Windows Application Using .NET and SAS® to Produce Custom Rater Reliability Reports
Sailesh Vezzu, Educational Testing Service, Princeton, NJ

ABSTRACT
Some of the common measures used to monitor rater reliability of constructed response items include Percent Exact Agreement, Percent Adjacent Agreement, Simple Kappa Coefficient, Weighted (Cicchetti-Allison, Fleiss-Cohen) Kappa Coefficient, Pearson Correlation Coefficient, Mean Difference, Standardized Mean Difference, and Matched-Pair T-Test. This paper discusses the development of a Windows application using Visual C#.NET and SAS to calculate various measures of rater reliability and produce custom reports. Users can select which measures to include in their reports and set threshold values to flag items with low rater reliability values.

INTRODUCTION
In the Research and Development division at Educational Testing Service (ETS), I work on several projects, which require me to calculate different measures of rater reliability in different ways. Depending on the purpose of the project, these measures sometimes need to be calculated per each item across several raters, per each rater across several items, or per each scoring session across all items and raters. In order to accommodate these scenarios, I built this application, which allows users to enter several parameters and produce customized reports with different measures of rater reliability.

The first section of this paper will discuss different measures of rater reliability and how to compute those using SAS. The second section of this paper will show how to build a Windows application using Visual C#.NET, get user input, and pass the input into SAS as parameters.

Figure 1 below shows the structure of a sample input dataset for this application. A typical input dataset includes the name of the item, the IDs of both rater 1 and rater 2, and the scores given by the raters. The data in this paper is for demonstration purposes only and is not actual data.

Figure 1, Sample Input Dataset
APP-FRIENDLY SAS CODING
To easily incorporate the SAS code into various applications, SAS code is organized into the following modules depending on SAS procedures used to compute these measures:

1. To calculate means, standard deviations, absolute mean difference and standardized mean difference
2. To calculate simple Kappa Coefficient and (Cicchetti-Allison, Fleiss-Cohen) weighted Kappa Coefficient
3. To calculate matched pair t-value and the corresponding p-value
4. To calculate Percent Exact Agreement and Percent Adjacent Agreement
5. To calculate Pearson Correlation Coefficient
6. To generate custom reports

All of these modules have an input dataset and output dataset as parameters. Any temporary datasets created during the module will be deleted at the end of the module using a DATASETS procedure.

MODULE 1: MEANS, STANDARD DEVIATIONS, MEAN DIFFERENCES
Proc Means in SAS can be used to calculate all the means, standard deviations, absolute mean differences, and standardized mean differences.

Absolute mean difference is the absolute value of the difference between the 2 scores, and the standardized mean difference is the difference between the means expressed in terms of the pooled standard deviation.

%Macro get_ns_means_diffs(indat, outdat, var1, var2, byvar);
proc means data=&indat. noprint;
  var &var1. &var2. ;
  by &byvar. ;
  output out=t_1 (drop=_type_ _freq_)
    n=N_&var1._&var2. mean=Mean_&var1. Mean_&var2.
    STD=SD_&var1. SD_&var2. ;
run;
data &outdat. ;
  set t_1;
  mnd_&var1._&var2. = abs(Mean_&var1. - Mean_&var2.);
  stdd_&var1. &var2. = (Mean_&var2. - Mean_&var1.)/(sqrt((SD_&var1.*2) + (SD_&var2.*2))/2));
  keep &byvar. N_&var1. &var2. Mean_&var1. Mean_&var2. SD_&var1. SD_&var2.
  mnd_&var1. _&var2. stdd_&var1. _&var2. ;
run;
proc datasets nolist;
  delete t_1;
run;
quit;
%Mend;

As you can see in the code above, module 1 has input and output datasets as parameters. Temporary dataset “t_1” created during the module will be deleted at the end of the module. All of the modules used in this application expect the input dataset to be sorted by the “byvar”. This is more efficient than sorting the input dataset in each module. My personal preference in writing app-friendly SAS code is to never alter or even sort the input dataset inside a module. If for some reason I have to make changes to the input dataset, I create a separate temporary work dataset and make changes to it. Doing this can help avoid unnecessary hassles when working with several modules within the same application.
MODULE 2: KAPPA COEFFICIENTS

Kappa coefficients are measures of association between score 1 and score 2 after correcting for chance agreement rates. Kappa is a more appropriate statistic to use when you want to compare reliability of items with different scales.

Agree option in Proc Freq will produce the simple or unweighted Kappa coefficient and the Cicchetti-Allison or linear weighted Kappa coefficient. When (wt=fc) is used next to the Agree option in Proc Freq, it produces the simple Kappa coefficient and the Fleiss-Cohen or Quadratic weighted Kappa coefficient.

Kappa coefficients for balanced data

When there is an equal number of rows and columns in a crosstab between score1 and score 2, as shown in Figure 2 below, you have a simple case of balanced data. In this case, SAS computes Kappa coefficients without any problems.

<table>
<thead>
<tr>
<th></th>
<th>Score1 * Score2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counts</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>264</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>278</td>
</tr>
</tbody>
</table>

Figure 2. Balanced Data Example

%Macro get_kappas(indat, outdat, var1, var2, byvar);
  ....
  proc freq data = &indat.;
    tables &var1. * &var2. / norow nocol nopercent nocum agree;
    output out = kappas_stats1 kappa wtkap;
  run;
  ....
  proc freq data = &indat.;
    tables &var1. * &var2. / norow nocol nopercent nocum agree (wt=fc);
    output out = kappas_stats2 kappa wtkap;
  run;
  ....
%Mend;

The first version of the proc freq above produces the simple and linear weighted Kappa coefficients. The second version of the proc freq above produces the simple and the quadratic weighted Kappa coefficients.

Kappa coefficients for unbalanced data

When there is an unequal number of rows and columns in a crosstab between score1 and score 2, as shown in Figure 3, proc freq doesn’t produce the agree statistics.
In the above crosstab between score1 and score2, there is a total of 8 observations where score1 equals 4, while there are 0 observations where score2 equals 4, hence resulting in a case of unbalanced data. The SAS support Web site published a solution for this by adding a dummy observation where score2 equals 4 and assigning it a weight of 0, while giving a weight of 1 for the rest of the observations. This solution is easy and simple to apply, when you know in advance where the dummy observation needs to be added to your data. When you are running your program on several items, you don’t want to manually check for all items to see if you have unbalanced data. This solution of having a weight of 0 is only possible with SAS versions 9 or later.

One other possible solution to this problem is to add a set of dummy observations for all possible cells in the crosstab to cover all bases instead of checking the crosstab each time to see how your data is unbalanced.

```
%Macro get_kappas(indat, outdat, var1, var2, byvar);
    ...
    data for_dummy (drop=i j);
        do i = 1 to 10 by 1;
            do j = 1 to 10 by 1;
                &var1 = i;
                &var2 = j;
                flag = "dummy";
                output;
            end;
        end;
    run;
    end;

    data score;
        set &indat.;
        flag = "score";
        if &byvar. = "&tmp_by_var.";
    run;
    ...
    data t_1;
        set score dummy;
    run;
    proc freq data = t_1;
        tables flag * &var1. * &var2. / norow nocol nopercent nocum agree;
        output out = kappas_stats1 kappa wtkap ;
    run;
    ...
%Mend;
```

The snippets of code above show how to create a dataset with 100 dummy observations, append the dummy data to the original dataset, and use a flag variable to distinguish scoring observations from the dummy observations. Using the flag variable in the tables statement would produce Kappa statistics for each value of flag "score" and "dummy" and across all observations. To get our desired Kappa statistics, we can just filter the output dataset for flag=score"
MODULE 3: MATCHED PAIR T-TEST
A t-test is a commonly used hypothesis test for determining differences between 2 groups. Since our data here consists of pairs of scores for each paper, a matched pair t-test can be used to test for significant differences between 2 groups. The paired statement in proc ttest produces the Matched Pair t-value and the corresponding p-value. A p-value of 0.05 or lower means that the 2 groups are significantly different at an Alpha level of 0.05.

%macro get_mptvalue(indat, outdat, var1, var2, byvar);
proc ttest data = &indat.;
paired &var1. * &var2.;
by &byvar.;
ods output ttests = t_1(keep=&byvar. tvalue probt);
run;
...
%mend;

MODULE 4: PERCENT EXACT AND ADJACENT AGREEMENTS
To better understand Percent Exact Agreement and Percent Adjacent Agreement, let’s look at the crosstabs below in Figure 4 and Figure 5.

<table>
<thead>
<tr>
<th>Counts</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>264</td>
<td>74</td>
<td>4</td>
<td>0</td>
<td>292</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>69</td>
<td>12</td>
<td>1</td>
<td>92</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>10</td>
<td>28</td>
<td>5</td>
<td>44</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>5</td>
<td>13</td>
</tr>
<tr>
<td>Total</td>
<td>283</td>
<td>104</td>
<td>48</td>
<td>11</td>
<td>441</td>
</tr>
</tbody>
</table>

Figure 4, Crosstab of score1 * score2: Counts

<table>
<thead>
<tr>
<th>Percent</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>59.9%</td>
<td>5.4%</td>
<td>0.9%</td>
<td>0.0%</td>
<td>66.2%</td>
</tr>
<tr>
<td>2</td>
<td>2.3%</td>
<td>15.6%</td>
<td>2.7%</td>
<td>0.2%</td>
<td>20.9%</td>
</tr>
<tr>
<td>3</td>
<td>0.7%</td>
<td>2.3%</td>
<td>5.9%</td>
<td>1.1%</td>
<td>10.0%</td>
</tr>
<tr>
<td>4</td>
<td>0.2%</td>
<td>0.2%</td>
<td>1.4%</td>
<td>1.1%</td>
<td>2.9%</td>
</tr>
<tr>
<td>Total</td>
<td>63.0%</td>
<td>23.6%</td>
<td>10.9%</td>
<td>2.5%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Figure 5, Crosstab of score1 * score2: Percents

Percent Exact Agreement is the sum of all percents in red on the diagonal from Figure 5. Percent Adjacent Agreement is the sum of all percents in red and green from Figure 5. To calculate these measures in SAS, get the difference between score1 and score2, and then produce a frequency distribution of the difference variable. The percent of 0 differences is the Percent Exact Agreement, and the cumulative percent of 0s and 1s is the Percent Adjacent Agreement.

%Macro get_agree_0_adj_1(indat, outdat, var1, var2, byvar);
data t_1;
   set &indat.;
ad_&var1._&var2. = abs(&var1. - &var2.);
run;
...
proc freq data = t_1;
tables ad_&var1._&var2./out = t_2 outcum;
by &byvar.;
run;
...
data t_3(rename=(CUM_PCT=agr_&var1._&var2.));
   set t_2;
   if ad_&var1._&var2. = 0;
   keep &byvar. cum_pct;
run;
...
data t_4(rename=(CUM_PCT=adj1_&var1._&var2.));
set t_2;
if ad_&var1._&var2. = 1;
keep &byvar. cum_pct;
run;
...
%Mend;

MODULE 5: PEARSON CORRELATION COEFFICIENT
Pearson Correlation Coefficient is another measure of association between score1 and score2, which can be easily produced using proc corr in SAS.

%Macro get_correl(indat, outdat, var1, var2, byvar);
proc corr data=&indat. noprint outp=t_1 vardef=n;
var &var1. &var2.;
by &byvar.;
run;
...
%Mend;

MODULE 6: GENERATE CUSTOM REPORTS
Module 6 shows how to generate custom reports with only the measures specified by the user and flag cases with reliability values lower than the threshold values set by the user. In the next section of this paper, I will demonstrate how to design a Windows form and get input from the use. For now, let's assume that user input is available in the form of macro variables.

%let variable1 = score1;
%let variable2 = score2;
...
%let prt_means_sds = Yes; /* whether or not to print in the output file */
%let prt_ckappa = Yes;
%let prt_fkappa = No;
...
%let kappa_threshold = 0.60; /* threshold values */
%let ckappa_threshold = 0.70;
%let fkappa_threshold = 0.75;
%let pea_threshold = 60;
%let paa_threshold = 75;
...

To highlight the stats with reliability lower than the threshold values set by the user, use proc format to define formats.

Proc format;
...
   Value ckap_flags low - < &ckappa_threshold. ='graybb';
   Value fkap_flags low - < &fkappa_threshold. ='graybb';
   Value pea_flags low - < &pea_threshold. ='graybb';
run;
Proc report, along with ods tagset excelxp, can be used to generate output files in XML format which can easily be saved as Excel files.

```sas
%macro gen_report(outdat, fl, fn, ttl, var1, var2, byvar);
/*&fl. and &fn. are macro variables containing the output file location and name*/
ods tagsets.Excelxp style=printer file = "&fl.&fn.";
ods tagsets.Excelxp options
(  embedded_titles='yes'
  embedded_footnotes='yes'
  sheet_name ="&var1.&var2."
  zoom = '90'
  Frozen_Headers = '5'
);
proc report data = pro_to_rep nowd split='~'
  style=[just=center] ;
...  ...
  columns
    (&byvar.)
    ("&var1. by &var2."
    ("N" N_&var1._&var2.)
    ...
  %if &prt_ckappa. = Yes %then %do;
    LWKap_&var1._&var2.
  %end;
  %if &prt_fkappa. = Yes %then %do;
    QWKap_&var1._&var2.
  %end;
  ...
  ...
  %if &prt_ckappa. = Yes %then %do;
    define LWKap_&var1._&var2. / analysis mean "Linear wtd ~ kappa" center
    style={background=ckap_flags.};
  %end;
  %if &prt_fkappa. = Yes %then %do;
    define QWKap_&var1._&var2. / analysis mean "Quad wtd ~ kappa" center
    style={background=fkap_flags.};
  %end;
  ...
  ...
run;
ods tagsets.Excelxp close;
proc datasets nolist;
delete pro_to_rep;
run; quit;
%mend;
```

%if statements, along with the appropriate macro variables, can be used to control what is being printed in the output file as shown above. Also, %if statements can be used to control which define statements get submitted.
RUNNING SAS CODE FROM VISUAL C#.NET
In this section, I will discuss how to design a Windows form using Visual C#.NET, get input from the user and pass the input to SAS, and call SAS programs from Visual C#.NET using various parameters received from the user through this Windows form.

DESIGNING A WINDOWS FORM
Open Visual Studio 2008 and create a new project. Select Visual C# and Windows application when creating this new project. You will then see the Visual Studio 2008 interface to design a Windows form, as shown in Figure 6 below.

Figure 6, Designing Windows form
On the left hand side of the screen, you will see common controls which can be dragged and dropped onto the blank Windows form. On the right hand side of the screen, you will see properties of the selected object. Start by dragging and dropping a textbox onto the form. As soon as you click on the textbox to select it, the properties windows on the right hand side will show the default properties of this textbox. You can easily rename it, change the size, add a default text, etc. by changing the properties window on the right. You can also drag and drop buttons to the form. You can assign a method to the event “button click”, by double clicking on the button. Double
clicking a button in this interface will cause Visual C#.NET to create a method and associate it with that button click.

To design a frontend (Windows form) for the rater reliability application, drag and drop a few labels, checkboxes, textboxes, and buttons to your form. Once you have these controls on your Windows form, rename them, resize them, assign default properties, and arrange them nicely on the form by changing their properties on the right hand side of the screen. Figure 7 shows some default threshold values for the flags in the form, which a user can change before clicking the submit button.

Once you double click the “Browse” button in the interface, Visual C#.NET will create a blank method in its code window. Then you just have to write the code to open and select the input dataset.

```csharp
private void browse1_Click(object sender, EventArgs e)
{
    OpenFileDialog fd1 = new OpenFileDialog();
    fd1.Title = "Select sas dataset";
    fd1.Filter = "sas datasets (*.sas7bdat)|*.sas7bdat|All files (*.*)|*.*";
    fd1.FilterIndex = 1;
    if (fd1.ShowDialog() == DialogResult.OK)
    {
        sasds_tbl.Text = fd1.FileName;
    }
}
```
The Visual C# code shown above creates a new file dialog window, asks the user to select a SAS dataset, and sets the default filter for the file dialog window to ".sas7bdat" files. Once the user selects the appropriate input file, the filename is written in the textbox shown next to the "Browse" button. Once you double click the submit button, Visual C#.NET creates a blank method in the code window. Then you can write the code to get all the input from the user and submit the SAS modules with appropriate window parameters.

GETTING USER INPUT

Figure 8 below shows a screenshot of the Windows application in action (running), where the user entered the necessary input.

![Windows application in action](image)

Figure 8, Windows application in action

The user of this application picks the SAS dataset, enters variable names, picks the statistics to be printed in the output file, changes the threshold values if needed, and clicks the submit button. All the input that the user entered in this form is gathered and read as C# string variables.

```csharp
private void submit_button_Click(object sender, EventArgs e)
{
    // reading textboxes
    string input_sas_file_path_name = sasds_tbx1.Text.ToString();
    string var1 = var1_tbx.Text.ToString();
    string var2 = var2_tbx.Text.ToString();
    ...
    // reading checkboxes input
    if (ckappa_cb.Checked == true) ckappa = "Yes";
    if (fkappa_cb.Checked == true) fkappa = "Yes";
    ...
    // reading threshold values
    string ckap_thvalue = ckap_th.Text.ToString();
```
string fkap_thvalue = fkap_th.Text.ToString();
...
...
}

Now that you have all the input from the user in the form of C# string variables, you have to pass them onto SAS and submit the SAS program.

**SUBMITTING SAS CODE FROM VISUAL C#.NET**

To work with SAS from Visual C#.NET, you need to add SAS references. To do this, open the solution explorer in Visual Studio, right click on “references” and choose “add reference”. Add the SAS Workspace Manager reference on the COM tab. Adding this reference will allow Visual C#.NET to communicate with SAS without actually opening the SAS environment in Windows.

Also add the “using statements” in C# code for convenience.

```csharp
using SAS;
using SASWorkspaceManager;
```

Let’s define a new StringBuilder “sascode” in C# and append all the sas code we wish to submit to that string.

```csharp
private void submit_button_Click(object sender, EventArgs e)
{
...
...
StringBuilder sascode = new StringBuilder();
...

sascode.Append("%let input_directory = " + input_directory + ";");
...

sascode.Append("%let variable1 = " + var1 + ";");
...

sascode.Append("%let variable2 = " + var2 + ";");
...

sascode.Append("%let ckappa = " + ckappa + ";");
...

sascode.Append("%let fkappa = " + fkappa + ";");
...

sascode.Append("%let ckappa_threshold = " + ckap_thvalue + ";");
...

sascode.Append("%let fkappa_threshold = " + fkap_thvalue + ";");
...

}
```

Now that you have all the input from the user assigned to global parameters in SAS, the next step is to submit the SAS program calling the modules with the necessary parameters. To do this, simply append the location of the SAS program calling the macros to the “sascode” with a `%inc` statement.

```csharp
private void submit_button_Click(object sender, EventArgs e)
{
...

sascode.Append("%inc 'C:\SXV6600\raterrel\eval_net.sas';");
...
```

```csharp
```
To submit the SAS code you appended to “sascode”, create a new instance of the SAS workspace and submit the “sascode” through the language service.

```java
...
WorkspaceManager wsm = new WorkspaceManager();
ILanguageService langservice = sasworkspace.LanguageService;
langservice.Submit(sascode.ToString());
...
```

Figure 7 below shows the output XML file that is created, which can easily be saved as an Excel file.

![Output file](image)

The grey cells in the above output indicate the reliability values lower than the threshold levels set by the user.

**VIEWING SAS LOG AND LISTING**

You can use MessageBoxes in Visual C#.NET to ask the user if he or she wants to view the SAS log and listing.

![View SAS Log?](image)

![View SAS Listing?](image)
If the user clicked yes, you can create a new form, then add a rich textbox to display the SAS log and/or listing.

```csharp
private void submit_button_Click(object sender, EventArgs e)
{
    if (MessageBox.Show("View SAS Log?", "View SAS Log", MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
    {
        string saslog = langservice.FlushLog(10000000).Replace("/", "\"));
        RichTextBox rtb = new RichTextBox();
        rtb.Size = new System.Drawing.Size(895, 545);
        Form f2 = new Form();
        f2.Text = "SAS Log";
        f2.ClientSize = new System.Drawing.Size(900, 550);
        f2.Controls.Add(rtb);
        rtb.Text = saslog;
        f2.Show();
    }
    //To flush sas listing use this
    if (MessageBox.Show("View SAS Listing?", "View SAS Listing", MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
    {
        string saslisting = langservice.FlushList(100000000).Replace("\", "\"));
        RichTextBox rtb2 = new RichTextBox();
        rtb2.Size = new System.Drawing.Size(895, 545);
        Form f3 = new Form();
        f3.Text = "SAS Listing";
        f3.ClientSize = new System.Drawing.Size(900, 550);
        f3.Controls.Add(rtb2);
        rtb2.Text = saslisting;
        f3.Show();
    }

    //...}

    //Figure 10, Viewing SAS Log and Listing
```
CONCLUSION
Applying the same principle used in this paper, SAS and Visual C#.NET can be used to build Windows and Web applications for doing data analysis and producing custom reports. SAS is a great tool for doing data analysis, and Visual C#.NET provides a quick and easy way to build Windows and Web interfaces to interact with the user.

This paper presents only snippets of the code. If you would like to view the complete code, please feel free to contact the author using the contact information listed below.

REFERENCES

ACKNOWLEDGEMENTS
I would like to thank my colleagues and friends at ETS for supporting this paper and my professional development and my beautiful wife Meg for her endless support in all I do and her help with editing this paper.

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

Sailesh Vezzu
Educational Testing Service
Rosedale Road, MS 20T
Princeton, NJ 08541
(609) 734 -5786
svezzu@ets.org

TRADEMARKS
SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® Indicates USA registration.