or no secondary key.

Example 02 is the same as Example 01 but keeping two fields for out=shoes_sorted.

Example 03 is non-destructive as far as shoes are concerned however any time region product are duplicated on the data= side, the duplicates are eliminated on the out= side via the NODUPKEY option

Example 04 is a non-destructive sort as far as shoes are concerned however any time region product are duplicated on the data= side, the duplicates are eliminated on the out= side via the NODUPKEY and placed in the dupout= file.

Example 05 **>>IS DESTRUCTIVE<<** as far as the data= file is concerned the duplicates are eliminated on data= file.

Example 06 is a non-destructive sort by (secondary sort key) Descending product by (primary sort key) ascending region.

```
          /*Proc SORT samples*/
(01)      proc sort data=temp.shoes out=temp.shoes_sorted;by region product;run;

(02)      proc sort data=temp.shoes (keep=region product) out=temp.shoes_sorted;by region product;run;

(03)      proc sort data=temp.shoes out=temp.shoes_sorted NODUPKEY;by region product;run;

(04)      proc sort data=temp.shoes out=temp.shoes_sorted NODUPKEY dupout=temp.shoes_duplicate;by region product;run;

(05)      proc sort data=temp.shoes_ nodupkey ;by region product;run;

(06)      proc sort data=temp.shoes out=temp.shoes_sorted;by region Descending product;run;
```

Proc Contents is extremely useful in any number of ways as self-documentation documentation for another person's use of the file.

```
proc contents data=sashelp.class;run;
```

The following is an abbreviated report:

Data Set Name	TEMP.CLASS	**Observations**	19
Member Type	DATA	**Variables**	5

Alphabetic List of Variables and Attributes			
#	**Variable**	**Type**	**Len**
3	Age	Num	8
4	Height	Num	8
1	Name	Char	8
2	Sex	Char	1
5	Weight	Num	8

Figure 29: Sample abbreviated Proc Contents

Proc Freq provides an easy way to get a distribution (counts) of a field or fields in a file. Before running a proc Freq you may want run a Proc Contents and maybe a Proc print (obs=100) to get a feeling for the data you are dealing with. The example shows only one field in the Tables. You could code Tables age Height Sex and maybe even Weight.

proc freq data=sashelp.class;tables age ;**run**;

Age	**Frequency**	**Percent**	**Cumulative Frequency**	**Cumulative Percent**
11	2	10.53	2	10.53
12	5	26.32	7	36.84
13	3	15.79	10	52.63

14	4	21.05	14	73.68
15	4	21.05	18	94.74
16	1	5.26	19	100

Figure 30: Sample output of a simple Proc Freq

This example of a Proc Freq gives you less detail in the report and can be very useful.

proc freq data=sashelp.class;tables age /nocol norow nopercent ;**run**;

Age	Frequency	Cumulative Frequency
11	2	2
12	5	7
13	3	10
14	4	14
15	4	18
16	1	19

Figure 31: Proc Freq sample with simplified output

This example of Proc Freq creates an output file of the results which can be used in subsequent job steps.

proc freq data=sashelp.class;tables age /out=temp.Freq_results nocol norow nopercent;**run**;
proc contents data= temp.Freq_results;**run**;
proc print noobs data= temp.Freq_results;**run**;

Age	Frequency	Cumulative Frequency

11	2	2
12	5	7
13	3	10
14	4	14
15	4	18
16	1	19

Figure 32: Output using Out= within a Proc Freq

Partial **Proc Contents** of Freq_results

#	Variable	Type
1	Age	Num
2	COUNT	Num
3	PERCENT	Num

Figure 33: Partial Proc Contents of the Out= file within a Proc Freq

Partial **Proc Print** results

Age	COUNT	PERCENT
11	2	10.5263
12	5	26.3158

Figure 34: Partial Proc Print of the Out file within a Proc Freq

Proc Print is also frequently used reporting method. The following example shows a number of options that can be used to make the output more useful and readable. The Label= option changes the report column heading. Title can be used for the report title and you can code multiple title lines (Title, Title1-Title9).

proc print noobs data=temp.Freq_results Label; var Age Count;

```
label age='Age of Student' Count='Count';
title1 Distribution of Class Ages;
run;
```

Distribution of Class Ages

Age of Student	Count
11	2
12	5
13	3
14	4
15	4
16	1

Figure 35: Proc Print sample with Label and Title

Proc Summary is an excellent way to summarize numeric fields with the option of having 'by' breaks. 'Output out=' creates an output file which either be used in a Proc Print or in another job step. 'sum()=' sums up a given field either by the 'by' breaks or totals. The 'By' breaks can either be Class or By. If Class is coded the Class field(s) are internally sorted. Not that with extremely large files, SAS could give you an execution time error because SAS could run out of memory for the sort. Using By fields(s) instead of Class assumes that the Data= files is coming into the Proc Summary already sorted by the By field(s).

In this example of using Class, Proc Summary automatically generates two fields: _Type_ and _Frequency_. In this example, the first row out has a _TYPE_ value of 0 (zero) which contains grand totals for the file. _TYPE_=1 contains the Class break totals for Region. _Frequency_ is a count of records for the grand total and each Class break. Bothe of these automatically generated fields will come in handy for subsequent job step processing.

Note that if By is coded instead of Class, no grand totals are generated. If neither Class nor By is coded, the out= file consists of the grand totals of the field(s) with a _type_ and _frequency_ field.

```
Proc Summary data=sashelp.shoes;
        output out=temp.Shoes_summary
        sum(Inventory)= sum(Returns)= sum(Sales)= sum(Stores)=;
        class region; /*Can use by here instead of class BUT….*/
run;
```

Region	_TYPE_	_FREQ_	Inventory	Returns	Sales	Stores
	0	395	$99,105,051	$1,172,092	$33,851,566	4601
Africa	1	56	$7,101,073	$74,087	$2,342,588	532
Asia	1	14	$1,176,139	$10,895	$460,231	65
Canada	1	37	$13,110,709	$129,394	$4,255,712	442
Central America/Caribbean	1	32	$10,173,878	$126,898	$3,657,753	539
Eastern Europe	1	31	$7,952,471	$86,701	$2,394,940	379
Middle East	1	24	$14,208,749	$206,880	$5,631,779	397
Pacific	1	45	$7,971,291	$77,129	$2,296,794	356
South America	1	54	$5,986,094	$102,851	$2,434,783	632
United States	1	40	$16,582,397	$187,502	$5,503,986	617
Western Europe	1	62	$14,842,250	$169,755	$4,873,000	642

Figure 36: Proc Summary Out= file output

PROC MEANS will analyze all numeric variables in your input data set and produce an analyses. Five default statistical measures are calculated:

N - Number of observations with a non-missing value of the analysis variable

MEAN - Mean (Average) of the analysis variable's non-missing values
STD - Standard Deviation
MAX - Largest (Maximum) Value
MIN - Smallest (Minimum) Value

Proc Means data=sashelp.shoes;**run**;

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Stores	**Number of Stores**	395	11.6481013	8.8736315	1	41
Sales	**Total Sales**	395	85700.17	129107.23	325	1298717
Inventory	**Total Inventory**	395	250898.86	351514.63	374	2881005
Returns	**Total Returns**	395	2967.32	4611.74	10	57362

Figure 37: Partial Proc Means sample output

Proc Print is capable of summary information in a fairly set format that can be useful. The By field must be coming into the Proc Print already sorted.

```
Proc Print data=sashelp.shoes noobs ;
        sum Inventory Returns Sales Stores;
        by region; /*Must be sorted by the 'by' field(s)*/
        title1 Sum of Fields by Region;
run;
```

Partial report:

Sum of Fields by Region

Region=Africa

Product	Subsidiary	Stores	Sales	Inventory	Returns
Boot	Addis Ababa	12	$29,761	$191,821	$769

Region		**642**	**$4,873,000**	**$14,842,250**	**$169,755**
		4601	**$33,851,566**	**$99,105,051**	**$1,172,092**

Figure 38: Partial sample of a Proc Print with Sum and by

Proc Export can be useful to create files in a non-SAS format. As with other proocs, Proc Export has many options which the user can review before using. This code was run locally on a PC and not on a server creating an Excel spreadsheet.

```
/* Run Locally on PC, NOT on server */
proc export data=sashelp.shoes
    outfile="H:\Data\ISSAS Users Group\Shoes.xls"
    replace dbms=Excel;
run;
*NOTE: File "H:\Data\ISSAS Users Group\Shoes.xls" will be created if the export process succeeds.;
```

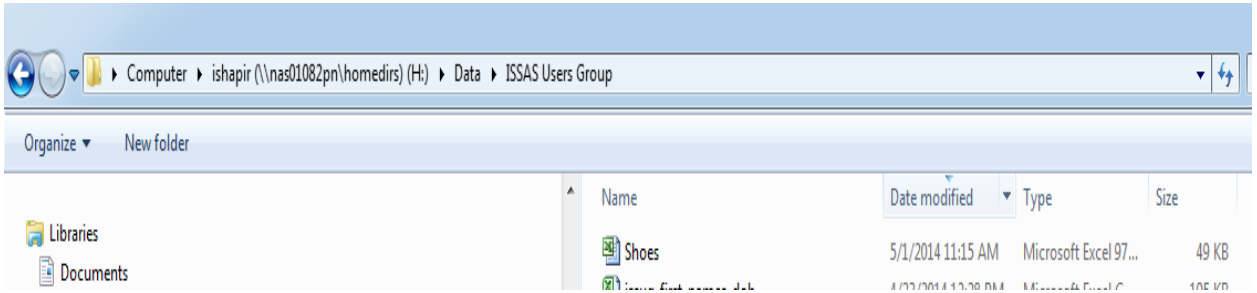


Figure 39: Sample Proc Export creating an Excel xls spreadsheet.

This Proc Export was run locally on a PC and not on a server creating an Excel csv file.

```
proc export data=temp.shoes /*Proc Export – CSV file*/
        outfile="H:\Data\ISSAS Users Group\Shoes.csv"
        replace ;
run;
*NOTE: 396 records were written to the file 'H:\Data\ISSAS Users Group\Shoes.csv'.;
```

Proc Import reads in a non-SAS file creating a SAS file. This example reads in the Excel file created in Figure 39. Like Proc Export, Proc Import has many options which can be useful.The Getnames options if coded as *yes* takes thecolumn headings from the first row of the Excel spreadsheet as field names in the Out SAS formatted file.

```
proc import REPLACE
   Datafile=  "H:\Data\ISSAS Users Group\Shoes"
          Out=temp.Shoes_xls_imported
          dbms=Excel;
          getnames=yes;
run;
*NOTE: TEMP.SHOES_XLS_IMPORTED data set was successfully created.;
```

The Proc Import Guessingrows=n option is only valid with csv files. Guessingrows=n indicate the number of rows the IMPORT procedure scans in the input file to determine the appropriate data type and length of columns. The scan data process scans from row 1 to the number that is specified by the GUESSINGROWS option.

```
proc import REPLACE
    Datafile=  "H:\Data\ISSAS Users Group\Shoes.csv"
    Out=       temp.Shoes_csv_imported;
    getnames=yes;    Guessingrows=100;
run;
*395 rows created in TEMP.SHOES_CSV_IMPORTED                from H:\Data\ISSAS Users Group\Shoes.csv.
NOTE: TEMP.SHOES_CSV_IMPORTED data set was successfully created.;
ODS – Output Delivery System
```

This ODS code writes a report in HTML Excel format on a remote server. The user either can open this file and save it on a PC folder in Excel Workbook format or copy parts/all of the report to an existing xls or xlsx report file:

```
Rsubmit;
ods html body="/dss/ishapir/Procs05.xls";
proc print data=sashelp.shoes noobs ;
Title1 Fields by Region;
run;
ods html close;
endrsubmit;
;
```

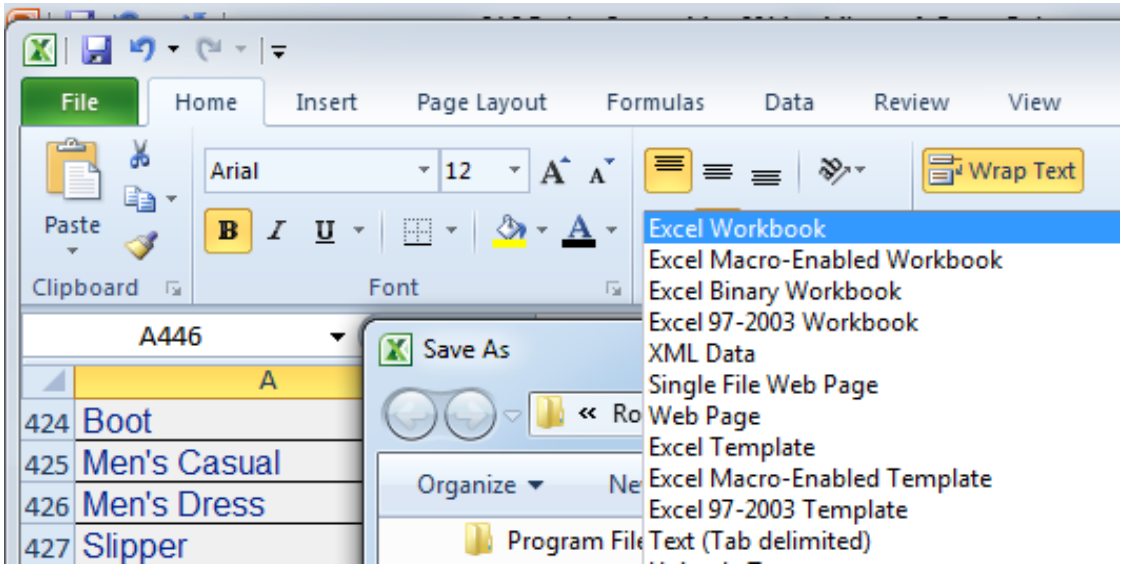
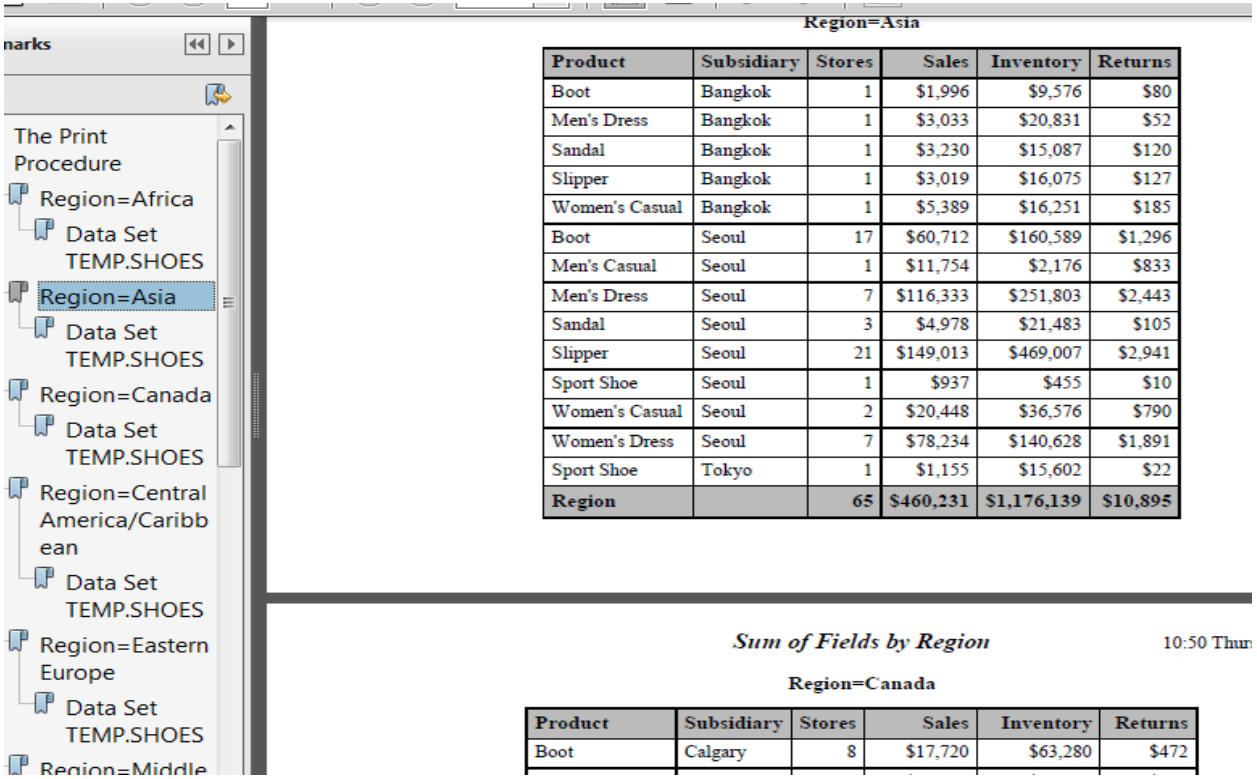


Figure 40: Options for saving the ODS output in Excel

This ODS code writes a report in HTML PDF format on a remote server.

```
Rsubmit;
ods pdf body="/dss/ishapir/Procs05.pdf";
proc print data=shoes noobs ;
        sum Inventory Returns Sales Stores;
        by region;
        title1 Sum of Fields by Region;
run;
ods pdf close;
Rendsubmit;
```


narks
The Print Procedure
Region=Africa
Data Set TEMP.SHOES
Region=Asia
Data Set TEMP.SHOES
Region=Canada
Data Set TEMP.SHOES
Region=Central America/Caribbean
Data Set TEMP.SHOES
Region=Eastern Europe
Data Set TEMP.SHOES
Region=Middle
Region=Asia
Product | Subsidiary | Stores | Sales | Inventory | Returns
Boot | Bangkok | 1 | $1,996 | $9,576 | $80
Men's Dress | Bangkok | 1 | $3,033 | $20,831 | $52
Sandal | Bangkok | 1 | $3,230 | $15,087 | $120
Slipper | Bangkok | 1 | $3,019 | $16,075 | $127
Women's Casual | Bangkok | 1 | $5,389 | $16,251 | $185
Boot | Seoul | 17 | $60,712 | $160,589 | $1,296
Men's Casual | Seoul | 1 | $11,754 | $2,176 | $833
Men's Dress | Seoul | 7 | $116,333 | $251,803 | $2,443
Sandal | Seoul | 3 | $4,978 | $21,483 | $105
Slipper | Seoul | 21 | $149,013 | $469,007 | $2,941
Sport Shoe | Seoul | 1 | $937 | $455 | $10
Women's Casual | Seoul | 2 | $20,448 | $36,576 | $790
Women's Dress | Seoul | 7 | $78,234 | $140,628 | $1,891
Sport Shoe | Tokyo | 1 | $1,155 | $15,602 | $22
Region | | 65 | $460,231 | $1,176,139 | $10,895
Sum of Fields by Region
10:50 Thur
Region=Canada
Product | Subsidiary | Stores | Sales | Inventory | Returns
Boot | Calgary | 8 | $17,720 | $63,280 | $472

References and Proceedings
General SAS® questions can be answered online via a search engine search such as:
SAS Proc Format
SAS Conversion of Characters to Numerics
SAS Proceedings Proc tabulate

SAS® Online Information
http://support.sas.com/documentation/
http://support.sas.com/onlinedoc/913/docMainpage.jsp

Conclusion:

You have officially passed **SA-02-2014 SAS 101 For Newbies--Not to Little, Not Too Much--Just Right** and know how to spell SAS forwards and backwards
May the SAS Force Be With You

Minion Stuart aka **Ira Shapiro**
Minion Stuart's email: ira_shapiro@uhc.com
Phone: 651.247.9906