MWSUG 2016 – Paper AA17

Using SAS® to Generate p-Values with Monte Carlo Simulation

Brandy R. Sinco, MS, University of Michigan, Ann Arbor, MI
Edith Kieffer, PhD, University of Michigan, Ann Arbor, MI
Michael S. Spencer, PhD, University of Michigan, Ann Arbor, MI
Michael Woodford, PhD, Wilfrid Laurier University, Ontario, Canada
Gloria Palmisano, BS, MA, CHASS Center, Detroit, MI
Gretchen Piatt, PhD, University of Michigan, Ann Arbor, MI
Michele Heisler, M.D., University of Michigan, Ann Arbor, MI

ABSTRACT

Background. When multiple comparisons are made between groups, the type-1 error rate, typically .05, becomes inflated. The statistical literature indicates that Monte Carlo simulation, using bootstrap and permutation tests, is an effective method to address the inflated error risk from multiple comparisons.

Objective. Discuss the theory behind bootstrap and permutation tests, and demonstrate how to generate bootstrapped p-values for commonly used analysis methods. Techniques to be covered include comparisons of counts and percentages, t-tests for means, linear regression, logistic regression, and linear mixed models.

SAS Procedure Focus. For comparisons between counts and percentages, PROC FREQ has user-friendly features to adjust for Monte Carlo simulation. Although PROC TTEST does not have multiple comparison adjustment, PROC MULTTEST is an excellent alternative. For logistic regression, PROC LOGISTIC has a Monte Carlo simulation option on both the Estimate and LSMEANS statements.

If linear regression is done with PROC GLM, the LSMEANS, but not the Estimate statement, contains a Monte Carlo option. Similarly, PROC MIXED, for linear mixed models, offers Monte Carlo adjustment only on the LSMEANS statement. However, SAS procedures for linear models that do not offer Monte Carlo adjustment on the Estimate statement, can easily have their output routed to PROC PLM for Monte Carlo adjustment of estimates.

Examples. Bootstrapped p-values will be demonstrated with SAS code from analyses of data from a study of LGBTQ college students by multiple race and gender categories, and from a diabetes study with three treatment groups across four time points.

Outline

1)_Why Multiple Comparisons Increase the Type 1 Error Rate	Page 2
2)_Remedies in the Statistical Literature	Pages 2 - 3
3)_Brief Overview of Monte Carlo Simulation, Bootstrapping, and Permutation Tests	Pages 3 - 6
4)_Bootstrapping with PROC SURVEYSELECT	Pages 6 - 7
5)_PROC FREQ - Percentages and Counts.	Pages 7 - 8
6)_PROC MULTTEST for T-Tests	Pages 8 - 9
7) PROC NPAR1WAY for Wilcoxon or Kruskal-Wallis Tests	Pages 9 - 10
8)_PROCS LOGISTIC and GENMOD for Binary, Count, and Categorical Regressions	Page 10
9)_PROC GLM and PLM for Linear Regression	Page 11
10)_PROCS MIXED, GLIMMIX, and PLM for Linear Mixed Models	Pages 11 - 12
11)_Conclusions	Page 12
12)_References	Page 13
13)_Contact Information	Page 14

Why Multiple Comparisons Increase the Type 1 Error Rate.

- Let α = probability of a type 1 error, i.e., probability of rejecting the null hypothesis when the null hypothesis is true. I.E., concluding that the test statistic indicates a significant result, such as a difference between two means, when the result is really insignificant.
- Let α' = probability of at least one type 1 error on m tests.
- Example of Type 1 Error: Concluding that there is a significant difference between groups 1 and 2, when no difference exists.
- If $\alpha = .05$ and two comparisons are made, such as group 1 to control and group 2 to control, the probability of not making a type 1 error on either test = $(1 - \alpha)^2 = (1 - .05)^2 = .9025$. The probability of making a type 1 error on at least one of the two tests = $1 - (1 - \alpha)^2 = .0975$.
- If m tests are done, the type 1 error, $\alpha' = 1 - (1 - \alpha)^m$.
- For $\alpha = .05$ and $m = 10$, $1 - (1 - \alpha)^m = 0.5987$. α' = probability of at least one type one error, increase from .05 to .5987.
- Multiple comparisons increase α'. Goal of adjusting for multiple comparisons is to set $\alpha' = \alpha$.

Remedies for Multiple Comparisons in the Statistical Literature.

Three remedies for the inflated α from multiple comparisons are: to use a smaller α, use tests that have a family-wise type 1 error of α for m tests, or to use Monte Carlo simulation.

One remedy proposed in the statistical literature is to use a smaller α, α_m, for each of the m hypothesis tests, so that overall type-1 error threshold for the m tests will be α. For example, the Bonferroni[1] method proposes that $\alpha_m = \alpha/m$. The Šidák[2] correction proposes that $\alpha_m = 1 - (1 - \alpha)^{1/m}$. So, if the original $\alpha = .05$ and m = 10 comparisons, $\alpha_m = .005$ with the Bonferroni adjustment and .005116 with the Šidák correction.

The problem with these approaches is that α will become extremely small with large m, and the investigator may end up with a type 2 error, which is concluding that an effect is non-signficant when the effect really is significant. One example would be concluding that a blood pressure medication does not lower blood pressure, when it really does. Simulation studies indicate that the Bonferroni and Šidák corrections are overly conservative[3].

The second remedy is to use alternative test statistics to the Student T and Fisher F that have a family-wise type 1 error rate of α for m tests. For example, Tukey's Procedure compares combinations of pairs of m means, 2 at a time, by using Tukey's Q statistic instead of the Student T. The total number of comparisons allowed under Tukey's procedure is m(m-1)/2. Scheffe's Procedure can be used to compare pairs of means or linear contrasts of multiple sample means, and uses a modified version of the F statistic. Tukey's procedure is better for comparing multiple differences between means. Whereas, Scheffe's procedure is better for linear contrasts of multiple sample means. Further, Dunnett's procedure is used to compare treatment means to a control, and uses the D statistic instead of the T statistic, and takes into account that (m - 1) comparisons are being made[4].

Many SAS procedures, such as GLM, LOGISTIC, GENMOD, MIXED, and GLIMMIX allow for multiple comparison adjustment of p-values on the LSMeans statement with the "Adjust=" option.

- ADJUST=BON for Bonferroni.
- ADJUST=DUNNETT for Dunnett.
- ADJUST=SCHEFFE for Scheffe
- ADJUST=SIDAK for Sidak.
- ADJUST=TUKEY for Tukey.

The third remedy is to use Monte Carlo simulation, which generates the p-values by using re-sampling procedures. While all three remedies are available in SAS, Monte Carlo simulation is the most reliable and efficient method[5]. In SAS, Monte Carlo simulation can be used to adjust p-values for multiple comparisons, by either using bootstrapping or by using permutation tests, in many procedures by using the ADJUST=SIMULATE option. The different approaches for various SAS procedures will be discussed in the sections below on specific procedures.

Brief Overview of Monte Carlo Simulation, Bootstrapping, and Permutation Tests

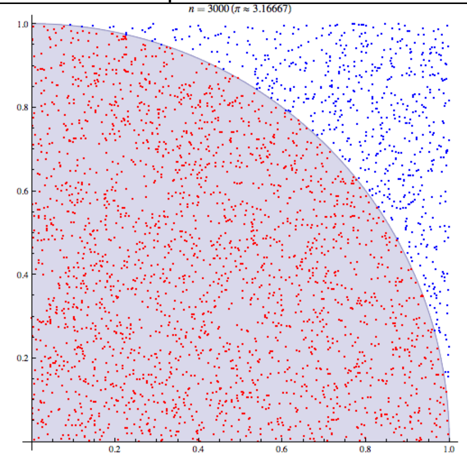
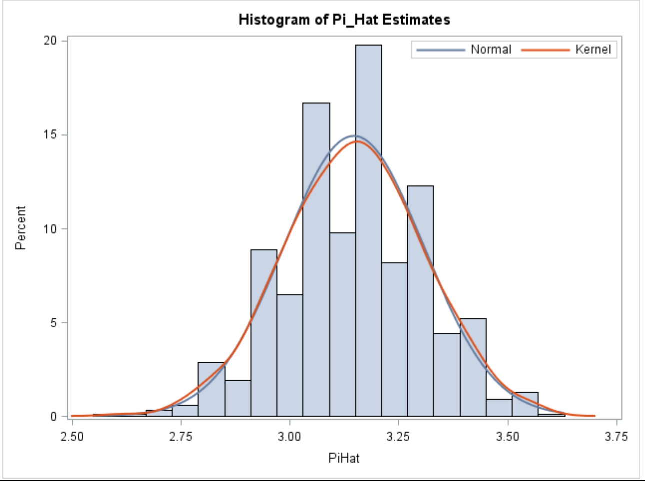
Monte Carlo Simulation involves taking many random samples from a population to estimate a parameter or a function of random variables[6]. In theory, the grand mean from the collection of random samples will converge to the parameter estimate. I will illustrate what Monte Carlo simulation is by examples rather than by complex equations.

For the first example, consider a coin toss.

- To simulate the probability of the coin producing a head, we could draw one random sample from a uniform [0, 1] pseudo-random uniform variable. A value ≥ 0.50 would indicate a head and a value < .5 would indicate a tail. This would be a simulation, but not a Monte Carlo simulation.
- To make this a Monte Carlo simulation, draw a large number of pseudo-random samples from the interval [0,1]. Then, estimate the probability of head by taking the mean of the probabilities of rolling a head from all of the samples.

For the second example, consider estimating the value of π from a circle of radius R and a square with each side of length R. Let C = area of the circle = πR^2 and S = area of the square = $4R^2$.
Then, π = 4C/S, where C/(4S) is the probability of a sample point being inside a circle.

Figure 1: Monte Carlo Simulation to Estimate π

Circle and Square	**Histogram of Pi Estimates**
	

Based on this Monte Carlo Simulation, the mean of InsideCircle = 0.7869 and π = 4 × 0.7869 = 3.1475.

SAS Program to Estimate π From 1000 Pseudo-Random Samples of 100

```
Data Circle;
Do B=1 to 1000;
      Do I=1 to 100;
            X=RanUni(8242016);
            Y=RanUni(8242016);
            R=Sqrt((X**2)+(Y**2));
            If R<=1 then InsideCircle=1;
            else InsideCircle=0;
            Output;
      End;
End;
Run;

PROC MEANS data=circle;
var InsideCircle;
Run;
/* Histogram of Pi Estimates */
PROC MEANS data=circle noprint;
Class B;
var InsideCircle;
Output out=circleby mean=PiQuarterHat;
run;

/* Delete first row because it contains grand mean */
data circleby;
set circleby;
if _N_=1 then delete;
PiHat=4*PiQuarterHat;
run;

Title 'Histogram of Pi_Hat Estimates';
PROC SGPLOT data=CircleBy;
 histogram PiHat;
 density PiHat;
 density PiHat / type=kernel;
  keylegend / location=inside
  position=topright;
run;
```

How Repeated Random Sampling Got the Name Monte Carlo Simulation

The modern version of the Monte Carlo simulation, known as the Markov Chain Monte Carlo method,was invented in the late 1940s by Stanislaw Ulam, while working on nuclear weapons projects at the Los Alamos National Laboratory. Ulam recalled his inspiration for multiple random experiments when he was recovering from an illness and experimented with estimating the probability that a rare solitaire hand would occur, both by combinatorics calculations and by observing the number of successful plays. Then, he thought about extending the concept of “count of successful plays” to nuclear physics problems, and worked with John von Neumann to develop the algorithm. Due to the top secret nature of his work, this new method needed a “code” name. Nicholas Metropolis, a colleague of Ulam and Neumann, suggested using the name “Monte Carlo” after a casino where his uncle liked to gamble.

Bootstrapping and Permutation Tests.

Bootstrapping is used to estimate a confidence interval for a parameter estimate, such as a regresssion coefficient or a percentage or a mean. Whereas, the purpose of permutation tests is to test differences between groups.

Let's focus on bootstrapping first. Bootstrapping was invented by Bradley Efron and is a method for estimating the empirical distribution and confidence interval for a test statistic[7]. Suppose we have a random sample of x, size n, with an empirical density function, f. $x_1, x_2, ..., x_n \sim f(x)$. $x_1, x_2, ..., x_n$ are assumed to be independent and identically distributed. Let T(x) be a test statistic for which we want to find a (1 - α) confidence interval.

To estimate the empirical distribution of T(x) by bootstrapping, we take B independent random samples with replacement of size n, and calculate T(x) for each replication.
The steps for bootstrapping are as follows:

1. Take a random sample of size n with replacement from the sample set.
2. Calculate the statistic of interest, T(x), for the random sample.
3. Repeat steps 1 and 2 B times (B is usually >=10,000).
4. Create a relative frequency histogram of the B statistics of interest, T(x), by placing a probability of 1/B at each estimated statistic.

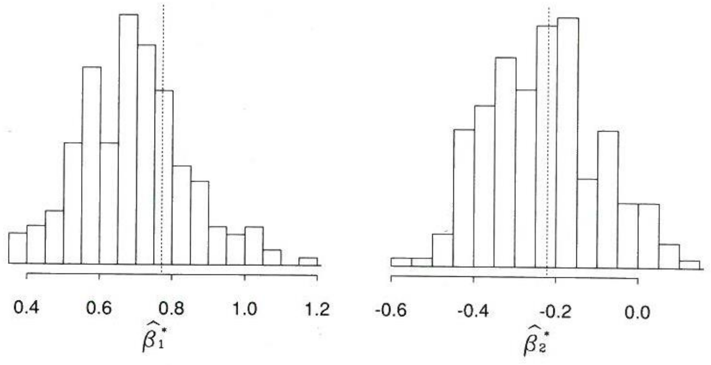
Point Estimate of T(x), $\hat{T}_B = \frac{1}{B}\sum_{b=1}^{B} T\left(x^*_b\right)$, where x^*_b = bth bootstrap sample from $x_1, x_2, ..., x_n$.

Standard error of T(x), $\hat{se}_B = \hat{T}_B = \sqrt{\frac{1}{(B-1)}\sum_{b=1}^{B}\left(T\left(x^*_b\right) - \hat{T}_B\right)^2}$

Then, estimate (1 - α)*100% confidence interval for T(x) from the (α/2) and (1 - α/2) percentiles of the bootstrap density function.

Bootstrapping to estimate confidence intervals of statistics that are functions of more than one random variable, x, such as a correlation coefficient, regression coefficient, and even effect size[8]. At the 2015 Conference on Statistical Practice, Erin Smith and Jason Osborne co-authored a presentation on bootstrapping confidence intervals for effect sizes[9].

Figure 2: Histograms of Bootstrap Distributions for Regression Coefficient Estimates from Efron & Tibshirani (1993)



Permutation tests[8].

Let x be a random sample of size n, with n_1 in group 1 and n_2 in group 2. Let T(x) be a test statistic, such as a T for a mean test or a X^2 for a percentage.   Then, randomly divide x into groups of n_1 and n_2. Recompute the test statist, $T_p(x)$, for each combination of x randomly subsetted into sizes of n_1 and n_2. The total number of subsets will be $\frac{n!}{n_1!n_2!}$.

Let M = number of trials for which $|T_p(x)| \geq |T(x)|$.
Let N = $n!/(n_1!n_2!)$.
Then, the p-value from the permutation test, also known as an “exact test” = M/N.

If the dataset is too large to calculate the test statistic for N subsets, Monte Carlo sampling will be used.

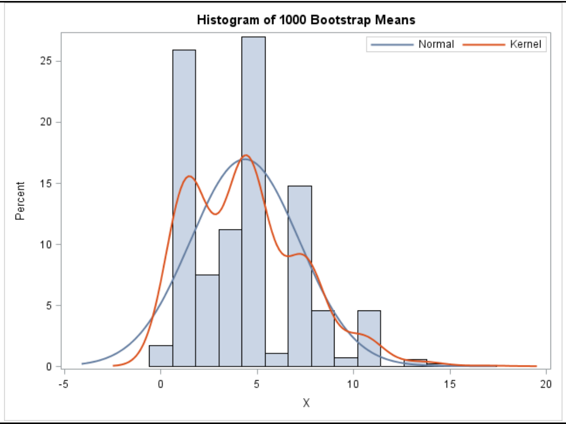
Bootstrapping with PROC SURVEYSELECT

For a simple bootstrapping example, let’s bootstrap a sample mean and generate a 95% confidence interval.

```
/* Create original dataset */
Data VerizonSubset;
Input X @@;
Datalines;
3.12 0.00 1.57 19.67 0.22 2.20
;
Run;

/* Create 1000 bootstrap samples, sample with replication */
PROC SURVEYSELECT data=VerizonSubset Seed=7252016 method=urs n=6 reps=1000
out=bootsamp1000 outhits;
run;

Title 'Bootstrap Means 1000';
ods html path="c:\temp";
PROC MEANS data=bootsamp1000;
Class Replicate;
var  X;
Output Out=BootMeans1000;
run;
ods html close;

/* Delete the first observation, because it’s the aggregate mean */
Data BootMeans1000;
Set BootMeans1000;
Keep Replicate _STAT_ X;
if Replicate NE . and  _STAT_='MEAN';
Run;

Title 'Bootstrap Means 1000';
PROC MEANS data=bootmeans1000;
var X;
run;
```

Figure 3: Bootstrapped Means Histogram and SAS Code

```
Title 'Histogram of 1000 Bootstrap
Means';
ods html path="c:\temp";
PROC SGPLOT data=BootMeans1000;
    histogram X;
    density X;
    density X / type=kernel;
     keylegend / location=inside
     position=topright;
run;
```



PROC FREQ - Percentages and Counts[10,11].

In PROC FREQ, p-values with Monte Carlo simulation can be created by using the exact statement with the MC option. The exact statement tells SAS to use a permutation test, which also known as the Fisher Exact Test and is based on the hypergeometric distribution. When the MC option is included on the Exact statement, SAS computes Monte Carlo estimates regardless of the sample size. PROC FREQ will not do any Monte Carlo simulations unless the MC option is included.

According to the PROC FREQ documentation, the following equations are used to compute the Monte Carlo p-value.

- t = observed test statistic
- M = number of samples with Test Statistic >= t
- N = total number of samples
- P_{MC} = p-value adjusted with Monte Carlo simulation = M/N.
- se(P_{MC}) =standard error of the Monte Carlo p-value = sqrt(P_{MC}(1-P_{MC})/(N-1))

For an example, I will use PROC FREQ to compare depression, suicide, and demographics between cisgendered and transgendered college students. Cisgender refers to a person having the gender identity that was assigned at birth and transgendered refers to seeing one's gender identity in a different way. In the PROC FREQ code below, Monte Carlo simulation is invoked by the MC option.

```
PROC FREQ Data=intersect;
  Tables (PHQ3Cat Suicide_NY Audit_BRS NHW disability_diagnosis_NY
SES_pell_NY conference_attend) * gender_trans / ChiSq Exact
 NoCum;
  Exact Fisher / MC seed=03202015;
Run;
```

The SAS output will display the p-value from Fisher's exact test, in addition to p-value with Monte Carlo adjustment. For the depression outcome, the p-value from Fisher's exact test without Monte Carlo simulation is p < .0001. With Monte Carlo simulation, p = .0018. Tables like the one below will be displayed for each variable in the tables statement.

Monte Carlo Estimate for the Exact Test	
Pr <= P	0.0018
99% Lower Conf Limit	0.0007
99% Upper Conf Limit	0.0029
Number of Samples	10000
Initial Seed	3202015

PROC MULTTEST for T-Tests[12]

Use PROC MULTTEST to do a T-Test with Monte Carlo adjustment because PROC TTEST does not have this option. In PROC MULTTEST, using bootstrap sampling within-strata is a more time efficient method of doing t-tests, rather than doing permutation tests. The Monte Carlo p-value will be based on bootstrap samples within the groups to be tested.

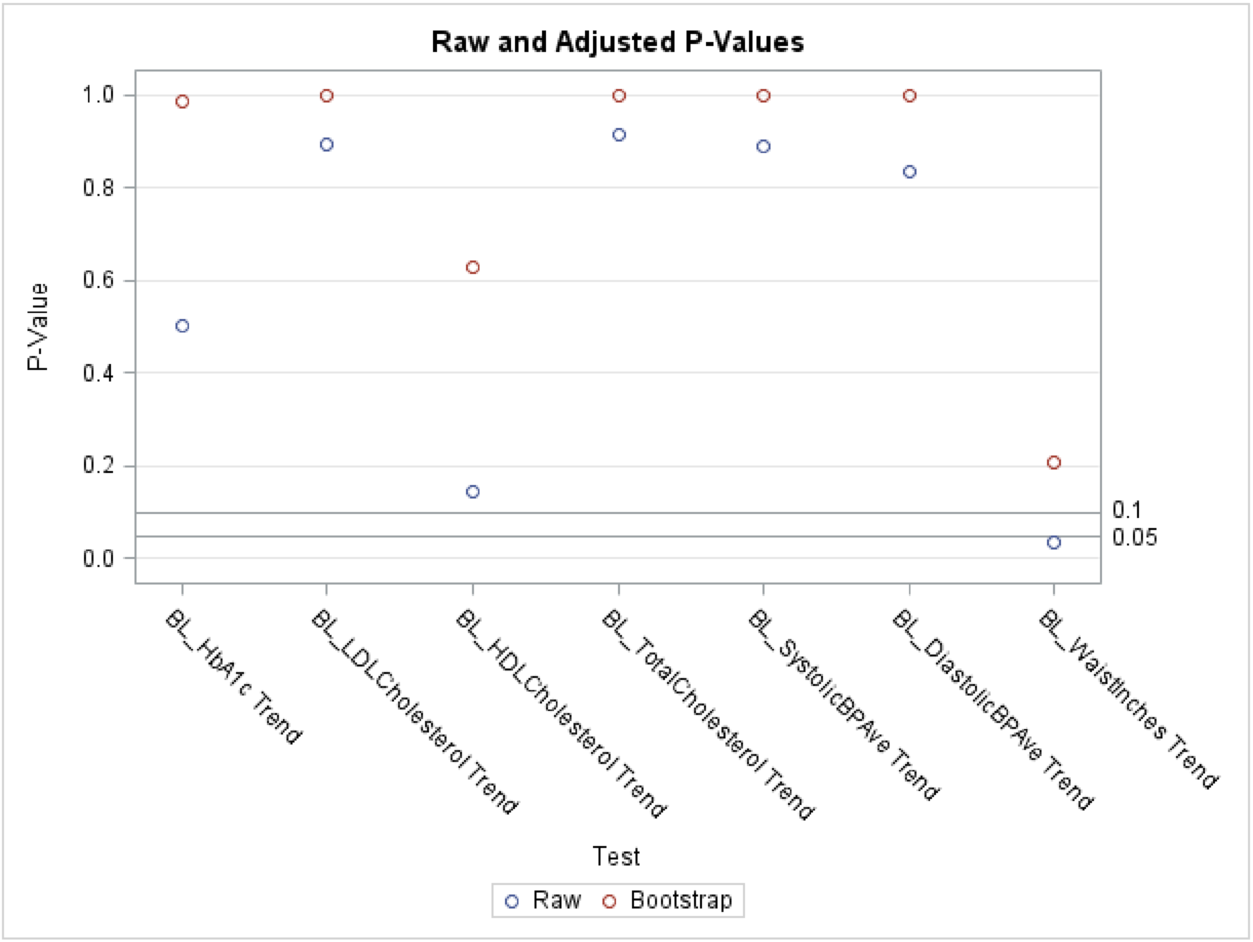
The PByTest plot is highly recommended and illustrates the effect of multiple comparison adjustment. The example below is from a diabetes study.

```
PROC MULTTEST data=across bootstrap nsample=10000 seed=4102015 outsamp=res
plots=PByTest(vref=0.05 0.1);
 test mean(BL_HbA1c BL_LDLCholesterol BL_HDLCholesterol BL_TotalCholesterol BL_SystolicBPAve
BL_DiastolicBPAve BL_WaistInches);
 class RandomizationBin;
run;
```

The output from PROC MULTTEST will include both the raw and bootstrapped p-values. The plot and table indicate that the p-value for waist circumference (BL_WaistInches) is significant without Monte Carlo adjustment, but loses significance after adjustment.

p-Values			
Variable	Contrast	Raw	Bootstrap
BL_HbA1c	Trend	0.5044	0.9877
BL_LDLCholesterol	Trend	0.8950	1.0000
BL_HDLCholesterol	Trend	0.1454	0.6284
BL_TotalCholesterol	Trend	0.9158	1.0000
BL_SystolicBPAve	Trend	0.8921	1.0000
BL_DiastolicBPAve	Trend	0.8367	1.0000
BL_WaistInches	**Trend**	**0.0358**	**0.2086**

Figure 3: Raw and Adjusted P-Values from PROC MULTTEST



PROC NPar1Way for Wilcoxon or Kruskal-Wallis Tests

The Wilcoxon test is used to compare non-normal, but ordinal, data between two groups. The Kruskal-Wallis test is an extension of the Wilcoxon test for more than two groups. In SAS, both tests are implemented in PROC NPar1Way with the same syntax. If there more than two groups, SAS will perform a Kruskal-Wallis test with the Wilcoxon statement. The syntax is similar to PROC FREQ and the equations for p_{MC} are the same as for PROC FREQ.

In this example, I am comparing class attendance from two treatment groups and am excluding the control group, indicated by Randomization=0 because they didn't have access to the classes. The p-value without Monte Carlo simulation is .4673 and with Monte Carlo simulation is .2350. So. Adjusting with Monte Carlo simulation does not always raise the p-value.

```
PROC NPAR1WAY wilcoxon data=fromacc.mainvar_cohort3;
 class Randomization;
 var TotalClasses;
 exact wilcoxon/ MC n=100000 seed=102752011;
where Randomization NE 0;
run;
```

Monte Carlo Estimates for the Exact Test	
One-Sided Pr >= S	
Estimate	0.2350
99% Lower Conf Limit	0.2315
99% Upper Conf Limit	0.2385

PROCS LOGISTIC and GENMOD for Binary, Count, and Categorical Regressions.

Of all of the SAS linear models procedures that I use, PROC LOGISTIC has the most user friendly features for generating Monte Carlo p-values for estimates. All that is needed is the “adjust=simulate” option on the “Estimate” statement. By using a seed, the estimate statements will always produce the same values.
While NSAMP=10000 could be added to the estimate statement, the SAS default is 12,604. SAS produces the confidence interval for the estimate by bootstrapping[13].

In the example below, I am estimating the odds of depression among transgendered college students. The model indicated that outness, i.e. feeling safe disclosing one’s identity significantly reduced the odds of depression. In this case and on many estimate statements, I have seen similar values for the raw p-values and the p-values with Monte Carlo simulation.

```
PROC LOGISTIC Descending Data=Intersect;
Class SoCat4_BRSN/param=ref;
Model PHQ3Cat=age  SoCat4_BRSN NHW
disability_diagnosis_NY SES_pell_NY
social_support stdsupportNNA_AK6
pride1r_cen pride3r_cen LGBTQ_outness_cen
pride1r_cen*NHW pride3r_cen*NHW  LGBTQ_outness_cen*NHW
LGBQHostility LGBTQ_interpersonal_micro conference_attend/iplots link=cumlogit aggregate
CLOdds=Both;
Estimate 'Salience POC' pride1r_cen 1 /adjust=simulate seed=2052015;
Estimate 'Salience NHW' pride1r_cen 1 pride1r_cen*NHW 1 /adjust=simulate seed=2052015;
Estimate 'Disturb POC' pride3r_cen 1 /adjust=simulate seed=2052015;
Estimate 'Disturb NHW' pride3r_cen 1 pride3r_cen*NHW 1 /adjust=simulate seed=2052015;
Estimate 'Outness POC' LGBTQ_outness_cen 1 /adjust=simulate seed=2052015;
Estimate 'Outness NHW' LGBTQ_outness_cen 1 LGBTQ_outness_cen*NHW 1 /adjust=simulate
seed=2052015;
where gender_trans=1;
run;
```

Estimate Adjustment for Multiplicity: Simulated						
Label	PHQ - 3 categories	Estimate	Standard Error	z Value	Pr > |z|	Adj P
Outness POC	2	-1.4649	0.4738	-3.09	0.0020	0.0017

In contrast, PROC GenMod does not have an “adjust=” option on the estimate statement, but does have this option on the LSMEANS and LSMEstimate statement. To obtain Monte Carlo p-values from the estimate statement, the output needs to be processed through PROC PLM, which I will demonstrate with PROC MIXED.

PROCS GLM and PLM for Linear Regression.

For linear regression, PROC REG does not have an “Estimate” statement. Although, PROC GLM does have an estimate statement, only LSMEANS and LSMEstimate have the “adjust=” option. Like PROC GENMOD, the only way to currently get Monte Carlo p-values for linear regression estimates is to route the results from the “estimate” statement in PROC GLM to PROC PLM. In the next section on PROC MIXED, the syntax for using PROC PLM will be displayed and the same syntax can be used with PROCS GLM and GENMOD.

Procs Mixed, GLIMMIX, and PLM for Linear Mixed Models.

Like PROCS GENMOD and GLM, PROC MIXED has an “adjust=” option on the LSMeans and LSMEstimate statements, but not on the Estimate statement. For PROC MIXED, the solution is to use PROC PLM, which offers post-processing options for many SAS linear models procedures.

Let’s begin with the longitudinal model for the change in HbA1c (hemoglobin A1c blood sugar measurement) over time. In the model below, I am estimating the mean change in HbA1c for three different treatment groups, indicated by RandomizationN, with covariates of baseline value and medication intensification, indicated by MoreDiabMeds. The most important detail of this SAS code is the “Store MixRes” statement, which stores the PROC MIXED processed data for PROC PLM.

```
PROC MIXED Data= A_Long Method=REML NOCLPRINT
plots(only)=(StudentPanel(conditional box));

Class REACHID TimepointN RandomizationN;

Model HbA1c=BL_HbA1c*RandomizationN TimepointN RandomizationN
TimepointN*RandomizationN HSGrad MoreDiabMeds / Solution
Influence(effect=REACHID Est) ddfm=KR;
Repeated TimepointN / Type= CS Subject=REACHID R RCorr;
Estimate 'Ref M6 - BL'  Intercept 1  RandomizationN 0 0 1 TimePointN 0 0 1
TimepointN*RandomizationN 0 0 0 0 0 0 0 0 1 BL_HbA1c*RandomizationN 0 0
&BLRef
  MoreDiabMeds .3588 HSGrad .306;

Estimate 'Trt1 M6 - BL' Intercept 1  RandomizationN 0 1 0 TimePointN 0 0 1
TimepointN*RandomizationN 0 0 0 0 0 0 0 1 0 BL_HbA1c*RandomizationN 0 &BLGrp1
0
  MoreDiabMeds .3588 HSGrad .306;

Estimate 'Trt2 M6 - BL' Intercept 1  RandomizationN 1 0 0 TimePointN 0 0 1
TimepointN*RandomizationN 0 0 0 0 0 0 1 0 0 BL_HbA1c*RandomizationN &BLGrp2 0
0
  MoreDiabMeds .3588 HSGrad .306;
Store MixRes;
run;
```

Then, I re-write the estimate statements in PROC PLM with the “adjust=simulate” option and PROC PLM produces the estimates with p-values adjusted for Monte Carlo simulation. Last, route the PLM output to PROC Print to display.

```
PROC PLM Restore=MixRes;
Estimate 'Ref M6 - BL'  Intercept 1  RandomizationBinN 0 1 TimePointN 0 0 1
TimepointN*RandomizationBinN 0 0 0 0 0 1 BL_HbA1c*RandomizationBinN 0 &BLRef
  MoreDiabMeds .3588 HSGrad .306 / adjust=simulate(NSAMP=10000 SEED=3282016);
/* HbA1c */
```

```
Estimate 'Trt M6 - BL'  Intercept 1  RandomizationBinN 1 0 TimePointN 0 0 1
TimepointN*RandomizationBinN 0 0 0 0 1 0 BL_HbA1c*RandomizationBinN &BLGrp 0
  MoreDiabMeds .3588 HSGrad .306 / adjust=simulate(NSAMP=10000 SEED=3282016);

ODS Output Estimates=PLMEst;
Run;
```

The output from PROC PLM will produce the following table, indicating the p-values for the HbA1c difference estimates between time points and treatment group are not being distorted by multiple comparisons.

Label	Estimate	StdErr	Probt	Adjustment	Adjusted p
Ref M6 - BL	-0.103	0.1533	0.5025	Simulated	0.4951
Trt1 M6 - BL	**-0.38**	**0.1377**	**0.0065**	**Simulated**	**0.0077**
Trt2 M6 - BL	-0.789	0.175	<.0001	Simulated	<.0001
Trt2-Trt1 M6-BL	-0.408	0.2226	0.0685	Simulated	0.0659
AveTrt1,2- RefM6-BL	**-0.471**	**0.1899**	**0.0142**	**Simulated**	**0.0146**
Int Effect1 M6 - BL	-0.277	0.2067	0.182	Simulated	0.1897
Int Effect2 M6 - BL	**-0.686**	**0.2326**	**0.0037**	**Simulated**	**0.0039**

According to the online SAS documentation, I was surprised to learn that PROC GLIMMIX does have an “Adjust=” option on the estimate statement. So, PROC PLM is not needed for Monte Carlo simulated p-values when using PROC GLIMMIX. According to SAS tech support, PROC GLIMMIX has the “adjust = simulate” feature on the “estimate” statement because it’s a more recent SAS procedure. Although the developers might add the “adjust = “ option to the “estimate” statement in PROC MIXED in the future, the current work-around is to use store statement in Proc Mixed and then use Proc PLM to get the p-values for the estimates with Monte Carlo simulation.

Conclusions.

- ✓ Monte Carlo simulation is an effective method of adjusting for multiple comparisons and is available in SAS for comparisons of percentages and means, the Wilcoxon and Kruskal-Wallis tests, linear and logistic regression, and mixed models.
- ✓ When comparing percentages, use PROC FREQ with the Exact statement and the MC option for Monte Carlo simulation.
- ✓ For the T-Test, use PROC MULTTEST with the “test mean” option and remember the “plots=PByTest” option on the PROC MULTTEST statement.
- ✓ For linear regression, use PROC GLM and then PROC PLM for post-processing of estimates.
- ✓ For linear mixed models, use PROC MIXED, followed by PROC PLM.
- ✓ For generalized linear mixed models with binary or categorical outcomes, PROC GLIMMIX already has an “Adjust=Simulate” option on the estimate statement.

REFERENCES

1. Bonferroni CE. Teoria statistica delle classi e calcolo delle probabilità. *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commerciali di Firenze*. 1936.

2. Šidák ZK. Rectangular confidence regions for the means of multivariate normal distributions): 626–633. *JASA*. 1967;62(318):626-633.

3. Hsu JC, Nelson B. Multiple comparisons in the general linear model. *Journal of Computational and Graphical Statistics*. 1998;7(1):23-41.

4. Kutner MH. *Applied linear statistical models.* Boston: McGraw-Hill Irwin; 2005:xxviii, 1396 p.

5. Edwards D, Berry JJ. The efficiency of simulation-based multiple comparisons. *Biometrics*. 1987;43:913-928.

6. Sawilowsky SS. Target article: You think you've got trivials? *Journal of modern applied statistical methods*. 2003;2(1):218-225.

7. Efron B. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*. 1979;7(1):1-26.

8. Efron B, Tibshirani R. *An introduction to the bootstrap.* New York: Chapman & Hall; 1993:xvi, 436 p.

9. Smith E, Osborne JW. Bootstrapping confidence intervals for effect sizes (and other weird things). . 2015;Conference on Statistical Practice, New Orleans.

10. Agresti A. A survey of exact inference for contingency tables. *Statistical Science*. 1992;7(1):131-153.

11. Mehta CR, Patel NR. A network algorithm for performing fisher's exact test in r × c contingency tables. *Journal of the American Statistical Association*. 1983;78(382):427-434.

12. Westfall PH, Young SS. *Resampling-based multiple testing: Examples and methods for P-value adjustment.* New York: Wiley; 1993:xvii, 340 p.

13. Mehta CR, Patel NR, Senchaudhuri P. Efficient monte carlo methods for conditional logistic regression. *Journal of the American Statistical Association*. 2000;95(449):99-108.

CONTACT INFORMATION

Your comments and questions are welcome.

Brandy R. Sinco, Statistician and Programmer/Analyst
University of Michigan School of Social Work
1080 S. University St.
Box 183
Ann Arbor, MI 48109-1106

Phone: 734-763-7784

E-Mail: brsinco@umich.edu