STAT 425 and STAT 625
Statistical Software

Lecture 7

More on Working with Dates

Data Processing: Iterative Loops

Saida Zardi

Reading Data Values from Text Data

Four kinds of display of dates in this example:

Original text data 

```
001 10/21/1950 05122003 08/10/65 23Dec2005
002 01/01/1960 11122009 09/13/02 02Jan1960
```

```
data Four_Dates;
   infile 'C:\books\learning\Dates.txt' truncover;
   input @1  Subject    $3.
         @5 DOB        mmddyy10.
         @16 VisitDate mmddyy8.
         @26 TwoDigit  mmddyy8.
         @34 LastDate  date9.;
   format DOB VisitDate date9.
          TwoDigit LastDate mmddyy10.;
run;
```

Listing of data set Four_Dates

Subject	DOB	VisitDate	TwoDigit	LastDate
001	21OCT1950	12MAY2003	08/10/1965	12/23/2005
002	01JAN1960	12NOV2009	09/13/2002	01/02/1960

Note: The added TRUNCOVER (or PAD) option, the INFILE statement, prevent errors when reading raw data files with dates in fixed columns format.

Computing the number of years between two dates

- If you want to compute a person's age from the date of birth, you can use the SAS function YRDIF as in this program:

```
data Ages;
   set Four_dates;
   Age = yrdif(DOB,VisitDate);
run;
title "Listing of Ages";
proc print data=Ages noobs;
 var DOB VisitDate Age;
run;
```

Listing of Ages

DOB	VisitDate	Age
21OCT1950	12MAY2003	52.5562
01JAN1960	12NOV2009	49.8630

- If you want to know the Age as of a person's last birthday, without a fraction of a year, you can use the INT (integer) function:

  Age = INT(YRDIF(DOB, VisitDate));

- If you want to round the Age to the nearest year, you can use the ROUND function:

  Age = ROUND(YRDIF(DOB, VisitDate));

- And you can also use a simple operations as in:

  Age= (VisitDate- DOB)/365.25

Demonstrating a Date Constant

How do you compute a person's age as of a certain date?

```
data Ages;
   set Four_Dates;
   Age = yrdif(DOB,'01Jan2017'd);
run;
title "Listing of Ages";
proc print data=Ages;
   format Age 5.1;
run;
```

Listing of Ages

Obs	Subject	DOB	VisitDate	TwoDigit	LastDate	Age
1	001	21OCT1950	12MAY2003	08/10/1965	12/23/2005	66.2
2	002	01JAN1960	12NOV2009	09/13/2002	01/02/1960	57.0

Computing the current Date

What if you want to compute a quantity based on today's date:
use the TODAY function as in the following program:

```
data Ages;
    set Four_Dates;
    Age = yrdif(DOB,today());
run;
```

You can also use the DATE function that performs an identical task as the TODAY function.

Extracting the day of the week, the month and the year

The following program shows how we can extract the weekday, the day of the month, the month and the year

```
data Extract;
    set Four_Dates;
    Day = weekday(DOB);
    DayOfMonth = day(DOB);
    Month = Month(DOB);
    Year = year(DOB);
run;
title "Listing of Extract";
proc print data=Extract noobs;
    var DOB Day -- Year;
run;
```

Listing of Extract

DOB	Day	DayOfMonth	Month	Year
21OCT1950	7	21	10	1950
01JAN1960	6	1	1	1960

Creating a SAS date

MDY(month, day year) allows you to create a SAS date value
This is useful when you have a data set whee the day, month and year is provided but there's no corresponding date value, for example.

```
data MDY_Example;
    set Learn.Month_Day_Year;
    Date = mdy(Month, Day, Year);
    format Date mmddyy10.;
run;
proc print data=MDY_Example;
run;
```

Listing of Extract

Obs	Month	Day	Year	Date
1	10	21	1950	10/21/1950
2	3	.	2005	.
3	5	7	2000	05/07/2000

Substituting a day of the month when the day value is missing

The MISSING function tests if there's any missing data.

```
data Substitute;
   set Learn.Month_Day_Year;
   if missing(Day) then Date = mdy(Month,15,Year);
   else Date = mdy(Month,Day,Year);
   format Date mmddyy10.;
run;
proc print data=substitute;
run;
```

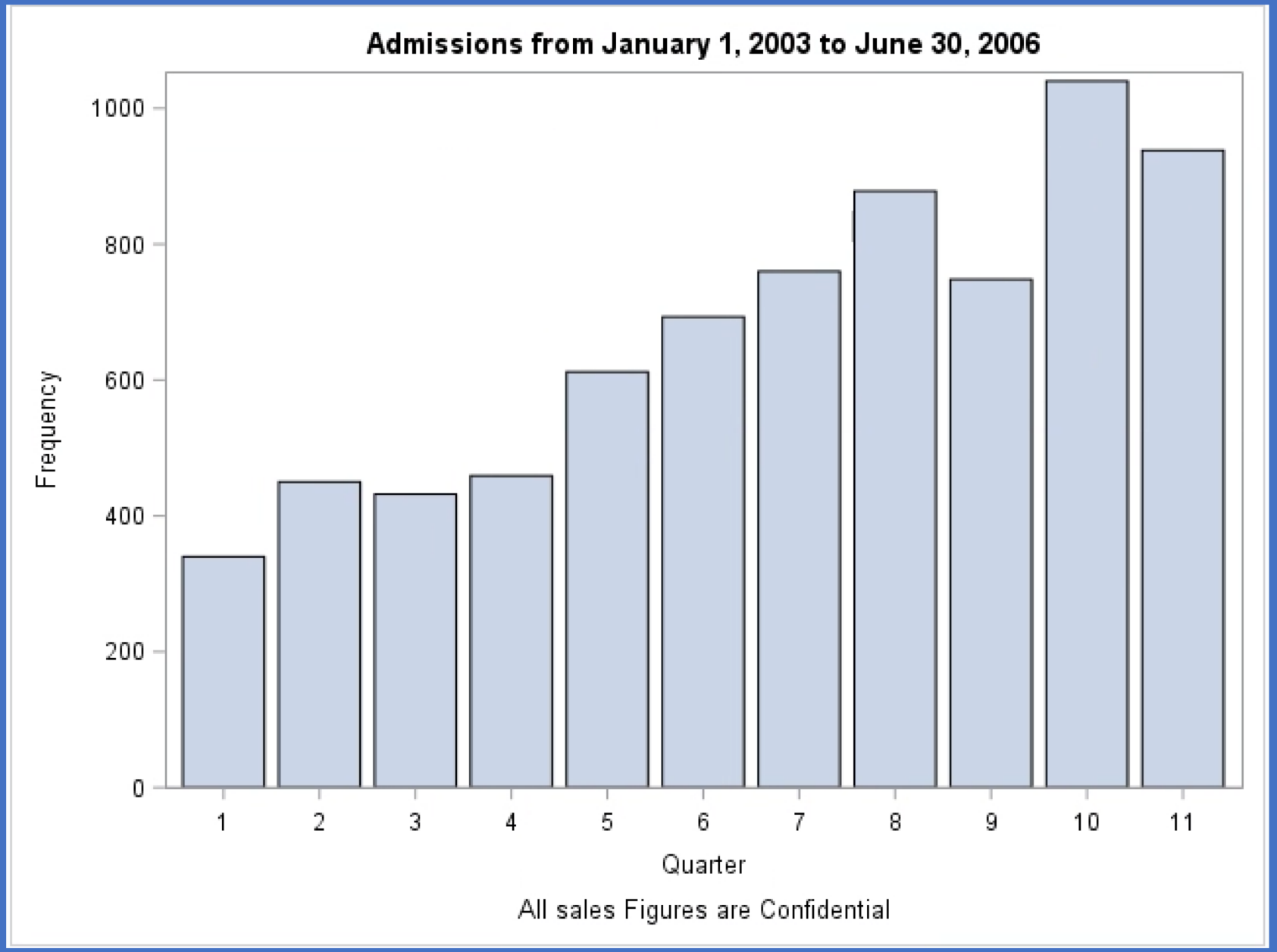
Listing of Extract

Obs	Month	Day	Year	Date
1	10	21	1950	10/21/1950
2	3	.	2005	03/15/2005
3	5	7	2000	05/07/2000

Using date interval functions

Two functions: INTCK and INTNX: each takes three arguments

```
data Frequency;
   set Learn.Hosp(keep=AdmitDate
                  where=(AdmitDate between '01Jan2003'd and
                        '30Jun2006'd));
   Quarter = intck('qtr','31Dec2002'd,AdmitDate);
run;
title "Admissions from January 1, 2003 to June 30, 2006";
proc sgplot data=Frequency;
   vbar Quarter;
run;
```


Admissions from January 1, 2003 to June 30, 2006
Frequency
1000
800
600
400
200
0
1
2
3
4
5
6
7
8
9
10
11
Quarter
All sales Figures are Confidential

In this program we use the INTEX function to compute Dates 6 months after Discharge

```
data Followup;
    set Learn.Hosp_Discharge;
    FollowDate = intnx('month',Discharge,6);
    format FollowDate date9.;
run;
proc print data=followup;
run;
```

Admissions from January 1, 2003 to June 30, 2006

Obs	Discharge	Patient	FollowDate
1	29NOV2003	879	01MAY2004
2	30NOV2003	880	01MAY2004
3	04SEP2003	883	01MAR2004
4	28AUG2003	884	01FEB2004
5	04SEP2003	885	01MAR2004
6	26AUG2003	886	01FEB2004
7	31AUG2003	887	01FEB2004
8	25AUG2003	888	01FEB2004
9	16NOV2003	913	01MAY2004
10	15NOV2003	914	01MAY2004

```
data Followup;
   set Learn.Hosp_Discharge;
   FollowDate = intnx('month',Discharge,6,'sameday');
   format FollowDate date9.;
run;

title "Listing of Data Set Follow-Up";
proc print data=Followup noobs;
run;
```

Listing of Data Set Follow-Up

Discharge	Patient	FollowDate
29NOV2003	879	29MAY2004
30NOV2003	880	30MAY2004
04SEP2003	883	04MAR2004
28AUG2003	884	28FEB2004
04SEP2003	885	04MAR2004
26AUG2003	886	26FEB2004
31AUG2003	887	29FEB2004
25AUG2003	888	25FEB2004
16NOV2003	913	16MAY2004
15NOV2003	914	15MAY2004

Performing Iterative Processing: Looping

DO groups

- IF – THEN statement can only have one action. If you add the keywords DO and END, then you can add more actions:

```
IF condition THEN DO;
    action ;
    action;
End;
```

The Do statement causes all SAS statements coming after it to be treated as a unit until a matching END statement appears.

The DO statement, the END statement and all other statements in between are called a DO group.

Program that doesn't use the DO statement

```
data Grades;
    length Gender $ 1
           Quiz   $ 2
           AgeGroup $ 13;
    infile 'J:\CLASSES\STAT46\Examples\Grades.txt' missover;
    input Age Gender Midterm Quiz FinalExam;
    if missing(Age) then delete;
    if Age le 39 then AgeGroup = 'Younger group';
    if Age le 39 then Grade    = .4*Midterm + .6*FinalExam;
    if Age gt 39 then AgeGroup = 'Older group';
    if Age gt 39 then Grade    = (Midterm + FinalExam)/2;
run;
title "Listing of Grades";
proc print data=Grades noobs;
run;
```

Program that uses the SAS statement

```
*8-2;
data Grades;
    length Gender $ 1
           Quiz   $ 2
           AgeGroup $ 13;
    infile 'J:\CLASSES\STAT46\Examples\Grades.txt' missover;
    input Age Gender Midterm Quiz FinalExam;
    if missing(Age) then delete;
    if Age le 39 then do;
       AgeGroup = 'Younger group';
       Grade = .4*Midterm + .6*FinalExam;
    end;
    else if Age gt 39 then do;
       AgeGroup = 'Older group';
       Grade = (Midterm + FinalExam)/2;
    end;
run;
title "Listing of Grades";
proc print data=Grades noobs;
run;
```

Output :

Listing of Grades

Gender	Quiz	AgeGroup	Age	Midterm	FinalExam	Grade
M	B-	Younger group	21	80	82	81.2
M	B+	Younger group	35	87	85	85.8
F		Older group	48	.	76	.
F	A+	Older group	59	95	97	96.0
M		Younger group	15	88	93	91.0
F	A	Older group	67	97	91	94.0
F	C-	Younger group	35	77	77	77.0
M	C	Older group	49	59	81	70.0

The Sum Statement

The SUM statement is used in two ways:

1- to calculate totals such as amonth-to-date totals

2 –Create a counter: a variable that is incremented by a fixed amount

On each iteration of a data step.

```
data Revenue;
    input Day : $3.
          Revenue : dollar6.;
    Total = Total + Revenue; /* Note: this does not work */
    format Revenue Total dollar8.;
datalines;
Mon $1,000
Tue $1,500
Wed  .
Thu $2,000
Fri $3,000
;
title "Listing of Revenue";
proc print data=Revenue noobs;
run;
```

Listing of Revenue

Day	Revenue	Total
Mon	$1,000	.
Tue	$1,500	.
Wed	.	.
Thu	$2,000	.
Fri	$3,000	.

Attempting to create accumulative total (1st attempt)

```
data Revenue;
    input Day : $3.
          Revenue : dollar6.;
    Total = Total + Revenue; /* Note: this does not work */
    format Revenue Total dollar8.;
datalines;
Mon $1,000
Tue $1,500
Wed  .
Thu $2,000
Fri $3,000
;
title "Listing of Revenue";
proc print data=Revenue noobs;
run;
```

Listing of Revenue

Day	Revenue	Total
Mon	$1,000	.
Tue	$1,500	.
Wed	.	.
Thu	$2,000	.
Fri	$3,000	.

Creating a Cumulative Total with the retain statement (Second Attempt)

```
data Revenue;
   retain Total 0;
   input Day : $3.
         Revenue : dollar6.;
   Total = Total + Revenue; /* Note: this does not work */
   format Revenue Total dollar8.;
datalines;
Mon $1,000
Tue $1,500
Wed  .
Thu $2,000
Fri $3,000
;
title "Listing of Revenue";
proc print data=Revenue noobs;
run;
```

Listing of Revenue

Total	Day	Revenue
$1,000	Mon	$1,000
$2,500	Tue	$1,500
.	Wed	.
.	Thu	$2,000
.	Fri	$3,000

Creating a Cumulative Total with the retain statement (Third Attempt)

A Sum statement takes the form: ***variable + expression***

```
data Revenue;
   retain Total 0;
   input Day : $3.
         Revenue : dollar6.;
   if not missing(Revenue) then Total = Total + Revenue;
   format Revenue Total dollar8.;
datalines;
Mon $1,000
Tue $1,500
Wed   .
Thu $2,000
Fri $3,000
;
title "Listing of Revenue";
proc print data=Revenue noobs;
run;
```

Listing of Revenue

Total	Day	Revenue
$1,000	Mon	$1,000
$2,500	Tue	$1,500
$2,500	Wed	.
$4,500	Thu	$2,000
$7,500	Fri	$3,000

Using a SUM statement to create a cumulative Total

A SUM statement does the following:

- Variable is retained
- Variable is initialized at 0
- Missing Values are ignored
- We obtain the following program:

```
data Revenue;
    input Day : $3.
          Revenue : dollar6.;
    Total + Revenue;
    format Revenue Total dollar8.;
datalines;
Mon $1,000
Tue $1,500
Wed   .
Thu $2,000
Fri $3,000
;
```

Using a SUM statement to set a counter:

This is a counter that counts the missing values by using MissCounter:

```
data Test;
    input x;
    if missing(x) then MissCounter + 1;
datalines;
2
.
7
.
;

run;
title 'Listing of Test';
Proc print data=Test;
run;
```

Listing of Test

Obs	x	MissCounter
1	2	0
2	.	1
3	7	1
4	.	2

The iterative DO Loop

```
data Compound;
   Interest = .0375;
   Total = 100;

   Year + 1;
   Total + Interest*Total;
   output;

   Year + 1;
   Total + Interest*Total;
   output;

   Year + 1;
   Total + Interest*Total;
   output;

   format Total dollar10.2;
run;
title "Listing of Compound";
proc print data=compound noobs;
run;
```

The statement Year+1 is a SUM statement

The value of TOTAL is also computed using a SUM statement

The increment here is by an expression: Interest * Total

Listing of Compound

Interest	Total	Year
0.0375	$103.75	1
0.0375	$107.64	2
0.0375	$111.68	3

An iterative DO Loop

- In this program: YEAR is first set to 1, the lower limit in the DO range.
- All the statements up to the END statement are executed and YEAR is automatically incremented.
- SAS tests if the new value of YEAR is between the lower and the upper limit.
  - ❖ If it is, the the statement in the DO group execute again,
  - ❖ If not, the program continues at the 1st line following the END statement.

```
data Compound;
   Interest = .0375;
   Total = 100;
   do Year = 1 to 3;
      Total + Interest*Total;
      output;
   end;
   format Total dollar10.2;
run;
title "Listing of Data Set Compound";
proc print data=Compound noobs;
run;
```

Listing of Data Set Compound

Interest	Total	Year
0.0375	$103.75	1
0.0375	$107.64	2
0.0375	$111.68	3

One form of the iterative DO statement follows:

DO index-variable = start to stop by increment;

If you leave off the ***increment,*** it defaults to ***1.***

Using an iterative DO Loop:

This program does not have input data.
It generates a value of n in the DO Loop, computes the square root
And outputs an observation in the data set. This continues for all values from 1 to 10.

```
data Table;
    do n = 1 to 10;
        Square = n*n;
        SquareRoot = sqrt(n);
        output;
    end;
run;
title "Table of Squares and Square Roots";
proc print data=table noobs;
run;
```

Table of Squares and Square Roots

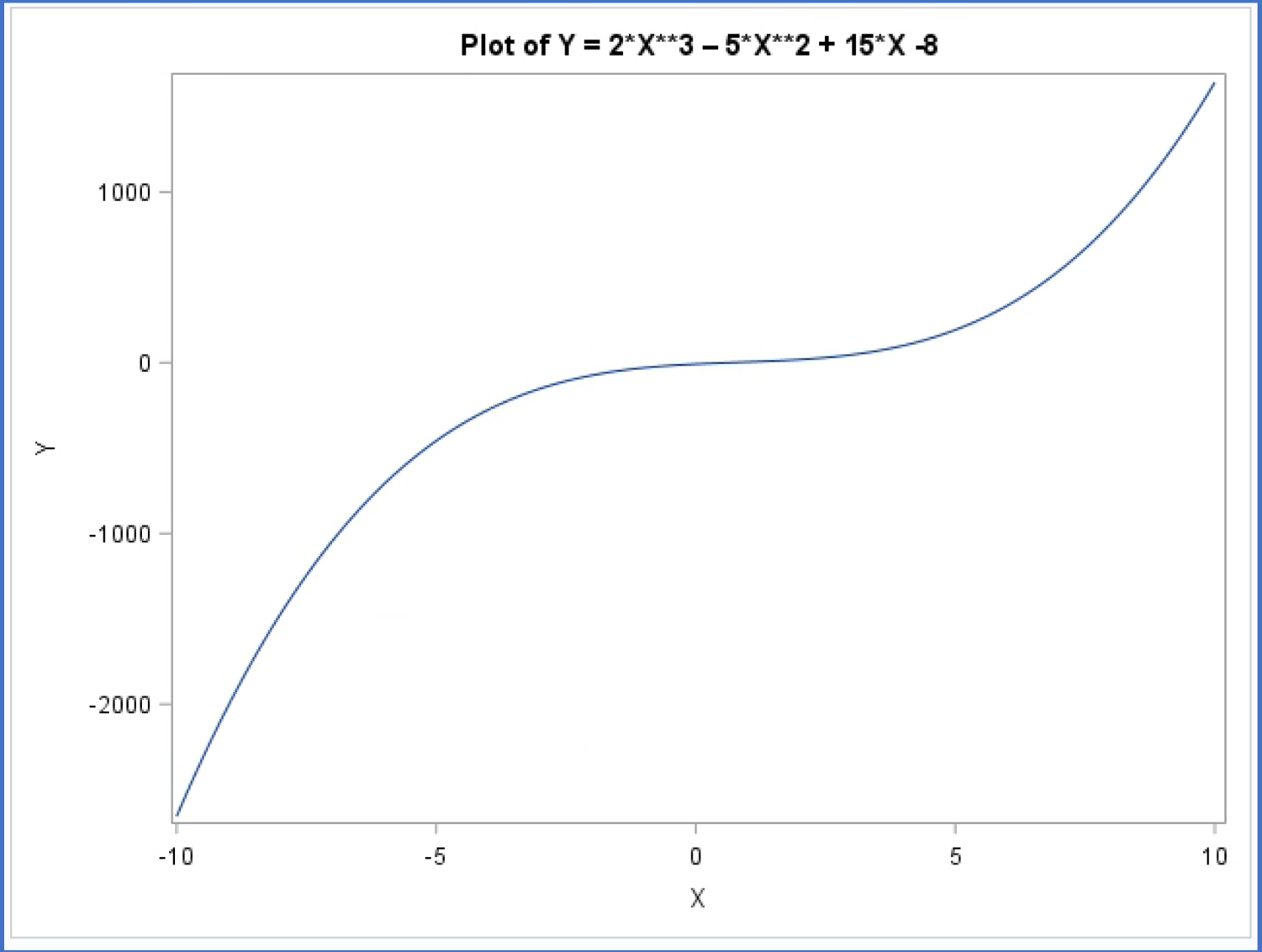
n	Square	SquareRoot
1	1	1.00000
2	4	1.41421
3	9	1.73205
4	16	2.00000
5	25	2.23607
6	36	2.44949
7	49	2.64575
8	64	2.82843
9	81	3.00000
10	100	3.16228

Using an interative DO Loop to graph an equation:

In this program, you have an equation relating X to Y and want a graph of Y versus X for values of X from -10 to +10.
The iterative DO Loop makes it easy:

```
data Equation;
    do X = -10 to 10 by .01;
       Y = 2*X**3 - 5*X**2 + 15*X -8;
       output;
    end;
run;
title "Plot of Y = 2*X**3 - 5*X**2 + 15*X -8";
proc sgplot;
    series x=X y=Y;
run;
```

This is the Output of the previous program



Other forms of an iterative DO loop

Using character values in the DO Statement::

```
data Easyway;
    do Group = 'Placebo','Active';
        do Subj = 1 to 5;
            input Score @;
            output;
        end;
    end;
datalines;
250 222 230 210 199
166 183 123 129 234
;
title "Listing of Data Set Easyway";
proc print data=Easyway noobs;
run;
```

Listing of Data Set Easyway

Group	Subj	Score
Placebo	1	250
Placebo	2	222
Placebo	3	230
Placebo	4	210
Placebo	5	199
Active	1	166
Active	2	183
Active	3	123
Active	4	129
Active	5	234

DO WHILE and DO UNTIL statement

Instead of choosing a stopping value for an iterative DO Loop, when a condition is met is true, we can use a DO Until or a DO WHILE.

The following program is an example:

```
data Double;
    Interest = .0375;
    Total = 100;
    do until (Total ge 200);
        Year + 1;
        Total = Total + Interest*Total;
        output;
    end;
    format Total dollar10.2;
run;
title "Listing of Double";
proc print data=Double noobs;
run;
```

Listing of Double

Interest	Total	Year
0.0375	$103.75	1
0.0375	$107.64	2
0.0375	$111.68	3
0.0375	$115.87	4
0.0375	$120.21	5
0.0375	$124.72	6
0.0375	$129.39	7
0.0375	$134.25	8
0.0375	$139.28	9
0.0375	$144.50	10
0.0375	$149.92	11

An alternative to DO UNTIL is a DO While.
A DO while loop iterates as long as the condition following the WHILE is true.
The WHILE condition is tested at the top of the Loop rather that at the bottom.
The DO WHILE block does not execute even once if the condition is false.

Demonstrating a DO While Statement

```
data Double;
   Interest = .0375;
   Total = 100;
   do while (Total le 200);
      Year + 1;
      Total = Total + Interest*Total;
      output;
   end;
   format Total dollar10.2;
run;
proc print data=double noobs;
   title "Listing of Double";
run;
```

DO WHILE Loops are evaluated at the TOP:

```
data Double;
   Interest = .0375;
   Total = 300;
   do while (Total lt 200);
      Year + 1;
      Total = Total + Interest*Total;
      output;
   end;
   format Total dollar10.2;
run;
```

Caution when using Do Until statements

Itis important that the condition you place on a DO UNTIL statement becomes true at some point.

```
data Double;
    Interest = .0375;
    Total = 300;
    do while (Total lt 200);
        Year + 1;
        Total = Total + Interest*Total;
        output;
    end;
    format Total dollar10.2;
run;
```

Leave and Continue Statements

The LEAVE statement inside a DO Loop shifts control to the statement following the END statement. At the Bottom of the Loop.

```
data Leave_it;
   Interest = .0375;
   Total = 100;
   do Year = 1 to 100;
      Total = Total + Interest*Total;
      output;
      if Total ge 200 then leave;
   end;
   format Total dollar10.2;
run;
```

The Continue statement halts further statements within the DO loop from executing and continues iteration in the Loop.

```
data Continue_on;
    Interest = .0375;
    Total = 100;
    do Year = 1 to 100 until (Total ge 200);
        Total = Total + Interest*Total;
        if Total le 150 then continue;
        output;
    end;
    format Total dollar10.2;
run;
title "Listing of Data Set Continue_on";
proc print data=Continue_on noobs;
run;
```

Listing of Data Set Continue_on

Interest	Total	Year
0.0375	$155.55	12
0.0375	$161.38	13
0.0375	$167.43	14
0.0375	$173.71	15
0.0375	$180.22	16
0.0375	$186.98	17
0.0375	$193.99	18
0.0375	$201.27	19