Week3_Part2

September 13, 2019

1 GWU STAT 4197/6197

1.0.1 Week 3 SAS Code Examples (Part 2): Transforming Data

- Creating New Variable by Using
  - Assignment Statements
  - SELECT/WHEN/OTHERWISE Statements
  - IFC/IFN/WHICHC/WHICHN Functions
  - Case Expression in PROC SQL
  - SUM and RETAIN Statements

1.0.2 Creating a New Variable by Recoding Distinct Values

```
In [4]: data work.demographics;
         set sashelp.demographics;
         length region_name$ 22.;
         if region = 'AFR' then region_name = 'Africa';
         else if region = 'AMR' then region_name = 'Americas';
         else if region = 'EUR'  then region_name= 'Europe';
         else if region = 'EMR' then region_name ='Eastern Mediterranean';
         else if region = 'SEAR' then region_name= 'South-East Asia';
         else if region = 'WPR' then region_name= 'Western Pacific';
         run;

         proc freq data=work.demographics;
          tables region_name;
        run;
```

```
Out[4]: <IPython.core.display.HTML object>
```

```
In [ ]: ### Creating
```

```
In [20]: options nocenter nodate nonumber;
         data Heart;
          length AgeAtDeath_group $25;
           set sashelp.heart;

           /*Character-type categorical variables using an assignment statement;
```

```
 IF-THEN/Else Statements are used to conditionally assign values to variables.*/

 if 36<=AgeAtDeath<=49 then AgeAtDeath_group='36-49 Years';
 else if 50<=AgeAtDeath<=64 then AgeAtDeath_group= '50-64 Years';
 else if 65<=AgeAtDeath<=79 then AgeAtDeath_group= '65-79 Years';
 else if 80<=AgeAtDeath<=94 then AgeAtDeath_group= '80-94 Years';
 else AgeAtDeath_group= ' ';

title "Age at Death Grouping Created with an Assignment Statement";
title2 '(IF-THEN/ELSE-IF-THEN in DATA Step)';

proc freq data=Heart;
 table AgeAtDeath_group;
run;
```

Out[20]: <IPython.core.display.HTML object>

1.0.3 Creating Character-Type Categorical Variables by Using a PUT Function in the Assignment Statement

In [23]:
```
proc format ;
 value agefmt low-49 = '36-49 Years'
              50-64 = '50-64 Years'
              65-79 = '65-79 Years'
              80-94 = '80-94 Years' ;
 data Heart;
 length AgeAtDeath_group $25;
 set sashelp.heart;
 *Character-type categorical variables using a PUT function;
 if ageatdeath ne . then ageatdeath_group=put(ageatdeath, agefmt.);

title 'Age at Death Grouping Created with an Assignment Statement';
title2 'and the PUT Function in DATA Step';
proc freq data=Heart;
table  ageatdeath_group ;
run;
title;
```

Out[23]: <IPython.core.display.HTML object>

In [26]:
```
data Heart;
  length AgeAtDeath_group AgeAtDeath_group_x $25;
  set sashelp.heart;
   if 36<=ageatdeath <=49 then
       DO;
        AgeAtDeath_group ='36-49 Years';
        AgeAtDeath_group_x= 'Adults';
       END;
    else if 50<=ageatdeath<=64 then
```

```
            DO;
              AgeAtDeath_group = '50-64 Years';
              AgeAtDeath_group_x = 'Middle-Aged Adults';
            END;

         else if ageatdeath>=65 then
            DO;
              AgeAtDeath_group = '65+ Years';
             AgeAtDeath_group_x ='Older Adults';
            END;

        title 'Listing of Multiple Variables Created with DO Group';
       title;
        proc freq data=Heart;
         tables AgeAtDeath_group AgeAtDeath_group_x;
       run;
```

```
Out[26]: <IPython.core.display.HTML object>
```

1.0.4 The IFC Function

- uses the IF-THEN/ELSE logic
- can be used to create a new Variable with this function that normally uses three arguments

  1st argument - a logical expression () – a condition (true/false) to be evaluated

  2nd argument - character value returned when true

  3rd argument – character value to returned when false #### The IFN Function

It is the same as the IFC function except that the IFN function returns the numeric value. For a logical expression with a missing value as in the following example, you can have a 4th argument for SAS to return the value in the 4th argument.

```
In [1]: proc format;
            value agefmt
               . = 'Unknown'
               1 = '36-64 Years'
               0 = '65+ Years';
        data IFC1_IFN1;
         length agedth_group_IFC1 $10;
         set sashelp.heart;
           agedth_group_IFC1 = IFC(36<=ageatdeath<=64, '36-64 Years', '65+ Years');
           agedth_LE64_IFN1 = IFN(36<=ageatdeath<=64, 1, 0);
           agedth_LE64_IFN1_formatted = put(agedth_LE64_IFN1, agefmt.);
          title1 'Ex12_IFC_IFN_Function.sas';
        proc freq;
        tables agedth_group_IFC1 agedth_LE64_IFN1 agedth_LE64_IFN1_formatted;
        run;
       title;
```

```
SAS Connection established. Subprocess id is 740
```

Out[1]: <IPython.core.display.HTML object>

In [29]:
```
*Ex13_IFN_Fourth_Argument.sas;
DATA Work.Ifn_Func;
INPUT property_value;
property_tax = ifn(property_value GE 150000,
               property_value*.02,
             property_value*.015, .);
format property_value property_tax dollar8.;
datalines;
150000
.
250000
100000
;
title1 'Ex13_IFN_Fourth_Argument.sas';
proc print data=Work.Ifn_Func;
run;
title1;
```

Out[29]: <IPython.core.display.HTML object>

1.0.5 WHICHC/WHICHN Functions

These two functions are used to search through a list of arguments and return the index of the first one that matches a given reference value.

In [11]:
```
options nonotes nocenter nodate nonumber nosource;
ods html close;
DATA _NULL_;
 x =  whichc("ECON", "STAT", "MATH", "ECON");
 y = whichn(8,7, 8,9);
 put x= y= ;
run;
```

Out[11]: <IPython.core.display.HTML object>

In [30]:
```
*Ex14_in_whichn_whichc.sas;
* Adapted from Joe Matise's code  (SAS-L) - 5/16/2016;
 data have1;
    set sashelp.class (obs=3);
    Age_13_dummy = whichn(13,age);
    Gender_male_dummy= whichc('M',sex);
  run;
 title1 'Ex14_in_whichn_whichc.sas';
 proc print data=have1 noobs;  run;
 title1;
```

```
Out[30]: <IPython.core.display.HTML object>
```

Ex20_SELECT_WHEN.sas
See example code on SELECT WHEN here

1.0.6 Using WHEN Statements in a SELECT Group

- Evaluating the when-expression When a select-expression Is Included
- Evaluating the when-expression When a select-expression Is Not Included
- Evaluating the when-expression When a statement-list Is Not Included Comparisons

1.0.7 Examples

- Example 1: SELECT with a select-expression
- Example 2: SELECT without a select-expression
- Example 3: SELECT without an IF Expression

```
In [40]: *Ex21_create_variable_in_SQL.sas (Part 1);
         options nocenter nodate nonumber;
         title1 'Ex21_create_variable_in_SQL.sas (Part 1)';
         PROC SQL;
         SELECT
           CASE
           WHEN weight <100 THEN '<100 lbs'
           WHEN weight GE 100 AND weight LT 120 THEN '100-<120 lbs'
           WHEN weight GE 120 AND weight LE 150 THEN '120-150 lbs'
           ELSE '120-150 lbs'
           END AS Weight_Cat label= 'Weight Category',
           count(*) as freq_count
         FROM sashelp.class
         group by Weight_Cat
         order by  freq_count desc;
         quit;
```

```
Out[40]: <IPython.core.display.HTML object>
```

1.0.8 The SUM Statement:

- (by default), creates the sum or accumulator variable that is automatically set to 0 before the first observation is read. The variable's value is retained from one iteration to the next, as if it had appeared in a RETAIN statement. The sum statement is equivalent to using the SUM function and the RETAIN statement.
- has an exprerssion that is evaluated and the results are added to the accumulator variable.
- automatically retains the variable across iterations of the DATA step
- ignores the missing values

```
In [41]: *Ex22_Retain_Sum_Statement.sas (Part 1);
         DATA temp ;
           INPUT month sales @@;
```

```
        Total_sales+sales;
    FORMAT Sales Total_sales dollar8.;
     DATALINES;
       1 4000 2 5000 3 . 4 5500 5 5000
       ;
title1 'Ex22_Retain_Sum_Statement.sas (Part 1)';
title2 'SUM Statement';
PROC PRINT noobs; run;
```

```
Out[41]: <IPython.core.display.HTML object>
```

1.0.9 The SUM Statement Coupled with the RETAIN Statement

- To reset the sum variable to a value other than zero, you need to include the accumulator variable in a RETAIN statement with an initial value.

```
In [2]: *Ex22_Retain_Sum_Statement.sas (Part 3);
        options nocenter nodate nonumber;
        DATA temp;
           RETAIN Total_sales 1000;
           INPUT month sales @@;
            Total_sales+sales;
           FORMAT Sales Total_sales dollar8.;
           DATALINES;
           1 4000 2 5000 3 . 4 5500 5 5000
           ;
        title 'Ex22_Retain_Sum_Statement.sas (Part 3)';
        title2 'RETAIN and SUM Statements';
        PROC PRINT noobs;
         var month sales Total_sales;
        RUN;
        title;
```

```
Out[2]: <IPython.core.display.HTML object>
```

1.0.10 The RETAIN Statement (as an Alternative to the SUM Statement)

- returns the value of the variable in the PDV acorss iterations of the DATA step
- initializes the retained variable to missing or a specified before the first iteration of the DATA step
- is a compile-time statement

The RETAIN statement has no effect on variables that are read with SET, MERGE, or UPDATE statements. Variables read from SAS data sets are retained automatically.

```
In [42]: *Ex22_Retain_Sum_Statement.sas (Part 2);
         DATA temp1 ;
            RETAIN Total_sales 0;
            FORMAT Sales Total_sales dollar8.;
```

```
   INPUT month sales @@;
    Total_sales= sum(Total_sales, sales);
   DATALINES;
   1 4000 2 5000 3 . 4 5500 5 5000
   ;
title1 'Ex22_Retain_Sum_Statement.sas (Part 2)';
title2 'RETAIN Statement';
PROC PRINT data=temp1;
  VAR month sales Total_sales;
run;
```

```
Out[42]: <IPython.core.display.HTML object>
```

1.0.11 The SUM Statement in DATA Step with BY-Group Processing

The SASHELP.CARS has 428 rows categorized into 38 makes. How to count the number of cars for each of these 38 makes?

Code Explanation (Programming for SAS Viya)

- The PROC SORT step and the DATA step with BY-Group Processing calculates the number of cars by MAKE.
- The data set must first be sorted to take advantage of the DATA step with By-Group processing.
- The program uses FIRST. processing to set the first observation of COUNT to 0 at the beginnibng of the each MAKE.
- The program uses LAST. processing to write the last observation of the MAKE that contains the final accumulated COUNT for each MAKE.

```
In [1]: *Ex22_Retain_Sum_Statement.sas  (Part 4);
        proc sort data = sashelp.cars out=cars; by make; run;
        data cars_x;
          set cars;
          count + 1;
          by make;
          if first.make then count = 1;
          if last.make;
        run;
        title 'Ex22_Retain_Sum_Statement.sas (Part 4)';
        title2 'SUM Statement';
        proc print data=cars_x;
        var make count;
        sum count;
        run;
        title;
```

```
SAS Connection established. Subprocess id is 5100
```