Chapter 8 - Programming in a DATA step

8.1. Storage bins for collections of values - ARRAYS
   8.1.1 Defining values in ARRAY variable list directly
   8.1.2 Inputting values in ARRAY variable list
   8.1.3 Changing missing value codes for numeric variables to “.”
   8.1.4 Recoding missing values for all numeric and character variables
   8.1.5 Creating multiple observations from a single record

8.2 Case Study 1: Monte Carlo P-value for test of spatial randomness [omit]

8.3 Remembering variable values across observations – RETAIN
   8.3.1 Processing multiple observations for an individual

8.4 Case Study 2: Randomization test for the equality of two populations [omit]

Summary
References
Exercises

8.1. Storage bins for collections of values - ARRAYS

Array = structure that is often used to contain a collection of variables.

Basic syntax for defining an array: ARRAY arrayname{number of array elements} variable-list. (Aside: {}, [ ] and ( ) can all be used in array definitions.)

8.1.1 Defining values in ARRAY variable list directly

Display 8.1: Converting temperature scales using arrays

data temps;
   array tempF(4) tempF1-tempF4 (32,50,68,86);
   array tempC(4) tempC1-tempC4;
   do itemp = 1 to 4;
      tempC(itemp) = 5/9*(tempF(itemp)-32);
   end;
   drop itemp;
proc print;
The printout from Display 8.1 includes the following output line:

<table>
<thead>
<tr>
<th>Obs</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32</td>
<td>50</td>
<td>68</td>
<td>86</td>
<td>0</td>
<td>10</td>
<td>20</td>
<td>30</td>
</tr>
</tbody>
</table>

8.1.2 Example 8.2: Inputting values in ARRAY variable list
Reading variable values into an array from datalines or an external file.

To illustrate, suppose the number of activities of daily living (ADL) that each resident of a nursing home could conduct was measured during 7 consecutive quarters.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Data D1;
ARRAY ADL{7} ADL1-ADL7;
  input ADL1-ADL7;
datalines;
6 6 5 5 5 4 3
;

Can also have the size of the array defined by counting the elements in the array by using the asterisk in “[*].”

Data D2;
ARRAY ADL{*} ADL1-ADL7;
  input ADL1-ADL7;
datalines;
6 6 5 5 5 4 3
;

Can define an array in terms of a collection of variable names. This is illustrated in the code snippet:

Data D3;
ARRAY ADL{*} t1 t2 t3 t4 t5 time6 time_7;
  input t1 t2 t3 t4 t5 time6 time_7;
8.1.3 Changing missing value codes for numeric variables to “.”

Suppose -999 was the missing value code in our ADL study. We want to replace all occurrences of “-999” with “.” before doing further analyses.

Display 8.2: Recoding missing data values in numeric variables in arrays

```sas
data D4;
array ADL{*} t1 t2 t3 t4 t5 time6 time_7;
input t1 t2 t3 t4 t5 time6 time_7;
D0 ielement = 1 to 7;
    if ADL{ielement}=-999 then ADL{ielement}=.;
END;
 dataplines;
6 6 5 5 5 4 3
7 -999 4 4 3 -999 2 ;
proc print;
title "Recoding missing values using Arrays using DO loop";
run;
```

The printout from Display 8.2 include lines

<table>
<thead>
<tr>
<th>Obs</th>
<th>t1</th>
<th>t2</th>
<th>t3</th>
<th>t4</th>
<th>t5</th>
<th>time6</th>
<th>time_7</th>
<th>ielement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>.</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>.</td>
<td>.</td>
<td>2</td>
</tr>
</tbody>
</table>

The list of variables included in an ARRAY can be implicitly defined as all numeric variables or all character variables or as all variables using the _NUMERIC_, _CHARACTER_ and _ALL_ options to ARRAY, respectively. The example below illustrates the power of this construction.
8.1.4 Recoding missing values for all numeric and character variables

Display 8.3: Recoding missing values in numeric and character variables using arrays

```
Data D5;
  input name $ sex $ t1 t2 t3 t4 t5 time6 time_7;
  ARRAY num_array{*} _NUMERIC_;
  ARRAY char_array{*} _CHARACTER_;

  /* recode the numeric variables */
  DO inum = 1 to dim(num_array);
    if num_array{inum}=-999 then num_array{inum}=.;
  END;

  /* recode the character variables */
  Do ichar = 1 to dim(char_array);
    if char_array{ichar}="-999" then char_array{ichar}=" ";
  END;

drop inum ichar;
datalines;
MrSmith -999 6 6 5 5 5 4 3
-999 F 7 -999 4 4 3 -999 2
;
proc print;
title "Recoding missing values using Arrays using DO loop";
run;
```

This program produces the following output:

```
Recoding missing values using Arrays using DO loop
Obs  name  sex  t1  t2  t3  t4  t5  time6  time_7
 1  MrSmith  sex  t1  t2  t3  t4  t5  time6  time_7
   1  MrSmith  F  6  6  5  5  5  4  3
   2                7 .  4  4  3 .  2
```

8.1.5 Creating multiple observations from a single record

Different procedures in SAS may expect data to have different structures.
Some of the multivariate procedures expect all of the variables to be defined in a single record.

For mixed models applied to longitudinal data, different measurements are expected to appear in different records.

It is important to be able to program the expansion of a single record to multiple records or the condensing of multiple records into a single record. We focus on the first of these tasks here.

Suppose we have data in the following form:

<table>
<thead>
<tr>
<th>Name</th>
<th>Sex</th>
<th>t1</th>
<th>t2</th>
<th>t3</th>
<th>t4</th>
<th>t5</th>
<th>time6</th>
<th>time_7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>M</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Jones</td>
<td>F</td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Fisher</td>
<td>M</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

that we wish to transform into

<table>
<thead>
<tr>
<th>Name</th>
<th>Sex</th>
<th>Time</th>
<th>ADL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>M</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Smith</td>
<td>M</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Smith</td>
<td>M</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Smith</td>
<td>M</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Smith</td>
<td>M</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Smith</td>
<td>M</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Smith</td>
<td>M</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>Jones</td>
<td>F</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>Jones</td>
<td>F</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Jones</td>
<td>F</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Jones</td>
<td>F</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Jones</td>
<td>F</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Jones</td>
<td>F</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Jones</td>
<td>F</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>Fisher</td>
<td>M</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Fisher</td>
<td>M</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Fisher</td>
<td>M</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Fisher</td>
<td>M</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Fisher</td>
<td>M</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Fisher</td>
<td>M</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Fisher</td>
<td>M</td>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>

Display 8.4: Expanding a single record into multiple records
What would happen if you ran this code without the `keep` statement?

### 8.2 Case Study: Monte Carlo P-value for test of spatial randomness [OMIT]

* P-values, the probability of observing a test statistic at least as extreme as the observed value of the test statistic when the null hypothesis is true, is commonly reported in statistical inference. * When the test statistic has a familiar form, standard statistical distributions (e.g. t, F, etc.) can be used to calculate this probability.

* In other cases, it might be very difficult to calculate a P-value if the test statistic does not have a familiar distributional form. In the example below, we will describe how `ARRAYS` are used in a Monte Carlo calculation of a P-value based upon an average nearest-neighbor test statistic.

The spatial distribution of some quantity of interest might be clustered or regularly spaced. Define the nearest neighbor distance for observation “i” as \( d(i) = \min(d_{i1}, d_{i2}, \ldots, d_{i(i-1)}, d_{i(i+1)}, \ldots, d_{in}) \) where \( d_{ij} = \) Euclidean distance between observation \( i \) and observation \( j \) and we assume that we have “\( n \)” observations in our data set. The average nearest neighbor distance is \( \Sigma d(i)/n \).

The strategy for this analysis is as follows:
1. Determine nearest-neighbor distances for the observed data configuration.
2. Calculate the average NN distance for the observed data.
3. Assuming the null hypothesis of complete spatial randomness is true, generate a sample of observations that are randomly distributed in the region of interest.
4. Calculate the average NN distance for this simulated set.
5. Repeat steps 3 and 4 a large number of times.
6. The Monte Carlo P-value is the proportional of generated samples that were more extreme than observed. In the case of testing for clustering, this involves testing to see if the average nearest neighbor distance from simulated data is SMALLER than observed in the original configuration.

The SAS program implementing this analysis is given in Display 8.5.

Display 8.5: Calculating a Monte Carlo P-value for testing average nearest neighbor distances

```sas
/* enter the observed data into data arrays */
data plot_obs;
array xobs(4) xobs1-xobs4;
array yobs(4) yobs1-yobs4;
do idat = 1 to 4;
   input xobs(idat) yobs(idat) @@;
end;
datalines;
.25 .65 .25 .75 .35 .65 .35 .75
;
/* Calculate the observed average nearest neighbor distance */
data plot1; set plot_obs;
array xobs(4) xobs1-xobs4;
array yobs(4) yobs1-yobs4;
array nnobs(4) nnobs1-nnobs4;

/* Determine the observed NN distance and average */
sumnnobs = 0;
do i=1 to 4;   * find NN distance for each point ;
   nnobs(i) = 100;  * initialize distances to be large;
   do j=1 to 4;     * compare the ith point to all others;
      d=sqrt( (xobs(i)-xobs(j))**2 + (yobs(i)-yobs(j))**2 );
      if (d<nnobs(i)) and (d>0) then nnobs(i)=d;
   end;
   sumnnobs=sumnnobs+nnobs(i);
end;
avgnnobs = sumnnobs/4;  * observed average NN distance;

/* Simulate plots under CSR */
data mccsr1; set plot1;
array xobs xsim1-xsim4; * observed data;
array yobs yobs1-yobs4;
array xsim xsim1-xsim4; * CSR simulated data;
```
array ysim ysim1-ysim4;
array nnobs nnobs1-nnobs4; * observed NN distances;
array nncsr nncsr1-nncsr4; * simulated NN distances;

/* Generate a large number of CSR plots with 4 trees */
/* CSR = completely spatially random */

* initialize counters of nn avg dist le or ge than observed;
numle = 0;
* numge = 0; * commented out since focus on clustering here;

nsim = 4000; * user-specified option;
seed1 = 12345; * if =0, then uses time since midnight;

do isim = 1 to nsim;

    do ii = 1 to 4;
        xsim(ii) = ranuni(seed1);
        ysim(ii) = ranuni(seed1);
    end;

    /* Find NN distance for the simulated trees */
    sumnncsr = 0;
    do i=1 to 4;
        nncsr(i) = 100; * initialize;
        do j=1 to 4;
            d=sqrt( (xsim(i)-xsim(j))**2 + (ysim(i)-ysim(j))**2 );
            if (d<nncsr(i)) and (d>0) then nncsr(i)=d;
        end;
        sumnncsr=sumnncsr+nncsr(i);
    end;
    avgnncsr = sumnncsr/4;

    /* Accumulate counts of patterns consistent with clustering */
    ile = (avgnncsr <= avgnnobs);
    numle = numle + ile;

    * ige = (avgnncsr >= avgnnobs);
    * numge = numge + ige;

    drop i j ii xobs1-xobs4 yobs1-yobs4 nnobs1-nnobs4
        sumnnobs sumnncsr;
    output;
end;    * of the isim - simulation loop;
Statistical Programming in SAS

Bailer

```
MC_P_value_Cluster = numle/nsim;

* write results to LOG file with PUT statements;
put "MC P-value (Clustering) = " MC_P_value_Cluster;
put "(based on " nsim " simulated plots)";
run;
```

Executing this code will write the results to the LOG file. The results of this calculation were

```
MC P-value (Clustering) = 0.007
(based on 4000 simulated plots)
```

8.3 Remembering variable values across observations – RETAIN

8.3.1 Example: Processing multiple observations for an individual

* Problem motivation: individuals were tracked over time. These individuals were participating in a program where they received services in the community.

* They could disenroll from the program at any time (say when admitted to a hospital or nursing home); however, they could re-enroll at some future date.

* Suppose we had the following goals:
  i) count the number of transitions for each individual;
  ii) record the total time in the program for each individual;
  iii) record the length of the first enrollment in the program. The code in Display 8.6 addresses these objectives.

Display 8.6: Processing variable number of observations per subject
data test;
  input id xstart xstop;
datalines;
  1 15 25
  2 10 12
  2 18 22
  3 6 12
  3 14 15
  3 17 23
;
proc print;
run;
data test2; set test; by id;        **** Comment 1;
  array start[9] start1-start9;
  array stop[9] stop1-stop9;
  array times[9] times1-times9;
  retain count 0;                   **** Comment 2;
  retain start1-start9 stop1-stop9 times1-times9;
* initialize count and arrays with new ID;
  if FIRST.id=1 then do;            **** Comment 3;
    count = 0;
    do ii=1 to 9;
      start{ii} = .;
      stop{ii} = .;
      times{ii} = .;
    end;
  end;
  count = count + 1;            **** Comment 4;
  start{count} = xstart;
  stop{count} = xstop;
  times{count} = xstop - xstart;
  if LAST.id=1 then do;             **** Comment 5;
    first_time = times(1);
    total_time = sum(of times1-times9);
    keep id count first_time total_time;
    output;  * output results if last obs for ID;
  end;
run;
proc print;
run;
The processing of these observations produces the following

<table>
<thead>
<tr>
<th>Obs</th>
<th>id</th>
<th>count</th>
<th>first_time</th>
<th>total_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>6</td>
<td>13</td>
</tr>
</tbody>
</table>

**** Comment 1: The contents of the original data set `test` are imported into a new data set `test2` by the variable `id`. This creates two internal variables, `FIRST.ID` and `LAST.ID`. The `FIRST.ID=1` if the record corresponds to the first record for a particular `id` and =0 for all other records. The `LAST.ID=1` if the record corresponds to the last record for a particular `id` and =0 for all other records. For the first individual (`id=1`), `FIRST.ID=1` and `LAST.ID=1` since there is only one record for this individual. For the third individual (`id=3`), `FIRST.ID=1` and `LAST.ID=0` for the data line “3 6 12”, `FIRST.ID=0` and `LAST.ID=0` for the data line 3 14 15, and `FIRST.ID=0` and `LAST.ID=1` for the data line 3 17 23.

**** Comment 2: The two lines starting with `RETAIN` define the counter of the number of enrollments for an individual and arrays for enrollment (`START`), disenrollment (`STOP`) and time spent in a particular enrollment (`TIMES`). `RETAIN` is used to make sure these values don’t disappear while other lines for the same individual are being processed. The `ARRAYS` are set up to have at most 9 separate enrollment times.

**** Comment 3: When it is the first record for an individual, we initialize the counter and the array elements. Note that the array elements are initially set to missing since most individuals are expected to have fewer than this maximum.

**** Comment 4: Load the arrays with information from the current record and calculate the time enrolled for this entry.

**** Comment 5: When it is the last record for an individual, we accumulate the total time (note and missing values are ignored by the `sum` function), define the length of the first stay, and output the results.

8.4 Case Study 8.2: Randomization test for the equality of two populations [OMIT]

Randomization tests (Edgington 1995, Good 2001) can also be used to obtain P-values.
If the two distributions are the same, then the label of group membership, i.e. group 1 vs. group 2, could be thought of as being randomly assigned to each observation. In a randomization test for two groups, we could consider all \( \binom{n_1 + n_2}{n_1} \) possible assignments of labels to observations. For each assignment, we can calculate the value of the test statistic and compare it to the observed value of the test statistic. The proportion of test statistics from the randomization process that exceed the observed test statistic is the randomization P-value. As an alternative to the complete randomization, we might take a sample from the collection of \( \binom{n_1 + n_2}{n_1} \) possible assignments of labels to observations, and calculate a P-value based about this collection. The code to implement this procedure is given in the example below.

Data are read from a member nitrofen of an external permanent SAS library. These data are the number of young produced by *Ceriodaphnia dubia* exposed to a rice herbicide, nitrofen. The control and 160 μg/l conditions are selected for analysis as we see below.

```sas
libname class 'D:\baileraj\Classes\Fall 2008\sta402\data';

data test; set class.nitrofen;
  if conc=0 | conc=160;

proc print;
  title NITROFEN: print of (0, 160) concentrations;
  var conc total;
  run;
```

<table>
<thead>
<tr>
<th>Obs</th>
<th>conc</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>27</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>32</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>34</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>33</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>36</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>34</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>33</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>30</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>24</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>31</td>
</tr>
<tr>
<td>11</td>
<td>160</td>
<td>29</td>
</tr>
<tr>
<td>12</td>
<td>160</td>
<td>29</td>
</tr>
<tr>
<td>13</td>
<td>160</td>
<td>23</td>
</tr>
<tr>
<td>14</td>
<td>160</td>
<td>27</td>
</tr>
<tr>
<td>15</td>
<td>160</td>
<td>30</td>
</tr>
<tr>
<td>16</td>
<td>160</td>
<td>31</td>
</tr>
<tr>
<td>17</td>
<td>160</td>
<td>30</td>
</tr>
<tr>
<td>18</td>
<td>160</td>
<td>26</td>
</tr>
</tbody>
</table>
We next examine the results from a t-test comparing the mean responses in the control and 160 μg/l conditions. From this section of code and the accompanying results, we see that the two group t-test resulted in a P-value of 0.035 (assuming equal variances).

\begin{verbatim}
proc ttest;
  title NITROFEN: t-test of (0, 160) concentrations;
  class conc;
  var total;
run;
\end{verbatim}

Display 8.7: Output from PROC TTEST

<table>
<thead>
<tr>
<th>conc</th>
<th>N</th>
<th>Mean</th>
<th>Mean</th>
<th>Mean</th>
<th>SD</th>
<th>SD</th>
<th>SD</th>
<th>Std Err</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>28.827</td>
<td>31.4</td>
<td>33.973</td>
<td>2.4737</td>
<td>3.5963</td>
<td>6.5654</td>
<td>1.1372</td>
</tr>
<tr>
<td>160</td>
<td>10</td>
<td>26.612</td>
<td>28.3</td>
<td>29.988</td>
<td>1.6229</td>
<td>2.3594</td>
<td>4.3073</td>
<td>0.7461</td>
</tr>
<tr>
<td>Diff (1-2)</td>
<td>0.2424</td>
<td>3.1</td>
<td>5.9576</td>
<td>2.2981</td>
<td>3.0414</td>
<td>4.4977</td>
<td>1.3601</td>
<td></td>
</tr>
</tbody>
</table>

\begin{verbatim}
proc transpose data=test prefix=xx out=tran_out;
  var total;
run;
data obs_test; set tran_out;
  type = 'O';    * identifies original data;
run;
proc print data=obs_test;
  title 'Randomization test: observed data';
\end{verbatim}

PROC PLAN to generate a collection of permutations of the indices 1 through 20. The 10 observations corresponding to the first 10 indices in the permuted set are assigned for group 1 for a particular permutation, while the remaining observations are assigned to group 2. These permutations are read into a SAS data set for additional processing.

TRANSPOSE procedure is used to place each permuted set of data onto a single line.
run;

proc plan;
  factors test=4000 ordered in=20;
  output out=d_permut;
run;

/*
PLAN used to generate a set of indices for the
randomization test and then use TRANSPOSE to package the
output
*/

proc transpose data=d_permut prefix=in
  out=out_permut(keep=in1-in20); by test;
run;

proc print data=out_permut;
run;

data _null_; set obs_test;
  file 'D:\baileraj\Classes\Fall 2008\sta402\SAS-
  programs\week7-perm.data';
  put type xx1-xx20;
run;

data _null_; set out_permut;
  type = 'P'; * permutation data;
  file 'D:\baileraj\Classes\Fall 2008\sta402\SAS-
  programs\week7-perm.data' mod;
/* mod option adds lines to existing file */
  put type in1-in20;
run;

/*/ week7-perm.data ...
O 27 32 34 33 36 34 33 30 24 31 29 29 23 27 30 31 30 26 29 29
P 8 14 4 11 3 2 12 1 6 13 17 9 15 16 5 19 20 7 10 18
P 12 2 8 10 13 7 9 16 4 19 15 3 5 14 17 1 20 11 6 18
P 18 17 13 14 5 8 19 16 3 12 11 9 10 7 2 20 4 6 1 15
.
/*

data perm_data;
  array both{20} x1-x10 y1-y10;
  /* array for observed values */
  array ins{20} in1-in20;
  /* index array */
  array perms{20} xp1-xp10 yp1-yp10;
/* array for permuted values */

infile 'D:\baileraj\Classes\Fall 2003\sta402\SAS-programs\week7-perm.data';

input type $ @;
if type='O' then do;
  input x1-x10 y1-y10;
  obs_diff = mean(of x1-x10) - mean(of y1-y10);
  retain obs_diff x1-x10 y1-y10;
end;
else do;
  input in1-in20;
  do ii = 1 to 20;
    perms{ii} = both{ ins[ii] };
  end;
  perm_diff = mean(of xp1-xp10) - mean(of yp1-yp10);
  perm_ge = (perm_diff >= obs_diff); * 1-tailed;
  perm_2tail = (abs(perm_diff) >= abs(obs_diff)); * 2-tailed;
  keep obs_diff perm_diff perm_ge perm_2tail;
  output;
end;

/*
FREQ is used to tabulate the relative frequency of
Randomization procedures yielding test statistics at
Least as extreme as observed in the data
*/

proc freq data=perm_data;
title 'NITROFEN: randomization test -> upper tail P-value';
table perm_ge perm_2tail;
run;

Display 8.9: Output from a randomization test of two population mean equality

<table>
<thead>
<tr>
<th>perm_ge</th>
<th>Frequency</th>
<th>Percent</th>
<th>Cumulative Frequency</th>
<th>Cumulative Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3927</td>
<td>98.18</td>
<td>3927</td>
<td>98.18</td>
</tr>
</tbody>
</table>
Thus we see that the two-tailed P-value from the randomization procedure, 0.0383, is quite similar to the P-value obtained from the t-test.

References


