Paper 91-2010

Plotting Summary ROC Curves from Multiple Raters Using the SmryROCPlot Macro

Matthew T. Karafa, PhD.[†], Nancy Obuchowski, PhD.[†],
[†]Quantitative Health Sciences - Cleveland Clinic Foundation , Cleveland, OH

Abstract

As an alternative to existing solutions provided by others, the authors created a macro to construct an ROC curve using average measures. The major problem with generating such a curve is for a given false positive rate (FPR, or 1- specificity), a rater may not have an actual measurement. Thus to accurately create the plot, we need to interpolate between the nearest actually observed points for a given reader to generate the Summary ROC plot. SmryROCPlot() can do this interpolation for you with minimal inputs. The user provides the SmryROCPlot() macro with variable names for the true event state, the observed event state, and the rater's identification. The user then also lists the FPR's at which the plot should be generated. The macro steps through each rater's data, and either uses the actual measured specificity if it is available or calculates a simple linear interpolation between the nearest observed FPR's for each "plot" FPR. The macro then provides an output data set, which can be used by PROC GPLOT to make a nice looking Summary ROC curve.

Introduction

In diagnostic testing literature, the multiple rater design is used to account for differences when the evaluator of the test results must make some subjective interpretation. To do this we have several raters evaluate the same set of patients' results and score them. Designs like these allow for accounting of the variations in a rater's ability to perform the interpretations. The typical method for determining the ROC area would be to construct multiple curves (one for each rater) and take the average of the individual statistic to summarize for the test as a whole. It is desirable to have a corresponding ROC curve to plot for this area, however the normal methods employed to picture such a curve underestimate the discrimination ability of the diagnostic test(1). If you merely ignore the rater and treat as one large sample, the ROC curve and corresponding area are markedly underestimated. To this end the authors created a macro to construct an ROC curve using average measures, similar methods of Swets and Pickett(2). The major problem with generating such a curve is for a given false positive rate (FPR, or 1-specificity), a rater may not have an actual measurement at that FPR. Thus to accurately create the plot, we need to interpolate between the nearest actually observed points for a given reader to generate the Summary ROC plot. SmryROCPlot() can do this interpolation for you with minimal inputs.

The Problem

The proposed method is simple, at each point of FPR for which we would like to generate a plot, we need an average sensitivity across all raters. If all raters used the same cutpoints and have observable values at all of the interested FPRs this is simple, we have points to average. However this is almost never the case, usually we are forced to interpolate between two known points of FPR and specificity for a given rater. For

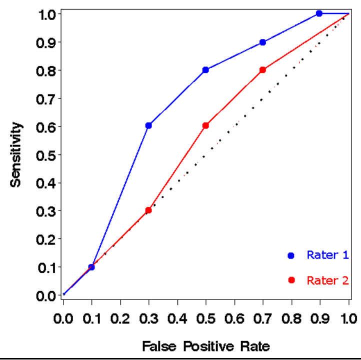

Figure 1: Theoretical ROC plot two raters that use different cut-points. A summary plot needs to interpolate Rater 2's sensitivities at FPRs 0.1 & 0.9.

example take Figure 1, here we have 2 simple ROC curves from two raters, one with 5 observed points and one with only 3. If we wished to take the average ROC curve for these at each of the 5 FPRs that Rater 1 observes, we will need to interpolate what the value of Rater 2 is at an FPR of 0.1 and 0.9. The methods proposed is to use a simple interpolation between the two points, using "y=mx+b" from Algebra 101:

$$Sens_{Plot} = Sens_{Upper} - \left(\left(\frac{Sens_{Upper} - Sens_{Lower}}{FPR_{Upper} - FPR_{Lower}}\right)\right) + \left(\left(\frac{Sens_{Upper} - Sens_{Lower}}{FPR_{Upper} - FPR_{Lower}}\right) \times FPR_{Plot}\right) \qquad \text{EQ 1}$$

This interpolation gives us an estimate of Rater 1's sensitivity **IF WE HAD OBSERVED** that FPR from them.

While this seems relatively straight forward, for the statistician to do this manually could be quite time consuming with many raters. What we would like the macro to do is step through a list of "Plot FPR's" and generate the "Plot Sensitivity" for us.

Programming Strategy

First the macro needs to get some information from the statistician. The statistician provides the SmryROCPlot() macro with variable names for the true event state, the observed event state, and the rater's identification. So a sample call would look something like this:

```
%SmryROCPlot(ds = TestingData,
             Truth = GroundTruth,
             Observation = Unaidedreadconfidence,
             IDvar = readerid,
             PlotFPRs =0 0.025 0.05 0.1 to 1.0 by 0.1,
             OutData = Plot2,
             GroupLabel = Unaided);
run;
```

The user also lists the FPRs at which the plot should be generated. A nice feature that was implemented was to allow for a "human" readable list for these FPR's. The macro puts the resulting data into the data specified in &OutData. (which will be called "outdata" by default). Additionally the macro will accept a group label in the event multiple calls to SmryROCPlot() are used to generate multiple summary curves for the same plot.

As this macro will be in the "wild" in my department, the first part of the macro then begins to check the validity of the parameters and returns appropriate comments to the log in the event of being passed something it's not expecting. Any aspiring coder will share horror stories of end users attempting to put the wrong values into a program and then being blamed for major problems. The best way to prevent these from happening is smart coding. That starts with ensuring that the user enters the correct information and knows when it was THEIR fault rather than the macro's fault. SmryROCPlot() checks through the entire parameter list, and stops after announcing any errors. To do this we first create a Boolean macro variable which is false if there are no errors and true if we've found any:

```
%let __SmryROCPlot_Err=0;
```

We then check our parameters for validity, and reset this to true when there's a problem:

```
%if %length(&Truth.)=0 %then %do;
    %put ERROR: NO TRUTH VARIABLE SUPPLIED;
    %let __SmryROCPlot_Err=1;
%end;
%if %length(&Observation.)=0 %then %do;
    %put ERROR: NO OBSERVATION VARIABLE SUPPLIED;
    %let __SmryROCPlot_Err=1;
%end;
```

Notice that each of the %put statements that provide the error messages starts with "ERROR:" just like a stock SAS error message. This enables the Log system to highlight the error messages properly. Some error checks are trivial, was the variable even provided like these or did the correct value get entered. However using this methodology, more interesting and difficult checks can be preformed. For example this segment of code checks to make sure the provided variable is even in the dataset listed:

```
proc sql noprint;
    select name into:__AllVarsin_DS separated by " "
    from sashelp.vcolumn
    where libname='WORK' & memname=%upcase("&ds.");
    quit;
run;
%if %index(%upcase(&__AllVarsin_DS.),%upcase(&Truth.)) eq 0 %then %do;
    %put ERROR: &Truth. is not found in &ds..;
    %let __SmryROCPlot_Err=1;
%end;
```

It does this by first getting a list from the sashelp.vcolumn dataset which is created by default and using proc sql to get a macro variable list separated by a blank space of all the variables in the dataset. We then use %index() to check to ensure the variable provided is actually IN the data. I can't begin to tell you how much these 10 lines of code have saved me in later debugging when someone just mistyped "Gender" as "Gedner".

Once we check all the errors if there's a problem, we stop the macro using:

```
%if &__SmryROCPlot_Err. %then %do;
    data _null_;
       abort 3;
    run;
%end;
```

This stops processing and sends an error code different from the standard, so we know it was a problem in the macro as opposed to other areas of the program. This lets the saavy user know better where to look for trouble in a longer code block.

Another coding trick I use in development of such proc replacement type macros is to use formal "DEBUG" parameters in my code. I then use the macro variable &DEBUG. as a Boolean variable to tell me if I should put out the debugging information. Once development is finished and this moves to production, these serve as invaluable aids for discovery of troubles when the users inevitably break code with data that I never thought would hit the program. For example this section:

```
%if &DEBUG eq 1 %then %do;
    %let DEBUGCount = %trim(%left(%sysevalf(&DebugCount. + 1)));
    %put ****************************************************;
    %put DEBUG Point &DebugCount ;
    proc print data = ReadersOut;
        var &IDvar.
            %do Popem = 1 %to &NumObsCuts.;
                Sens&PopEm. FPR&PopEm.
            %end;;
    run;
    %put ****************************************************;
%end;
```

runs a proc print of the Raters database. This ONLY occurs if the debugging is turned on, so it never even gets called if debug is off. I use a macro variable to count which debug point is being produced to make them easier to find in the log. By initializing &DEBUGCount. to zero at the outset, and adding in the second line, I never have to keep a count in my code, I just run the code and it tells me it's the "Xth" debug point. Very useful in tracking down errors later.

To parse the &PlotFPRs. list we use a built in macro that parses a list in the format of "A to B by C" into a delimited list of numbers. In this case we wanted to add in the ability to specify additional numbers in a form like: "A B C X to Y by Z" which is handled by parsing each space delimited word and creating the list. Once the PlotFPR is fixed into a listing of the plot points, the macro steps through each cut-point and creates a rater specific data set. The sensitivity is merely the number observed over the cutpoint Each row of this data has all of the sensitivities and FPR's for each rater at each of the "K" observed cut-points. Since we have no idea how many cut-points will be in the data a priori, we must use a loop to step through and create them:

```
proc sql;
    create table ReadersOut as
        select distinct &IDvar.
           %do Popem = 1 %to &NumObsCuts.;
              , sum((&Observation. gt %scan(&Observationlist.,%sysevalf(&Popem.-1),"|")) and
                    (&Truth. eq 1))/&NumberTrue  as Sens&Popem.,
                sum((&Observation. gt %scan(&Observationlist.,%sysevalf(&Popem.-1),"|")) and
                    (&Truth. eq 0))/&NumberFalse as FPR&Popem.
           %end;
        from &ds.
        group by &IDvar.
        order by &IDvar.;
    quit;
run;
```

Finally the dataset called "intermediate" steps through each of the PlotFPRs and compares them to the rater's data. For each PlotFPR the actual measured specificity is used or if it is available or calculates a simple linear interpolation between the nearest observed FPR's for each "plot" FPR. The macro determines the "nearest" observed FPR" by looking stepping through the 3 arrays:

```
array FPR4Plot(&NumPlotPts.) _TEMPORARY_ (&PlotFPRs_Fixed.);
array FPRActual(*) %do Popem = 1 %to &NumObsCuts.; FPR&PopEm. %end;;
array SensActual(*) %do Popem = 1 %to &NumObsCuts.; Sens&PopEm. %end;;
```

For each value in the Temporary Array "FPR4Plot" we plan to export an observation to the final dataset, which can be used by PROC GPLOT to make a nice looking Summary ROC curve. The macro uses a brute force nested loop, starting lower at 0 and upper at 1 for both FPR and sensitivity. While not the most efficient, at each FPR4plot this ensures the closest value is used to determine the extrapolation.

The final result is a data set that can be passed to proc gplot to make the ROC curve:

```
%SmryROCPlot(ds = TestingData,
             Truth = GroundTruth,
             Observation = ConcurrentReadConfidence,
             IDvar = readerid,
             PlotFPRs =0 0.025 0.05 0.1 to 1.0 by 0.1,
             OutData = Plot1,
             GroupLabel = Concurrent);
run;
%SmryROCPlot(ds = TestingData,
             Truth = GroundTruth,
             Observation = Unaidedreadconfidence,
             IDvar = readerid,
             PlotFPRs =0 0.025 0.05 0.1 to 1.0 by 0.1,
             OutData = Plot2,
             GroupLabel = Unaided);
run;
%SmryROCPlot(ds = TestingData,
             Truth = GroundTruth,
             Observation = SecondReadConfidence,
             IDvar = readerid,
             PlotFPRs =0 0.025 0.05 0.1 to 1.0 by 0.1,
             OutData = Plot3,
             GroupLabel = Second Read);
run;
```

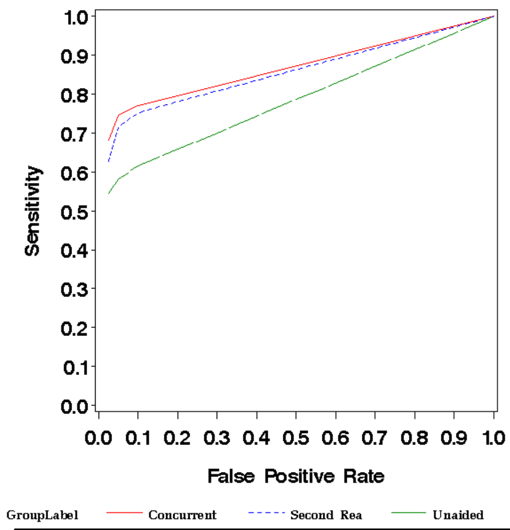

Figure 2: The resulting ROC plot from three calls to SmryROCPlot(). Plot describes raters for 3 types of assistance on reading a colonoscopy.

```
data plots;
    set plot1
        plot2
        plot3;
run;

goptions reset=all;
goptions device = TIFFP gsfname=graf gsfmode=replace rotate=landscape
         cback=white colors=(blue red)ftext=zapfb ftitle=zapfb;
filename graf "~/graphs/Fig1.tif";

symbol1 color=red height=1.5 line=1 interpol=l;
symbol2 color=blue height=1.5 line=2 interpol=l;
symbol3 color=green height=1.5 line=5 interpol=l;

axis1 label=(h=1.5 font=swissb 'False Positive Rate') order=0 to 1 by 0.1
      value=(h=1.5 font=swissb) length = 12cm minor = none;

axis2 label=(h=1.5 font=swissb angle=90 'Sensitivity') order=0 to 1 by 0.1
      value=(h=1.5 font=swissb) length = 12cm minor = none;

proc gplot data = plots;
    plot SummarySens*SummaryFPR = GroupLabel / haxis=axis1 vaxis=axis2;
run;
quit;
```

Conclusion

Here we present a simple macro that creates a dataset that can be used to generate a summary ROC plot. This macro uses a two embedded macros to parse a complex "list" of FPR points. The user can enter a more human readable form (X to Y by Z) or a simple delimited list (A,B,C,…) or a mix of the two for the list definition. These macros could easily be extracted and placed in a macro bank for use in other designs. It also highlights the use of temporary arrays to store the FPR list and PROC SQL to generate macro variables and macro lists.

At its core, this macro is rather simple statistically. However, the programming challenges it presented in terms of parameter processing and eventual user friendliness added to the complexity of coding. Most of the techniques presented here (parameter checking, the DEBUG routines) are useful to adapt to your general macro coding practice

References

(1) Obuchowski NA. New methodological tools for multiple-reader ROC studies. Radiology 2007; 243:10-12.

(2) Swets JA Pickett RM. Evaluation of diagnostic systems: methods from signal detection theory. New York, NY: Academic Press, 1982

Contact Information

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

Name:	Matthew Karafa, PhD
Enterprise:	Cleveland Clinic Foundation
Address:	9500 Euclid Avenue / JJN3-01
City, State ZIP:	Cleveland, Ohio 44195
Phone:	(216) 445-9556
Fax:	(216) 444-8023
E-mail:	karafam@ccf.org
Web:	http://www.lerner.ccf.org/qhs/

Appendix 1: The %SmryROCPlot () macro

```
%macro SmryROCPlot (ds = _LAST_,
                   Truth = ,
                   TruthFmt = ,
                   Observation = ,
                   IdVar = ,
                   PlotFPRs = ,
                   OutData = OutData,
                   GroupLabel = "PlotGp",
                   DEBUG = 0
                  );
    %local DEBUG DEBUGCount __SmryROCPlot_Err;


    ***************************************************************************;
    ** Couple of Local Macros - for parameter processing;
    ***************************************************************************;

    %macro ParseToShorthand(InVar);
        %local _X_ _Y_ _Z_ _Temp_ _Counter_ __ParseToShorthand_Err;
        %let __ParseToShorthand_Err=0;

        %if %length(&InVar.)=0 %then %do;
            %put ERROR: NO VARIABLE SUPPLIED;
            %let __ParseToShorthand_Err=1;
        %end;
        %if not((%scan(&InVar.,1," ") gt 0)) or
            not((%scan(&InVar.,3," ") gt 0)) %then %do;
            %put ERROR: EXPECTING 'X TO Y BY Z' TYPE FORMAT;
            %let __ParseToShorthand_Err=1;
        %end;
        %if &__ParseToShorthand_Err. eq 0 %then %do;
           %let _X_ = %scan(&InVar.,1," ");
           %let _Y_ = %scan(&InVar.,3," ");
           %if %scan(&InVar.,5," ") gt 0 %then %let _Z_ = %scan(&InVar.,5," ");
           %else %let _Z_ = 1;
           %let _Counter_ = 0;
           %let _Val_ = &_X_.;

           %do %while (%sysevalf(&_Val_. le &_Y_.));
               %if %length(&_Temp_.) eq 0 %then %let _Temp_ = %sysevalf(&_Val_.);
               %else %let _Temp_ = &_Temp_.  %sysevalf(&_Val_.);
               %let _Counter_ = %sysevalf(&_Counter_. + 1);
               %let _Val_ = %sysevalf(&_Val_. + %sysevalf(&_Z_.));
           %end;
        %end;
        %if &__ParseToShorthand_Err. eq 1 %then %let _Temp_ = _ERROR_;
        &_Temp_.
    %mend;

    %macro ParseMixedShorthand(InVar);
        %local _CurrWd_ _NxtWd_ _X_ _Y_ _Z_ __TempCnt__ _Temp_
               __ParseMixedShorthand_Error;
        %let __ParseMixedShorthand_Error = 0;

        %if %upcase(%scan(&InVar.,1," ")) = TO %then %do;
            %put ERROR: &InVar. begins with 'TO' expecting a number.;
            %let __ParseMixedShorthand_Error=1;
            %let _TEMP_ = _ERROR_;
        %end;
        %else %do;
            %let __TempCnt__ = 0;
            %let _NxtWd_ = %scan(&InVar.,%sysevalf(&__TempCnt__+1)," ");
            %do %while (%length(&_NxtWd_.) gt 0);
               %put Current Word = **&_CurrWd_.**        Next Word = **&_NxtWd_.**;
               %if %upcase(&_NxtWd_.) eq TO %then %do;
                   %let _A_ = %scan(&InVar.,%sysevalf(&__TempCnt__.)," ");
                   %let _B_ = %scan(&InVar.,%sysevalf(&__TempCnt__.+2)," ");
                   %if %upcase(%scan(&InVar.,%sysevalf(&__TempCnt__.+3)," ")) eq BY %then
                   %do;
                       %let _C_ = %scan(&InVar.,%sysevalf(&__TempCnt__.+4)," ");
                       %let __TempCnt__ = %sysevalf(&__TempCnt__.+4);
                   %end;
```

```
                %else %do;
                    %let _C_ = 1;
                    %let __TempCnt__ = %sysevalf(&__TempCnt__.+2);
                %end;
                %let _CurrWd_ = %ParseToShorthand(&_A_. TO &_B_. BY &_C_.);
                %if %length(&_TEMP_.) eq 0 %then
                    %let _TEMP_ = %trim(%left(&_CurrWd_.));
                %else %let _TEMP_ =&_TEMP_. %trim(%left(&_CurrWd_.));
                %if %length(%scan(&InVar.,%sysevalf(&__TempCnt__+1)," ")) gt 0 and
                    %length(%scan(&InVar.,%sysevalf(&__TempCnt__+2)," ")) eq 0 %then
                    %let _TEMP_ =&_TEMP_.
                            %trim(%left(%scan(&InVar.,%sysevalf(&__TempCnt__+1)," ")));
            %end;
            %else %do;
                %if %length(&_TEMP_) eq 0 %then %let _TEMP_ = %trim(%left(&_CurrWd_.));
                %else  %let _TEMP_ =&_TEMP_. %trim(%left(&_CurrWd_.));
                %let _CurrWd_ = %trim(%left(&_NxtWd_.));
                %if %length(%scan(&InVar.,%sysevalf(&__TempCnt__+1)," ")) gt 0 and
                    %length(%scan(&InVar.,%sysevalf(&__TempCnt__+2)," ")) eq 0 %then
                    %let _TEMP_ =&_TEMP_.
                            %trim(%left(%scan(&InVar.,%sysevalf(&__TempCnt__+1)," ")));
            %end;
            %let __TempCnt__ = %sysevalf(&__TempCnt__+1);
            %let _NxtWd_ = %scan(&InVar.,%sysevalf(&__TempCnt__+1)," ");
        %end;
    %end;
    &_Temp_.
%mend;

***************************************************************************;
%if &DEBUG eq 1 %then %let DEBUGCount = 0;
***************************************************************************;
** Parameter Checking Section;
***************************************************************************;
%let __SmryROCPlot_Err=0;

%if %length(&Truth.)=0 %then %do;
    %put ERROR: NO TRUTH VARIABLE SUPPLIED;
    %let __SmryROCPlot_Err=1;
%end;
%if %length(&Observation.)=0 %then %do;
    %put ERROR: NO OBSERVATION VARIABLE SUPPLIED;
    %let __SmryROCPlot_Err=1;
%end;
%if %length(&IDVAR.)=0 %then %do;
    %put ERROR: NO ID VARIABLE SUPPLIED;
    %let __SmryROCPlot_Err=1;
%end;
%if %length(&PlotFPRs.)=0 %then %do;
    %put ERROR: NO ID VARIABLE SUPPLIED;
    %let __SmryROCPlot_Err=1;
%end;

proc sql noprint;
    select name into:__AllVarsin_DS separated by " "
    from sashelp.vcolumn
    where libname='WORK' & memname=%upcase("&ds.");
    quit;
run;
%if %index(%upcase(&__AllVarsin_DS.),%upcase(&Truth.)) eq 0 %then %do;
    %put ERROR: &Truth. is not found in &ds..;
    %let __SmryROCPlot_Err=1;
%end;
%if %index(%upcase(&__AllVarsin_DS.),%upcase(&IDVar.)) eq 0 %then %do;
    %put ERROR: &IdVar. is not found in &ds..;
    %let __SmryROCPlot_Err=1;
%end;
%if %index(%upcase(&__AllVarsin_DS.),%upcase(&Observation.)) eq 0 %then %do;
    %put ERROR: &Observation. is not found in &ds..;
    %let __SmryROCPlot_Err=1;
%end;

%let PlotFPRs_Fixed = %ParseMixedShorthand(&PlotFPRs.);
%if %index(&PlotFPRs_Fixed., _ERROR_) gt 0 %then %do;
    %put ERROR: BAD FORMAT PARAMATER: PlotFPRs.;
    %let __SmryROCPlot_Err=1;
%end;

%let NumPlotPts = 0;
%do %while (%scan(&PlotFPRs_Fixed.,&NumPlotPts.+1," |") ne );
    %let NumPlotPts=%sysevalf(&NumPlotPts. + 1);
%end;

%if &__SmryROCPlot_Err. %then %do;
    data _null_;
       abort 3;
    run;
%end;

%if &DEBUG eq 1 %then %do;
    %let DEBUGCount = %trim(%left(%sysevalf(&DebugCount. + 1)));
    %put ****************************************************;
    %put DEBUG Point &DebugCount ;
    %put   Truth Variable        = **&Truth.**;
    %put   TruthFmt              = **&TruthFmt.**;
    %put   ID Variable           = **&IDVar.**;
    %put   Observation Variable  = **&Observation.**;
    %put   Plot FPRs             = **&PlotFPRs.**;
    %put   Plot FPRs Fixed       = **&PlotFPRs_Fixed.**;
    %put ****************************************************;
%end;

***************************************************************************;
** Get Data Set Info Section;
***************************************************************************;
proc sql noprint;
    select &IDvar. into :LastIdVar from &ds;
    select distinct &Observation. into :Observationlist separated by "|"
        from &ds.
        order by &Observation.;
    select count(distinct &IDvar.) into :ReaderCount from &ds.;
    select sum(&Truth. eq 1) into :NumberTrue from &ds. where &IdVar. eq &LastIdVar.;
    select sum(&Truth. eq 0) into :NumberFalse from &ds. where &IdVar. eq &LastIdVar.;
    quit;
run;

%let NumObsCuts = 0;
%do %while (%scan(&Observationlist.,&NumObsCuts.+1,"|") ne );
    %let NumObsCuts=%sysevalf(&NumObsCuts. + 1);
%end;

%if &DEBUG eq 1 %then %do;
    %let DEBUGCount = %trim(%left(%sysevalf(&DEBUGCount. + 1)));
    %put ****************************************************;
    %put DEBUG Point &DebugCount ;
    %put   Last IDVar      = **&LastIdVar.**;
    %put   Reader Count    = **&ReaderCount.**;
    %put   Number True     = **&NumberTrue.**;
    %put   Number False    = **&NumberFalse.**;
    %put   Observationlist = **&Observationlist.**;
    %put   NumObsCuts      = **&NumObsCuts.**;
    %put ****************************************************;
%end;
```

```
***************************************************************************;
** Process ROC Sens and FPR Section;
***************************************************************************;

proc sql;
    create table ReadersOut as
        select distinct &IDvar.
           %do Popem = 1 %to &NumObsCuts.;
               , sum((&Observation. gt %scan(&Observationlist.,%sysevalf(&Popem.-1),"|")) and
                     (&Truth. eq 1))/&NumberTrue  as Sens&Popem.,
                 sum((&Observation. gt %scan(&Observationlist.,%sysevalf(&Popem.-1),"|")) and
                     (&Truth. eq 0))/&NumberFalse as FPR&Popem.
           %end;
        from &ds.
        group by &IDvar.
        order by &IDvar.;
    quit;
run;

%if &DEBUG eq 1 %then %do;
    %let DEBUGCount = %trim(%left(%sysevalf(&DebugCount. + 1)));
    %put *****************************************************;
    %put DEBUG Point &DebugCount ;
    proc print data = ReadersOut;
        var &IDvar.
            %do Popem = 1 %to &NumObsCuts.;
                Sens&PopEm. FPR&PopEm.
            %end;
            ;
    run;
    %put *****************************************************;
%end;

data Intermediate;
    set ReadersOut;
    by &IDvar.;
    array FPR4Plot(&NumPlotPts.) _TEMPORARY_ (&PlotFPRs_Fixed.);
    array FPRActual(*) %do Popem = 1 %to &NumObsCuts.; FPR&PopEm. %end;;
    array SensActual(*) %do Popem = 1 %to &NumObsCuts.; Sens&PopEm. %end;;

    do __Counter_1_ = 1 to dim(FPR4Plot);
        LowFPR = 0; UppFPR = 1; LowSens = 0; UppSens = 1;
        PlotFPR = FPR4Plot[__Counter_1_];
        do __Counter_2_ = 1 to dim(FPRActual);
            if LowFPR eq 0 and FPRActual[__Counter_2_] LE FPR4Plot[__Counter_1_] then
              do;
                LowFPR = FPRActual[__Counter_2_];
                LowSens = SensActual[__Counter_2_];
              end;
            else if LowFPR NE 0 and FPRActual[__Counter_2_] LE FPR4Plot[__Counter_1_]
              then do;
                if FPRActual[__Counter_2_] GT LowFPR then do;
                    LowFPR = FPRActual[__Counter_2_];
                    LowSens = SensActual[__Counter_2_];
                end;
              end;

            if UppFPR eq 1 and FPRActual[__Counter_2_] GE FPR4Plot[__Counter_1_]
              then do;
                UppFPR = FPRActual[__Counter_2_];
                UppSens = SensActual[__Counter_2_];
              end;
            else if UppFPR NE 1 and FPRActual[__Counter_2_] GE FPR4Plot[__Counter_1_]
              then do;
                if FPRActual[__Counter_2_] LT UppFPR then do;
                    UppFPR = FPRActual[__Counter_2_];
                    UppSens = SensActual[__Counter_2_];
                end;
              end;
        end;

        Slope = (UppSens - LowSens) / (UppFPR - LowFPR );
        Intercept = UppSens - Slope*UppFPR;
        PlotSens = Intercept+(PlotFPR*Slope);
```

```
            output;
        end;
    run;

    %if &DEBUG eq 1 %then %do;
        %let DEBUGCount = %trim(%left(%sysevalf(&DEBUGCount. + 1)));
        %put *****************************************************;
        %put DEBUG Point &DebugCount ;
        Proc Print data = Intermediate;
            var &IdVar PlotFPR PlotSens LowFPR LowSens UppFPR UppSens;
        run;
        %put *****************************************************;
    %end;

    proc sql noprint;
        create table &OutData. as
            select "&GroupLabel." as GroupLabel,
                   PlotFPR as SummaryFPR,
                   mean(PlotSens) as SummarySens
             from Intermediate
               group by PlotFPR
               order by PlotFPR;
        quit;
    run;
%mend; * SmryROCPlot();
```