

Things to get out of conferences/workshops/minicourses

I started going to conferences outside of NM in my last 18 months of graduate school, and didn't have a very good idea of what to get out of them or why they are important.

My understanding of why they are valuable and what you can try to get out of them has evolved over the last 10 years of going. Here are some things that are important about them, especially for academics.

Things to get out of conferences/workshops/minicourses

First, the more obvious things...

1. Share your knowledge. There's a saying: "If a tree falls in the forest and no one is around to hear it, does it make a sound?" Research is a bit like this. If you figure out something new, but don't share this new knowledge, what difference does it make?
2. Learn new information. Of course you can learn specific information from specific talks, and it is good to try to learn what is going on in your area. More specifically, you can
 - ▶ learn what is cutting edge in a particular area
 - ▶ learn about new areas and see if they appeal to you
 - ▶ learn about new trends in statistics (what the heck is Big Data?)
 - ▶ even if you can't follow the details of a talk, you might learn about new areas, or learn about computer programs or packages that you weren't aware of.

Things to get out of conferences/workshops/minicourses

Less obvious things

1. Conferences are a place to network. If you are interested in applying to a postdoc at a certain school, you could talk to a conference presenter from that school and ask what it is like. Although not very common, it is possible to pick up new collaborators by talking to presenters after their talk. This has happened to me at 4 conferences/events in 10 years (maybe 20-30 conferences), including last week. Although it doesn't happen very often, it is very important when it does happen, especially for academics.
2. Although this might be a little bit rare, someone in the audience might be on a search committee for a job you are or will be applying to (this happened to me for my New Zealand job).
3. You might get an idea for a new research project. You might see a talk and think, "that's not how I would approach this problem..." Rather than just be critical of the presenter, you might use it as an opportunity to try a new method on your own, and it can become a research project. Or it just might lead you to think of new approaches. *So and so applied bootstrapping to their problem. What if I apply bootstrapping to my problem?...*

Restructuring the data with PROC TRANSPOSE

Another way of changing data from multiple observations per patient to one observation per patient and vice versa is through PROC TRANSPOSE.

PROC TRANSPOSE

```
/* transposing the original data */  
/* I assume data is sorted by id */  
title "Using PROC TRANSPOSE";  
proc transpose data=bp out=bp_Transpose1;  
  by id;  
  var bp1-bp4;  
run;
```

Using PROC TRANSPOSE

| | Obs | id | _NAME_ | COL1 |
|--------------------------------|-----|------|--------|------|
| proc print data=bp_Transpose1; | 1 | 0001 | bp1 | 130 |
| run; | 2 | 0001 | bp2 | 138 |
| | 3 | 0001 | bp3 | 140 |
| | 4 | 0001 | bp4 | 136 |
| | 5 | 0002 | bp1 | 150 |
| | 6 | 0002 | bp2 | 145 |
| | 7 | 0002 | bp3 | 144 |
| | 8 | 0002 | bp4 | 142 |
| | 9 | 0003 | bp1 | 121 |
| | 10 | 0003 | bp2 | 135 |
| | 11 | 0003 | bp3 | 122 |
| | 12 | 0003 | bp4 | 140 |
| | 13 | 0004 | bp1 | 118 |
| | 14 | 0004 | bp2 | 115 |
| | 15 | 0004 | bp3 | 126 |
| | 16 | 0004 | bp4 | 120 |

PROC TRANSPOSE

The idea is that for each ID variable, the columns become rows and the rows become columns. For ID 0001, the data was a row which was 1×4 , so now for ID 0001, the data is 4×1 .

Note that the age variable was lost in the process, so we'll have to do something to get it back.

The variable names have now become a column of data. Usually, you wouldn't want to keep this (although sometimes you might want to), and instead you would want COL1 to be called blood pressure.

PROC TRANSPOSE: wide to narrow

```
/* transposing the original data */
/* I assume data is sorted by id */
title "Using PROC TRANSPOSE";
proc transpose data=bp
    out=bp_Transpose1(rename=(col1=BP) drop=_name_);
    by id age;
    var bp1-bp4;
run;
```

Using PROC TRANSPOSE

```
proc print data=bp_Transpose1;
run;
```

| Obs | id | age | BP |
|-----|------|-----|-----|
| 1 | 0001 | 67 | 130 |
| 2 | 0001 | 67 | 138 |
| 3 | 0001 | 67 | 140 |
| 4 | 0001 | 67 | 136 |
| 5 | 0002 | 61 | 150 |
| 6 | 0002 | 61 | 145 |
| 7 | 0002 | 61 | 144 |
| 8 | 0002 | 61 | 142 |
| 9 | 0003 | 72 | 121 |
| 10 | 0003 | 72 | 135 |
| 11 | 0003 | 72 | 122 |
| 12 | 0003 | 72 | 140 |
| 13 | 0004 | 51 | 118 |
| 14 | 0004 | 51 | 115 |
| 15 | 0004 | 51 | 126 |
| 16 | 0004 | 51 | 120 |

PROC TRANSPOSE

If you have missing data due to missing observations at some time points, then this will show up as periods in the wide data, but you might want these rows to be deleted. For example,

```
0001 67 130
0001 67 138
0001 67 140
0001 67 .
0002 61 150
0002 61 145
0002 61 144
0002 61 142
```

To have the row deleted, so that the number of rows is equal to the number of times the patient received a measurement, use

```
out=bo_Transpose1(rename(col1=BP) drop=_name_
                  where BP is not null);
```


PROC TRANSPOSE: narrow to wide

```
proc transpose data=bp_Transpose1  
               out=bp_Transpose2;  
  by id;  
  var BP;  
run;  
  
proc print data=bp_Transpose2; run;
```

Using PROC TRANSPOSE

| Obs | id | _NAME_ | COL1 | COL2 | COL3 | COL4 |
|-----|------|--------|------|------|------|------|
| 1 | 0001 | BP | 130 | 138 | 140 | 136 |
| 2 | 0002 | BP | 150 | 145 | 144 | 142 |
| 3 | 0003 | BP | 121 | 135 | 122 | 140 |
| 4 | 0004 | BP | 118 | 115 | 126 | 120 |

PROC TRANSPOSE: a whole dataset

To transpose an entire dataset, you can omit the BY statement:

```
filename foo url "http://math.unm.edu/~james/citytemp.txt";

data city;
  infile foo dlm="09"x firstobs=1 obs=5;
  input city :$50. jan apr jul oct rain days snow years :10.;
run;

proc transpose data=city out=city2;
  var jan apr jul oct rain days;
run;

proc print data=city2; run;
```

The SAS System

| Obs | _NAME_ | COL1 | COL2 | COL3 | COL4 | COL5 |
|-----|--------|-------|-------|--------|--------|-------|
| 1 | jan | 22.2 | 35.70 | 15.80 | 35.80 | 42.7 |
| 2 | apr | 46.6 | 55.60 | 36.30 | 54.10 | 61.6 |
| 3 | jul | 71.1 | 78.50 | 58.40 | 73.00 | 80.0 |
| 4 | oct | 49.3 | 57.30 | 34.10 | 55.20 | 62.8 |
| 5 | rain | 38.6 | 9.47 | 16.08 | 47.07 | 50.2 |
| 6 | days | 136.0 | 60.00 | 115.00 | 126.00 | 115.0 |

Macros (Chapter 25)

The idea behind Macros is that before SAS compiles your code, it checks for macro statements and macro variables. These tell SAS to find and replace macro code with appropriate SAS code, and then to execute the resulting code, which no longer has macros in it.

The use of macros can let you do things like have loops over data steps instead of (or in addition to) loops within data steps. You can also loop over procedures. This also allows you to have variables (in the form of macro variables) that are not part of a datastep.

Macro variables for dates

A built-in macro variable is the current date. If you want the date as part of your output (such as in title statements), you might want the date to reflect the date that the SAS code was run, which would need to be changed every day that you modified your program.

Instead, you can use a macro variable for date, such as `&sysdate` and `&sysptime` so that when SAS compiles your code, it replaces these macro variables with the current date and time based on your system. This saves you from having to manually update the date and time every time you run your code.

Macro variables for dates

An example is something like this:

```
title "Current date: &sysdate, &systemtime"  
proc print data=mydata; run;
```

When SAS sees your code, it first converts it to something like title
"Current date: 28OCT2014, 11:31"
proc print data=mydata; run;

Then SAS executes this code with the macro variables replaced. Note also that you should use double quotes instead of single quotes when working with macro variables (with single quotes it will print &sysdate as a string and not replace it). For previous examples we've done, double quotes versus single quotes hasn't mattered.

Macro variables

You can have user-defined macro variables as well using %LET. Here you define a macro variable to be equivalent to some string, and when SAS runs, it will replace the appropriate string before executing any code. Let's say you are not sure which variables you want to analyze, but you want to do multiple analyses with these variables. For example, you might have variables such as age, systolicBP, diastolicBP, cholesterol, height, and weight. You might want to run several analyses such as correlations, regression, and a survival analysis. If you want to run different models, say some that include both measures of blood pressure, others that only include systolic BP, you have to change your variable list in multiple procedures. Instead, you can use a macro variable to keep track of the variables you want analyzed, and only have to change the list once.

Macro variables with %LET

Here %let tells SAS to find and replace all instances of the string
&predictors
with the string
age sytstolicBP cholesterol height weight

```
%let predictors= age sytstolicBP cholesterol height weight;  
proc means data=patients; var &predictors; run;  
proc corr data=patients; var &predictors; run;  
proc reg data=patients; diastolicBP = &predictors; run;
```

Macro variables

Macro variables can also be useful for doing loops. For example, in a simulation, you might have to repeat a procedure a random number size a variable number of times. Using macro variables lets the parameters of your simulation be stated all at once at the top of your code, making your code easier to modify.

Simulation to illustrate the Central Limit Theorem

```
%let n=10;
%let iter=1000;
%let lambda=2.5;
%let seed=20141025;

title "Illustration of Central Limit Theorem using exponential
distributions with sample size n=&n and &iter iterations";
data sim;
  do i=1 to &iter;
    do j=1 to &n;
      /* argument of ranexp is the seed */
      x = ranexp(&seed)/&lambda;
      output;
    end;
  end;
run;

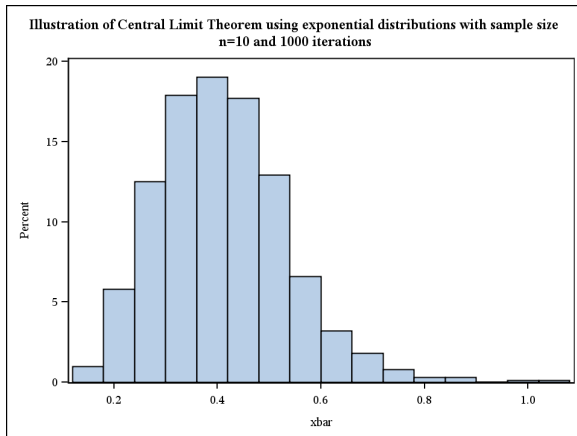
proc means data=sim;
  by i;
  var x;
  output out=simout mean=xbar;
run;

proc print data=simout; run;

ods pdf file="clt.pdf";
proc sgplot data=simout;
  histogram xbar;
run;
ods pdf close;
```

Simulation to illustrate the Central Limit Theorem

05:30 Tuesday, October 28, 2014 1



The syntax `%MACRO` lets you write a macro that is similar to writing a function in R or other languages. This is a way to get around the fact that in SAS, you can't write your own procedure. So, instead, a collection of procedures data steps, etc. can be written as a macro and then that macro can be called with different parameters. The macro is closed by writing `%MEND` for Macro End.

The following example simulates rolling a die with n sides starting with values $k, k + 1, \dots, k + n - 1$.

```
%macro roll(n,Start,End);  
data generate;  
  do Subj = 1 to &n;  
    x = int((&End - &Start + 1)*ranuni(0) + &Start);  
    output;  
  end;  
run;  
title "Randomly Generated Data Set with &n Obs";  
title2 "Values are integers from &Start to &End";  
proc print data=generate noobs; run;  
%mend
```

Macros as functions

The MPRINT option allows you to see how your macro is evaluated in the log file, and this can be useful for debugging.

```
1      %macro roll(n,Start,End);
2      data generate;
3          do Subj = 1 to &n;
4              x = int((&End - &Start + 1)*ranuni(0) + &Start);
5              output;
6          end;
7      run;
8
9      title "Randomly Generated Data Set with &n Obs";
10     title2 "Values are integers from &Start to &End";
11     proc print data=generate noobs; run;
12     %mend;
13
14     /* this options prints values of the macros to the log */
15     options mprint;
16
17     /* generate 10 random numbers between 1000 and 1100 */
18     %roll(10,1000,1100);
MPRINT(ROLL):  data generate;
MPRINT(ROLL):  do Subj = 1 to 10;
MPRINT(ROLL):  x = int((1100 - 1000 + 1)*ranuni(0) + 1000);
MPRINT(ROLL):  output;
MPRINT(ROLL):  end;
MPRINT(ROLL):  run;
```

Using Macros as functions

The above code defines the macro but doesn't actually do anything, much like a function definition.

To get the macro to do something useful, you have to call it. For example, writing

```
%roll(10000,1,100)
```

generates a dataset with 10,000 rows where each row is a random number between 1 and 100.

Using macros to clean up code

You could also use a very short macro just to save typing later. For example, if you are creating many discrete uniforms for you code, it might be a bit tedious to always write

```
int((&End - &Start + 1)*ranuni(0) + &Start);
```

So you could just have a macro like this

```
%macro roll(n, Start, End); int((&End - &Start +  
1)*ranuni(0) + &Start); %mend;
```

so that you only have to type `%roll(n, &start, &end)` to use the macro. Note that there is no semicolon when you call a macro. This is because SAS is just replacing the text.

Macro variables within strings

Sometimes you want to have a macro variable in the middle of a string. For example, suppose I have data sets d1a, d2a, d3a, etc. and want to let the numeral be a macro variable. The following will not work

```
%let n=1;
data d1;
    set d&na;
    input x y z;
run;
```

Why doesn't this work?

Macro variables within strings

The problem is that SAS is looking for a macro variable called `&na` instead of `&n`. To tell SAS to stop reading the name of the macro variable, you can put a period in.

```
%let n=1;
data d1;
    set d&n.a;
    input x y z;
run;
```

And this will correctly resolve as `set d1a;`.

Macro variables within strings

Sometimes the syntax can be a little funny. For example, suppose you have files `data1.txt`, `data2.txt`, etc. To read them in you could use the following

```
%let n=1;
data d1;
  infile "data&n..txt";
  input x y z;
run;
```

What happens if you remove one of the periods in the above code?

Macrovariables within strings

In the previous example, if you remove one of the periods, it SAS will correctly read in the macro variable `&n`, but then it will ignore the next period and instead concatenate the strings `'data1'` and `txt`, so it will look for a file called `data1txt` with no period and probably won't find the file.

Looping over datasteps with macro %DO

Suppose you want to read in several datafiles, and you want to perform the same analysis on each file. You can loop over data sets using macros. For example, suppose you have data files called month1.txt, month2.txt, etc. Here is a macro way of handling all of these files.

```
%macro loop;
%let i=100;
/* note that you can initialize a variable in a \%DO loop */
%do j = 1 %to &i;
data d&i;
    infile = "data&i..txt";
    input x y z; run;
title = "Data&i..txt";
proc reg data = d&i;
    model x = y + z; run;
%end;
%mend;
%loop;
```

Looping over datasets

```
%macro means;
%do i = 1 %to 5;
data data&i;
    infile "data&i..txt";
    input x;
run;

proc means data=data&i;
    var x;
    title "Average average of data set &i";
run;
%end;
%mend means;

options mprint;
%means;
```

Looping over datasets

lst

Average average of data set 1

The MEANS Procedure

Analysis Variable : x

| N | Mean | Std Dev | Minimum | Maximum |
|---|------------|------------|------------|------------|
| 7 | 41.0000000 | 11.4746097 | 26.0000000 | 62.0000000 |

Average average of data set 2

The MEANS Procedure

Analysis Variable : x

| N | Mean | Std Dev | Minimum | Maximum |
|---|------------|------------|------------|------------|
| 7 | 38.1428571 | 15.3452335 | 14.0000000 | 62.0000000 |

Average average of data set 3

The MEANS Procedure

Analysis Variable : x

| N | Mean | Std Dev | Minimum | Maximum |
|---|------------|------------|------------|------------|
| 7 | 41.0000000 | 25.1329797 | 12.0000000 | 92.0000000 |

Looping over data sets

The previous code was equivalent to writing

```
data data1;  infile "data1.txt";  input x; run;
proc means data=data1; var x;
title "Average average of data set 1"; run;
data data2;  infile "data2.txt";  input x; run;
proc means data=data2; var x;
title "Average average of data set 2"; run;
data data3;  infile "data3.txt";  input x; run;
proc means data=data3; var x;
title "Average average of data set 3"; run;
data data4;  infile "data4.txt";  input x; run;
proc means data=data4; var x;
title "Average average of data set 4"; run;
data data5;  infile "data5.txt";  input x; run;
proc means data=data5; var x;
title "Average average of data set 5"; run;
```

Scripting

Another way of thinking about macros in SAS is that they are similar to *scripting*. Scripting is a topic that we won't get into much but it is extremely common in scientific research. Other languages are better at handling scripting, particularly `python` and `perl`, which are full-blown languages but were designed with scripting in mind.

Often in scripting, you have a sequence of programs that you want to run in a certain order, and often the output of one program is the input for another program. For example, you might decide that you regularly want to run SAS to analyze your data, but you want to generate graphics in R. In this case, your output from your SAS code could be used as input for R code used to generate the graphics. An advantage for this strategy of using computing programs is that you can use the strengths of each program (efficiency in SAS, graphing ability and flexibility in R).

Scripting

Another way to do this kind of looping is to write a loop in R (or Python or Perl or) that generates the SAS code. That might sound a little odd, but it is actually very similar to using SAS macros. Essentially, the SAS macro language is another language from typical SAS syntax, and it is a language that generates SAS syntax. So instead of using the SAS macro language, you could use another language instead if that is easier.

Here is R code to generate the SAS code:

```
for(i in 1:5) {  
  write(paste("data data",i,sep=""),file="loop.sas",append=T)  
  write(paste("infile \"data\",i,\".txt\";",sep=""),file="loop.sas",append=T)  
  write("input x;",file="loop.sas",append=T)  
  write("run;",file="loop.sas",append=T)  
  write(paste("proc means data=data",i,sep=""),file="loop.sas",append=T)  
  write("var x;",file="loop.sas",append=T)  
  write(paste("title \"Average of data set", i),file="loop.sas",append=T)  
  write("run;")  
}
```

In addition to R to SAS, you could use SAS to write SAS code even without using SAS macros, by using data `_null_` and PUT statements written to a file other than the log file. Ultimately, this might be more flexible than using the SAS macro language because you would have all of the subtle string functions, logical operators, arithmetic operators, and so forth available in SAS as opposed to the more limited SAS macro language. This would generally require running SAS twice, though: once to run data `_null_` to generate the SAS code you want, then running that SAS code from another file.

Another example of scripting might be that to run your SAS code on 10 different data sets, instead of using a macro to loop over data sets, you run a loop in shell script (usually this will be linux or Mac OS X). This has the advantage that jobs can be run separately on different processors such as in parallel computing. It is increasingly common for household computers to have multiple CPUs as well...

For this approach, instead of SAS using a different infile each time it is run, you could have it read in a file called `temp.txt` and the data needed is temporary copied to this file.

Generating variable lists

Another use of loops is to generate variable lists.

Suppose I have a data set that has variables such as Q1a, Q1b, Q2a, Q2b, ..., Q100b, in that order. Suppose I want to generate a list of variables that has Q1a, Q2a, Q3a, ..., Q100a. I can't type

Q1a-Q100a

to generate the list, because the hyphen only works if the numbers occur at the end of the string. Typing Q1a--Q100a also doesn't work because it will include variables that have b's in the string. An alternative is to write a macro (see next slide).

Generating variable lists

```
%macro varlist;  
%do i = 1 %to 100;  
Q&I.a  
%end;  
%mend varlist;  
  
proc print data=mydata;  
  var %varlist  
run;
```

does the trick. It is a bit less typing than Q1a, Q2a, For variables with longer names (for example species names that indexed by specimen number), you are also less likely to make a typo. This is at the expense of more sophisticated code with more difficult syntax, but this is a tradeoff which might or might not be worthwhile.

Using macro variables as global variables

Instead of first defining a macro variable using a %LET statement outside of a data step, you can define a value to be a macro variable from within a data step, which can then be subsequently used in later data steps or procedures. This is an alternative to having an entire column just have one constant value.

To do this, you don't use a %LET statement at all. Instead you can use CALL SYMPUT(macro variable name, value) from within a data step.

Using macro variables as global variables

Suppose I have a set of points and want to calculate a reference line based on the mean. We could use the temperature data as an example. The goal here is to create a globally accessible variable representing the mean temperature, and to use this in another procedure. In this case, as a REFLINE for a plot.

Using macro variables as global variables

```
filename foo url "http://math.unm.edu/~james/normtemp.txt";

data new;
  infile foo dlm='09'x firstobs=2;
  input degrees sex $ age;

run;

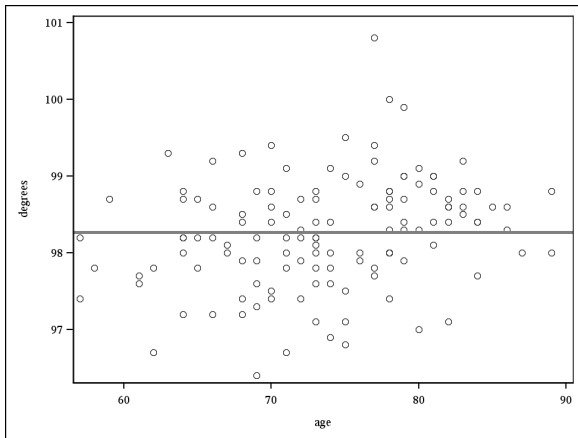
/* create output dataset with only one observation */
proc means data=new;
  var degrees;
  output out = tempout(keep=aveTemp)
         mean = aveTemp;
run;

/* creates macro variable meanTemp */
data _null_;
  set tempout;
  call symput("meanTemp",aveTemp);
run;

ods pdf file="tempplot.pdf";
proc sgplot data=new;
  scatter x=age y=degrees;
  refline &meanTemp / lineattrs=(thickness=3);
run;
ods pdf close;
```


Using macro variables as global variables

1



SYMGET

SYMGET is a complement to SYMPUT. Instead of putting a value to a global macro variable, SYMGET retrieves the value of a macrovariable, such as in

```
rate = symget("aveTemp");
```

CALL

In addition to CALL SYMPUT, there are other functions that can be used with CALL. Functions used with CALL do not return a value, but assign a value as part of the function. For example, the following both generate a random variable and assign it to x:

```
x = ranexp(seed, n, p);
```

`call ranexp(seed,n,p,x);` There is no particular advantage to doing it this way. There are a few other functions that CALL can be used with.

CALL SYSTEM

CALL SYSTEM allows SAS to use functions from the operating system, and it is operating-system dependent. A really cool application is being able to process all files of a certain type in the current directory when it is unknown how many files there are and they are not systematically named.

CALL SYSTEM: part 1 of code to read in all data files

```
/* generate a list of files in the directory with
   the given format (start with d, end with .txt) */
data _null_;
  call system("ls d*.txt > files.txt");
run;

/* generate a dataset consisting of names of files */
data datasets;
  length file $20;
  infile "files.txt";
  input file;
  n = _n_;
run;

/* create a macro variable that has the number of files */
data datsets;
  set datasets;
  by n;
  if last.n then call symput('n',n);
run;
```

CALL SYSTEM: part 2 of code to read in all data files

```
/* read in all files in the current directory */
%macro printdata;
%do i=1 %to &n;
/* This is a bit tricky. I'm creating macro variables
   that are the names of all the files I'm interested in */
data _null_;
  set datasets;
  if _n_ = &i then call symput('myfile',file);
run;

data d&i;
  infile "&myfile";
  input x;
run;

title "Data from &myfile";
proc print data=d&i;
run;
%end;
%mend;

options mprint;
%printdata;
```

CALL SYSTEM

Data from data3.txt

| Obs | x |
|-----|----|
| 1 | 34 |
| 2 | 45 |
| 3 | 34 |
| 4 | 26 |
| 5 | 44 |
| 6 | 12 |
| 7 | 92 |

Data from data4.txt

| Obs | x |
|-----|----|
| 1 | 11 |
| 2 | 12 |
| 3 | 34 |
| 4 | 45 |
| 5 | 34 |
| 6 | 26 |
| 7 | 44 |
| 8 | 42 |
| 9 | 62 |

Data from data5.txt

| Obs | x |
|-----|----|
| 1 | 34 |
| 2 | 45 |
| 3 | 34 |
| 4 | 26 |
| 5 | 54 |
| 6 | 52 |
| 7 | 52 |
| 8 | 9 |

Having macro variables with special characters

A macro variable can be just a short-hand for a string. However, it normally can't include a semi-colon because this is a special character that ends a %LET statement. You can get around this.

To allow a semi-colon at the end, you can use a macro string function. For example

```
let myprint = proc print%str(;
```

where %STR is a string function in the macro language. There are several macro statements that mimic SAS commands, such as %UPCASE, %SUBSTR, %SCAN, %LEFT, %INDEX, %TRIM, %LEFT, and %VERIFY, %DO ... %TO and %IF ... %THEN. Usually, macro statements occur within macros, but some can occur outside of macros, such as %LET, and %PUT (which mimics PUT).

Merging a variable number of files

Suppose you want to merge several files all in one step, but the number of files might vary from case to case. An example might be if you have one file per math class, and you want to create a giant file that has each student listed once with information across all math classes, so you merge by student ID. You can use a macro statement to do this. Suppose there are 50 math classes one semester with datasets class1, class2, ...

```
%let nclasses=50;
%macro mathmerge;
data mathclasses;
    merge %do i=1 %to &nclasses;
        class&i %end;;
    by id;
run;
%mend;
```

Note the use of the semi-colons. The first is to end the macro statement %END. The second ends the merge statement within the data step.

Another application?

This is from a book called *Professional SAS Programming Secrets* by Rick Aster and Rena Seidman that I have based on SAS version 6 (1991):

“The fact that macros are more difficult to read can be an advantage if you are distributing a program that you don’t want the user to read. Although you can’t currently prevent an experienced an experienced SAS programmer from finding out the contents of a macro, you could at least make it difficult for the average SAS user.”

I was very surprised to read this. I haven’t found this attitude very often! On the other hand, try reading

http://en.wikipedia.org/wiki/Obfuscation_%28software%29#Examples
for examples of deliberately obscure code.

Combining macros with the Output Delivery System

The Output Delivery System (ODS) can be used to generate output other than just sending the results of procedures to the Results Viewer window. So far, all that we've used ODS for is to generate pdf files from graphics procedures, but there are other uses.

A common use of the ODS is to extract information from results of PROCs. For example, you might want to run a procedure on numerous data sets, or run different models on the same data set, and save just one piece of information from each one, such as a p -value or the log-likelihood. You might then want to gather these p -values (or other statistic) together and put them in a data set.

Combining macros with ODS

A common application for wanting to extract information from a procedure is to do a simulation to test the effects of model violation, or a power analysis to determine the probability of rejecting H_0 when H_0 is false. In this cases, you might want to generate a large number of simulated data sets, run a PROC on them, and extract the p -value. This might tell you the effects of model violation (violating assumptions) on type I error by looking at the proportion of p -values that are less than 0.05.

For this type of application, since we are generating multiple data sets and analyzing them the same way, it would make sense to use a macro to loop over data steps and procedures.

ODS

First we'll look at just using ODS to influence the output of PROC UNIVARIATE.

```
filename foo url "http://math.unm.edu/~james/normtemp.txt";

data temperature;
  infile foo dlm="09"x;
  input temperature sex $ age;
run;

ods listing select BasicMeasures;
title "First PROC UNIVARIATE";
proc univariate data=temperature;
run;

title "Second PROC UNIVARIATE";
proc univariate data=temperature;
run;
```

Note that the ODS options only apply to the first PROC UNIVARIATE statement. The second instance of PROC UNIVARIATE uses several pages of output (so it isn't shown).

```

First PROC UNIVARIATE

The UNIVARIATE Procedure
Variable: temperature

Basic Statistical Measures

Location                                Variability
Mean      98.24923      Std Deviation      0.73318
Median    98.30000      Variance      0.53756
Mode      98.00000      Range      4.50000
                                Interquartile Range      0.90000
First PROC UNIVARIATE

The UNIVARIATE Procedure
Variable: age

Basic Statistical Measures

Location                                Variability
Mean      73.76154      Std Deviation      7.06208
Median    74.00000      Variance      49.87293
Mode      73.00000      Range      32.00000
                                Interquartile Range      10.00000
    
```

Note: The mode displayed is the smallest of 2 modes with a count of 10.

To make the ODS options last for more than one procedure, you can use the PERSIST option, as in

```
ods listing select BasicMeasures (persist);
```

You can also EXCLUDE some of the output, as in

```
ods listing exclude BasicMeasures (persist);
```

Which retains most of the PROC UNIVARIATE output but omits some of the basic statistics. For one UNIVARIATE statement for this statement, this reduces the output from 142 lines to 104 lines.

ODS in SAS Studio

The code I showed from linux SAS doesn't work in SAS Studio, and it gives an error saying that SELECT is not available and that no listing is open. You can use ODS to write output to a text file, for example in the following.

```
1 options nodate;  
2  
3 filename foo url "http://math.unm.edu/~james/normtemp.txt";  
4  
5  
6 data temperature;  
7     infile foo dlm="09"x;  
8     input temperature sex $ age;  
9  
10 run;  
11  
12 ods listing file="/home/jamdeg/output.txt";  
13 proc univariate data=temperature;  
14 run;  
15 ods listing close;
```


ODS in SAS Studio

To limit the SAS output and save the output to a file, use two ODS statements.

```
12 ods html file="/home/jamdeg/output.html";
13 ods select BasicMeasures;
14 proc univariate data=temperature;
15 run;
16 ods html close;
```

The UNIVARIATE Procedure
Variable: temperature

| Basic Statistical Measures | | | |
|----------------------------|----------|---------------------|---------|
| Location | | Variability | |
| Mean | 98.24923 | Std Deviation | 0.73318 |
| Median | 98.30000 | Variance | 0.53756 |
| Mode | 98.00000 | Range | 4.50000 |
| | | Interquartile Range | 0.90000 |

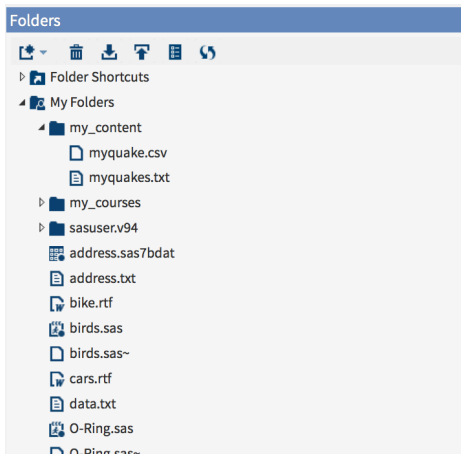
The UNIVARIATE Procedure
Variable: age

| Basic Statistical Measures | | | |
|----------------------------|----------|---------------------|----------|
| Location | | Variability | |
| Mean | 73.76154 | Std Deviation | 7.06208 |
| Median | 74.00000 | Variance | 49.87293 |
| Mode | 73.00000 | Range | 32.00000 |
| | | Interquartile Range | 10.00000 |

Note: The mode displayed is the smallest of 2 modes with a count of 10.

ODS in SAS Studio

The generated file shows up on the left under Folders depending on the path you specified.



ODS in SAS Studio

Saving as html makes the output in the same pretty format as in SAS studio. Here I generated the html file in SAS studio, saved to my local computer, then opened the file output.html from my local computer

```
12 ods html file="/home/jamdeg/output.html";
13 proc univariate data=temperature;
14 run;
15 ods html close;
```

The UNIVARIATE Procedure
Variable: temperature

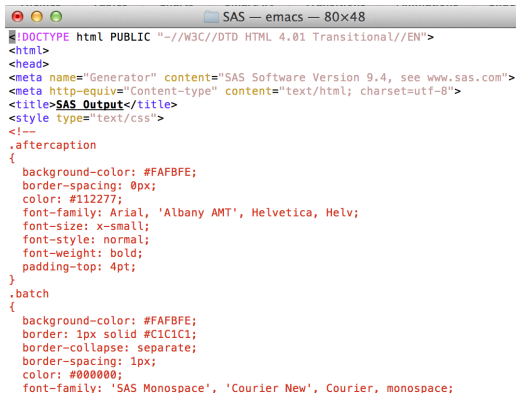
| Moments | | | |
|-----------------|------------|------------------|------------|
| N | 130 | Sum Weights | 130 |
| Mean | 98.2492308 | Sum Observations | 12772.4 |
| Std Deviation | 0.73318316 | Variance | 0.53755754 |
| Skewness | -0.0044191 | Kurtosis | 0.7804574 |
| Uncorrected SS | 1254947.82 | Corrected SS | 69.3449231 |
| Coeff Variation | 0.74624824 | Std Error Mean | 0.06430442 |

| Basic Statistical Measures | | | |
|----------------------------|----------|---------------------|---------|
| Location | | Variability | |
| Mean | 98.24923 | Std Deviation | 0.73318 |
| Median | 98.30000 | Variance | 0.53756 |
| Mode | 98.00000 | Range | 4.50000 |
| | | Interquartile Range | 0.90000 |

| Tests for Location: Mu0=0 | | | |
|---------------------------|-----------|----------|------------------|
| Test | Statistic | p Value | |
| Student's t | t | 1527.877 | Pr > t < .0001 |

ODS in SAS Studio

The html file can be opened up in a plain text editor (or Word or Whatever) and edited by hand, so if you wanted to omit some of the output, you could manipulate this file. Since it is html, this also makes it useful for putting your results on webpages.



```
SAS — emacs — 80x48
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta name="Generator" content="SAS Software Version 9.4, see www.sas.com">
<meta http-equiv="Content-type" content="text/html; charset=utf-8">
<title>SAS Output</title>
<style type="text/css">
<!--
.aftercaption
{
    background-color: #FAFBFE;
    border-spacing: 0px;
    color: #112277;
    font-family: Arial, 'Albany AMT', Helvetica, Helv;
    font-size: x-small;
    font-style: normal;
    font-weight: bold;
    padding-top: 4pt;
}
.batch
{
    background-color: #FAFBFE;
    border: 1px solid #C1C1C1;
    border-collapse: separate;
    border-spacing: 1px;
    color: #000000;
    font-family: 'SAS Monospace', 'Courier New', Courier, monospace;
```

Different procedures might have different statistics that can be displayed and controlled by ODS. For PROC REG, you can specify that you want the ANOVA table, or that you want FitStatistics, for example.

ODS in SAS Studio

The generated file shows up on the left under Folders depending on the path you specified.

```
3 filename foo url "http://math.unm.edu/~james/normtemp.txt";
4
5
6 data temperature;
7   infile foo dlm="09"x;
8   input temperature sex $ age;
9
10 run;
11
12 ods listing file="/home/jamdeg/regression1.txt";
13 ods select Anova;
14 proc reg data=temperature;
15 model temperature=age;
16 run;
17 ods listing close;
18 quit; /* quit helps the ods options not carry over
19        to the next PROC */
20
21
22
23 ods listing file="/home/jamdeg/regression2.txt";
24 ods select FitStatistics;
25 proc reg data=temperature;
26 model temperature=age;
27 run;
28 quit;
```

ODS in SAS Studio

The generated file shows up on the left under Folders depending on the path you specified.

output.html

output.txt

regression1.txt

regression2.txt

temp.rtf

The REG Procedure
Model: MODEL1
Dependent Variable: temperature

| Analysis of Variance | | | | | |
|----------------------|-----|----------------|-------------|---------|--------|
| Source | DF | Sum of Squares | Mean Square | F Value | Pr > F |
| Model | 1 | 4.46176 | 4.46176 | 8.80 | 0.0036 |
| Error | 128 | 64.88316 | 0.50690 | | |
| Corrected Total | 129 | 69.34492 | | | |

The REG Procedure
Model: MODEL1
Dependent Variable: temperature

| | | | |
|----------------|----------|----------|--------|
| Root MSE | 0.71197 | R-Square | 0.0643 |
| Dependent Mean | 98.24923 | Adj R-Sq | 0.0570 |
| Coeff Var | 0.72466 | | |

To have control over what output is reported by a procedure, you need to know the names of different SAS tables that are output by different procedures. You can do this putting an ODS TRACE that will give the names of the tables in the log file.

ODS in SAS Studio

The log file lists different outputs generated by the PROC

```
69
70
71
72     ods listing file="/home/jamdeg/regression2.txt";
73     ods trace on;
74     proc reg data=temperature;
75     model temperature=age;
76     run;
```

Output Added:

```
-----
Name:      NObs
Label:     Number of Observations
Template:  Stat.Reg.NObs
Path:      Reg.MODEL1.Fit.temperature.NObs
-----
```

Output Added:

```
-----
Name:      ANOVA
Label:     Analysis of Variance
Template:  Stat.REG.ANOVA
Path:      Reg.MODEL1.Fit.temperature.ANOVA
-----
```

ODS in SAS Studio

The error message is a SAS Studio problem...don't worry too much about this.

Output Added:

Name: FitStatistics
Label: Fit Statistics
Template: Stat.REG.FitStatistics
Path: Reg.MODEL1.Fit.temperature.FitStatistics

Output Added:

Name: ParameterEstimates
Label: Parameter Estimates
Template: Stat.REG.ParameterEstimates
Path: Reg.MODEL1.Fit.temperature.ParameterEstimates

Output Added:

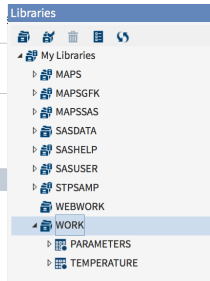
Name: DiagnosticsPanel
Label: Fit Diagnostics
Template: Stat.REG.Graphics.DiagnosticsPanel
Path: Reg.MODEL1.ObswiseStats.temperature.DiagnosticPlots.DiagnosticsPanel

ERROR: Cannot write image to DiagnosticsPanel1.png. Please ensure that proper disk permissions are set.
ERROR: Cannot write image to DiagnosticsPanel1.png. Please ensure that proper disk permissions are set.

ODS in SAS Studio

Note that this creates an output data set that can be treated just like data.

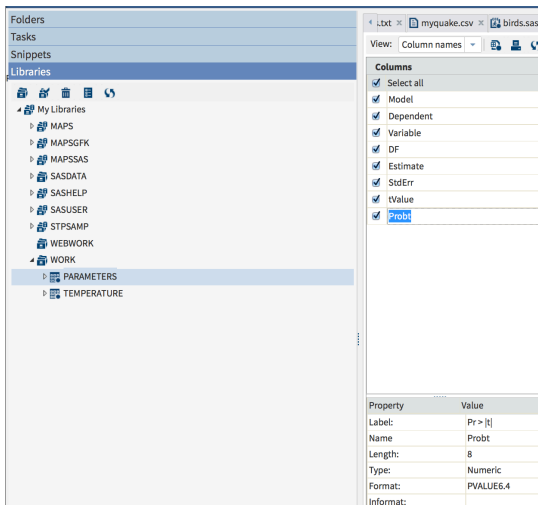
```
15 ods output ParameterEstimates=parameters;  
16 ods select none;  
17 proc reg data=temperature;  
18 model temperature=age;  
19 run;  
20  
21 ods select all;  
22 proc print data=parameters;  
23 var probt;  
24 run;  
25
```



| Obs | Model | Dependent | Variable | DF | Estimate | StdErr | tValue | Probt |
|-----|--------|-------------|-----------|----|----------|---------|--------|--------|
| 1 | MODEL1 | temperature | Intercept | 1 | 96.30675 | 0.65770 | 146.43 | <.0001 |
| 2 | MODEL1 | temperature | age | 1 | 0.02633 | 0.00888 | 2.97 | 0.0036 |

ODS in SAS Studio

What type of variable is Probt?



The screenshot displays the SAS Studio interface. On the left, the 'Libraries' pane shows a tree structure under 'My Libraries'. The 'PARAMETERS' library is selected. On the right, the 'Columns' pane shows a list of variables. The variable 'Probt' is selected. Below the 'Columns' pane, a table displays the properties of the selected variable.

| Property | Value |
|-----------|-----------|
| Label: | Pr > t |
| Name | Probt |
| Length: | 8 |
| Type: | Numeric |
| Format: | PVALUE6.4 |
| Informat: | |

It looks like it is numeric, and a format was used to display the inequality. This means that SAS has saved more information than just that the p-value is less than $< .0001$. Sometimes in genetics, we make literally 100s of thousands of hypothesis tests, so using a Bonferroni correction, we might care whether the p-value is 10^{-6} versus 10^{-8} .

How can we get a more precise value for the p -value?

ODS in SAS studio

We'll reprint the p -values and reformat them to something more precise. This is more precise than I would expect SAS to be – it might have 10-15 digits of precision or so.

```
21 ods select all;  
22 proc print data=parameters;  
23 var probt;  
24 format probt 32.31;  
25 run;  
26
```

| Obs | Probt |
|-----|-----------------------------------|
| 1 | .00000000000000000000000000000000 |
| 2 | .00359148925664424600000000000000 |

We're now ready to try combining macros with ODS. Here is an example of a simulation study that we can do with what we have.

Suppose X_1, \dots, X_{10} are i.i.d. (independent and identically distributed) exponential random variables with rate 1. If you perform a t -test at the $\alpha = 0.05$ level for whether or not the mean is 1, what is the type 1 error?

If X_1, \dots, X_{10} are normal with mean 1, standard deviation σ , then you expect that you will reject H_0 5% of the time when H_0 is true. In this case, H_0 is true (the mean is 1), but the assumption of normality in the t -test is violated, and this might effect the type 1 error rate.

ODS in SAS studio

First we'll do one data set and run PROC TTEST once with TRACE ON to figure out what table we want to save.

```
13 %let seed=1453155;
14
15 data d1;
16   do i=1 to 10;
17     x = ranexp(&seed);
18     output;
19   end;
20 run;
21
22 ods trace on;
23 proc ttest data=d1 h0=1;
24   var x;
25 run;
26 ods trace off;
27
```

```
79
80   ods trace on;
81   proc ttest data=d1 h0=1;
82     var x;
83   run;
```

Output Added:

```
-----
Name:      Statistics
Label:     Statistics
Template:  Stat.TTest.Statistics
Path:      Ttest.x.Statistics
-----
```

Output Added:

```
-----
Name:      ConFLimits
Label:     Confidence Limits
Template:  Stat.TTest.ConFLimits
Path:      Ttest.x.ConFLimits
-----
```

Output Added:

```
-----
Name:      TTests
Label:     T-Tests
Template:  Stat.TTest.TTests
Path:      Ttest.x.TTests
-----
```


ODS in SAS studio

It looks like we want the third table, which was called TTests

The TTEST Procedure
Variable: x

| N | Mean | Std Dev | Std Err | Minimum | Maximum |
|----|--------|---------|---------|---------|---------|
| 10 | 0.8501 | 0.4259 | 0.1347 | 0.1050 | 1.4820 |

| Mean | 95% CL Mean | Std Dev | 95% CL Std Dev |
|--------|-------------|---------|----------------|
| 0.8501 | 0.5455 | 1.1548 | 0.4259 |
| | 0.2930 | | 0.7776 |

| DF | t Value | Pr > t |
|----|---------|---------|
| 9 | -1.11 | 0.2947 |

If you look at the data set in Work Folder, the name of the variable with the p -value is again `Probt`, although when PROC TTEST runs, it labels the variable by `Pr > |t|`.

ODS in a macro

Here's a start, I run PROC TTEST 3 times on 3 generated data sets. This creates 3 small data sets with p-values.

```
%macro sim;

%do i=1 %to 3;
  data d1;
    do i=1 to 10;
      x = ranexp(15151515 + &i);
      output;
    end;
  run;

  ods output TTests=temp&i;
  ods select TTests;
  proc ttest data=d1 h0=1;
    var x;
  run;

  title "Print &i";
  proc print data=temp&i;
  run;
%end;
%mend;
%sim;
```

ODS in a macro: using concatenation merge to put all p-values in one data set

```
%macro sim;

%do i=1 %to 10000;
data d1;
  do i=1 to 10;
    x = ranexp(15151515 + &i);
    output;
  end;
run;

ods output TTests=temp;
ods select none;
proc ttest data=d1 h0=1;
  var x;
run;

%if &i=1 %then %do;
data pvalues;
  set temp;
  keep Probt;
run;
%end;

%else %do;
data pvalues;
  set pvalues temp;
  keep Probt;
run;
%end; /* end else */

%end; /* end do at top of macro */
%mend; /* end macro */
%sim;

proc print data=pvalues;
run;
```

ODS dataset of p -values

You should be able to now use your dataset of p -values to analyze your p -values. Particular questions of interest would be (1) how many p -values are below 0.05 (This is the type I error rate), and (2) the distribution of the p -values.

ODS dataset of p-values

In my case, I had trouble doing anything directly with the dataset `pvalues`. Nothing seemed to print in my output (for example PROC MEANS). So, in my case, I just output my dataset `pvalues` to an external file and then read it in again using a new SAS program. This is slightly inelegant, but it means I can start from scratch in case any of my ODS settings changed things and caused problems.//

This approach could also be useful in case I wanted to generate p-values in SAS and analyze them or plot them in another program like R, or if I just want to save those p-values for later reference.

ODS dataset of p-values

```
ods listing file="pvalues.txt";
proc print data=pvalues;          head -30 pvalues.txt
run;
ods listing close;

data pvalues;
  if Probt < .05 then type1=1;
  else type1=0;
run;
quit;

ods select all;
proc means data=pvalues;
  var type1;
run;

ods pdf file="pvalue.pdf";
proc sgplot data=pvalues;
  histogram Probt;
run;
ods pdf close;
```

T-Test 1000

| Obs | Probt |
|-----|--------|
| 1 | 0.3977 |
| 2 | 0.3571 |
| 3 | 0.3594 |
| 4 | 0.7764 |
| 5 | 0.5935 |
| 6 | 0.3053 |
| 7 | 0.3605 |
| 8 | 0.9266 |
| 9 | 0.4465 |
| 10 | 0.5570 |
| 11 | 0.8304 |
| 12 | 0.0062 |
| 13 | 0.7380 |
| 14 | 0.7901 |
| 15 | 0.6612 |
| 16 | 0.6396 |
| 17 | 0.4201 |
| 18 | 0.5442 |
| 19 | 0.2656 |
| 20 | 0.0207 |
| 21 | 0.9250 |
| 22 | 0.2574 |
| 23 | 0.4046 |
| 24 | 0.6112 |
| 25 | 0.0288 |
| 26 | 0.3867 |

ODS dataset of p-values

Here I analyze the output of a SAS program using a second SAS program. `pvalues2.txt` is a cleaned up version of `pvalues.txt` that removed header information and so forth.

```
data pvalues;
  infile "pvalues2.txt";
  input n x;
  if x<.05 then type1=1;
  else type1=0;
run;

proc means data=pvalues;
  var type1;
run;

ods pdf file="pvalueplot.pdf";
proc sgplot data=pvalues;
  histogram x;
run;
ods pdf close;
```

| The SAS System | | | | |
|---------------------------|-----------|-----------|---------|-----------|
| The MEANS Procedure | | | | |
| Analysis Variable : type1 | | | | |
| N | Mean | Std Dev | Minimum | Maximum |
| 1000 | 0.0840000 | 0.2775266 | 0 | 1.0000000 |

ODS dataset of p-values

Note that the MEANS procedure calculated the mean of the 0s and 1s and got 0.084. This means there were 84 (out of 1000) observations that had a p-value less than 0.05. How did it get the standard deviation? Note that each observation is a Bernoulli trial, and the standard deviation of a Bernoulli trial is $\sqrt{p(1-p)}$, so we would estimate this to be $\sqrt{.084(1-.084)} = .2773878$. Why is this (slightly) different from the standard deviation reported by PROC MEANS?

Also, is there evidence that there is an inflated type I error? Is 0.084 significantly higher than $\alpha = 0.05$?

Statistical inference and simulations

Sometimes we find ourselves using statistical inference just to interpret our own simulations, rather than for interpreting data.

Some scientists have the attitude that if a phenomenon is real, then you shouldn't need to statistics to see it in the data. Although I don't share this point of view, because a lot of data is too complicated to interpret by "eye", I sort of feel this way with simulations, though.

If you're not sure whether 0.084 is significantly higher than 0.05 (meaning there really is inflated type 1 error), you could either get a confidence interval around 0.084, or you could just do a larger simulation so that you could be really sure without having to construct confidence intervals. In this case, the confidence interval does exclude 0.05, so there is evidence to think that the type 1 error rate is somewhat inflated due to violation of the assumption of normally distributed samples.

What about the distribution of p -values?

The p -value is a random variable. It is a function of your data, much like \bar{X} and $\widehat{\sigma^2}$, so it is a sample statistic. What should the distribution of the p -value be under the null hypothesis for a test statistic? If you use $\alpha = 0.05$, this means that 5% of the time the p -value should be below α . More generally, for any α ,

$$P(p\text{-value} < \alpha) = \alpha$$

. The p -value therefore has the same CDF as a uniform random variable. So p -values should be uniformly distributed for appropriate statistical tests when the null hypothesis and all assumptions are true. This is true for tests based on continuous test-statistics. For discrete problems, it might not be possible for $P(p\text{-value} < \alpha) = \alpha$.

The distribution of p -values

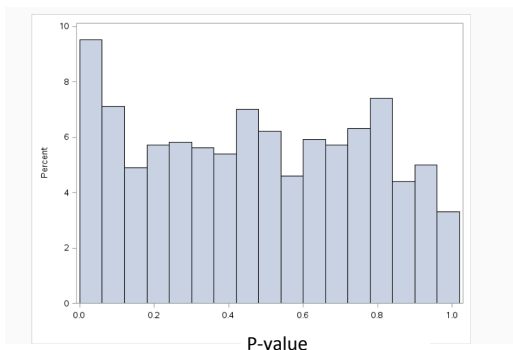
Here is a more technical explanation of why p -values are uniformly distributed for continuous test statistics, when the null is true, and for hypotheses $H_0 : \mu = \mu_0, H_A : \mu > \mu_0$ (i.e., I'll just consider a one-sided test). For this one-sided test, the p -value is $P(T \geq t)$, where T is the test-statistic.

$$\begin{aligned} 1 - F(t) &= P(T > t) = P(F(T) > F(t)) = 1 - P(F(T) \leq F(t)) \\ &\Rightarrow P(F(T) \leq F(t)) = F(t) \end{aligned}$$

Because $0 \leq F(t) \leq 1$, this means that $F(T)$ has a uniform distribution (since it has the same CDF). If U is uniform(0,1), then so is $1 - U$, so $1 - F(t)$ is also uniform(0,1). But note that $1 - F(t) = P(T \geq t)$, which is the p -value.

ODS dataset of p -values

Here is the distribution of the p -values represented by a histogram. Typically uniform distributions have flatter looking histograms with 1000 observations, so the p -values here do not look uniformly distributed. Again, this would be clearer if we did more than just 1000 simulations.



A different way to simulate this problem

Instead of simulating 1000 data sets of 10 observations, I could have just simulated all of the data all once and indexed the 1000 sets of 10 observations (similar to what I did for the Central Limit Theorem example). In this case, I would want to use PROC TTEST separately on each of the 1000 experiments.

Moral: There's more than one way to do things.

PROC TTEST using a BY statement

```
data d1;
  do i=1 to 1000;
    do j=1 to 10;
      x = ranexp(15151 + 100*j + i);
      output;
    end;
  end;
run;

title "T-Test";
ods output TTests = pvalues;
proc ttest data=d1 h0=1;
  by i;
  var x;
run;

proc print data=pvalues;
run;
```

PROC TTEST using a BY statement

T-Test

| Obs | i | Variable | tValue | DF | Probt |
|-----|----|----------|--------|----|--------|
| 1 | 1 | x | 0.92 | 9 | 0.3839 |
| 2 | 2 | x | -1.41 | 9 | 0.1908 |
| 3 | 3 | x | -0.67 | 9 | 0.5175 |
| 4 | 4 | x | -1.06 | 9 | 0.3167 |
| 5 | 5 | x | 0.37 | 9 | 0.7170 |
| 6 | 6 | x | 0.39 | 9 | 0.7027 |
| 7 | 7 | x | 1.15 | 9 | 0.2809 |
| 8 | 8 | x | -1.05 | 9 | 0.3208 |
| 9 | 9 | x | -0.57 | 9 | 0.5823 |
| 10 | 10 | x | -0.37 | 9 | 0.7214 |
| 11 | 11 | x | -0.60 | 9 | 0.5644 |
| 12 | 12 | x | 1.15 | 9 | 0.2811 |
| 13 | 13 | x | -1.23 | 9 | 0.2498 |
| 14 | 14 | x | -0.16 | 9 | 0.8792 |
| 15 | 15 | x | -0.22 | 9 | 0.8303 |
| 16 | 16 | x | 0.02 | 9 | 0.9824 |
| 17 | 17 | x | 0.10 | 9 | 0.9188 |

P-value

How to do this in R?

It's a little easier in R. Here's how I would do it:

```
x <- rexp(10000)
x <- matrix(x,ncol=1000)
pvalue <- 1:1000
for(j in 1:1000) {
  pvalue[j] <- t.test(x[,j],mu=1)$p.value
}
sum(pvalue<=.05)/1000 # this is the type I error rate
hist(pvalue)
```