Week 08-09 [31+ Oct.] Class Activities

File:  week-08-09-DATA-step-prog-30oct08.doc
Directory:  \\Muserver2\USERS\B\baileraj\Classes\sta402\handouts

Formatting, basic DATA step manipulations and programming

7. Formatting, data recoding and basic DATA step manipulations
   1. Internal representations and output displays
   2. Date and time formats
   3. Recoding and transforming variables in a DATA step
   4. Ordering how tasks are done – precedence of operations
   5. What goes and what stays in a data set – DROP, KEEP, IF, WHERE, OUTPUT
   6. Structured thinking about writing programs – pseudo-code and modules
   7. CASE STUDY 7.1: Is the two-sample t-test robust to heterogeneous variances?
   8. CASE STUDY 7.2: Monte Carlo integration to estimate Pr(0<Z<1.645) for Z~N(0,1)
   9. CASE STUDY 7.3: Simple percentile-based bootstrap
  10. Throw out your tables of statistical distributions – CDF, PDF, QUANTILE
  11. Generating variables using random number generators – RAND

Exercises
7.1. Internal representations and output displays

- getting data with formatted values into a SAS data set (via “informats”) - how to process data and store an input variable

- displaying data values (“formats”) - how to display values of a particular variable

Data (crudely classified) into 4 types: character, numeric, date and time.
(One might argue that only two types are necessary since date and time data are essentially numeric but it is convenient to make this distinction when discussing formats

Common formats (character):
Characters: $w.

Display 7.1: Reading character and numeric variables with implicit and explicit formats

```sas
data mrexample;
* Lunneborg (1994) - body weight brain example;
  input species $ bodywt brainwt @@;
datalines;
beaver 1.35 8.10 cow 465.00 423.00 wolf 36.33 119.50 goat 27.66 115.00
guipig 1.04 5.50 diplodocus 11700.00 50.00 asielephant 2547.00 4603.00
donkey 187.10 419.00 horse 521.00 655.00 potarmonkey 10.00 115.00
cat 3.30 25.60 giraffe 529.000 680.00 gorilla 207.00 406.00
human 62.00 1320.00 afrelephant 6654.00 5712.00 triceratops 9400.00 70.00
rheomoney 6.80 179.00 kangaroo 35.00 56.00 hamster 0.12 1.00
mouse 0.023 0.40 rabbit 2.50 12.10 sheep 55.50 175.00
jaguar 100.00 157.00 chimp 52.16 440.00 brachiosaurus 87000.00 154.50
rat 0.28 1.90 mole 0.12 3.00 pig 192.00 180
;
```

```sas
data mrexample2;
  length species $ 15;
  input species bodywt brainwt @@;
datalines;
beaver 1.35 8.10 cow 465.00 423.00 wolf 36.33 119.50 goat 27.66 115.00
guipig 1.04 5.50 diplodocus 11700.00 50.00 asielephant 2547.00 4603.00
donkey 187.10 419.00 horse 521.00 655.00 potarmonkey 10.00 115.00
cat 3.30 25.60 giraffe 529.000 680.00 gorilla 207.00 406.00
human 62.00 1320.00 afrelephant 6654.00 5712.00 triceratops 9400.00 70.00
rheomoney 6.80 179.00 kangaroo 35.00 56.00 hamster 0.12 1.00
mouse 0.023 0.40 rabbit 2.50 12.10 sheep 55.50 175.00
jaguar 100.00 157.00 chimp 52.16 440.00 brachiosaurus 87000.00 154.50
rat 0.28 1.90 mole 0.12 3.00 pig 192.00 180
```
```sas
ods rtf;
proc print data=mrexample;
  id species;
run;
proc print data=mrexample2;
  id species;
run;
ods rtf close;
```

Display 7.2: Output produced after reading character and numeric variables

```sas
data mrexample;
  input species $ bodywt brainwt @@;
length species $ 15;
  input species bodywt brainwt @@;

  species   | bodywt | brainwt |
  -------------------------------------
  beaver    | 1.35   | 8.1     |
  Cow       | 465.00 | 423.0   |
  Wolf      | 36.33  | 119.5   |
  Goat      | 27.66  | 115.0   |
  guipig    | 1.04   | 5.5     |
  diplodoc  | 11700.00 | 50.0 |
  asieleph  | 2547.00 | 4603.0 |
  donkey    | 187.10 | 419.0   |
  horse     | 521.00 | 655.0   |
  potarmon  | 10.00  | 115.0   |
  Cat       | 3.30   | 25.6    |
  giraffe   | 529.00 | 680.0   |
  gorilla   | 207.00 | 406.0   |
  human     | 62.00  | 1320.0  |
  afreleph  | 6654.00 | 5712.0 |
  tricerat  | 9400.00 | 70.0   |
  rhemonke  | 6.80   | 179.0   |
  kangaroo  | 35.00  | 56.0    |
  hamster   | 0.12   | 1.0     |
  mouse     | 0.02   | 0.4     |
```
data mrexample;
  input species $  bodywt  brainwt @@;
  length species $   15;
  input species  bodywt  brainwt @@;

  species     |   bodywt |    brainwt |
  -----------------|----------|-----------|
    rabbit      |   2.50   |    12.1   |
    sheep       |   55.50  |   175.0   |
    jaguar      |  100.00  |   157.0   |
    chimp       |   52.16  |   440.0   |
  brachiosaurus |  87000.0 |   154.5   |
    Rat         |   0.28   |    1.9    |
    mole        |   0.12   |    3.0    |
    Pig         |  192.00  |   180.0   |

Common formats (numeric):
  w. (w.d)
  BESTw.
  COMMAw.d
  Ew.
  DOLLARw.d

Display 7.3: DATA step displays of numeric formats

```sas
data numeric_format_show;
  /* character formatting illustrated first */
  test_num = 1277695.384;
  put 'BEST6. / BEST9. / BEST12.';
  put test_num BEST6. ;
  put test_num BEST9. ;
  put test_num BEST12. ;
  put '-------------------------------';
  put 'COMMA7. / COMMA10.1 / COMMA11.3';
  put test_num COMMA9. ;
  put test_num COMMA12.1 ;
  put test_num COMMA13.3 ;
  put '-------------------------------';
  put 'E7.';
  put test_num E7. ;
  put '-------------------------------';
  put '7. / 10.1 / 11.3';
  put test_num 8. ;
  put test_num 12.1 ;
  put test_num 13.3 ;
  put '-------------------------------';
  put 'DOLLAR7. / DOLLAR10.2';
  put test_num DOLLAR9. ;
  put test_num DOLLAR12.2 ;
```

run;

Display 7.4: Output from SAS LOG with examples using numeric formats `put` to the log

<table>
<thead>
<tr>
<th>Format</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Value 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEST6. / BEST9. / BEST12.</td>
<td>1.28E6</td>
<td>1277695.4</td>
<td>1277695.384</td>
</tr>
<tr>
<td>COMMA7. / COMMA10.1 / COMMA11.3</td>
<td>1,277,695</td>
<td>1,277,695.4</td>
<td>1,277,695.384</td>
</tr>
<tr>
<td>E7.</td>
<td>1.3E+06</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7. / 10.1 / 11.3</td>
<td>1277695</td>
<td>1277695.4</td>
<td>1277695.384</td>
</tr>
<tr>
<td>DOLLAR7. / DOLLAR10.2</td>
<td>$1277695</td>
<td>$1277695.38</td>
<td></td>
</tr>
</tbody>
</table>
7.1.1 Defining your own FORMATS / INFORMATS

For defining numeric formats, the general form of this statement is

```
proc format
   <options: LIBRARY= SAS library/catalog to contain formats/informats.>;  
   value name <value-range-set(s)>;
```

For defining character formats, the general form of this statement is

```
proc format
   <options: LIBRARY= SAS library/catalog to contain formats/informs.>;  
   value $name <value-range-set(s)>;
```

Informats are analogously defined except that `value` is replaced by `invalue`. 
Example: User-defined format for numeric variable

Display 7.5: Constructing a user-defined format for a numeric variable

```sas
data toyexample;
  input literacy @@;
  literacy_too = literacy;
datalines;
-99 25.55 53 53.5 73.7 83 99.9 107 . ;
proc format;
  value literacyfmt
    0-53='First quartile'
    53<-76='Second quartile'
    76<-90='Third quartile'
    90<-100='Fourth quartile'
    . = 'Missing'
    OTHER = 'Invalid';
data toyexample2; set toyexample;
  format literacy literacyfmt.;
ods rtf;
proc print;
  run;
proc means;
  var literacy literacy_too;
  run;
ods rtf close;
```

Display 7.6: Output from printing dataset with a user-defined variable

<table>
<thead>
<tr>
<th>Obs</th>
<th>literacy</th>
<th>literacy_too</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Invalid</td>
<td>-99.00</td>
</tr>
<tr>
<td>2</td>
<td>First quartile</td>
<td>25.55</td>
</tr>
<tr>
<td>3</td>
<td>First quartile</td>
<td>53.00</td>
</tr>
<tr>
<td>4</td>
<td>Second quartile</td>
<td>53.50</td>
</tr>
<tr>
<td>5</td>
<td>Second quartile</td>
<td>73.70</td>
</tr>
<tr>
<td>6</td>
<td>Third quartile</td>
<td>83.00</td>
</tr>
<tr>
<td>7</td>
<td>Fourth quartile</td>
<td>99.90</td>
</tr>
<tr>
<td>8</td>
<td>Invalid</td>
<td>107.00</td>
</tr>
<tr>
<td>9</td>
<td>Missing</td>
<td>.</td>
</tr>
</tbody>
</table>
Display 7.7: Output from PROC MEANS

<table>
<thead>
<tr>
<th>Variable</th>
<th>N</th>
<th>Mean</th>
<th>Std Dev</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>literacy</td>
<td>8</td>
<td>49.5812500</td>
<td>65.6923019</td>
<td>-99.0000000</td>
<td>107.0000000</td>
</tr>
<tr>
<td>literacy_too</td>
<td>8</td>
<td>49.5812500</td>
<td>65.6923019</td>
<td>-99.0000000</td>
<td>107.0000000</td>
</tr>
</tbody>
</table>

7.2. Character, numeric, time and date formats

Large number of formats that can be used to read and display dates and times.

Dates might be recorded as 30jun10, 30jun2010, 063010 or even 30.06.08.

All require the use of different format to read them into SAS (see 7.8)

Display 7.8: Formats for reading common date-values into SAS variables

```
data;
  input @1 indate1 date7. @9 indate2 date9.
    @19 indate3 mmddyy. @26 indate4 ddmmyy8.;
;
datalines;
30jun10 30jun2010 063010 30.06.10
;
do rtf;
proc print;
run;
do rtf close;
```

Why are 30jun10, 30jun2010, 063010 and 30.jun.10 all displayed as 18443?

Display 7.9: Result of reading dates into SAS variables

<table>
<thead>
<tr>
<th>Obs</th>
<th>indate1</th>
<th>indate2</th>
<th>indate3</th>
<th>indate4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18443</td>
<td>18443</td>
<td>18443</td>
<td>18443</td>
</tr>
</tbody>
</table>

Dates are stored in SAS as the number of days from a selected point in time.

- default origin of time for SAS is January 1, 1960.
- Thus, June 30, 2010 is 18443 days away from January 1, 1960.
• You may want to store dates numerically; however, displaying these dates numerically is not useful. To display the dates in a familiar format, assign one of the date formats to the variables containing the dates as we do in Display 7.10.

Display 7.10: Assigning date formats to all of the numeric variables in a data set

data;
  input @1 indate1 date7. @9 indate2 date9. @19 indate3 mmddyy. @26 indate4 ddmmyy8.;
  format _numeric_ date9.;
;
  datalines;
  30jun10 30jun2010 063010 30.06.10
;
  ods rtf;
  proc print;
  run;
  ods rtf close;

The *informs*, date7 date9 mmddyy and ddmmyy8, are used to read the dates in the data set in the data set in Display 7.10 while the date9 *format* is assigned to each of these variables.

Display 7.11: Output from PROC PRINT with dates appropriately formatted

<table>
<thead>
<tr>
<th>Obs</th>
<th>indate1</th>
<th>indate2</th>
<th>indate3</th>
<th>indate4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30JUN2010</td>
<td>30JUN2010</td>
<td>30JUN2010</td>
<td>30JUN2010</td>
</tr>
</tbody>
</table>

Why store dates numerically?

• You may want to construct variables from dates.

• For example, **DOB** and **DOD** are variables corresponding to dates of birth and death, respectively, of individuals in your study. If **DOB** and **DOD** are stored as numeric variables, then age at death could be calculated directly, \( \text{AGE AT DEATH} = (\text{DOD} - \text{DOB})/365.25 \).
Display 7.12: Various Date formats illustrated

```sas
data date_format_show;
    start = 0;
    put start date9.;
    today = 17700;  * days since Jan 1, 1960;
    put '-------------------------------';
    put 'DATE7. / DATE9.';
    put today date7.;
    put today date9.;
    put '-------------------------------';
    put 'DAY2. / DAY7.';
    put today day2.;
    put today day7.;
    put '-------------------------------';
    put 'EURDFDD8.';
    put today eurdfdd8.;
    put '-------------------------------';
    put 'MMDDYY8. / MMDDYY6.';
    put today mmddyy8.;
    put today mmddyy6.;
    put '-------------------------------';
    put 'WEEKDATE15. / WEEKDATE29.';
    put today weekdate15.;
    put today weekdate29.;
    put '-------------------------------';
    put 'WORDDATE12. / WORDDATE18.';
    put today worddate12.;
    put today worddate18.;
run;
```

Display 7.13: Results from SAS LOG of displaying date using different formats

<table>
<thead>
<tr>
<th>Date Format</th>
<th>Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE7. / DATE9.</td>
<td>01JAN1960</td>
</tr>
<tr>
<td>DAY2. / DAY7.</td>
<td>17JUN08</td>
</tr>
<tr>
<td>EURDFDD8.</td>
<td>17JUN2008</td>
</tr>
<tr>
<td>MMDDYY8. / MMDDYY6.</td>
<td>17JUN2008</td>
</tr>
<tr>
<td>WEEKDATE15. / WEEKDATE29.</td>
<td>Tue, Jun 17, 08</td>
</tr>
</tbody>
</table>
TIME also needs to be started as an elapsed number relative to some reference.

- In SAS, time is recorded as the number of seconds that have elapsed since midnight.
- Display 7.14 includes code to define three variables using time and date constants (the “t” and “d” behind the character expressions defines these variables as times and dates), and them displays the internal storage value and a formatting of this value.

Display 7.14: Time and date formats illustrated

```
data;
time_date_origin = 0;
nowtime = '09:00't;
today = '17jun2008'd;
put time_date_origin @20 time_date_origin
datetime13.;
put nowtime @20 nowtime time9.;
put today @20 today date9.;
```

As we see in Display 7.15, 9:00 corresponds to 9*60*60=32400 seconds since midnight, and 17 June 2008 corresponds to 17700 days since January 1, 1960.

Display 7.15: The origin of time and dates in SAS and other formatted values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>01JAN60:00:00</td>
</tr>
<tr>
<td>32400</td>
<td>9:00:00</td>
</tr>
<tr>
<td>17700</td>
<td>17JUN2008</td>
</tr>
</tbody>
</table>
Example: simple illustration reading date, time and currency data into a SAS data set

Suppose your data was in the following format …

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234567890123456789012345678901234567890</td>
<td>[column guides]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01/01/1960</td>
<td>01:00:00</td>
<td>$100.22</td>
<td></td>
</tr>
<tr>
<td>09/29/2003</td>
<td>09:49:59</td>
<td>$12693.79</td>
<td></td>
</tr>
</tbody>
</table>

Display 7.16: Reading date, time and currency data into SAS

```sas
data test;
  input @1 date MMDDYY10. @21 time TIME8. @31 money DOLLAR10.2;
datalines;
01/01/1960 01:00:00 $100.22
09/29/2003 09:49:59 $12693.79;

ODS RTF;
proc print;
title print of date and time w/o formatting – internal SAS representation;
  var date time money;
run;
proc print;
title print of date and time w/ formatting;
  var date time;
  format date MMDDYY10. time TIME8. money DOLLAR10.2;
run;
ODS RTF CLOSE;
```

Display 7.17: Output from PROC PRINT of data containing date, time and currency variables

<table>
<thead>
<tr>
<th>Obs</th>
<th>date</th>
<th>time</th>
<th>money</th>
<th>Obs</th>
<th>date</th>
<th>time</th>
<th>money</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>3600</td>
<td>100.22</td>
<td>1</td>
<td>01/01/1960</td>
<td>1:00:00</td>
<td>$100.22</td>
</tr>
<tr>
<td>2</td>
<td>15977</td>
<td>35399</td>
<td>12693.79</td>
<td>2</td>
<td>09/29/2003</td>
<td>9:49:59</td>
<td>$12,693.79</td>
</tr>
</tbody>
</table>
7.3. Recoding and transforming variables in a DATA step

Display 7.18: Data transformations before fitting a polynomial regression model

```sas
data nitrofen;
  infile '\\Muserver2\USERS\B\BAILERAJ\public.www\classes\sta402\data\ch2-dat.txt' firstobs=16 expandtabs missover pad ;
  input @17 conc 3. @49 total 2.;
  sqrt_total = sqrt(total);  * transformed response variable;
  cconc = conc - 157;        * construct mean-centered concentration;
  cconc2 = cconc*cconc;      * quadratic term;
ods rtf BODYTITLE;
ods graphics on;
proc reg data=nitrofen;
  model sqrt_total = cconc cconc2;  * fit the polynomial reg. model;
run;
ods graphics off;
ods rtf close;
```

Display 7.19: ODS statgraphics output from a polynomial regression

[Graphs depicting regression diagnostics: residual plots, predicted value against residual, predicted value against studentized residuals, studentized residuals against leverage, studentized residuals against observation number, normal probability plot of residuals, fit-mean residual plot, and a table of goodness-of-fit statistics.]
Boolean tricks to recode …

Option 1: IF-THEN

```sas
idino = 0;   * define the indicator of dinosaurs;
if (species="diplodoc" or species="tricerat" or species="brachios") then
  idino=1;
```

Option 2: Boolean evaluation

```sas
idino = (species="diplodoc" or species="tricerat" or species="brachios");
```

Display 7.20: Defining indicator variables for different treatments to fit an anova model

```sas
options nodate formdlim="-";
data meat;
  input condition $ logcount @@;
  iPlastic = (condition= "Plastic");
  iVacuum = (condition= "Vacuum");
  iMixed = (condition= "Mixed");
  iCO2 = (condition= "Co2");
datalines;
Plastic 7.66 Plastic 6.98 Plastic 7.80
Vacuum 5.26 Vacuum 5.44 Vacuum 5.80
Mixed 7.41 Mixed 7.33 Mixed 7.04
Co2 3.51 Co2 2.91 Co2 3.66
;
title "bacteria growth under 4 packaging conditions";
ODS RTF bodytitle;
proc print;
title "Print to check indicator variable construction";
  run;
proc reg data=meat;
title "Regression with indicator variables: alt. to one-way anova model";
  model logcount = iPlastic iVacuum iMixed;
  run;
proc glm data=meat;
title "One-way anova model";
  class condition;
  model logcount = condition;
  run;
ods rtf close;
```

Display 7.21: OUTPUT from PROC PRINT to confirm indicator variable coding

<table>
<thead>
<tr>
<th>condition</th>
<th>logcount</th>
<th>iPlastic</th>
<th>iVacuum</th>
<th>iMixed</th>
<th>iCO2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plastic</td>
<td>7.66</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Plastic</td>
<td>6.98</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>condition</td>
<td>logcount</td>
<td>iPlastic</td>
<td>iVacuum</td>
<td>iMixed</td>
<td>iCO2</td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
<td>----------</td>
<td>---------</td>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>Plastic</td>
<td>7.80</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Vacuum</td>
<td>5.26</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Vacuum</td>
<td>5.44</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Vacuum</td>
<td>5.80</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mixed</td>
<td>7.41</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Mixed</td>
<td>7.33</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Mixed</td>
<td>7.04</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Co2</td>
<td>3.51</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Co2</td>
<td>2.91</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Co2</td>
<td>3.66</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Example: Comparing coding when constructing categories (or why you test with missing values)

Display 7.22: Constructing indicator variables for different categories of a numeric variable

```sas
data toyexample;
  input literacy @@;
cat_literacy1 = 1*(0<literacy<=53) + 2*(53<literacy<=76)
  + 3*(76<literacy<=90) + 4*(90<literacy<=100);
cat_literacy2 = 1*(literacy<=53) + 2*(53<literacy<=76)
  + 3*(76<literacy<=90) + 4*(90<literacy<=100);
if ( (literacy NE .) AND (0<=literacy<=100) ) then
  cat_literacy3 = 1*(literacy<=53) + 2*(53<literacy<=76)
  + 3*(76<literacy<=90) + 4*(90<literacy<=100);
if ( (literacy EQ .) OR (100<literacy)
  OR (literacy<0) ) then cat_literacy4=.;
else if (liter <=53) then cat_literacy4=1;
else if (liter <=76) then cat_literacy4=2;
else if (liter <=90) then cat_literacy4=3;
else cat_literacy4=4;

datalines;
-99 25.55 73.7 83 99.9 107 .
;
ods rtf;
proc print;
run;
ods rtf close;
```

Display 7.23: Output from PROC PRINT for the different indicators

<table>
<thead>
<tr>
<th>Obs</th>
<th>liter</th>
<th>cat_literacy1</th>
<th>cat_literacy2</th>
<th>cat_literacy3</th>
<th>cat_literacy4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-99.00</td>
<td>0</td>
<td>1</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>25.55</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>73.70</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>83.00</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>99.90</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>107.00</td>
<td>0</td>
<td>0</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>7</td>
<td>.</td>
<td>0</td>
<td>1</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>
7.4. Ordering how tasks are done – precedence of operations

Display 7.24: DATA block illustration for order of operations comparisons

```sas
data preced_test;
  x1a = 3*2**2;
  x1b = (3*2)**2;
  x2a = 3-2/2;
  x2b = (3-2)/2;
  x3a = -2**2;
  x3b = (-2)**2;
  put '-------------------------';
  put '|' Order of operations   |';
  put '|' illustrated           |';
  put '-------------------------';
  put ' 3*2**2 = ' x1a;
  put '(3*2)**2 = ' x1b;
  put ' 3-2/2 = ' x2a;
  put '(3-2)/2 = ' x2b;
  put ' -2**2 = ' x3a;
  put ' (-2)**2 = ' x3b;
run;
```

Display 7.25: Output from the SAS LOG for the order of operations illustration

```
<table>
<thead>
<tr>
<th>Order of operations</th>
<th>illustrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>3*2**2 = 12</td>
<td></td>
</tr>
<tr>
<td>(3*2)**2 = 36</td>
<td></td>
</tr>
<tr>
<td>3-2/2 = 2</td>
<td></td>
</tr>
<tr>
<td>(3-2)/2 = 0.5</td>
<td></td>
</tr>
<tr>
<td>-2**2 = -4</td>
<td></td>
</tr>
<tr>
<td>(-2)**2 = 4</td>
<td></td>
</tr>
</tbody>
</table>
```

* Ultimately, there is a simple moral to this cautionary programming tale: use PARENTHESES when concerned that operations need to be conducted in a specific order.
7.5. What goes and what stays in a data set—KEEP, IF, WHERE, OUTPUT

- commands **DROP** and **KEEP** specify the variables to be eliminated from or retained in, respectively, a particular SAS dataset.

- statements **WHERE** and **IF** allow you to specify values of variables that you want to retain in a SAS data set. For example, the statement

  ```sas
  if literacy EQ .;
  ```

7.6. Structured thinking about writing programs – pseudo-code and modules

Two basic ideas are advocated here:

- first modularize your code

- write “pseudo-code” that will be operationalized as functioning programming statements during implementation.

Basic idea in Modular Programming: when faced with a large and complex task, subdivide this task into smaller components. These smaller components can then be implemented and tested before combined into a larger program. Often these components are separate modules that may be implemented as separate subroutines or functions; however, these may reflect discrete ideas that need to be implemented with a module as well. Variants of these subroutines or functions can be encountered in SAS Macro programs (Ch. 9) or in SAS/IML modules (Ch. 10).
• Pseudo-coding is a strategy for writing out in words or sentence fragments what should occur in a large programming task.

### 7.7. CASE STUDY 7.1: Is the two-sample t-test robust to heterogeneous variances?

**H₀:** \( \mu_1 = \mu_2 \),

T.S.: \( t_{stat} = \frac{\bar{Y}_1 - \bar{Y}_2}{s_p \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}} \) where \( s_p^2 = \frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2} \)

assumes that the observations from the two populations are independent and that the two populations are normally distributed with a common variance.

* In this case study, we conduct a Monte Carlo simulation to investigate the impact of unequal variance on the tests of mean equality when using the pooled-variance t-test. In particular, we investigate the operating characteristics of this test (Type I error rates, power).

As we implement a study of the study of heterogeneous variances on the pooled-variance t-test, let’s think about some of the tasks that need to be implemented to conduct this study. We need to:

• specify the conditions to be generated

• generate data sets reflecting these conditions

• calculate the test statistic

• accumulate results over numerous simulated data sets

Program may

• have input parameters (the conditions studied)

• produce output (a summary of the results over repeated simulated experiments)
• generate data
• calculate a test statistic

STEP 1: start with a program that includes only comments.

Display 7.30: Pseudo-code/comments for the t-test simulation

```sas
/* Problem: Explore whether t-test really is robust to violations of the equal variance assumption
   
   Strategy: See if the t-test operates at the nominal Type I error rate when the unequal variance assumption is violated
   
   */
   /* specify the conditions to be generated */
   /* generate data sets reflecting these conditions */
   /* calculate the test statistic */
   /* accumulate results over numerous simulated data sets */
```

STEP 2: identify variables that define simulation conditions

need to specify …

• number of simulated experiments (Nsims)

• sample sizes (N1, N2)

• population means (mu_1, mu_2)

• standard deviations (sig_1, sig_2)

• seed (myseed) for the random number generator so that we can generate the same sequence for our testing.

Display 7.31: Specifying the simulation conditions for the t-test simulation

```sas
/* Problem: Explore whether t-test really is robust to violations of the equal variance assumption
   
   Strategy: See if the t-test operates at the nominal Type I error rate when the unequal variance assumption is violated
   
   */
```
Statistical Programming in SAS

Bailer

Streamlined code for generating data and calculating the test statistic:

```sas
/* specify the conditions to be generated */
Nsims = 1;       * number of simulated experiments;
Myseed = 65432;  * specify seed for random number sequence;
N1 = 10;      * sample sizes from populations 1 and 2;
N2 = 10;
Mu_1 = 0;     * mean/sd of population 1;
Sig_1 = 1;
Mu_2 = 0;     * mean/sd of population 2;
Sig_2 = 1;

/* generate data sets reflecting these conditions */
* generate N1 observations ~ N(mu_1, sig_1^2)  ;
* generate N2 observations ~ N(mu_2, sig_2^2)  ;

/* calculate the test statistic */
/* accumulate results over numerous simulated data sets */
```

**STEP 3:** add the sample code for generating the data and the code for calculating the test statistic.

**Display 7.32:** Adding the data generation code to the t-test simulation

```sas
/* Problem: Explore whether t-test really is robust to
violations of the equal variance assumption
Strategy: See if the t-test operates at the nominal
Type I error rate when the unequal variance
assumption is violated */

Data simulate_2group_t;
Nosims = 1;       * number of simulated experiments;
Myseed = 65432;  * specify seed for random number sequence;
N1 = 10;      * sample sizes from populations 1 and 2;
N2 = 10;
Mu_1 = 0;     * mean/sd of population 1;
Sig_1 = 1;
Mu_2 = 0;     * mean/sd of population 2;
Sig_2 = 1;
```

21
do iexpt = 1 to Nsims;
/* generate data sets reflecting these conditions */
* generate N1 observations ~ N(mu_1, sig_1^2) ;
do ix = 1 to N1;
   group = 1;
   Y = mu_1 + sig_1*rannor(myseed);
   output;
end;
* generate N2 observations ~ N(mu_2, sig_2^2) ;
do ix = 1 to N2;
   group = 2;
   Y = mu_2 + sig_2*rannor(myseed);
   output;
end;
/* calculate the test statistic */
/* accumulate results over numerous simulated data sets */
end; /* of the do-loop over simulated experiments;*/
ods rtf bodytitle;
proc print;
run;
proc means;
var y;
class group;
run;
ods rtf close;

STEP 4: calculating the test statistic

* calculate using PROC TTEST (need to see what TTEST produces)

Display 7.33: Determining output objects produced by PROC TTEST

/* Problem: Explore whether t-test really is robust to violations of the equal variance assumption*/

Strategy: See if the t-test operates at the nominal Type I error rate when the unequal variance assumption is violated

*/

data simulate_2group_t;
Nsims = 1; /* number of simulated experiments;*/
Myseed = 65432; /* specify seed for random number sequence;*/
N1 = 10; /* sample sizes from populations 1 and 2;*/
N2 = 10;
Statistical Programming in SAS

Mu_1 = 0;     * mean/sd of population 1;
Sig_1 = 1;

Mu_2 = 0;     * mean/sd of population 2;
Sig_2 = 1;

do iexpt = 1 to Nsims;

/* generate data sets reflecting these conditions */

* generate N1 observations ~ N(mu_1, sig_1^2)  ;
do ix = 1 to N1;
  group = 1;
  Y = mu_1 + sig_1*rannor(myseed);
  output;
end;

* generate N2 observations ~ N(mu_2, sig_2^2)  ;
do ix = 1 to N2;
  group = 2;
  Y = mu_2 + sig_2*rannor(myseed);
  output;
end;

end;  * of the do-loop over simulated experiments;

/* calculate the test statistic */
ods rtf;
ods trace on/listing;
proc ttest data=simulate_2group_t; by iexpt;
class group;
var Y;
run;
ods trace off;
ods rtf close;

Display 7.34: ODS TRACE listing of output objects and the PROC TTEST output

Output Added:

<table>
<thead>
<tr>
<th>Name:</th>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label:</td>
<td>Statistics</td>
</tr>
<tr>
<td>Template:</td>
<td>Stat.TTest.Statistics</td>
</tr>
<tr>
<td>Path:</td>
<td>Ttest.ByGroup1.Y.Statistics</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>group</th>
<th>N</th>
<th>Mean</th>
<th>Std Dev</th>
<th>Std Err</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>0.1294</td>
<td>1.1156</td>
<td>0.3528</td>
<td>-1.7786</td>
<td>1.5453</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>0.4339</td>
<td>0.6476</td>
<td>0.2048</td>
<td>-0.4954</td>
<td>1.6836</td>
</tr>
<tr>
<td>Diff(1-2)</td>
<td>10</td>
<td>-0.3045</td>
<td>0.9121</td>
<td>0.4079</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Output Added:

<table>
<thead>
<tr>
<th>Name:</th>
<th>Conflimits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label:</td>
<td>Confidence Limits</td>
</tr>
</tbody>
</table>

23
Template: Stat.TTest.ConfLimits
Path: Ttest.ByGroup1.Y.ConfLimits

<table>
<thead>
<tr>
<th>group</th>
<th>Method</th>
<th>Mean</th>
<th>95% CL Mean</th>
<th>Std Dev</th>
<th>95% CL Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>0.1294</td>
<td>-0.6687</td>
<td>1.1156</td>
<td>0.7673</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>0.4339</td>
<td>-0.0294</td>
<td>0.8971</td>
<td>0.6476</td>
</tr>
<tr>
<td>Diff(1-2)</td>
<td>Pooled</td>
<td>-0.3045</td>
<td>-1.1615</td>
<td>0.9121</td>
<td>0.6892</td>
</tr>
<tr>
<td>Diff(1-2)</td>
<td>Satterthwaite</td>
<td>-0.3045</td>
<td>-1.1768</td>
<td>0.5679</td>
<td></td>
</tr>
</tbody>
</table>

Output Added:

---
Name: TTests
Label: T-Tests
Template: Stat.TTest.TTests
Path: Ttest.ByGroup1.Y.TTests

<table>
<thead>
<tr>
<th>Method</th>
<th>Variances</th>
<th>DF</th>
<th>t Value</th>
<th>Pr &gt;</th>
<th>t</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pooled</td>
<td>Equal</td>
<td>18</td>
<td>-0.75</td>
<td>0.4650</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Satterthwaite</td>
<td>Unequal</td>
<td>14.447</td>
<td>-0.75</td>
<td>0.4674</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Output Added:

---
Name: Equality
Label: Equality of Variances
Template: Stat.TTest.Equality
Path: Ttest.ByGroup1.Y.Equality

<table>
<thead>
<tr>
<th>Method</th>
<th>Num DF</th>
<th>Den DF</th>
<th>F Value</th>
<th>Pr &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Folded F</td>
<td>9</td>
<td>9</td>
<td>2.97</td>
<td>0.1208</td>
</tr>
</tbody>
</table>

Display 7.35: Displaying contents of data set created by ODS OUTPUT

```sas
ods output TTests=Out_TTests;
proc ttest data= simulate_2group_t; by iexpt;
  class group;
  var Y;
  run;
ods output close;
ods rtf bodytitle;
proc print;
  run;
ods rtf close;
```

Display 7.36: Contents of data set created by ODS OUTPUT

<table>
<thead>
<tr>
<th>Obs</th>
<th>iexpt</th>
<th>Variable</th>
<th>Method</th>
<th>Variances</th>
<th>tValue</th>
<th>DF</th>
<th>Probt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Y</td>
<td>Pooled</td>
<td>Equal</td>
<td>-0.75</td>
<td>18</td>
<td>0.4650</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Y</td>
<td>Satterthwaite</td>
<td>Unequal</td>
<td>-0.75</td>
<td>14.447</td>
<td>0.4674</td>
</tr>
</tbody>
</table>
STEP 5: put this all together . . .

Display 7.37: ODS output version of the t-test simulation program

```sas
/* Problem: Explore whether t-test really is robust to violations of the equal variance assumption */

data simulate_2group_t;
Nsims = 4000;    * number of simulated experiments;
Myseed = 65432;  * specify seed for random number sequence;
N1 = 10;        * sample sizes from populations 1 and 2;
N2 = 10;
Mu_1 = 0;       * mean/sd of population 1;
Sig_1 = 1;
Mu_2 = 0;       * mean/sd of population 2;
Sig_2 = 1;
do iexpt = 1 to Nsims;
  /* generate data sets reflecting these conditions */
  * generate N1 observations ~ N(mu_1, sig_1^2)  
  do ix = 1 to N1;
    group = 1;
    Y = mu_1 + sig_1*rannor(myseed);
    output;
  end;
  * generate N2 observations ~ N(mu_2, sig_2^2)  
  do ix = 1 to N2;
    group = 2;
    Y = mu_2 + sig_2*rannor(myseed);
    output;
  end;  * of the do-loop over simulated experiments;
  /* calculate the test statistic */
  /* Note: ODS TRACE was used to determine the output object containing the test statistics. This included the pooled-variance t-test and the Satterthwaite df approximation for the t-test allowing for unequal variances */
  ods output TTests=Out_TTests;
  proc ttest data= simulate_2group_t; by iexpt;
    class group;
    var Y;
    run;
  ods output close;
```

Observations:

- pooled variance procedure and the Satterthwaite procedure reject 4.78% and 4.65% of the time, respectively, when the null hypothesis was true and the data are from normal populations with homogeneous variances.

- Rejection rates were based on 4000 simulated experiments. The bound on the Monte Carlo error of estimation is approximately $B = 2 \sqrt{(0.05)(0.95)/4000} = 0.007$ suggesting that you would expect a procedure operating at $\alpha=0.05$ to possess an observed Type I error rate of between 0.043 and 0.057 of the cases when 4000 simulated experiments are considered.

- If we repeat this program changing $\text{sig}_2=2$ and holding all other variables constant, i.e. the second population has a variance that is $4 \times$ greater than the first population, we observe 5.33% and 4.83% Type I error rates in the pooled variance t-test and separate variance t-test, respectively.

- probably don’t want to have to change hard-coded values each time you want to investigate other simulation conditions. Macro variables are a nice solution to this quandary.
7.7a. CASE STUDY 7.1 (revisited using more DATA STEP programming): Is the two-sample t-test robust to heterogeneous variances?

- Alternative would be do all of the test statistic calculation as part of the DATA step.
- One change that we need to consider is the structure for producing the data.
- If we want to do all of the calculation in the DATA step, then the production of a separate data line for each observation is probably not the best way to proceed.

Display 7.38: Generating data as one record per simulated experiment

```sas
data simulate_2group_t;
Nsims = 4000;  * number of simulated experiments;
Myseed = 65432;  * specify seed for random number sequence;
N1 = 10;      * sample sizes from populations 1 and 2;
N2 = 10;
Mu_1 = 0;     * mean/sd of population 1;
Sig_1 = 1;
Mu_2 = 0;     * mean/sd of population 2;
Sig_2 = 1;
do iexpt = 1 to Nsims;
  * generate N1=10 observations ~ N(mu_1, sig_1^2) ;
   X1 = mu_1 + sig_1*rannor(myseed);
   X2 = mu_1 + sig_1*rannor(myseed);
   X3 = mu_1 + sig_1*rannor(myseed);
   X4 = mu_1 + sig_1*rannor(myseed);
   X5 = mu_1 + sig_1*rannor(myseed);
   X6 = mu_1 + sig_1*rannor(myseed);
   X7 = mu_1 + sig_1*rannor(myseed);
   X8 = mu_1 + sig_1*rannor(myseed);
   X9 = mu_1 + sig_1*rannor(myseed);
   X10 = mu_1 + sig_1*rannor(myseed);
  * generate N2=10 observations ~ N(mu_2, sig_2^2) ;
   Y1 = mu_2 + sig_2*rannor(myseed);
   Y2 = mu_2 + sig_2*rannor(myseed);
   Y3 = mu_2 + sig_2*rannor(myseed);
   Y4 = mu_2 + sig_2*rannor(myseed);
   Y5 = mu_2 + sig_2*rannor(myseed);
   Y6 = mu_2 + sig_2*rannor(myseed);
```

Y7 = mu_2 + sig_2*rannor(myseed);
Y8 = mu_2 + sig_2*rannor(myseed);
Y9 = mu_2 + sig_2*rannor(myseed);
Y10 = mu_2 + sig_2*rannor(myseed);

output;

/* calculate the t-statistic */

Display 7.39: Storing simulated data in arrays

data simulate_2group_t;
array x{10} x1-x10;   * storage for sample from population 1;
array y{10} y1-y10;   * storage for sample from population 2;

Nsims = 4000;    * number of simulated experiments;
Myseed = 65432;  * specify seed for random number sequence;

N1 = 10;      * sample sizes from populations 1 and 2;
N2 = 10;

Mu_1 = 0;     * mean/sd of population 1;
Sig_1 = 1;

Mu_2 = 0;     * mean/sd of population 2;
Sig_2 = 1;

do iexpt = 1 to Nsims;

* generate N1=10 observations ~ N(mu_1, sig_1^2) ;
do isample1 = 1 to N1;
   x{isample1} = mu_1 + sig_1*rannor(myseed);
end;

* generate N2=10 observations ~ N(mu_2, sig_2^2) ;
do isample2 = 1 to N2;
   y{isample2} = mu_2 + sig_2*rannor(myseed);
end;

output;

/* calculate the t-statistic */
Display 7.40: Pseudo-code/comments for the t-test simulation in DATA

```sas
/* calculate the t-statistic */
* >>>> calculate sample means and variances ;
* >>>> calculate pooled variance and t-statistic ;
* >>>> calculate P-value ;
```

Display 7.41: Simulation code for studying the t-test using only DATA step programming

```sas
data simulate_2group_t;
array x{10} x1-x10;   * storage for sample from population 1;
array y{10} y1-y10;   * storage for sample from population 2;
Nsims = 4000;    * number of simulated experiments;
Myseed = 65432;  * specify seed for random number sequence;
N1 = 10;      * sample sizes from populations 1 and 2;
N2 = 10;
Mu_1 = 0;     * mean/sd of population 1;
Sig_1 = 1;
Mu_2 = 0;     * mean/sd of population 2;
Sig_2 = 1;
	do iexpt = 1 to Nsims;
* generate N1=10 observations ~ N(mu_1, sig_1^2)  ;
		do isample1 = 1 to N1;
			x{isample1} = mu_1 + sig_1*rannor(myseed);
		end;
* generate N2=10 observations ~ N(mu_2, sig_2^2)  ;
		do isample2 = 1 to N2;
			y{isample2} = mu_2 + sig_2*rannor(myseed);
		end;
/* calculate the t-statistic */
* >>>> calculate sample means and variances ;
	xbar = mean(of x1-x10);
	ybar = mean(of y1-y10);
	xvar = var(of x1-x10);
	yvar = var(of y1-y10);
* >>>> calculate pooled variance and t-statistic ;
	s2p = (9*xvar + 9*yvar)/18;

tstat = (xbar-ybar)/sqrt(s2p*(2/10));
* >>>> calculate P-value ;
Pvalue = 2*(1-CDF('t',abs(tstat),18));
Reject05 = (Pvalue <= 0.05);
```
7.8. **CASE STUDY 7.2: Monte Carlo integration to estimate an integral**

Display 7.42: pseudo-code for the Monte Carlo integration

```plaintext
/* generate data [x, f(x)] in a rectangle containing the f(x) = density for N(0,1) */

/* determine proportion of points that lie below f(x) */

/* derive the area estimate and place a bound on the error of estimation */
```

- dimensions of the rectangular over which we generate the data? The interval on the x-axis is easy – [0, 1.645]. The interval for the y-axis is not hard but you need to think a little.

The standard normal density takes the form: \( f(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}x^2\right) \). The maximum of this function occurs at \( x=0 \) so the interval for the y-axis should range from 0 to \( \frac{1}{\sqrt{2\pi}} = 0.399 \). So, we generate an (x,y) point in the rectangle with lower left corner [0,0] and upper right corner [1.645, 0.400]

- To estimate the area, first determine the proportion of randomly generated points that lie below the density and then multiply this proportion by the area. In other words, the area under the curve = \{area of the rectangle\} \times \{proportion of randomly generated points below the density\} = \{ 1.645 \times (0.400) \} \times \{ \frac{\# \text{points with } y \leq f(x)}{\# \text{(x, y) points randomly generated}} \}.

- Monte Carlo error – by using the ideas for inference for a single population proportion.

The \( 100 \times (1-\alpha)\% \) confidence interval for a proportion is \( p \pm z_{\alpha/2} \sqrt{\frac{p(1-p)}{n}} \) where “p” is the observed proportion based on “n” trials.
Display 7.43: Expansion of the program to do Monte Carlo integration (pseudo-code)

| REPEAT { |
| /* generate data \([x, f(x)]\) in a rectangle containing the |
| \[f(x) = \text{density for } N(0,1)\] */ |
| \[x = 1.645*\text{ranuni(seed1)};\] |
| \[y = 0.400*\text{ranuni(seed1)};\] |
| /* determine proportion of points that lie below \(f(x)\) */ |
| \[i\_\text{under} = (y<= (1/\sqrt{2*\pi})*\exp(-x*x/2);\] |
| } |
| \[p\_\text{est} = \text{sum}(i\_\text{under})/\text{number of simulated points};\] |
| /* derive the area estimate and place a bound on the |
| error of estimation */ |
| \[\text{AUC}\_\hat{\text{hat}} = \text{Area}\_\text{Rectangle} * p\_\text{est};\] |
| \[\text{SE}\_\text{AUC}\_\hat{\text{hat}} = \text{Area}\_\text{Rectangle} * \sqrt{p\_\text{est} * (1-p\_\text{est}) / \text{n}\_\text{pts};}\] |
| \[\text{LCL} = \text{AUC}\_\hat{\text{hat}} - \text{zmult} * \text{SE AUC}\_\hat{\text{hat}};\] |
| \[\text{UCL} = \text{AUC}\_\hat{\text{hat}} + \text{zmult} * \text{SE AUC}\_\hat{\text{hat}};\] |
| /* write out results */ |

Display 7.44: Program to generate a Monte Carlo estimate of \(P(0<Z<1.645)\)

| data area_est; |
| retain n\_under 0; * initialize counter; |
| seed1 = 98765; |
| const = 1/\sqrt{2*\arccos(-1)}; * bring constant calc. outside loop; |
| n\_pts = 4000; |
| zmult = 1.96; * 95% Confidence interval requested; |
| Area\_rectangle = 1.645*.400; |
| do ii = 1 to n\_pts; * REPEAT { . . . ; |
| /* generate data \([x, f(x)]\) in a rectangle containing the |
| \[f(x) = \text{density for } N(0,1)\] */ |
| \[x = 1.645*\text{ranuni(seed1)};\] |
| \[y = 0.400*\text{ranuni(seed1)};\] |
| /* determine proportion of points that lie below \(f(x)\) */ |
| \[i\_\text{under} = (y<= \text{const} * \exp(-x*x/2);\] |
| \[n\_\text{under} = n\_\text{under} + i\_\text{under};\] |
| \[p\_\text{est} = n\_\text{under}/ii;\] |
| /* derive the area estimate, SE and CI */ |
| \[\text{AUC}\_\hat{\text{hat}} = \text{Area}\_\text{Rectangle} * p\_\text{est};\] |
| \[\text{SE AUC}\_\hat{\text{hat}} = \text{Area}\_\text{Rectangle} * \sqrt{p\_\text{est} * (1-p\_\text{est}) / \text{n}\_\text{pts};}\] |
| \[\text{LCL} = \text{AUC}\_\hat{\text{hat}} - \text{zmult} * \text{SE AUC}\_\hat{\text{hat}};\] |
| \[\text{UCL} = \text{AUC}\_\hat{\text{hat}} + \text{zmult} * \text{SE AUC}\_\hat{\text{hat}};\] |
| output; |
| end; * . . . } ; |
| /* write out the results */ |
| put "Area est. = " \text{AUC}\_\hat{\text{hat}}; |
| put "(SE = " \text{SE AUC}\_\hat{\text{hat}} "); |
| put "CI: [" \text{LCL }," \text{UCL } "]; |
| /* create annotate data set for adding vertical reference line and text */ |
data add_horiz_ref;
   retain xsys ysys "2" function;
   length function $6;
   function= "move";  x=0; y=0.45; output;
   function= "draw";  x=3500; y=0.45; output;
   function= "label"; x=1500; y=0.455;
       text="Pr(0<Z<1.645)=0.45";
       output;
   x=3000; y=0.4325; text="LCL"; output;
   x=3000; y=0.4425; text="point est."; output;
   x=3000; y=0.4525; text="UCL"; output;
run;

ods rtf bodytitle;
/* generate plot of estimate of P(0<Z<1.645) plus point-wise CI */
goptions reset=global;
symbol i=join;
axis1 order=0.38 to 0.48 by 0.01 label=(angle=90 "Estimated Area");
axis2 order=0 to 3500 by 500 label=('Number of simulated data points");
proc gplot data=area_est;
   plot AUC_hat*ii LCL*ii UCL*ii/vaxis=axis1 haxis=axis2 overlay
       annotate=add_horiz_ref;
   where 10 <= ii <=3500;
run;

/* set up formats and labels for point of randomly generated points */
proc format;
   value underfmt 1="Under curve"
                   0="Over curve";
   goptions reset=global;
   axis1 label=(angle=90 "Randomly generated Y-coordinate");
   axis2 label=('Randomly generated X-coordinate");
symbol1 value=$ cv=black;
symbol2 value=dot cv=black;
proc gplot data=area_est;
   plot y*x=iunder / vaxis=axis1 haxis=axis2;
   format iunder underfmt.;
run;
ods rtf close;

Display 7.45).

Display 7.45: Output from SAS LOG: area estimate, standard error and confidence interval for the Monte Carlo integral estimate

<table>
<thead>
<tr>
<th>Area est. = 0.4443145</th>
</tr>
</thead>
<tbody>
<tr>
<td>(SE = 0.0048719495)</td>
</tr>
<tr>
<td>CI: [0.4347654791 ,0.4538635209]</td>
</tr>
</tbody>
</table>

Here, the estimate of the P(0 < Z < 1.645) is 0.444 (95% CI: 0.435, 0.454).
Plot of these quantities from the 10th to the 2000th point is produced in Display 7.46.

Display 7.46: Plot of estimated $P(0<Z<1.645)$ along with pointwise CI vs. number of simulated data points. Horizontal reference line at 0.45 is included.

Display 7.47: Plot of the simulated points below the normal density (shaded circles) or above the normal density (open diamonds).
7.9. CASE STUDY 7.3: Simple percentile-based bootstrap

- Bootstrap is arguably one of the most important tools to be added to the data analyst’s toolkit in recent memory (see, e.g. Efron and Tibshirani 1993).

- Key idea in this method is that if the sample distribution is a reasonable approximation to the population distribution, then you could approximate the sampling distribution of a statistic by sampling from the sample/data distribution.

- For each resample, you calculate the test statistic of interest.

- Distribution of this test statistic over bootstrap resamples can be used to construct a variety of quantities including standard error estimates and confidence intervals.

- Simplest form of a confidence interval construction is to simply use appropriate quantiles of the bootstrap distribution to serve as a confidence interval. For example, a 90% percentile-based confidence interval would be constructed from the 5th and 95th percentiles of the bootstrap distribution. (Other more complicated constructions of the bootstrap CIs may be preferred, e.g. BCa, can be constructed, and you are encouraged to learn more about this.)

Display 7.48: Psedo-code/comments for constructing the percentile-based bootstrap CI

```bash
/* input the data */

/* construct the t-based confidence interval for the mean response */

/* calculate the bootstrap-based CI for the mean response */

* generate bootstrap resamples of the data vector;

* calculate the mean for each resample;

* select the 5th and 95th percentiles from the bootstrap distribution of the mean;
```
Display 7.49: Constructing the percentile-based bootstrap CI for a population mean

/* input the data */
/* From "cars" data set found on http://lib.stat.cmu.edu. Note: Only mpg data from 1982 cars considered. */
data in_data;	input mpg @@;
datalines;
28 27 34 31 29 27 24 23 36 37 31 38 36 36 34 38 32 38 25 38 26 22 32 36
27 27 44 32 28 31;

ODS RTF bodytitle;
/* construct the t-based confidence interval for the mean response */
proc tabulate data=in_data alpha=0.10;
var mpg;
table mpg, LCLM UCLM; * 90% CI requested for the mean MPG;
run;

/* calculate the bootstrap-based CI for the mean response */
data boot_data;
array mpg{31} mpg1-mpg31;
array bmpg{31} bmpg1-bmpg31;
input mpg1-mpg31;
* generate 4000 bootstrap resamples of the 31 element data vector;
do i=1 to 4000;
do ii = 1 to 31;
ipick = int(31*ranuni(27549)+1);
bmpg(ii) = mpg(ipick);
end;
boot_mean = mean(of bmpg1-bmpg31); * calculate the test statistic;
keep boot_mean;
output boot_data;
end;
datalines;
28 27 34 31 29 27 24 23 36 37 31 38 36 36 34 38 32 38 25 38 26 22 32 36
27 27 44 32 28 31;
* select the 5th and 95th %tiles from the bootstrap distrib. of the mean;
proc tabulate data=boot_data;
var boot_mean;
table boot_mean, P5 P95;
run;
proc gchart data=boot_data; * histogram of the bootstrap means;
vbar boot_mean;
run;
ODS RTF close;

The 90% t-based confidence interval for the mean response was 30.07 to 33.35 while the percentile-based bootstrap confidence interval for the mean response was 30.10 to 33.26.
7.10. **Throw out your tables of statistical distributions – CDF, PDF, QUANTILE**

- Quantiles from particular reference distributions in order to define rejection regions or to determine multipliers of standard errors for constructing confidence intervals.

- May need to look up areas under certain distributions to construct P-values.

- Tables of quantiles and probabilities often occupied the inside pages and appendices of introductory statistics books.

- The good news is that these tables can be replaced with a good use of functions available in SAS, namely the CDF and QUANTILE functions.

- The CDF function is used to determine the Pr(Y ≤ c) where Y is a random variable following one of many distributions that are implemented in SAS. For example, continuous distributions such as uniform, beta, chi-square, F, gamma, normal, t, and log-normal are included as are discrete distributions such as Bernoulli, binomial, Poisson and negative binomial are also included. The general syntax of the CDF function is `CDF('distribution-name', c, parameter-list-sep-commas).`

- The CDF function is a nice general way of requesting lower-tail probabilities. As an aside, SAS had a number of different CDF-named functions that still work in SAS 9.2. For example, `CDF('normal',c)` is equivalent to `probnorm(c)` and `CDF('t',c)` is equivalent to `probt(c)`.

**Display 7.50: CDF function in SAS**

```sas
data cdf_examples;
/* Z ~ N(0,1) table values */
 norm_area_left = cdf("Normal", -1.645);

 norm_area_right = 1-cdf("Normal", -1.645); /* area above -1.645 under N(0,1); */

/* T ~ t(df) table values */
t_area_left_06 = cdf("T", -1.645, 6); /* area <= -1.645 for t(df=6); */
t_area_left_60 = cdf("T", -1.645, 60); /* area <= -1.645 for t(df=60); */
t_area_left_600 = cdf("T", -1.645, 600); /* area <= -1.645 for t(df=600); */

/* Pr(Y<=m) for Y ~ binomial(m(successes), p=prob of success=0.5, n=trials=4) */
bin_cdf_0 = CDF('binomial', 0, 0.50, 4);
bin_cdf_1 = CDF('binomial', 1, 0.50, 4);
bin_cdf_2 = CDF('binomial', 2, 0.50, 4);
```

36
bin_cdf_3 = CDF('binomial', 3, 0.50, 4);
bin_cdf_4 = CDF('binomial', 4, 0.50, 4);

p0 = bin_cdf_0; /* Pr(Y=m) for Y ~ binomial(p=0.5, n=4) */
p1 = bin_cdf_1 - bin_cdf_0;
p2 = bin_cdf_2 - bin_cdf_1;
p3 = bin_cdf_3 - bin_cdf_2;
p4 = bin_cdf_4 - bin_cdf_3;

ods rtf bodytitle;
proc print;
  run;
ods rtf close;

Display 7.51: Output from PROC PRINT of normal and binomial CDF function in SAS

<table>
<thead>
<tr>
<th>norm_area_left</th>
<th>norm_area_right</th>
<th>df</th>
<th>t_area_left_06</th>
<th>t_area_left_60</th>
<th>t_area_left_600</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.049985</td>
<td>0.95002</td>
<td>6</td>
<td>0.075536</td>
<td>0.052600</td>
<td>0.050247</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bin_cdf_0</th>
<th>bin_cdf_1</th>
<th>bin_cdf_2</th>
<th>bin_cdf_3</th>
<th>bin_cdf_4</th>
<th>p0</th>
<th>p1</th>
<th>p2</th>
<th>p3</th>
<th>p4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0625</td>
<td>0.3125</td>
<td>0.6875</td>
<td>0.9375</td>
<td>1</td>
<td>0.0625</td>
<td>0.25</td>
<td>0.375</td>
<td>0.25</td>
<td>0.0625</td>
</tr>
</tbody>
</table>

Observation: difference in CDFs was used to calculate the probability of the 5 possible values of a binomial random variable when 4 trials were conducted.

- Calculation of the probability mass function is unnecessary given other functions available in SAS.

- In fact, SAS provides a general function for returning the value of the probability density function (pdf) of a continuous random variable or the probability mass function (pmf) of a discrete random variable using the PDF function.

- General syntax of this function is PDF('distribution-name', c, parameter-list-sep-commas). A large collection of distributions can be employed here as well. Display 7.52 contains the program to calculate Pr(Y=m) for m=0,1,2,3,4 for Y~binomial(n=4, π=0.5) and the value of the standard normal density at z=0 using the PDF function.
Display 7.52: Program illustrating PDF/PMF function use in SAS

```sas
data pdf_examples;
/* Pr(Y=m) for Y ~ binomial(p=prob of success=0.5, n=trials=4) */
bin_0 = PDF('binomial', 0, 0.50, 4);
bin_1 = PDF('binomial', 1, 0.50, 4);
bin_2 = PDF('binomial', 2, 0.50, 4);
bin_3 = PDF('binomial', 3, 0.50, 4);
bin_4 = PDF('binomial', 4, 0.50, 4);
/* Z ~ N(0,1) value of phi(0) */
normal_density_0 = pdf("Normal", 0);
ods rtf bodytitle;
proc print;
run;
ods rtf close;
```

Aside: display includes the binomial P(Y=m) and the standard normal density evaluated at z=0, i.e. $\frac{1}{\sqrt{2\pi}} = 0.399$.

Display 7.53: Output from PROC PRINT of normal and binomial PDF function in SAS

<table>
<thead>
<tr>
<th>bin_0</th>
<th>bin_1</th>
<th>bin_2</th>
<th>bin_3</th>
<th>bin_4</th>
<th>normal_density_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0625</td>
<td>0.25</td>
<td>0.375</td>
<td>0.25</td>
<td>0.0625</td>
<td>0.39894</td>
</tr>
</tbody>
</table>
Example: generate a graphical comparison of a t(df=4) with a standard normal density.

Display 7.54: Comparing a standard normal density to a t4.

```sas
data t_vs_z;
  do y= -3.5 to 3.5 by .001;
    t_density = PDF('t', y, 4);
    z_density = PDF('Normal', y);
    output;
  end;

data add_text;
  retain xsys ysys "2" function;
  length function $ 6;
  function= "label"; x=2.75; y=0.06; text="t[df=4]"; output;
  x=1.0; y=0.28; text="Z"; output;
run;

goptions reset=global;
axis1 label=(angle=90 "Density");
axis2 label=("Value");
symbol1 interpol=join color=black line=2 width=1.5;
symbol2 interpol=join color=black line=1 width=1.5;
ods rtf bodytitle;
proc gplot data=t_vs_z;
  plot t_density*y z_density*y / overlay vaxis=axis1 haxis=axis2 annotate=add_text;
run;
ods rtf close;
```

Display 7.55: Plot comparing a standard normal density to a t4.
• final distributional function is noteworthy, namely the QUANTILE function.

• This function returns the value of a random variable associated some lower-tail probability.

• Given a particular cumulative distribution function \( F() \) and its associated parameters, this function will return the value of \( x \) such that \( F(x) = \text{probability} \). The syntax of this function is \( \text{QUANTILE('distribution-name', probability, parameter-list-sep-commas)} \) where the parameter list varies with the distribution invoked.

Display 7.56: QUANTILE function in SAS DATA Step programming

```sas
data quant_calc;
* z examples ;
zq_50 = QUANTILE('Normal',0.50);
zq_90 = QUANTILE('Normal',0.90);
zq_95 = QUANTILE('Normal',0.95);
zq_975 = QUANTILE('Normal',0.975);
put "Z: 50th percentile = " @25 zq_50;
put "Z: 90th percentile = " @25 zq_90;
put "Z: 95th percentile = " @25 zq_95;
put "Z: 97.5th percentile = " @25 zq_975;

* binomial examples;
binq_50 = QUANTILE('Binomial',0.50,.50,4);
binq_90 = QUANTILE('Binomial',0.90,.50,4);
binq_95 = QUANTILE('Binomial',0.95,.50,4);
binq_975 = QUANTILE('Binomial',0.975,.50,4);
put "Binomial: 50th percentile = " @35 binq_50;
put "Binomial: 90th percentile = " @35 binq_90;
put "Binomial: 95th percentile = " @35 binq_95;
put "Binomial: 97.5th percentile = " @35 binq_975;
run;
```

Display 7.57: Log with results of using the QUANTILE function

```
Z: 50th percentile =  -1.15194E-17
Z: 90th percentile =  1.2815515655
Z: 95th percentile =  1.644853627
Z: 97.5th percentile = 1.9599639845

Binomial: 50th percentile = 2
Binomial: 90th percentile = 3
Binomial: 95th percentile = 4
Binomial: 97.5th percentile = 4
```
Example: Quantiles for a t-distribution with 4 degrees of freedom and a z-distribution are calculated for probabilities from 0.01 to 0.99 by 0.01.

Display 7.58: SAS code to produce a quantile-quantile plot of a $t_4$ vs. standard normal

```
data t_vs_z_quant;
  do prob = .01 to .99 by .01;
    t_quantile = QUANTILE('t', prob, 4);
    z_quantile = QUANTILE('Normal', prob);
    output;
  end;

  goptions reset=global;
  axis1 label=(angle=90 "t Quantile") order=(-4 to 4 by 1) length=3.4in;
  axis2 label="Z Quantile" order=(-4 to 4 by 1) length=3.4in;
  symbol1 interpol=join color=black line=1 width=1.5;

  ods rtf bodytitle;
  proc gplot data=t_vs_z_quant;
    plot t_quantile*z_quantile / vaxis=axis1 haxis=axis2;
  run;
  ods rtf close;
```

Display 7.59: Output containing QQplot of a $t_4$ vs. standard normal

The curvature in this display is indicative of the heavier tails encountered in the $t_4$ vs. standard normal distribution, e.g. $P(Z \leq -2) \approx P(t_4 \leq -3)$.
Comments:

- As with the CDF, previous incarnations of SAS included (and still include) a collection of specific functions to calculate quantiles such as PROBIT (for standard normal) and TINV (for t-distribution).

- With the advent of QUANTILE, these other quantile functions are unnecessary.

- Other SAS functions of possible interest that are associated with distributions include survival functions (SDF) and functions that return the log of the CDF/PDF/SDFs (logLOGCDF, LOGPDF, LOGSDF).
7.11. Generating variables using random number generators – RANUNI, RANNOR, RAND

Historically, SAS functions for generating random deviates from normal distributions were of the form

```
"ran####"
```

where #### denotes some distribution such as binomial (#### = BIN), Cauchy (####=CAU), Exponential (####=EXP), Poisson (####=POI) and Triangular (####=TRI) to name a few of the options. In addition, a random deviate can be generated from a probability distribution specified as a table (rantbl).

Currently, the single function RAND may be preferred - syntax of this function is

```
RAND('distribution-name', parameter-list-sep-commas)
```

where the parameter list varies with the distribution invoked. The seed for the RAND invocation is set using the STREAMINT routine.
Example: Triangular random deviate generated and displayed

Display 7.60: Generating and displaying a triangular random deviate

```sas
data triangular;
  call streaminit(34567);
  do inum = 1 to 1400;
    mynum = RAND('Triangle', 0.70);
    * h=0.70 is a parameter of the triag. distn. ;
    output;
  end;
  goptions reset=global;
  axis1 label=(angle=90 "Frequency");
  axis2 label=('Triangular deviate');
  ods rtf bodytitle;
  proc gchart data=triangular;
    vbar mynum / raxis=axis1 maxis=axis2;
  run;
  ods rtf close;
```

Display 7.61: Histogram of 1400 random triangular deviates
Example: rolling two dice.

The probability of rolling a “6” is implicit from the specification of the other 5 probabilities, i.e. 
\[ p(6) = 1 - [p(1) + \ldots + p(5)] \].

Display 7.62: SAS code to simulate rolling a pair of fair 6-sided dice

```
data diceroll;   * rolling two 6-sided balanced dice;   
call streaminit(34567); 
do inum = 1 to 6000;  * 6000 rolls . . . ; 
die1 = RAND('Table', 1/6, 1/6, 1/6, 1/6, 1/6);   
die2 = RAND('Table', 1/6, 1/6, 1/6, 1/6, 1/6);   
sum7plus = (die1+die2)>=7;  * sum >=7;   
output;   
end;   
ods rtf bodytitle;   
proc freq;   
   table die1 / nocum nocum testf=(1000,1000,1000,1000,1000,1000);   
   table sum7plus;   
run;   
ods rtf close; 
```

Display 7.63 contains the relative frequency of each die face along with the result of the X² 
goodness of fit test suggesting that these data are consistent with the die 1 being fair (X²=0.6020, 
df=5, P-value<0.9879).

Display 7.63: Frequently of results from the 6000 rolls of the first die

<table>
<thead>
<tr>
<th>die1</th>
<th>Frequency</th>
<th>Test Frequency</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>998</td>
<td>1000</td>
<td>16.63</td>
</tr>
<tr>
<td>2</td>
<td>985</td>
<td>1000</td>
<td>16.42</td>
</tr>
<tr>
<td>3</td>
<td>998</td>
<td>1000</td>
<td>16.63</td>
</tr>
<tr>
<td>4</td>
<td>1002</td>
<td>1000</td>
<td>16.70</td>
</tr>
<tr>
<td>5</td>
<td>998</td>
<td>1000</td>
<td>16.63</td>
</tr>
<tr>
<td>6</td>
<td>1019</td>
<td>1000</td>
<td>16.98</td>
</tr>
</tbody>
</table>