I. WORKING WITH DATES IN SAS

How do you work with date variables, date functions, and display dates in SAS? Here is some data (this is actually one line per observation, but it wrapped):

```
115 Capitola                          10/28/1999  10:00 AMStorm Surge               2
2     1.0M
116 CAZ043                             12/6/1999   1:50 PMHeavy Surf                1
0     150K
117 CAZ001                            12/14/1999   7:30 AMHeavy Seas                1
0         0
118 CAZ091>092 - 095                    2/3/2000  11:30 AMWind                     1
4     100K
```

The program below reads the data and creates a temporary data file called **storms**. Note that the dates are read in the **DATA STEP**, and the format mmddyy10. is used to read the date variable. Grab the program:

http://www.stat.ucla.edu/~vlew/stat130b/week3/calstorms.sas it is also called

http://www.stat.ucla.edu/~vlew/stat130b/week3/calstormsprogram.txt

```
filename in url
         "http://www.stat.ucla.edu:80/~vlew/stat130b/week3/calstorms.prn";
data storms(drop=idno);
infile in;
input idno
      location $35.
      @40 ddate mmddyy10.
      report_time time10.
      event $26.
      deaths
      injuries
      cost $;
proc print;
 format ddate mmddyy10. report_time timeampm8.;
run;
```

The output of the **PROC PRINT** without the format statement is presented below. Compare the dates in the data to the values of DDATE. Note that for Capitola the input date is 10/28/1999 and the value for DDATE is 14545. This is because dates are stored internally in SAS as the number of days from Jan 1,1960. If we had dates before 1960 the value of DDATE would be negative. A date exactly on January 1, 1960 would be zero.

115	Capitola	14545	36000	Storm Surge	2	2	1.0M
116	CAZ043	14584	49800	Heavy Surf	1	0	150K
117	CAZ001	14592	27000	Heavy Seas	1	0	0
118	CAZ091>092 - 095	14643	41400	Wind	1	4	100K

In order to see the dates in a way that we understand you would have to format the output. We use the MMDDYY10**.** format to see dates in the form of the original**.** This is specified on a **FORMAT** statement during the proc print.

Here is the output produced by the **PROC PRINT** statement with the right format:

```
115 Capitola                10/28/1999 10:00 AM Storm Surge          2       2    1.0M
116 CAZ043                  12/06/1999  1:50 PM Heavy Surf           1       0    150K
117 CAZ001                  12/14/1999  7:30 AM Heavy Seas           1       0    0
118 CAZ091>092 - 095        02/03/2000 11:30 AM Wind                 1       4    100K
```

From SAS:

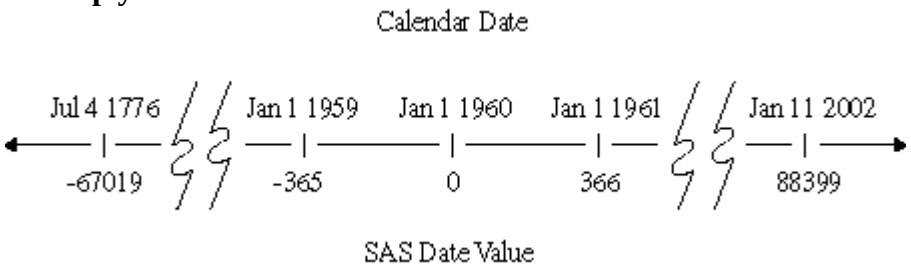
SAS date value
is a value that represents the number of days between January 1, 1960, and a specified date. SAS can perform calculations on dates ranging from A.D. 1582 to A.D. 19,900. Dates before January 1, 1960, are negative numbers; dates after are positive numbers.
SAS date values account for all leap year days, including the leap year day in the year 2000.

SAS date values can reliably tell you what day of the week a particular day fell on as far back as September 1752, when the calendar was adjusted by dropping several days. SAS day-of-the-week and length-of-time calculations are accurate in the future to A.D. 19,900.

SAS time value
is a value representing the number of seconds since midnight of the current day. SAS time values are between 0 and 86400

So in our example above, 10:00AM is the same as 36000 (60 * 60 * 10) to SAS.

To help you understand SAS dates



Look at the following data. At first glance it looks like the dates are so different that they couldn't be read. They do have two things in common:

1) they all have numeric months,
2) they all are ordered month, day, and then year.

```
John 1 1 1960
```

```
Mary 07/11/1955
Joan 07-11-1955
Kate 11.12.1962
Mark 06081959
```

These dates can be read with the same format, **MMDDYY11.** An example of the use of that format in a **DATA STEP** follows.

```
DATA dates;
   INPUT  name $ 1-4 @6 bday mmddyy11.;
CARDS;
John 1 1 1960
Mary 07/11/1955
Joan 07-11-1955
Kate 11.12.1962
Mark 06081959
;
RUN;
PROC PRINT DATA=dates;
   FORMAT bday date9. ;
RUN;
```

The results of the above **PROC PRINT** show that all of the dates are read correctly.

```
OBS    NAME          BDAY

 1     John     01JAN1960
 2     Mary     11JUL1955
 3     Joan     11JUL1955
 4     Kate     12NOV1962
 5     Mark     08JUN1959
```

There are a number of SAS informats and formats available to you when you are working with dates and times:

Date and Time

DATEw.	Reads date values in the form ddmmmyy or ddmmmyyyy
DATETIMEw.	Reads datetime values in the form ddmmmyy hh:mm:ss.ss or ddmmmyyyy hh:mm:ss.ss
DDMMYYw.	Reads date values in the form ddmmyy or ddmmyyyy
EURDFDEw.	Reads international date values
EURDFDTw.	Reads international datetime values in the form ddmmmyy

hh:mm:ss.ss or ddmmmyyyy hh:mm:ss.ss

EURDFMYw. Reads month and year date values in the form mmmyy or mmmyyyy

JULIANw. Reads Julian dates in the form yyddd or yyyyddd

MMDDYYw. Reads date values in the form mmddyy or mmddyyyy

MONYYw. Reads month and year date values in the form mmmyy or mmmyyyy

TIMEw.
Reads hours, minutes, and seconds in the form hh:mm:ss.ss

TODSTAMPw.
Reads an eight-byte time-of-day stamp

YYMMDDw.
Reads date values in the form yymmdd or yyyymmdd

YYMMNw.
Reads date values in the form yyyymm or yymm

YYQw.
Reads quarters of the year

Sometimes you will see dates broken into three separate variables month, **year**, and **day**.

```
12 1957 21
 1 1960 1
11 1962 12
 1 1988 22
```

One option is to read these data with each portion of the date as a separate variable and then use a function to convert the variables into a SAS date:

```
DATA dates;
   INPUT  month 1-2 year 4-7 day 9-10;
   bday=MDY(month,day,year);
CARDS;
12 1957 21
 1 1960 1
11 1962 12
 1 1988 22
;
```

```
RUN;
PROC PRINT DATA=dates;
   FORMAT bday date9. ;
RUN;
```

The results of the proc print follow. Notice we used a different date format for bday.

Obs	month	year	day	bday
1	12	1957	21	21DEC1957
2	1	1960	1	01JAN1960
3	11	1962	12	12NOV1962
4	1	1988	22	22JAN1988

A. SAS Dates and Y2K

Consider the following data, which are the same as above except that only 2 digits are used to signify the year, and year appears last. Suppose these are birthdates of people who are currently alive:

```
 7 11 18
 7 11 48
 1  1 60
10 15 90
12 10 00
```

Reading the data is the same as the example above using the program:

http://www.stat.ucla.edu/~vlew/stat130b/week3/yearcutoff2000.sas

```
DATA dates;
   INPUT  month day year ;
   bday=MDY(month,day,year);
CARDS;
 7 11 18
 7 11 48
 1  1 60
10 15 90
12 10 00
;
RUN;
proc print;
format bday mmddyy10.;
run;
```

The results of the **PROC PRINT**. What do you notice?

Obs	month	day	year	bday

1	7	11	18	07/11/2018
2	7	11	48	07/11/1948
3	1	1	60	01/01/1960
4	10	15	90	10/15/1990
5	12	10	0	12/10/2000

The first observation is showing a 2018 instead of 1918. By default, SAS assumes that two digit years less than 20 have a "20" in front of them. To get around this, use a SAS option called YEARCUTOFF. Get the program:

http://www.stat.ucla.edu/~vlew/stat130b/week3/yearcutoff1900.sas

```
options yearcutoff=1900;
DATA dates;
   INPUT  month day year ;
   bday=MDY(month,day,year);
CARDS;
7 11 18
7 11 48
1  1 60
10 15 90
12 10 00
;
RUN;
proc print;
format bday mmddyy10.;
run;
```

Obs	month	day	year	bday
1	7	11	18	07/11/1918
2	7	11	48	07/11/1948
3	1	1	60	01/01/1960
4	10	15	90	10/15/1990
5	12	10	0	12/10/1900

There is no complete answer to the Y2K problem, but with the **YEARCUTOFF=** option SAS helps. The ultimate answer is to use 4 digit years.

B. Computations with Elapsed Dates

SAS date variables make computations involving dates very convenient. For example, to calculate how long movies generally run (in days and weeks) use this program:

http://www.stat.ucla.edu/~vlew/stat130b/week3/movierun.sas

or

http://www.stat.ucla.edu/~vlew/stat130b/week3/movierunprogram.txt

```
options yearcutoff=1920;
filename in url
"http://www.stat.ucla.edu:80/~vlew/stat130b/week3/movies.csv";
data movies;
infile in delimiter = ',' MISSOVER DSD lrecl=111 firstobs=2;
      format title $49. ;
      format total_gross_receipts best12. ; /* try dollar12. */
      format opening_week_gross best12. ;
      format opening_day_screens best12. ;   /* try comma12. */
      format widest_release best12. ;
      format first_week_receipts best12. ;   /* this wasn't input correctly
*/
      format per_theater_receipts best12. ;
      format distributor $18. ;
      format opening_date mmddyy10. ; /* try mmddyyp10. */
      format closing_date mmddyy10. ;

  /* format controls how the variable will be presented */

      informat title $49. ;
      informat total_gross_receipts best32. ;
      informat opening_week_gross best32. ;
      informat opening_day_screens best32. ;
      informat widest_release best32. ;
      informat first_week_receipts best32. ;
      informat per_theater_receipts best32. ;
      informat distributor $18. ;
      informat opening_date mmddyy10. ;
      informat closing_date mmddyy10. ;

  /* informat controls how the variable will be read in with the input
statement */

input
       title $
       total_gross_receipts
       opening_week_gross
       opening_day_screens
       widest_release
       first_week_receipts
       per_theater_receipts
       distributor $
       opening_date
       closing_date
;;;;
percent_receipts=first_week_receipts/100;
days_run=datdif(opening_date,closing_date,'act/act');
weeks_run=days_run/7;
quarter_release=qtr(opening_date);
day_of_week=weekday(opening_date); /* 1= Sunday, 7=Saturday */
run;
proc sort;
  by descending weeks_run;
proc print;
 var title opening_date closing_date days_run weeks_run;
run;
```

and the output:

Obs	title	opening_ date	closing_ date	days_run	weeks_ run
1	Postman, The	06/14/1995	10/17/1996	491	70.1429
2	Deep Rising	01/30/1998	04/23/1999	448	64.0000
3	Desperate Measures	01/30/1998	04/16/1999	441	63.0000
4	Bulletproof	09/06/1996	10/31/1997	420	60.0000
5	Fargo	03/08/1996	04/10/1997	398	56.8571
6	Orgazmo	10/23/1998	11/19/1999	392	56.0000
7	Braveheart	05/24/1995	05/09/1996	351	50.1429
8	Life is Beautiful	10/23/1998	10/07/1999	349	49.8571
9	Bridges of Madison County, The	06/02/1995	04/18/1996	321	45.8571
10	Saving Private Ryan	07/24/1998	06/03/1999	314	44.8571

to calculate time in years from days, the general procedure is to divide by 365.25.

C. Other Useful Date Functions

We ran into some of these in the previous lecture:

Function	Description	Sample
month()	Extracts Month	m=MONTH(bday);
day()	Extracts Day	d=DAY(bday) ;
year()	Extracts Year	y=YEAR(bday);
weekday()	Extracts Day of Week	wk_d=WEEKDAY(bday);
qtr()	Extracts Quarter	q=QTR(bday);

D. Date Summary

- Dates are read with date formats, most commonly **DATE9.** and **MMDDYY10.**
- Date functions can be used to create date values from their components **(mdy(m,d,y))**, and to extract the components from a date value **(month(),day(), etc.)**.
- The **YEARCUTOFF** option may be used to control where the 2000 break comes if you have to read two digit years.

II. Sums and Counts in a Data Set

Let's look at the movie dataset again. We have a variable called total_gross_receipts and you want to accumulate a grand total of gross receipts across all observations of a data set.

Since all observations are automatically set to missing at the top of every loop of a DATA step the following equation will not work:

1) grand_total=grand_total+total_gross_receipts;

However, there exists an alternative statement that will produce the desired result:

2) grand_total+total_gross_receipts;

The reason the first statement does not work and the second one does work is that SAS interprets the following two statements 1) and 2) in very different ways:

In 1) the "+" is simply the addition operator of two variables. There is no implicit initializing (of total to the right of the "=" and to the left of the "+") or retaining (and therefore no accumulation). In other words, a statement such as x=y+z in a DATA step yields only row-specific values. If either y or z is missing, then x will be missing also.

Using statement 1) when the DATA step first begins, the first record creates the variable grand_total, but since it is automatically assigned the missing data value, there isn't a number in it and each row of the table (observation) has it's own value.

The DATA step, is like reading a spreadsheet, one line at a time, across the row from left to right. You can use 1); however you also need to use a RETAIN statement along with it:

RETAIN grand_total 0;
grand_total=grand_total+total_gross_receipts;

will give you the desired total, since it tells SAS to retain the value of grand_total after outputting the record to the next record and give it an initial value of 0.

To calculate the total sum of some variable across all observations, SAS provides a special type of statement that can stand alone. With statement 2) SAS interprets the "+" as part of the definition of a SUM statement. The SUM statement in 2) does several things implicitly, including:

- initializes the variable on the left of the "+" (total) with a value of 0.
- RETAINs (i.e., does not set to missing) the value of this variable from one data step iteration to the next.
- accumulates the row-by-row values of cost into the accumulator variable called grand_total.

A different kind of sum is to sum across variables within a single observation, use the SUM function:

sum_y = SUM(y1,y2,y3,y4);

The function N gives the number of variables in the current observation that have values.

n_y = N(y1,y2,y3,y4);

The function NMISS gives the number of variables in the current observation that have missing values.

nmss_y = NMISS(y1,y2,y3,y4);

The sum of the two variables created here, n_y and nmss_y, must add to the number of variables listed in parenthesis, 4.

Rather than list all the variable names, if you they are coded with numerical indices at the end of the name, here is a very convenient short-hand notation to use to compute the sum, number with values, and the number missing data:

```
sum_y =  SUM(of y1-y20);  /* the sum of all the non-missing data */
n_y    = N(of y1-y20);          /* the number of non-missing data*/
nmss_y = NMISS(of y1-y20);  /* the number of missing data*/
```

The syntax SUM(y1-y20) or SUM(y1--y20) for a range of variables will not work. For example, in the statement

```
 sum_y = SUM(y1-y20);
```

the portion within the () is treated as the difference between y1 and y20 -- the other 18 variables are ignored! Likewise, SAS interprets SUM(y1--y20) as SUM(y1-(-y20)), i.e. SUM(y1+y20), the sum of y1 and y20.

If you have missing data, the SUM function works very differently than adding all the variables with straight addition. For example, assume the following equation is used to add the yi's:

```
sum_y = y1 + y2 + y3 + y4;
```

If one or more of the individual yi's are missing, then sum_y is also set to missing. The SAS SUM function [e.g., SUM(y1,y2,y3,y4) and SUM(of y1-y4)] give the sum of the data that are NOT missing. For that reason, it is very important to use the COUNT or NMISS functions along with it.

3) An easy way to get grand_total is to use PROC MEANS.

```
proc means sum;
  var total_gross_receipts;
run;
```

Retain and the sum operator then work well if you need a "running" total. A simpler "proc means" with the SUM option can generate a total as well.

III. Use of Arrays

What is a SAS array?
A SAS array is a set of variables grouped together for the duration of a data step by being given a name in an ARRAY statement

```
array pop(5) ga sc nc va wv;
```

* array name is "pop"
* the sequence of variable names ga sc nc va wv is the "array list"
* the variables in the array list are the "array elements"
* each array element in this example has a position number in the array list from 1 to 5

They are convenient to use. For example: pop(3) refers to the variable in position # 3 in the array pop (nc in above example). The Array elements must all be of same type (numeric or character) and they can be variables which exist or which will be created in the data step.

Here are some examples of "one-dimensional" arrays:

array x(1:6) a b c d e f;	x(5) same as e
array x(0:5) a b c d e f;	x(4) same as e
array quiz(20) q1-q20;	equivalent array declarations
array quiz(1:20) q1-q20;	quiz(4) same as q4
array quiz(*) q1-q20;	subscript lower bound=1, upper bound=20
array quiz(20);	SAS creates quiz1-quiz20 as array elements
array color(1:3) $ 1 red blue green;	character array, elements have length=1 character color(2) same as blue
array pop(1:5) yr95-yr99;	pop(2) same as yr96
array pop(95:99) yr95-yr99;	pop(96) same as yr96
array x(*) _numeric_;	all numeric variables on the observation
array y(*) _character_;	all character variables on the observation
array z(*) _all_;	all variables on the observation

(there are two dimensional arrays, but that's advanced)

Why use SAS arrays?

- repeat an action or set of actions on each of a group of variables
- write shorter programs
- restructure a SAS data set to change the unit of observation

Here is a situation. You have been given survey data and you need to "reverse scales" for a number variables. Suppose each of these variables have five responses to questions on a surveys. For example, responses might run from from "strongly agree" to "strongly disagree." :

1. strongly agree
2. somewhat agree
3. neither agree nor disagree
4. somewhat disagree
5. strongly disagree

Long surveys may reverse the order of responses to break the monotony of the survey or "check" to see if the respondent is really answering the question.

When it comes time to perform the analysis, it would be nice if the data all follow the same order. You might try to fix the problem in the following manner:

```
if var1 = 1 then newvar1 = 5 ;
     else if var1 = 2 then newvar2 = 4 ;
     else if var1 = 3 then newvar2 = 3 ;
     else if var1 = 4 then newvar2 = 2 ;
     else if var1 = 5 then newvar2 = 1;
if var2=  1 then newvar2 =5;
     /* etc etc */
     /*or you might try to be clever: */
newvar1 = 6 - var1;
newvar2 = 6 - var2;
```

This kind of coding subject to error. It requires that each value of each question to be reversed be specified or that each variable be specified. A SAS ARRAY allows us to group all variables that are to be treated similarly in one area. Here is one way to use a SAS array to reverse the scoring on three questions:

```
array stuff {*} var1 var2 var3;
 do I = 1 to 3 ;
   stuff{I} = 6 - stuff{I} ;
 end ;
  drop I ;
```

This SAS code will reverse a five point scale by subtracting the current value of whichever question we are working with (when I = 1, stuff{I} is stuff{1} which is var1; when I = 2, it is var2 and when I = 3, it is var3) from 6. Missing values will remain missing.

Simple examples using one-dimensional arrays

1. Recode the set of variables A B C D E F G in the same way: if the variable has a value of 99 recode it to SAS missing.

```
array v(7) a b c d e f g;
do k=1 to 7;
   if v(k)= 99 then v(k)=.;
```

```
end;
```

2. Each observation of your data set has five variables SEX1 SEX2 SEX3 SEX4 SEX5 which give the sex (1=male, 2=female) of up to 5 persons. You want to count the number of males (MALES) and the number of females (FEMALES) on each observation.

```
array sex(1:5) sex1-sex5;
males=0;
females=0;
do i=1 to 5;
    if sex(i)=1 then males=males+1;
                else if sex(i)=2 then females=females+1;
end;
```

3. Recode all numeric variables in your data set as follows: if a variable has a value of 98 or 99 recode it to SAS missing.

```
array nvar(*) _numeric_;
do i=1 to dim(nvar);
    if nvar(i)=98 or nvar(i)=99 then nvar(i)=.;
end;
```