

SAS Data Management

March, 2006

Introduction.....	2
Reading Text Data Files.....	2
INFILE Command Options	2
INPUT Command Specifications.....	2
Working with String Variables	3
Upper and Lower Case String Values.....	3
Searching Text Variables	4
Extracting Substrings.....	4
Informats and Formats	5
Date and Time Variables:.....	5
Proc Contents.....	7
Data Organization Guidelines	7
Task 1	8
Date Calculations	8
Restructuring Data – "by subject" to "by measurement"	9
Keep	9
Array	9
Do – End.....	10
Output	10
Procedure OUTPUT Datasets.....	10
Graphing.....	11
TARGET and DEVICE options for Graph Output.....	11
Graphics Output to File.....	11
Selecting Observations – IF vs. WHERE	12
IF.....	12
WHERE	12
Task 2	13
Restructuring Data – "by measurement" to "by subject"	13
Read more than one input data line	13
Using First.byvar/Last.byvar	14
Creating Multiple Output Datasets	15
Match-Merging Datasets	15
Labeling Results.....	16
Variable Labels.....	16
Coding labels.....	16
Labels in Permanent Datasets	17
Sharing SAS Datasets with User-Defined Formats.....	19
Listing the Contents of a Format Catalog.....	19
Statistical Analysis Procedures	20
Common Subcommands.....	20
BY:.....	20
WHERE:	20
LABEL and FORMAT:	20
OUTPUT:	20
Frequently Used PROCs	21
Tabulations and Measures of Association	21
Univariate Descriptive Statistics	21
Correlations	21
T-Tests.....	22
Linear Regression	22
ANOVA	23
Appendix:.....	24

Introduction

This document assumes you have mastered the material covered in [Introduction to SAS on Windows](#) on the [Statistical Software at UMass](#) website. We assume you understand the function of the SAS data step, know how to set up a simple SAS dataset, create some new variables, understand the difference between temporary and permanent SAS datasets and how to work with each, and can run a few simple PROCs.

With that background, this document will help you develop your SAS skills by introducing new SAS concepts, commands and options needed to get beyond the simplest datasets, choose the correct data organization for your intended analysis and begin to write SAS programs to achieve your ends.

Reading Text Data Files

Space delimited text data files with no missing data and character data no longer than 8 characters long can usually be read with "plain vanilla" INFILE and INPUT commands. For most other text data formats you will need to use one or more of the many available options on the INFILE and INPUT commands.

If your data file is not being read correctly, check the INFILE and INPUT commands for options that may help.

INFILE Command Options

The most commonly used options on the INFILE command are:

DELIMITER – used to specify a character data delimiter other than blank. e.g. DLM=','

EXPANDTABS – used for TAB delimited data.

LRECL – specifies the input record length. Under Windows, LRECL is needed for input records over 256 characters long.

MISSEVER – prevents going to the next input record to fill missing data. This is almost always the correct choice.

FIRSTOBS, OBS – number of first and last record to read from input data file. If the data file has some explanatory text in the beginning, use FIRSTOBS to ignore the text.

Example: Read a tab delimited file with records up to 5000 characters long, starting with the 10th input line, ending with the 100th line, a total of 91 observations.

```
INFILE 'filename' EXPANDTABS LRECL=5000 MISSEVER FIRSTOBS=10 OBS=100;
```

INPUT Command Specifications

You can use the INPUT command to read data from specific positions, or free format.

Fixed Format: For data arranged in fixed positions, specify the position following the variable name:

```
data work.test;
input city $ 1-11 julytemp 13-16;
datalines;
denver 78
miami 97.8
vladivostok 72
run;
```

```
proc print; run;
```

Free Format: With free format data, the most common problems are reading string data values over 8 characters long, and embedded blanks in blank-delimited data.

For string data over 8 characters, you can use `:$n.` (where `n` is a number) to indicate the maximum length of a character variable. This reads a character string up to `n` characters long starting at the delimiter, and ending with the following delimiter or the end of line.

For single embedded blanks in blank-delimited data, use the `&` modifier. With the `&` modifier, **two** consecutive blanks are required to end the field. The following example needs both the `&` and `:$n` in order to read "San Francisco". Note that there are at least two blanks separating the city names and the temperature values:

```
data work.test;
input city & :$15. julytemp;
datalines;
denver      78
miami      97.8
san francisco 72
run;
proc print; run;
```

Working with String Variables

Upper and Lower Case String Values

You can mix upper and lower case freely in your SAS code. However, the values of string variables are case-sensitive. Consider the following:

```
Data work.cities;
infile datalines dlm=',';
input city & :$15. state$;
cards;
New York, NY
Buffalo, NY
Newark, NJ
Trenton, NJ
Princeton, NJ
Boston, MA
Houston, TX
San Antonio, TX
Galveston, TX
Los Angeles, ca
Sacramento, Ca
San Francisco, ca
Santa Clara, CA
run;
* select california cities. Data is case-sensitive;
proc print data= work.cities; where state='ca'; run;
```

The state codes 'CA', 'Ca', and 'ca' are distinct values. Hence, the above code prints Los Angeles and San Francisco, but not Sacramento or Santa Clara.

When dealing with string variables, if you wish to ignore case, be sure to convert strings to all upper (or lower) case before working with them. The following will print all four of the California cities:

```
* convert to uppercase before comparison;
proc print data= work.cities; where upcase(state)='CA'; run;
```

The above converts the value of STATE to upper case before comparing it to 'CA'. It does not change the stored value of STATE, so you will have to use the upcase function every time you need to work with variable STATE. In order to avoid having to do that, change the stored value of STATE in a data step:

```
* 'normalize' state codes to upper case;
data work.cities2;
  set work.cities;
  state=upcase(state);
run;
proc print data= work.cities2; where state='CA'; run;
```

Searching Text Variables

The INDEX function is very useful for searching string variables for text fragments. Here we identify cities that contain 'san' or 'ton'. We also use the LOWCASE function to avoid problems with upper-lower case mismatches:

```
* identify "saint" cities and "ton" cities;
data work.santon;
  set work.cities2;
  san=index(lowcase(city), 'san');
  ton=index(lowcase(city), 'ton');
run;
title 'san cities';
proc print data=work.santon; where san>0; run;
title 'ton cities';
proc print data=work.santon; where ton>0; run;
```

Are any of the selected cities unexpected?

Extracting Substrings

Here we read the same data as before, but as a single string variable containing the city and state names. Then we split this string into the city and state components:

```
* extract city & state from general text string;
Data work.cities;
  input citystate & :$25.;
  comma=index(citystate, ','); * locate comma;
  city=substr(citystate, 1, comma-1); * substring starting at 1;
  state=upcase(substr(citystate, length(citystate)-1, 2)); *last 2 characters;
cards;
New York, NY
Buffalo, NY
Newark, NJ
Trenton, NJ
Princeton, NJ
Boston, MA
Houston, TX
San Antonio, TX
Galveston, TX
Los Angeles, ca
Sacramento, Ca
```

```

San Francisco, ca
Santa Clara, CA
run;
title ;
proc print data=work.cities; run;

```

To get CITY, the INDEX function identifies the location of the comma, which separates the city from the state. The SUBSTR function then takes the portion of variable CITYSTATE starting at position 1, and ending at the last position before the comma. This is stored as CITY.

To get STATE, we want the last two characters of CITYSTATE. The LENGTH function returns the last non-blank position in CITYSTATE. SUBSTR then extracts the 2 characters starting one position before the end of CITYSTATE. Finally, this is converted to upper case and the result stored in variable STATE.

Informats and Formats

SAS uses Informats and Formats to interpret and display data in convenient ways.

Informats provide instructions for how to interpret data as it is read. Informats can be specified using an INFORMAT statement, or on the INPUT command following a colon after the variable name, as in the previous example.

Formats provide instructions for how to display a variable on output. Informats and Formats ALWAYS end with a period!

Example: Here we use the INFORMAT command to permit character values of up to 15 characters. Note that the INPUT statement does not have a \$ to indicate that CITY is character. The INFORMAT has already done this. The INPUT statement still needs the & to read the embedded blank in San Francisco, and the data still requires two blanks between the CITY name and the temperature.

The FORMAT statement requests that JULYTEMP be printed using 5 digits, three of which are to be reserved for the decimal portion. The decimal point uses one of those three places, so the data is displayed with two decimal places.

```

data work.test;
  informat city $15.; format julytemp 5.3;
  input city & julytemp;
  datalines;
denver          78
miami          97.8
san francisco  72
run;
proc print; run;

```

Date and Time Variables:

Data containing date or time values must be read with a suitable informat if they are to be used in any calculations. The Informat tells SAS to convert an input date or time to a SAS Date or SAS Time. SAS Dates are number of days from January 1, 1960. SAS Times are number of seconds since midnight. SAS Datetimes are number of seconds since midnight, January 1, 1960. SAS Date and time variables can be used in calculations (e.g. given a birthdate, you can calculate age), for sorting, etc.

This example demonstrates the DATETIME, MMDDYY, and MONYY Informats to read dates entered in three different ways. We also read a date using character (\$) informat, to show how the result differs from using a date informat. Then we experiment with several Format statements to see how they affect the display of the data. The Format for a variable can be different from the Informat, but the two must be compatible. If you do not assign a format to a Date or Time variable, it will be displayed as stored, in days or seconds.

```

Title "It LOOKS like a date.  But is it? or THAT doesn't look like
a date.  But it is!";
data work.dates;
input date1 :datetime18. date2 :mmddyy10. date3 :monyy7. date4 $;
/* Choose one of the following formats to see how they differ */
*format date1 datetime18. date2 date9. date3 monyy7.;
*format date1 datetime9. date2 date3 mmddyy10.;
*format date1 date9. date2 date3 date9.;
/* Convert DATETIME to DATE */
*date5=datepart(date1); *format date5 date9.;
datalines;
01jan1960:00:00:00 01/01/1960 jan1960 jan1960
01jan1960:01:00:00 01/01/1961 jan1961 jan1961
02jan1960:00:00:00 01/02/1960 Feb1960 Feb1960
15apr1960:00:00:00 04/15/1960 Apr1960 Apr1960
run;
proc print data=work.dates;
run;

/* sorting on dates. */
proc sort data=work.dates;
  by date4; run;
proc print data=work.dates;run;

```

Since Date variables are measured in days and Datetime variables in seconds, they cannot be successfully combined in calculations unless you first convert them to the same units. You can convert a Datetime (seconds) into a Date (days) using the DATEPART function. Depending on your version of SAS, dates entered in Excel or Access may import into SAS as DATETIME, even if you never entered a time portion in Excel/Access.

Character data that LOOK like dates (date4 in this example) cannot be used for calculations, and will not sort correctly. Compare the results of sorting on date4 to sorting on date3.

Proc Contents

PROC CONTENTS lists all information about a dataset – the number of observations, number of variables, whether or not the data is *known* to be sorted (i.e. whether it was sorted by SAS), the names of all variables, their informat, formats, labels...

Enter the following data into an Excel sheet. Make sure the dates in the third column have cell format mm/dd/yyyy. Save and **Close** the Excel file. (Winer525.xls)

Condition	Subject	Dob	Score1	Score2	Score3	Score4
1	1	05/01/1958	0	0	5	3
1	2	03/15/1970	3	1	5	4
1	3	10/10/1947	4	3	6	2
2	4	04/01/1975	4	2	7	8
2	5	06/13/1965	5	4	6	6
2	6	02/01/1962	7	5	8	9

Import this worksheet to SAS, as SAS dataset `work.wp525one`.

Run Proc Contents to display all information about the dataset. Observe whether variable `dob` has Format and Informat `DATETIME20.` or `DATE9.` If it is `DATETIME20.` you may need to convert it to a `DATE` using the `DATEPART` function.

The variables are listed alphabetically. Use the `POSITION` keyword to get an additional list in the order that the variables appear in the dataset.

```
Proc contents data=work.wp525one position; run;
```

Data Organization Guidelines

Whenever the same variable is measured on the same subject more than once, you have to decide whether to organize the data "by subject" or "by measurement". Note that in this context, "subject" means a unit of analysis, which is not always an individual. In the context of matched case-control studies, for example, the unit of analysis is the matched set.

- **By subject:** all measurements on the same subject are entered as different variables on one observation. The conditions of the measurements are not entered. They are implied by the multiple variables. Measurements that do not vary over condition are entered once. This is the organization used in the `wp525one` dataset. The scores measured on four occasions are separate variables. Trial number is not entered – we know that the result of the first trial is `score1`, the result of the second trial is `score2`, etc. Analyses requiring "by subject" organization include:
 - Correlations
 - Change scores
 - Paired t-test
 - Repeated-measures analysis of variance using PROC GLM
 - Regression

- **By measurement:** repeated measurements on the same subject are entered as a single variable on multiple observations. Variables are needed to uniquely identify the subject and the measurement condition. Variables that do not vary over the repeated condition will have the same value for all observations on the same subject. The arrangement of the forestry survey, *trees*, in task 2 uses this organization. Analyses requiring "by measurement" organization include:
 - Plotting repeated measurements
 - Modeling repeated responses for a subject (growth curves)
 - Repeated-measures analysis of variance using PROC MIXED

Although the "by subject" organization is more common, there is no absolute rule for choosing one of these data arrangements over the other. If all your planned analyses can be done with the same arrangement, your choice will be clear. If not, choose whichever makes the data entry faster or easier. If need be, you can always restructure the data to the other format using a program adapted from the ones introduced here.

Task 1

The *work.wp525one* dataset contains information on 3 subjects in each of two conditions. Each subject has a date of birth, and a score on a test at each of four trials(*). The dates for the four trials are the same for all subjects – Sept 15, Sept 22, Oct 6 and Nov 3, 2002.

Our task is to calculate the subject's age at the start of the experiment, September 15, 2002; get the average score for each trial in each condition; and compare the condition averages visually by plotting the average scores against trial number (or trial date), with separate lines for the two conditions.

Date Calculations

We begin by computing the subjects' age on September 15 and storing the dates of the trials in four new variables.

To assign a date value to a variable, use the form `var='ddMonyyyy'D`. The D at the end tells SAS to interpret the value in apostrophes as a Date. These variables need to be assigned a suitable output format.

Since SAS dates are number of days since Jan 1, 1960, we can compute age in days by subtraction. [Using SAS 8.2 or earlier, *dob* is not a SAS date variable; it is a Datetime variable. Therefore, we first have to convert it to a Date (i.e. convert seconds to days) using the DATEPART function.] We assign *dob* a date format, subtract it from *testdat1*, whose value is September 15, 2002, and divide by 365 to get age in years on September 15, 2002.

```
/* score1 baseline on Sept 15 2002; score2 at 1 week (Sept 22, 2002);
   score3 at 3 weeks (Oct 6, 2002); score4 at 7 weeks (Nov 3, 2002);*/
data work.dates2;
  set work.wp525one;
  testdat1='15sep2002'D; testdat2='22sep2002'D;
  testdat3='6Oct2002'D; testdat4='3Nov2002'D;
  *dob=datepart(dob);          *not needed with SAS 9;
  format testdat1-testdat4 dob mmddyy10.;
```

```
age=(testdat1-dob)/365.25;
run;
```

Restructuring Data – "by subject" to "by measurement"

Now we are ready to compute average scores for each trial x condition combination. The data are arranged using the "by subject" organization, with one observation per subject. The first subject's data in work.dates2 dataset is:

Condition	Subject	Dob	Score1	Score2	Score3	Score4	age	Testdat1	Testdat2	Testdat3	Testdat4
1	1	05/01/1958	0	0	5	3	44.37	9/15/2002	9/22/2002	10/6/2002	11/3/2002

The scores on the four trials are four variables. This data arrangement will allow us to compute the needed averages, but it is not suitable for plotting. In order to plot average scores against trial number or trial date, we need a "by measurement" organization. The four scores for each subject should be separate observations, with an additional variable to indicate the trial number. For the first subject, this would be:

Condition	Subject	Age	Testdate	Trial	Score
1	1	44.37	09/15/2002	1	0
1	1	44.37	09/22/2002	2	0
1	1	44.37	10/06/2002	3	5
1	1	44.37	11/03/2002	4	3

```
data work.wp525many (keep=condition subject age testdate trial score);
set work.dates2;
array s(4) score1-score4;
array d(4) testdat1-testdat4;
format testdate mmddyy10.;
do trial=1 to 4;
score=s(trial); testdate=d(trial);
output;
end;
run;
```

Keep

The KEEP= or DROP= options on the data command are used to limit which variables are saved to the output data set. By default, ALL variables named or implied in the data step are saved in the output dataset.

Array

The ARRAY command is useful as a means of referring to many variables with an indexed name. The index can then be used in loops to name any item in the array. In this example, s(1) means the first variable in the array s, score1. The variables in the array do not need to be of the form x1-x4. They could be A,B,C,D.

Do – End

DO – END enclose commands that are to be executed as a group. The DO can be iterative as in this example, or part of a logical structure:
IF (something) THEN DO; ...; END;

Output

The Data step is an implicit loop that is repeated for each input line. Normally, an observation is sent to the output dataset at the end of each iteration of the data step. The OUTPUT command is used to send an observation to the output dataset at some other point in the Data step. An observation is saved each time the OUTPUT command is executed. This could be once, more than once, or less than once per Data step iteration. In this example, the OUTPUT command is executed four times per data step, resulting in four observations saved to work.wp525many for each observation read from work.dates2.

Procedure OUTPUT Datasets

We can compute the condition x trial averages using the restructured dataset, wp525many. Here we use PROC MEANS to do this AND to save the results to a new dataset, to be used for the plots. We added trial as a second analysis variable so it would be included in the output dataset. Since each trial number corresponds to a particular date, the average trial number for any date is the original trial number.

```
/* Compute means of each trial for each condition.
   Save means to work.wp525means, for use with plot. */
proc means data=work.wp525many maxdec=3 fw=8;
  class condition testdate; ways 2;
  var trial score;
  output out=work.wp525means mean= ;
run;
```

The **WAYS** subcommand limits the means to only two-way combinations.

The **OUTPUT** subcommand of PROC MEANS specifies that the mean scores should be saved in the dataset WORK.WP525MEANS. Since there are two conditions and four dates, WP525MEANS will contain eight observations, regardless of the number of observations in the original dataset. If we had not used the WAYS subcommand the output dataset would contain additional observations for 0 and 1-way means.

In addition to the variables *condition*, *testdate*, *trial* and *score*, the output dataset contains two automatic variables:

FREQ indicates the number of observations used to compute each mean;
TYPE indicates which variables are involved in each average. **_TYPE_=0** is always the grand mean of all observations. The highest order combination is **_TYPE_ = 2ⁿ - 1** where n is the number of class variables. For this example, the highest order combination, condition x testdate, is **_TYPE_=3**.

Many SAS procedures can save output datasets containing the statistics computed in the PROC. The syntax and options will vary according to the PROC used. Consult SAS documentation for specifics.

Graphing

Now we can use the dataset `wp525means` saved from PROC MEANS to graph average scores against trial number and trial date for the two conditions.

```
/* Plot average scores for 4 trials for each condition.
   Plots use cell means saved from PROC MEANS
   symbol commands request connecting the means with lines
   Axis commands specify labeling and scale. */
goptions reset=all;
title 'Winer 525: Average Scores for Trials by Condition';
symbol1 value = dot interpol=join width = 2 color = black line=2;
symbol2 value = square interpol=join width = 2 color = black line = 20;
axis1 label = ('Average Score' justify = center);
proc gplot data=work.wp525means;
  plot score*trial=condition / vaxis=axis1;
  plot score*testdate=condition / vaxis=axis1;
run;
```

The **SYMBOL** commands specify two different symbols, to be used for the two conditions, and that the symbols are to be connected with line segments. The axis command defines the appearance of an axis. Without these commands, PROC GPLOT would use default symbols and axis attributes.

The **PLOT** command `score*trial=condition` says to plot score on the y-axis, trial on the x-axis, and use different symbols for each value of condition. The `VAXIS=axis1` option tells says to apply the earlier axis1 definition to the vertical axis.

The PLOT command is repeated to get a second graph using testdate as the x-variable. Observe that the plot using testdate is scaled according to the time elapsed between trial dates.

TARGET and DEVICE options for Graph Output

The graphics output from the above graph is intended for viewing on the computer screen. For printed output you get better results if you specify the type of printer you will use. You do this using the TARGET or DEVICE options on the GOPTIONS command preceding PROC GPLOT.

```
goptions reset=all target=ps;
```

The DEVICE option sends the graph directly to the printer or other device. TARGET shows you a preview on screen, which you can then send to the printer. PS is a generic postscript printer driver. For other driver names, you will need to peruse the list of available drivers. To see the list, use PROC GDEVICE.

```
proc gdevice; run;
```

Graphics Output to File

To put your graphics into a file for another application, use a driver appropriate to the target application – CGMOF97P creates cgm file formatted for Office 97 Portrait mode.

```
filename sasgraph 'C:\sasfiles\winer525plot.cgm';
goptions reset=all device=cgmof97p gsfname=sasgraph gsfmode=replace;
```

FILENAME specifies the file to use, and associates it with the name SASGRAPH. The GSFNAME option sends the output to the file associated with the name SASGRAPH. GSFMODE option says to replace the file contents with the current graph. Use GSFMODE=APPEND to add the current graph to the file.

Selecting Observations – IF vs. WHERE

In the previous example, if we had not used the WAYS subcommand on PROC MEANS, the *work.wp525means* output dataset would have contained additional observations of `_TYPE_ = 0, 1, and 2`, which we do not want to include in the plots. To select a subset of observations for analysis, you can use an **IF** statement in a DATA step, or a **WHERE** subcommand in any procedure.

IF

The IF command can only be used in a DATA step. This means you are creating a new dataset that includes only the selected observations. You can then use that dataset in any procedures that you wish to limit to the same selected observations.

```
DATA work.type3means;  
  set work.wp525means;  
  if (_TYPE_ = 3);  
run;
```

Only observations that satisfy the IF condition are sent to the output dataset. This method is advantageous if you will be doing many analyses with the same selected observations. Specify the dataset of selected observations on your procedures to be sure that your analysis is using the correct selection.

WHERE

The WHERE subcommand can be added to any procedure. Only observations that satisfy the WHERE condition are used by the procedure.

```
proc gplot data=work.wp525means;  
  where _type_ = 3;  
  plot score*trial=condition / vaxis=axis1;  
  plot score*testdate=condition / vaxis=axis1;  
run;
```

This method is more convenient for the occasional selection, or for using a variety of selections. It does not require a DATA step and does not change the dataset in any way. It applies only to the procedure where it's used. You can easily run different procedures with different selections, or re-run this procedure with a different WHERE subcommand to do another selection.

Task 2

The data used in this example is a small portion of a forestry survey (**). Trees were measured in 1984 and 1994, and the diameter of the trunk recorded. The two measurements for each tree were entered as separate observations, a "by measurement" organization. Tree and Year uniquely identify the subject and condition; species and location are constant for each tree; dbh ("diameter at breast height") is the measurement variable that changes at each measurement. The challenge is to calculate how much the diameter increased for each tree during the 10 years.

Restructuring Data – "by measurement" to "by subject"

The data and coding is listed in the Appendix. In order to compute the increase in trunk diameter for each tree, we need to have simultaneous access to the two measurements for a tree, so we can take the difference between them – a "by subject" organization.

Read more than one input data line

The data is arranged so the two measurements on the same tree are on two consecutive lines. Therefore, it might make sense to read two lines from the input file for each observation. A slash (/) in the INPUT list tells SAS to read from the next data line. The following program reads the 1984 data from one line, and the 1994 data from the next:

```
/* Read two data lines per observation */
DATA work.trees;
  INPUT tree species location y84 dbh84/ tr94 sp94 loc94 y94 dbh94;
  datalines;
  1 15 1 84 17.00
  1 15 1 94 18.30
  2 27 1 84 23.10
  2 27 1 94 22.80
;
run; proc print; run;
```

Looking at the log, we see some ominous notes:

```
NOTE: LOST CARD.
RULE:      +-----1-----2-----3-----4-----5-----6-----7-----
8-----+
65
tree=12 species=3 location=4 y84=94 dbh84=5.3 tr94=. sp94=. loc94=. y94=. dbh94=.
_ERROR_ =1
_N_ =12
NOTE: The data set WORK.TREES has 11 observations and 10 variables
```

We were expecting 12 trees, but there are only 11. We are not sure what the "LOST CARD" note means, but it doesn't sound good. Examining the output from PROC PRINT we observe that y84 has value 84 for trees 1-6, and 94 for trees 7-11. Evidently something went awry around tree 6 or 7. Looking at the original data file (see Appendix), we see that tree 6 does not have data for 1994. Reading two lines for every tree is not going to work unless every tree has two lines of data.

Using First.byvar/Last.byvar

Since we can't count on having the same number of data lines for each observation, we will read the data one line at a time, and then use a SAS program to re-structure it. This program is used to read the data.

```
/* Tree diameters are measured 10 years apart.
   We want to compute how much the diameter increased for each tree.
   Note that tree 6 is missing the second measurement.*/
DATA work.trees;
  INPUT tree species location year dbh ;
  datalines;
  1 15 1 84 17.00
  1 15 1 94 18.30
  2 27 1 84 23.10
  2 27 1 94 22.80
;
run;
```

This program combines the data for each tree into one observation and computes the difference in trunk diameter.

```
/* WORK.TREES must be sorted by TREE, YEAR */
DATA work.treegrowth (keep=tree species location dbh84 dbh94 growth);
  set work.trees; by tree;
  array d(2) dbh84 dbh94;
  retain dbh84 dbh94;
  if year=84 then d(1)=dbh;
  else if year=94 then d(2)=dbh;
  if last.tree then do;
    growth=dbh94-dbh84;
    output;
    do i=1 to 2; d(i)=.; end;
  end;
run;
```

Retain

Ordinarily, at the start of each iteration of the data step, all variables are cleared (set to missing). The RETAIN statement is used to prevent a variable from being cleared. In this example, we need to keep the tree diameter from the first measurement of a tree in order to be able to subtract it from the second measurement, when we find it.

First.byvar/Last.byvar

Since we have to combine two input observations for each tree into one, we only want to send an observation to the output dataset when we get to the last observation for a tree. When an input dataset is sorted, and the data step has a BY command with the sorting variable, then you can use FIRST.byvar and LAST.byvar to signal the first and last observation of each BY group.

Here we use LAST.TREE to determine when we have reached the last observation for a tree. When we get to that observation, we compute the difference in diameter and send the result to the output dataset. Then we clear the dbh variables in preparation for the next tree.

Creating Multiple Output Datasets

Here is another way to accomplish the task of combining the two years of information on each tree and calculating the difference in diameter. When reading the data, make separate datasets for each year, then match the observations for the same tree in the two datasets.

We can name two output datasets on the DATA statement. With a choice of two output datasets, the OUTPUT command needs to specify the dataset to which an observation is to be sent. Since each output dataset has observations from only one year, we drop the year variable.

```
DATA work.trees84 (drop=year) work.trees94 (drop=year);
  INPUT tree species location year dbh ;
  if year=84 then output work.trees84;
  else if year=94 then output work.trees94;
datalines;
  1 15 1 84 17.00
  1 15 1 94 18.30
  2 27 1 84 23.10
  2 27 1 94 22.80
;
```

Match-Merging Datasets

Files to be matched must be sorted on the BY variable used for matching. The data in this example is already sorted by TREE, so we can skip this step.

```
proc sort data=work.trees84; by tree; run; *sort if needed;
proc sort data=work.trees94; by tree; run;
```

We want the matched file as a permanent SAS dataset, so we assign a LIBNAME to specify where to save the file (the directory), and use the assigned reference as the first part of the dataset name, instead of WORK.

The variables in the two datasets to be matched have the same names – *tree*, *species*, *location*, *dbh*. The variable to be used for matching, TREE, must have the same name in the files to be matched, so that's fine. For all other variables, if the same variable appears in both files, the value found on the last file named on the MERGE command is the value saved in the output dataset. Species and location have the same value in both files for any given tree, so this is not a problem. Dbh, however, has different values in 84 and 94, and we need them both. Therefore, we have to rename this variable so it has different names for the two years.

```
libname a 'c:\sasfiles\';
DATA a.trees8494;
  merge trees84 (rename=(dbh=dbh84)) trees94(rename=(dbh=dbh94));
  by tree;
  growth=dbh94-dbh84;
run;
```

The MERGE command names the two files to be merged. The RENAME option is used on each input file to give dbh unique names for the two years. The BY subcommand

following MERGE specifies that the observations must be matched on the value of TREE.

Labeling Results

Variable names are often terse and not very revealing. Variable codes can be even more mysterious to anyone unfamiliar with the data – if gender is coded 1 and 2, which is the code for male, and which for female?

Assigning labels to the variables and to the codes will make your output more readily understandable.

Variable Labels

Variable labels are assigned using the LABEL command in the data step. If the data step creates a permanent SAS dataset, the variable labels are stored as part of the permanent dataset. The label is usually used automatically on output involving the labeled variable, but in a few cases you need to request it.

Example: In the trees dataset, we would like to assign meaningful labels to some variables. To do this, add a LABEL command to the data step that creates the dataset:

```
DATA work.trees;
  INPUT tree species location year dbh ;
  LABEL tree="TreeID" dbh="Diameter at Breast Height";
  datalines;
  1 15 1 84 17.00
  1 15 1 94 18.30
  2 27 1 84 23.10
  2 27 1 94 22.80
;
run;
```

Coding labels

The codes for SPECIES and LOCATION need to be labeled. Coding labels are assigned using the FORMAT command. Recall that a FORMAT tells SAS how to display a variable on output. We would like to tell SAS that when displaying the values of SPECIES, it should print "Ash" instead of 15, "Oak" instead of 27, etc. See the Appendix for the list of SPECIES and LOCATION codes.

SAS has many pre-defined formats for standard data types, but obviously printing the number 15 as "Ash" is not standard. We can define our own formats for SPECIES and LOCATION using PROC FORMAT:

```
/* Define display formats for species and location */
proc format;
  value sp_name 3='Maple' 10='Hickory' 15='Ash' 27='Oak' 31='Hemlock';
  value loc_name 1='NE' 2='SE' 3='SW' 4='NW';
run;
```

This creates two new formats, named `sp_name` and `loc_name` with the appropriate labels assigned to the numeric codes. Once the formats are defined, you associate

them with the variables using a FORMAT command, just as you would with a standard format:

```
DATA work.trees;
  INPUT tree species location year dbh ;
  LABEL tree="TreeID" dbh="Diameter at Breast Height";
  format species sp_name. location loc_name.;
  datalines;
  1 15 1 84 17.00
  1 15 1 94 18.30
  2 27 1 84 23.10
  2 27 1 94 22.80
;
run;
```

Note that PROC FORMAT must precede the data step in which it is to be used, and that, as with all formats, the format name is followed by a period on the FORMAT command. It is not followed by a period in the format definition in PROC FORMAT.

The FORMAT command to associate variables with formats can be used as a subcommand in any PROC, rather than in the data step. When you assign a format in a PROC, it applies only for the duration of that PROC. FORMATS assigned in the DATA step apply for the life of the dataset.

The trees dataset is in the WORK library, so its life is limited to the current SAS session. In each future session you would need to run the PROC FORMAT and the DATA step codes before you could run any PROCs on this data.

Labels in Permanent Datasets

When creating a permanent SAS dataset, labels and format assignments in the DATA step become part of the permanently stored information in the dataset. Non-standard Format **definitions**, however, are NOT stored in the dataset. Format definitions are stored in SAS Format Catalogs. Like SAS datasets, SAS Format Catalogs can be temporary or permanent.

Example: Suppose the trees dataset was created with the format assignments shown in the last example and stored as a permanent SAS dataset in the directory `c:\sasfiles`. In a new SAS session, you try to use this permanent dataset, using this code:

```
libname a 'c:\sasfiles\';
proc print data=a.trees (obs=5) label ; run;
```

You get the following errors in the log:

```
ERROR: Format SP_NAME not found or couldn't be loaded for variable species.
ERROR: Format LOC_NAME not found or couldn't be loaded for variable location.
```

The formats `sp_name` and `loc_name` are permanently associated with the data, but the format definitions have not been permanently saved, nor have they been defined in this session. Hence, SAS is unable to use the dataset.

Solution 1: Include code to define the formats in each session before you use the permanent SAS dataset:

```

proc format;
  value sp_name 3='Maple' 10='Hickory' 15='Ash' 27='Oak' 31='Hemlock';
  value loc_name 1='NE' 2='SE' 3='SW' 4='NW';
run;
libname a 'c:\sasfiles\';
proc print data=a.trees (obs=5) label ; run;

```

Solution 2: Save the user-defined formats in a permanent SAS Format Catalog.

Like permanent SAS Datasets, permanent SAS Format Catalogs are saved and used by referencing a SAS Library. Using SAS 8 or higher under Windows, SAS Datasets have the file extension `.sas7bdat`; SAS Format Catalogs have the extension `.sas7bcat`.

A SAS Library is a location (subdirectory/folder) where SAS files are stored. It is defined using the `LIBNAME` command.

a. Using a named format catalog:

To save the user-defined (i.e. non-standard) formats in a permanent SAS Format Catalog, define the SAS Library where SAS is to store the catalog, and add the `LIBRARY` option to `PROC FORMATS` to specify the name of the format catalog. Here a permanent SAS Format Catalog called `treesfmt.sas7bcat` is created in directory `c:\sasfiles`.

```

/* Save permanent Format Catalog. */
libname a 'c:\sasfiles\';
proc format library=a.treesfmt;
  value sp_name 3='Maple' 10='Hickory' 15='Ash' 27='Oak' 31='Hemlock';
  value loc_name 1='NE' 2='SE' 3='SW' 4='NW';
run;

```

The log shows the messages:

```

NOTE: Format SP_NAME has been written to A.TREESFMT.
NOTE: Format LOC_NAME has been written to A.TREESFMT.

```

To use your permanent SAS Dataset with user-defined formats in future SAS sessions you need to:

- assign a SAS Library to the location of the SAS Format Catalog
- specify the name of the SAS Format Catalog with the `fmtsearch` option

In this example, the dataset and the format catalog are stored in the same location, so we can use the same `LIBNAME` statement for both:

```

/* Use permanent Format Catalog. */
libname a 'c:\sasfiles\';
options fmtsearch=(a.treesfmt);
proc print data=a.trees (obs=5) label ; run;

```

Some points to note about permanent SAS Format Catalogs:

- The `LIBNAME` reference defines WHERE the catalog will be stored. It does not specify a filename, just as it does not specify a filename for SAS Datasets.
- The name of a SAS Format Catalog to create is specified using the `LIBRARY` option of the `PROC FORMAT` command.
- The name of an existing SAS Format Catalog to use is specified using the `fmtsearch` option on an `OPTIONS` command.

- SAS Format Catalogs are stored as separate files. They are not linked to SAS Datasets in any way. You are responsible for assigning the correct SAS Format Catalog to use with your SAS Datasets. If the SAS Dataset and the SAS Format Catalog are stored in different locations, you will need to assign a LIBNAME for each.

b. Using the default format catalog:

The default name for a SAS Format Catalog is formats.sas7bcat. To use this name for your permanent SAS Format Catalog you do not need to use the ffmtsearch option, but you must use the libname LIBRARY:

```
/* Save permanent Format Catalog using default name. */
libname library 'c:\sasfiles\';
proc format library=library;
  value sp_name 3='Maple' 10='Hickory' 15='Ash' 27='Oak' 31='Hemlock';
  value loc_name 1='NE' 2='SE' 3='SW' 4='NW';
run;

/* Use permanent Format Catalog with default name. */
libname library 'c:\sasfiles\';
proc print data=a.trees (obs=5) label ; run;
```

Using this code requires some care, as each SAS dataset does not necessarily have it's a separate associated format catalog. There can only be one format catalog with the default name in each sas library (folder). If you use this format catalog for more than one dataset, you must be sure there are no conflicting format definitions.

Using the default name is the only means of storing permanent SAS Format Catalogs in SAS version 6.12 or earlier.

Sharing SAS Datasets with User-Defined Formats

If you need to share a permanent dataset that contains user-defined (i.e. non-standard) formats, you must also share either the code that defines the formats or the permanent format catalog file where the format definitions are stored. This assumes that the person with whom you are sharing SAS datasets has the same version of SAS, on the same Operating System (e.g. Windows). If this is not the case, read [Transporting SAS Libraries](http://www-unix.oit.umass.edu/~statdata/software/Libraries) at <http://www-unix.oit.umass.edu/~statdata/software/>

If someone gives you a SAS dataset that contains user-defined formats and does NOT provide the necessary code definitions, you can force SAS to ignore the user-defined formats. With the NOFMTERR (No Format Error) option specified, SAS reverts to standard display formats whenever the user-defined formats are not available. This will enable you to work with the dataset, but you will not have the benefit of nicely labeled codes.

```
options nofmtterr;
```

Listing the Contents of a Format Catalog

Use PROC FORMAT with the FMTLIB option to list a description of all the formats stored in a format catalog:

```
proc format library=a.treesfmt ffmtlib; run;
```

If you specify a library without the catalog name, you will get the contents of the default format catalog, formats.sas7bcat:

```
proc format library=a fmtlib; run;
```

Statistical Analysis Procedures

There are a great many PROCs available for statistical analysis, each with its own set of options and subcommands. We will focus on the basic specifications required for a few of the most frequently used PROCs, and on some options/subcommands that are common to most PROCs.

Common Subcommands

Although each PROC has specific subcommands, there are some subcommands that can be used on most PROCs. A subcommand on a PROC affects only that one invocation of that PROC.

BY:

The BY subcommand, added to any PROC, repeats the PROC for each value of the BY variable. The data must be pre-sorted according to the BY variable. The following code will run PROC MEANS for the tree diameter measurements from year 84, and then from year 94.

```
proc sort data=work.trees; by year; run;
proc means data=work.trees;
  var dbh; by year; run;
```

WHERE:

The WHERE subcommand, added to any PROC, is used to select a subset of the observations for analysis. This code will run PROC MEANS on the tree diameter measurements of Hemlocks.

```
proc means data=work.trees;
  var dbh; where species=31 ; run;
```

LABEL and FORMAT:

The LABEL and FORMAT commands described earlier can be used as subcommands on any PROC to *temporarily* change the label or format associated with a variable. This code temporarily removes the user-defined formats for species, so it prints with the default numeric format.

```
libname library 'c:\sasfiles\';
proc freq data=library.trees;
  table species; format species; run;
```

OUTPUT:

The OUTPUT subcommand can be used on many, but by no means all, PROCs to save results as SAS datasets. We used it on PROC MEANS in Task 1. The syntax varies among PROCs. Sometimes it's an option, rather than a subcommand. We mention it

here as a reminder to look for it whenever you need to use the results of a PROC for further analysis in another PROC. If a PROC does not have an OUTPUT option or subcommand, it is often possible to save results in a SAS dataset by using the Output Delivery System (ODS). ODS is often more flexible than the OUTPUT option for saving results as SAS datasets. Use of ODS is not described in this document.

Frequently Used PROCs

The specifications shown here are not intended to be complete. They show the minimal code you will need to get some basic analyses. Consult online help or printed manuals for additional subcommands and options.

The illustrations use datasets created earlier:

- wp525one – initial "by subject" arrangement of data from Task 1
- trees – initial "by measurement" arrangement of data from Task 2
- trees8494 – final restructured "by subject" data from Task 2

In all illustrations we assume that the dataset is permanent, and is stored in the c:\sasfiles directory, which has been assigned the name LIBRARY.

Tabulations and Measures of Association

PROC FREQ reports the number of observations with each unique value of a variable; for multi-way tabulations it reports the number of observations with each unique combination of two or more variables. Various statistical tests are available as options on the TABLES subcommand.

```
LIBNAME library 'c:\sasfiles\';
PROC FREQ DATA=library.trees;
    TABLES species location;                *one-way tables;
    TABLES species*location/chisq; run;    *two-way table with Chisquare test;
```

Univariate Descriptive Statistics

Use PROC MEANS for simple descriptive statistics, such as means, standard deviations, medians, extremes, etc. For more extensive descriptive statistics, normal probability plots and tests of normality use PROC UNIVARIATE. The CLASS subcommand on either procedure provides analyses for subgroups. The results are similar to what you would get using the BY subcommand, but do not require the data to be pre-sorted.

```
LIBNAME library 'c:\sasfiles\';
PROC MEANS DATA=library.trees;            *describe dbh for each species*year;
    CLASS species year; VAR dbh; run;

*describe dbh with test of normality and lineprinter plots;
PROC UNIVARIATE DATA=library.trees  NORMAL PLOTS;
    VAR dbh; run;
```

Correlations

Use PROC CORR to get parametric (Pearson) or non-parametric (Kendall and Spearman) correlations, or Cronbach's Alpha. Correlations require more than one measurement level variable. Therefore we use the wp525one dataset from Task 1 for the example.

```

/* Pearson correlations. */
LIBNAME library 'c:\sasfiles\';
PROC CORR DATA=library.wp525one;
    VAR score1-score4;
    BY condition; run;

```

*all possible pairs of variables;
*separately for each condition;

T-Tests

PROC TTEST computes one and two-sample t-tests. For paired t-test, run a one sample t-test on the difference between the paired variables of interest.

We illustrate the independent sample t-test using the trees dataset to test for difference in tree diameter between the two years. This is not the most appropriate test for this effect, as it ignores the fact that the measurements were taken on the same trees.

```

/* Two-sample t-test. */
PROC TTEST DATA=library.trees;
    VAR dbh; CLASS year; run;

```

A better way to test the effect of time is to use a paired t-test, comparing each tree's diameter in 1994 to its diameter in 1984. For this we must use the restructured, "by subject" version of the dataset, `trees8494`, and test the average growth against zero. Variable growth was calculated for each tree as the difference between `dbh94` and `dbh84`.

```

/* Paired t-test of dbh84 & dbh94 is a One-sample t-test on growth */
PROC TTEST DATA=library.trees8494;
    VAR growth; run;

```

Linear Regression

Use PROC REG for simple and multiple linear regression. There are many choices of model selection algorithms, regression diagnostics, plots, output datasets, and other features. PROC REG requires the "by subject" data organization. We use `trees8494` to do a regression of `dbh94` on `dbh84`.

```

/* Simple linear regression with plots*/
PROC REG DATA=library.trees8494;
    MODEL dbh94=dbh84;          * dbh94 (dependent) on dbh84 (independent);
    PLOT dbh94*dbh84;          * dbh94 (y-axis) vs. dbh84 (x-axis);
    PLOT r.*dbh84;            * residual (y-axis) vs. dbh84 (x-axis);
    run;

```

ANOVA

There are several PROCs available for Analysis of Variance. PROC ANOVA is computationally more efficient, but it can only be used for one-way ANOVAs, and for balanced designs (equal cell-sizes). PROC GLM uses very similar syntax, and is suitable for more general, unbalanced designs. Except for very large designs, the computational efficiency of PROC ANOVA will not make any practical difference. Therefore, if in doubt, use PROC GLM for traditional analysis of variance computations. PROC MIXED is useful for repeated measures with missing values, or those that do not satisfy the traditional "sphericity" assumptions, but is beyond the scope of this introduction. Here we use the "by subject" `wp525one` dataset.

```
/* One-way Univariate ANOVA testing effect of Condition on Score1*/  
PROC GLM DATA=library.wp525one;  
  CLASS condition;  
  MODEL score1=condition;  
  run;  
  
/* Traditional Repeated Measures, "by subject" data organization.  
  1 between-subject, 1 within-subject factor.  
  Testing effect of Condition, Time, and Interaction.*/  
PROC GLM DATA=library.wp525one;  
  CLASS condition;  
  MODEL score1-score4=condition / nouni;  
  REPEATED time 4;  
  run;
```

(*) Data adapted from Winer, B.J., "Statistical Principles in Experimental Design", 2nd Ed. McGraw-Hill Book Company, 1962

(**) Data courtesy of B. Wilson, measurements from UMass Cadwell Forest.

Appendix:

The forestry data used in some of the examples comes from measurements at UMass Cadwell Forest, courtesy of B. Wilson. The data used in the examples are:

Tree ID	Species	Location	Year	DBH
1	15	1	84	17.00
1	15	1	94	18.30
2	27	1	84	23.10
2	27	1	94	22.80
3	3	2	84	4.70
3	3	2	94	5.10
4	31	2	84	5.75
4	31	2	94	7.00
5	3	2	84	10.00
5	3	2	94	11.00
6	3	3	84	4.60
7	10	3	84	12.50
7	10	3	94	13.80
8	3	3	84	7.70
8	3	3	94	8.10
9	3	3	84	8.55
9	3	3	94	6.60
10	3	3	84	6.20
10	3	3	94	9.30
11	31	4	84	5.10
11	31	4	94	6.40
12	3	4	84	4.75
12	3	4	94	5.30

The Species codes are:

Species Code	Species Name
3	Maple
10	Hickory
15	Ash
27	Oak
31	Hemlock

The Location codes are:

Location Code	Location Name
1	NE
2	SE
3	SW
4	NW