

Lecture 1.pdf

Lecture 2.pdf

Lecture 3.pdf

Lecture 4.pdf

Lecture 5.pdf

Lecture 6.pdf

Lecture 7.pdf

Lecture 8.pdf

Lecture 9.pdf

Lecture 10.pdf

Lecture 11.pdf

Lecture 12.pdf

Lecture 13.pdf

Lecture 14 .pdf

SAS Components



- This class
 - Base SAS – basic procedures and data management
- SAS has over 200 components, including
 - SAS/STAT – statistical analysis
 - SAS/GRAFPH – high quality graphics & presentations
 - SAS/ACCESS – reads data directly from databases
 - SAS/ETS – econometrics and time series
 - SAS/INSIGHT – data mining
 - SAS/QC – quality control
 - SAS/PH – clinical trials

Applications



- Why are we using it?
- What are some of the applications?
 - Statistical analysis
 - Data management
 - Creating reports

Today's Objectives

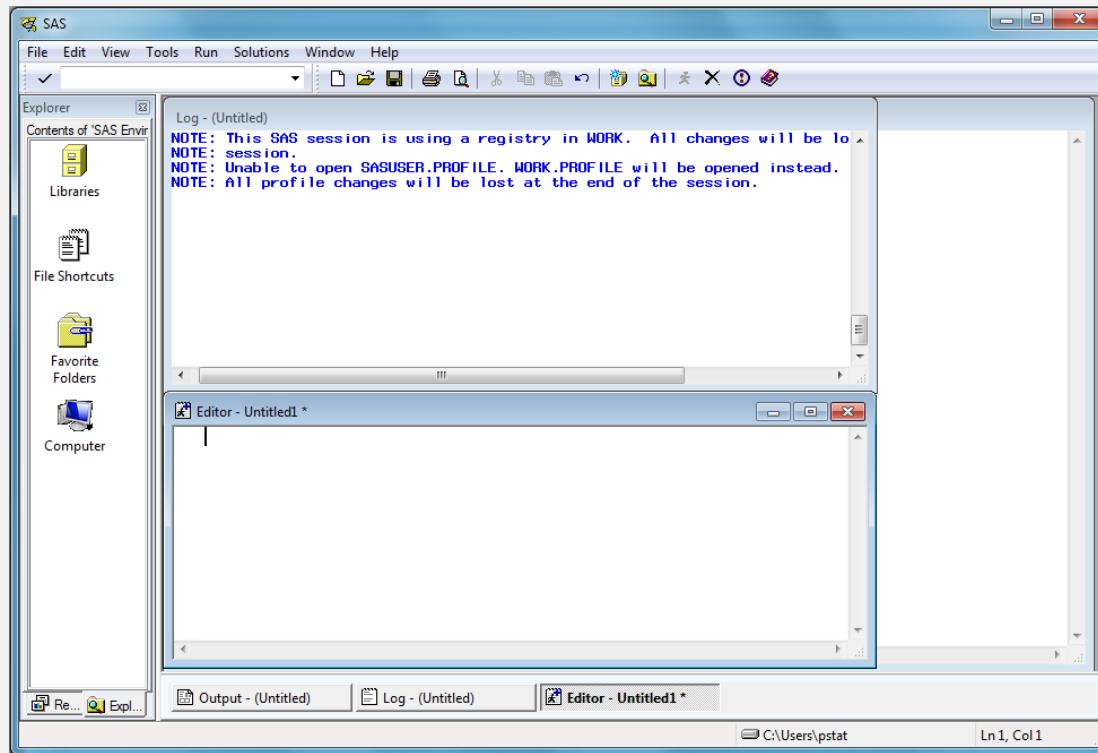


- Open the program
- Learn how to navigate the program
- Learn some basic syntax
- Create our first SAS program!

First Step



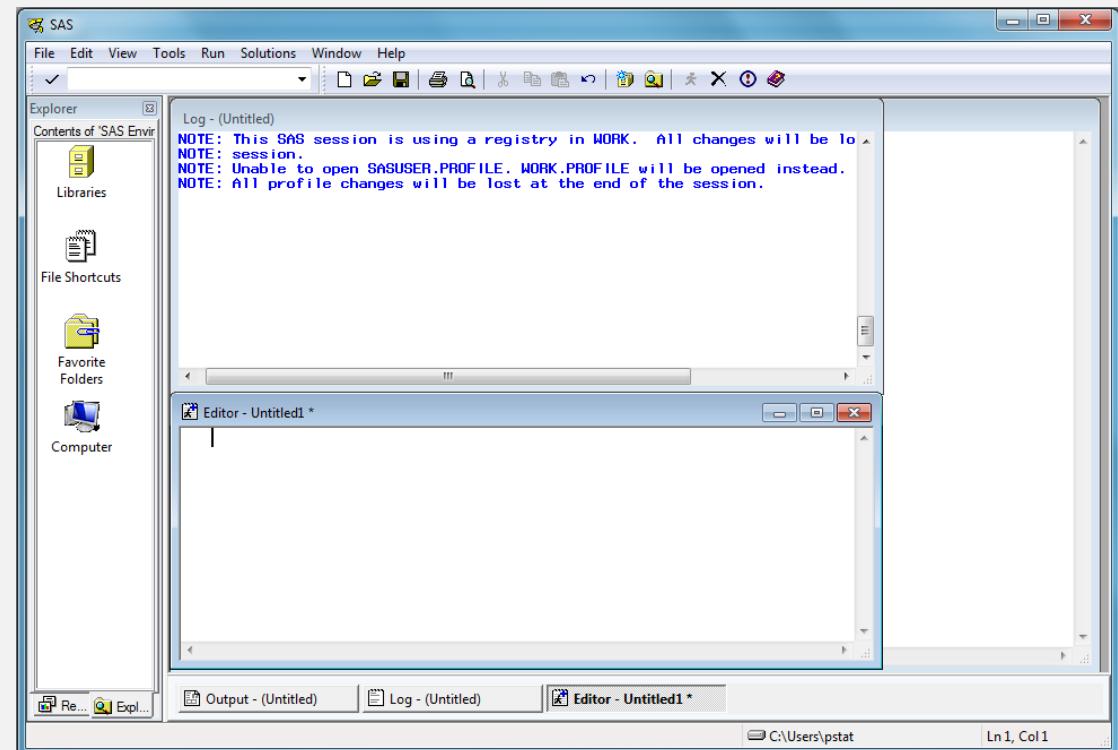
- Open SAS
 - Menu -> All Programs -> Math & Stats -> SAS -> SAS 9.4



The Main Windows



- Five Main Windows
 - Editor
 - Log
 - Output
 - Results
 - Explorer



Editor



- Edit, execute, and save SAS programs

The screenshot shows the SAS Editor interface. The title bar says "ex1 *". The main window displays a SAS program:

```
data objectives;
    input topics $ minutes;
    datalines;
    intro 5
    syllabus 10
    windows 5
    steps 5
    library 10
    print 20
    syntax 5
    quiz 5
    ;
    run;
proc print;
    run;
```

At the bottom, there are tabs for "Output - (Untitled)", "Log - (Untitled)", and the active tab "ex1 *".

Log



- Displays status messages regarding the execution of SAS procedures

The screenshot shows the SAS Log window titled "Log - tmp". The window displays the following SAS code and its execution results:

```
1  data objectives;
2      input topics $ minutes;
3      datalines;

NOTE: The data set WORK.OBJECTIVES has 8 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time          0.21 seconds
      cpu time           0.01 seconds

12  ;
13  run;
14  proc print;
NOTE: Writing HTML Body file: sashtml.htm
15  run;

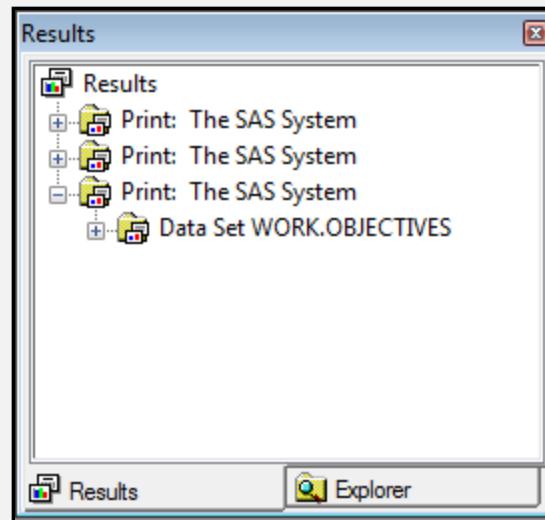
NOTE: There were 8 observations read from the data set WORK.OBJECTIVES.
NOTE: PROCEDURE PRINT used (Total process time):
      real time          1.79 seconds
      cpu time           0.45 seconds
```

The log window has a toolbar at the bottom with icons for Output, Log, and Results Viewer.

Results



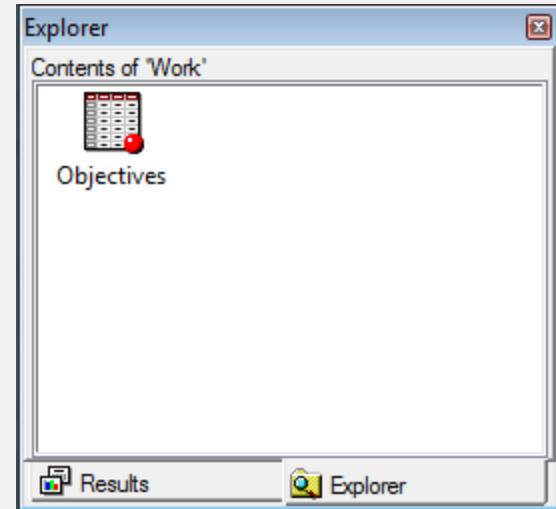
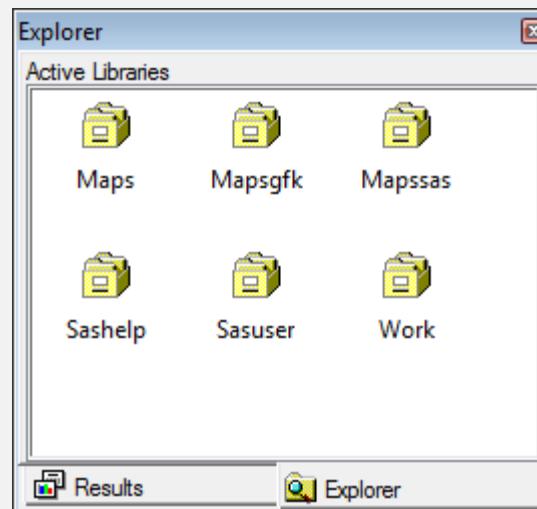
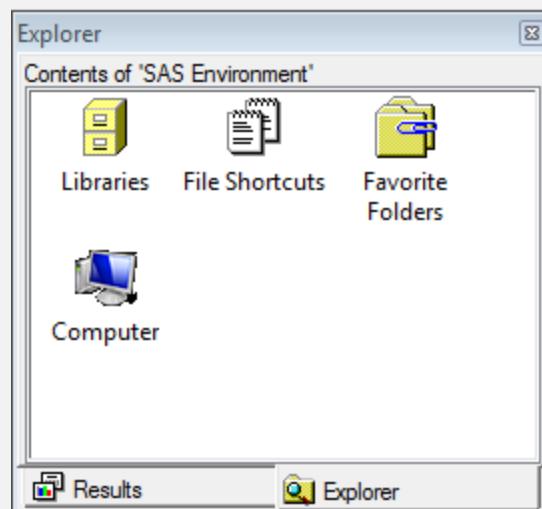
- Displays links to previously executed results



Explorer



- Navigate libraries, data sets, and other SAS objects



Output + Results Viewer



- Output window:

The SAS System			13:49 Wednesday, March 18, 2015	1
Obs	topics	minutes		
1	intro	5		
2	syllabus	10		
3	windows	5		
4	steps	5		
5	library	10		
6	print	20		
7	syntax	5		
8	quiz	5		

- Results viewer

- Default output for SAS 9.4
- A single, continuous html report
- Not affected by options like
 - ✖ Page size, page number, etc.

The SAS System		
Obs	topics	minutes
1	intro	5
2	syllabus	10
3	windows	5
4	steps	5
5	library	10
6	print	20
7	syntax	5
8	quiz	5

Another Useful Window: Table Editor

The image shows two overlapping windows of a "VIEWTABLE" application. The window on the left is titled "VIEWTABLE: Work.Objectives" and contains a table with columns "topics" and "minutes". The data is as follows:

	topics	minutes
1	intro	5
2	syllabus	10
3	windows	5
4	steps	5
5	library	10
6	print	20
7	syntax	5
8	quiz	5

The window on the right is titled "VIEWTABLE(New): (Untitled)" and contains a table with columns A through E. The data is as follows:

	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					
8					
9					

Getting Started



- How to submit/execute a program?

- Make sure the Editor window is active

- ✖ Submitting the entire program

- Run → Submit

- Click on the “running man” symbol in the toolbar



- Command line → Type ‘submit’ and hit Enter

- F8

- ✖ Submitting a portion of the program

- Highlight the portion you’d like to submit

- <right click> → Submit Selection

- F8

Getting Started



- How to save a program?
 - File → Save
 - Ctrl + S

File Extensions



- What are the file extensions for
 - A SAS program?
 - .sas
 - A SAS data set?
 - .sas7bdat
 - A SAS log file?
 - .log
 - A SAS output file?
 - .html (by default)

The Basics: SAS Process



The Basics



- SAS statements
 - Always begin with a keyword
 - Always end with a semicolon (;)
 - Are free format
 - i.e. Can begin at any location and end at any location
 - Entire program can be written on one line, or many lines
 - EXCEPT when using the datalines; statement
- SAS
 - Is not case sensitive
 - i.e. daTa nOtCaseSensitive;
 - EXCEPT in the case of string comparisons

The Basics



- Names of SAS data sets and variables must
 - Be no longer than 32 characters
 - Begin with a letter or underscore
 - Contain only letters, numbers, or underscores (_)

The Basics



- Comments
 - What are they?
 - Why should we use them?
 - Single line: begin with an asterick (*) and ends with a semicolon (;)
 - Multiple line: begins with a /* and ends with a */

The Basics



- SAS 9.4 has context-sensitive help.
 - Highlight a keyword and press F1

Libraries



- SAS file names are always contain 2 levels
 - Level 1: <library-name>
 - Level 2: <data-set-name>
 - ✖ i.e. <library-name>.<data-set-name>
- What does this mean?
 - SAS references folders called libraries when accessing SAS data sets
 - Libraries are simply pointers to folder locations on the disk drive
 - ✖ i.e. 'X:\PStat 130\data1'

Libraries



- SAS has 2 existing libraries:
 - work
 - sasuser
- **work**
 - Is a temporary library
 - Is the default library
- **sasuser**
 - Is a permanent library
- All other libraries must be assigned

Libraries

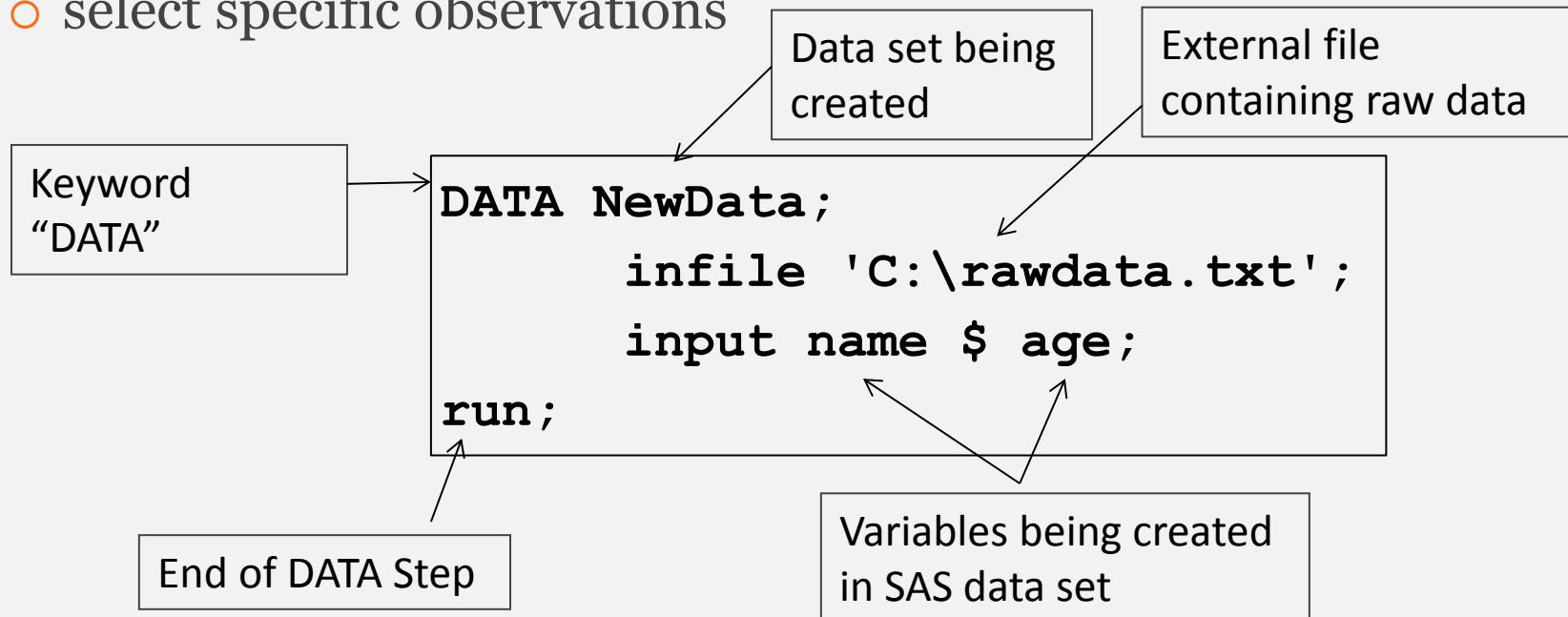


- Library references (libref) must
 - Start with a letter or an underscore
 - Be 8 characters or less
- To define a library, you need
 - The **libname** keyword
 - A user-defined libref name
 - A folder location
- General format:
 - `libname desktop 'C:\desktop';`

The DATA Step



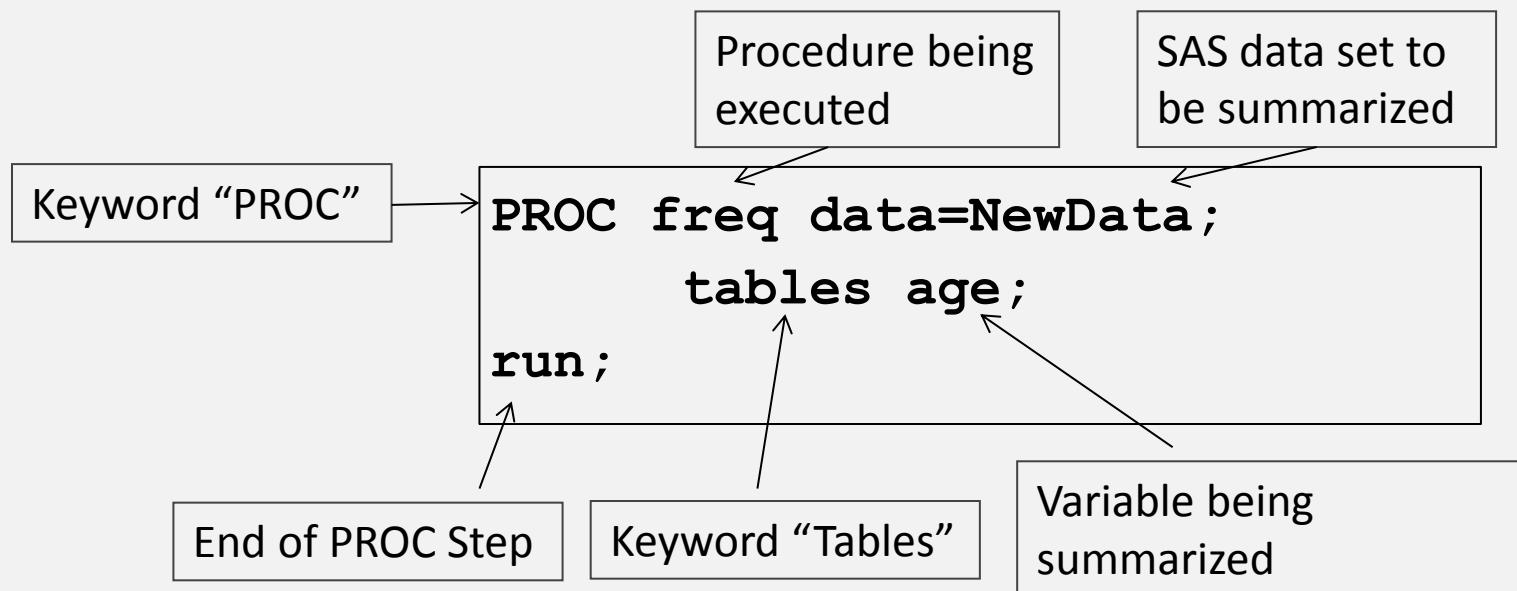
- DATA Step: A set of statements that
 - read in a data file
 - assign variable names, labels, and formats
 - select specific observations



The PROC Step



- PROC Step: A set of statements that
 - perform “utility” operations on a data set
 - analyze data
 - output results or reports



First SAS Program

```
DATA intelligence;  
    input IQ;  
datalines;  
99  
140  
125  
118  
104  
;  
run;  
  
PROC print;  
run;
```

i.e. work.intelligence

Obs	IQ
1	99
2	140
3	125
4	118
5	104

The SAS System 23:08 Tu

Obs	IQ
1	99
2	140
3	125
4	118
5	104

PSTAT 130

SAS Base Programming

Summer 2017

William Qiu

Department of Statistics and Applied Probability

UCSB

Class 2

Objectives

- Create a printed report using PROC PRINT
- Learn PROC PRINT syntax and options
- Define titles and footnotes to enhance reports
- Define descriptive column headings
- Use SAS system options

PROC PRINT

PROC PRINT

The PRINT procedure prints the observations in a SAS data set, using all or some of the variables. You can create a variety of reports ranging from a simple listing to a highly customized report that groups the data and calculates totals and subtotals for numeric variables.

- Simple Version

PROC PRINT will only let you print a single data set at a time

```
proc print;  
run;
```

- With Data File

```
proc print data=ia.empdata;  
run;
```

With Variable Selection: identifies the variables to print. PROC PRINT prints the variables in the order that you list them.

```
proc print data=ia.empdata;  
  var JobCode EmpID Salary;  
run;
```

Proc Print Example

```
Libname ia <insert folder reference>

proc print data=ia.empdata;
  var JobCode EmpID Salary;
run;
```

PROC PRINT Default Output

- Use **variable names as column headings**
- Question: how to change variable names?
- Eg. Month->Mon
- Display **all** variables contained in data set
- Question: how to select variables that we are interested in?
- Display **all** observations contained in data set
- Question: how to select a subset of observations that we want to display?
- Display variable values in their “**native**” format
- Question: how to change formats of variables? Eg. Decimal places

Defining Titles and Footnotes

Control the content, appearance and placement of title and footnote text.

- General form of the TITLE statement

TITLEn 'text'

Syntax: TITLE <n><ods-format-options><'text' | "text">;

n: specifies the relative line that contains the title line.

ods-format-options specifies formatting options for the ODS HTML, RTF, and PRINTER destinations., eg. Color=red, height=, (customize and enhance your output)

'text' | "text" specifies text that is enclosed in single or double quotation marks.

- General form of the FOOTNOTE statement

FOOTNOTEn 'text' ;

- Examples

- title1 'PSTAT 130 Homework #1' ; suppresses a title on line 1 and all lines after it:
- footnote2 'Confidential' ;

Defining Titles and Footnote

Features of titles:

- Titles appear at the top of the page of output
- The default title is “The SAS System”
- You can have more than title line
- The value of n can be from 1 to 10 and refers to title line
- An unnumbered TITLE is equivalent to TITLE1
- Titles remain in effect until they are changed
- The null TITLE statement , title; cancels all titles

Defining Titles and Footnote

Features of footnotes:

- Footnotes appear at the bottom of the page of output
- There is no default footnote (different from title!)
- You can have more than footnote
- The value of n can be from 1 to 10 and refers to footnote line
- An unnumbered FOOTNOTE is equivalent to FOOTNOTE1
- Footnotes remain in effect until they are changed
- The null FOOTNOTE statement, footnote ; cancels all footnote

Changing Titles and Footnotes

TITLEn or FOOTNOTEⁿ

- replaces a previous title or footnote with the same number
- Cancels all titles or footnotes with higher numbers

Defining Titles and Footnotes

PROC PRINT Code	Resultant Title(s)
proc print data=work.march; title1 'The First Line' ; title2 'The Second Line' ; run;	
proc print data=work.april; title2 'The Next Line' ; run;	
proc print data=work.may; title 'The Top Line' ; run;	
proc print data=work.june; title3 'The Third Line' ; run;	
proc print data=work.july; title; run;	Cancel all previous titles or footnotes, that are in effect.

The two TITLE statements below, specified for lines 1 and 3, define titles for the PROC PRINT output.

```
title1 'Heart Rates for Patients with';
title3 'Increased Stress Tolerance Levels';
proc print data=clinic.stress;
  var resthr maxhr rechr;
  where tolerance='I';
run;
```

In HTML output, title lines appear consecutively, without extra spacing to indicate skipped title numbers.

Heart Rates for Patients with Increased Stress Tolerance Levels

Obs	RestHR	MaxHR	RechR
2	68	171	133

In SAS listing output, title line 2 is blank, as shown below. Titles are centered by default.

```
Heart Rates for Patients with
Increased Stress Tolerance Levels

OBS      RestHR      MaxHR      RechR

      2          68          171        133
      3          78          177        139
      8          70          167        122
     11          65          181        141
     14          74          152        113
     15          75          158        108
     20          78          189        138
```

Assigning Column Labels

- Enhance your PROC PRINT report by labeling columns with more descriptive text.
- General form of the LABEL statement

```
LABEL variable= 'label'  
           variable= 'label' ;
```

Note: 1. labels can be up to 256 characters long. Enclose the label in quotation marks.

2. The LABEL statement applies only to the PROC step in which it appears.

- Examples:

```
LABEL Units= 'Number of Units'  
           DOB= 'Date of Birth' ;
```

Assigning Column Labels

libname ia 'X:\PStat 130\data1'; (code may be varied)

EmplID	LastName	FirstName	JobCode	Salary
0031	GOLDENBERG	DESIREE	PILOT	50221.62
0040	WILLIAMS	ARLENE M.	FLTAT	23666.12
0071	PERRY	ROBERT A.	FLTAT	21957.71

ia.empdata

```
proc print data=ia.empdata;
   title1 'Salary Report';
run;
```

Default Variable Names

Obs	Emp ID	LastName	FirstName	Job Code	Salary
1	0031	GOLDENBERG	DESIREE	PILOT	50221.62
2	0040	WILLIAMS	ARLENE M.	FLTAT	23666.12
3	0071	PERRY	ROBERT A.	FLTAT	21957.71
4	0082	MCGWIER-WATTS	CHRISTINA	PILOT	96387.39
5	0091	SCOTT	HARVEY F.	FLTAT	32278.40
6	0106	THACKER	DAVID S.	FLTAT	24161.14
7	0355	BELL	THOMAS B.	PILOT	59803.16
8	0366	GLENN	MARTHA S.	PILOT	120202.38

Assigning Column Labels

EmplID	LastName	FirstName	JobCode	Salary
0031	GOLDENBERG	DESIREE	PILOT	50221.62
0040	WILLIAMS	ARLENE M.	FLTAT	23666.12
0071	PERRY	ROBERT A.	FLTAT	21957.71

ia.empdata

```
proc print data=ia.empdata label;
  label LastName='Last Name'
        FirstName='First Name'
        Salary='Annual Salary';
  title1 'Salary Report';
run;
```

Obs	Emp ID	Last Name	First Name	Job Code	Annual Salary
1	0031	GOLDENBERG	DESIREE	PILOT	50221.62
2	0040	WILLIAMS	ARLENE M.	FLTAT	23666.12
3	0071	PERRY	ROBERT A.	FLTAT	21957.71
4	0082	MCGWIER-WATTS	CHRISTINA	PILOT	96387.39
5	0091	SCOTT	HARVEY F.	FLTAT	32278.40
6	0106	THACKER	DAVID S.	FLTAT	24161.14

This is also a case of using Multiple LABEL statement to assign multiple names. See next page for an example.

You can assign labels in separate LABEL statements ...

```
proc print data=clinic.admit label;  
  var age height;  
  label age='Age of Patient';  
  label height='Height in Inches';  
run;
```

...or you can assign any number of labels in a single LABEL statement.

```
proc print data=clinic.admit label;  
  var actlevel height weight;  
  label actlevel='Activity Level'  
    height='Height in Inches'  
    weight='Weight in Pounds';  
run;
```

Assigning Column Labels

EmplID	LastName	FirstName	JobCode	Salary
0031	GOLDENBERG	DESIREE	PILOT	50221.62
0040	WILLIAMS	ARLENE M.	FLTAT	23666.12
0071	PERRY	ROBERT A.	FLTAT	21957.71

ia.empdata

```
proc print data=ia.empdata split=' ' ;
  label LastName='Last Name'
        FirstName='First Name'
        Salary='Annual Salary';
  title1 'Salary Report';
```

Obs	Emp ID	Last Name	First Name	Job Code	Annual Salary
1	0031	GOLDENBERG	DESIREE	PILOT	50221.62
2	0040	WILLIAMS	ARLENE M.	FLTAT	23666.12
3	0071	PERRY	ROBERT A.	FLTAT	21957.71
4	0082	MCGWIER-WATTS	CHRISTINA	PILOT	96387.39
5	0091	SCOTT	HARVEY F.	FLTAT	32278.40
6	0106	THACKER	DAVID S.	FLTAT	24161.14
7	0355	BELL	THOMAS B.	PILOT	59803.16
8	0366	GLENN	MARTHA S.	PILOT	120202.38

Use the option
Split instead of
Label. Almost the
same.

Split option control
where text strings
split in labels.

Using SAS Options

- SAS system options are initialized with default settings when SAS is invoked. However, the default settings for some SAS system options vary both by operating environment and by site. Your on-site SAS support personnel might have created a default options table. The default options table is created by your site administrator and provides a global set of default values that are specific for your site. It reduces the need to duplicate options in every system configuration file at your site.
- General form of the OPTIONS statement:

```
OPTIONS option...;
```

- Run the following program to generate a list of all options (the list will appear in the LOG window):

```
PROC OPTIONS;  
RUN;
```

Using SAS System Options

Selected SAS system options:

- DATE vs. NODATE
 - Determines whether SAS displays the system date on output
- LINESIZE=width (LS=width)
 - Determines size of output line before line break
- PAGESIZE=n (PS=n)
 - Determines number of lines on page before page break
- NUMBER vs. NONUMBER
 - Determines whether SAS displays page numbers
- PAGENO=n
 - Determines starting page number
- Example: (we are going to see more in homework 1)
Options nodate nonumber ls=72;

Formatting Data Values

In your SAS reports, formats control how the data values are displayed. To make data values more understandable when they are displayed in your procedure output, you can use the **FORMAT** statement, which associates formats with variables.

Formats affects only how the data values appear in output, not the actual data values as they are stored in the SAS data set.

Objectives

- Display formatted values using SAS formats in a list report
- Create user-defined formats using the FORMAT procedure
- Apply user-defined formats to variables in a list report

Using SAS Formats

an example for preview

Enhance the readability of reports by formatting the data values

Salary Report					
Obs	Emp ID	Last Name	First Name	Job Code	Annual Salary
1	0031	GOLDENBERG	DESIREE	PILOT	\$50,221.62
2	0040	WILLIAMS	ARLENE M.	FLTAT	\$23,666.12
3	0071	PERRY	ROBERT A.	FLTAT	\$21,957.71
4	0082	MCGWIER-WATTS	CHRISTINA	PILOT	\$96,387.39
5	0091	SCOTT	HARVEY F.	FLTAT	\$32,278.40
6	0106	THACKER	DAVID S.	FLTAT	\$24,161.14
7	0355	BELL	THOMAS B.	PILOT	\$59,803.16
8	0366	GLENN	MARTHA S.	PILOT	\$120,202.38

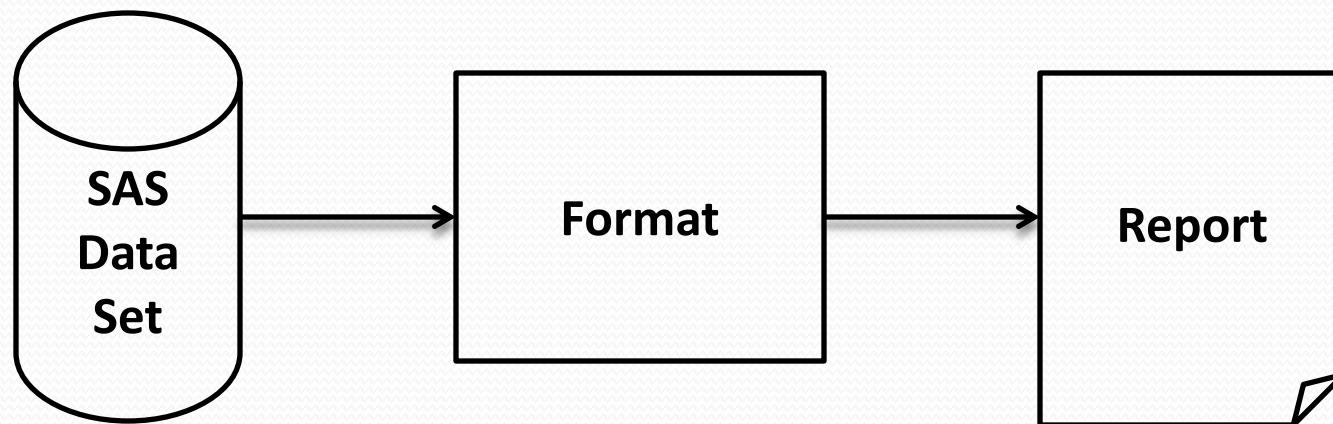
Using SAS Formats

Create custom formats to recode data values in a report

Salary Report in Categories					
Emp ID	Last Name	First Name	JobCode	Annual Salary	
0031	GOLDENBERG	DESIREE	Pilot	More than	50,000
0040	WILLIAMS	ARLENE M.	Flight Attendant	Less than	25,000
0071	PERRY	ROBERT A.	Flight Attendant	Less than	25,000
0082	MCGWIER-WATTS	CHRISTINA	Pilot	More than	50,000
0091	SCOTT	HARVEY F.	Flight Attendant	25,000 to	50,000
0106	THACKER	DAVID S.	Flight Attendant	Less than	25,000
0355	BELL	THOMAS B.	Pilot	More than	50,000
0366	GLENN	MARTHA S.	Pilot	More than	50,000

For a list of formats, see Formats, By Category in SAS Help

Formatting Data Values



Formatting Data Values

- General form of the FORMAT statement

FORMAT variable(s) format;

Variable(s) is the name of one or more variables whose values are to be written according to a particular pattern.

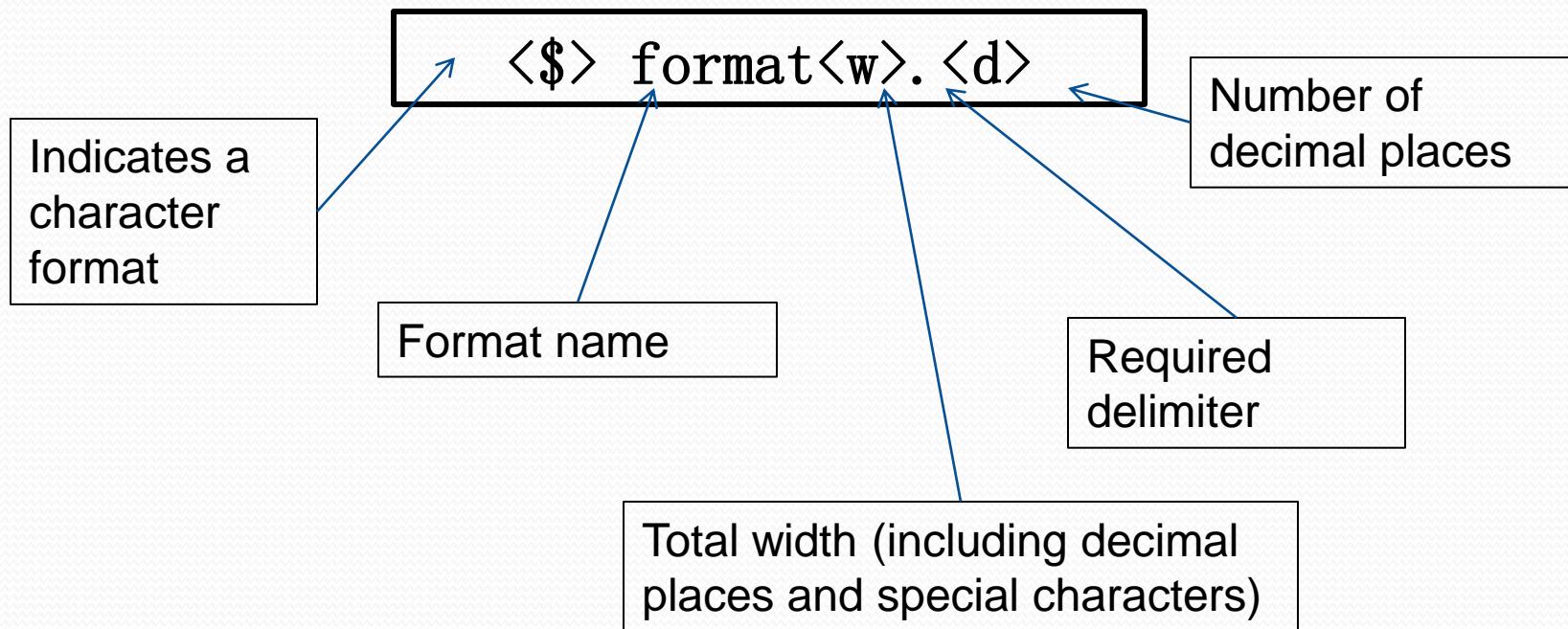
Format-name specifies a SAS format or a user-defined format that is used to write out the values.

Note: the FORMAT statement applies only to the PROC step in which it appears.

- Example:

```
proc print data=ia.empdata split=' ' ;
    format Salary dollar11.2;
run;
```

What is a SAS Format?



SAS Formats

- Selected SAS formats

Format/ Example	Result	Description
w.d 8 . 2	12234 . 21	standard numeric format Width=8, 2 decimal places
\$w. \$5 .	KATHY	standard character format width=5
COMMAw.d COMMA9 . 2	12 , 234 . 21	Commas in a number Width=9, 2 decimal places
DOLLARw.d DOLLAR10 . 2	\$12 , 234 . 21	Dollar signs and commas in a number Width=10, 2 decimal places

SAS Formats

Stored Value	Format	Displayed Value
27134.2864	COMMA12. 2	27, 134. 29
27134.2864	12. 2	27134. 29
27134.2864	DOLLAR12. 2	\$27, 134. 29
27134.2864	DOLLAR9. 2	\$27134. 29
27134.2864	DOLLAR8. 2	27134. 29
27134.2864	DOLLAR5. 2	27134
27134.2864	DOLLAR4. 2	27E3

Formatting Data Values

EmplID	LastName	FirstName	JobCode	Salary
0031	GOLDENBERG	DESIREE	PILOT	50221.62
0040	WILLIAMS	ARLENE M.	FLTAT	23666.12
0071	PERRY	ROBERT A.	FLTAT	21957.71

ia.empdata

```
proc print data=ia.empdata split=' ' ;
  label LastName='Last Name'
        FirstName='First Name'
        Salary='Annual Salary';
  format Salary dollar11.2;
  title1 'Salary Report';
run;
```

Note the difference between the original format and the new format in the output!

Obs	Emp ID	Last Name	First Name	Job Code	Annual Salary
1	0031	GOLDENBERG	DESIREE	PILOT	\$50,221.62
2	0040	WILLIAMS	ARLENE M.	FLTAT	\$23,666.12
3	0071	PERRY	ROBERT A.	FLTAT	\$21,957.71
4	0082	MCGWIERN-WATTS	CHRISTINA	PILOT	\$96,387.39
5	0091	SCOTT	HARVEY F.	FLTAT	\$32,278.40
6	0106	THACKER	DAVID S.	FLTAT	\$24,161.14
7	0355	BELL	THOMAS B.	PILOT	\$59,803.16
8	0366	GLENN	MARTHA S.	PILOT	\$120,202.38

Date Formats

MMDDYYw.

Format	Displayed Value
MMDDYY6.	101601
MMDDYY8.	10/16/01
MMDDYY10.	10/16/2001

DATEw.

Format	Displayed Value
DATE7.	16OCT01
DATE9.	16OCT2001

DATE Formats and Serial Date Values

Stored Value	Format	Displayed Value
0	MMDDYY8.	01/01/60
0	MMDDYY10.	01/01/1960
1	DATE9.	01JAN1960
-1	WORDDATE.	December 31, 1959
365	DDMMYY10.	31/12/1960
366	WEEKDATE.	Sunday, January 1, 1961

Creating User-defined Formats

To create and use your own formats,

1. Use the FORMAT procedure to create the format
2. Apply the format to specific variables by using a FORMAT statement

Create User-defined Formats

We now learned to associate formats with variables. But sometimes we might want to create custom formats for displaying variable values. For example, we can format a product number so that it is displayed as descriptive text.

With the FORMAT procedure, we can define our own formats for variables. We can store our formats temporarily or permanently, and we can display a list of all our formats and descriptions of their values.

General form of a PROC FORMAT step:

```
PROC FORMAT;  
    VALUE format-name  
        range1= 'label'  
        range2= 'label'  
        . . . ;  
RUN;
```

Creating User-defined Formats

Format-name

- Names the format you are creating (i.e., DATE7.)
- Cannot be more than 8 characters
- For character values, must have a dollar \$ as the first character , a letter or underscore as the second character, and no more than 6 additional characters, number and underscores
- For numeric values, must have a letter or underscore as the first character, and no more than 7 additional characters, number and underscores
- Cannot end in a number
- Cannot be the same as a SAS format
- Does not end with a period in the VALUE statement

Creating User-define Formats

Labels

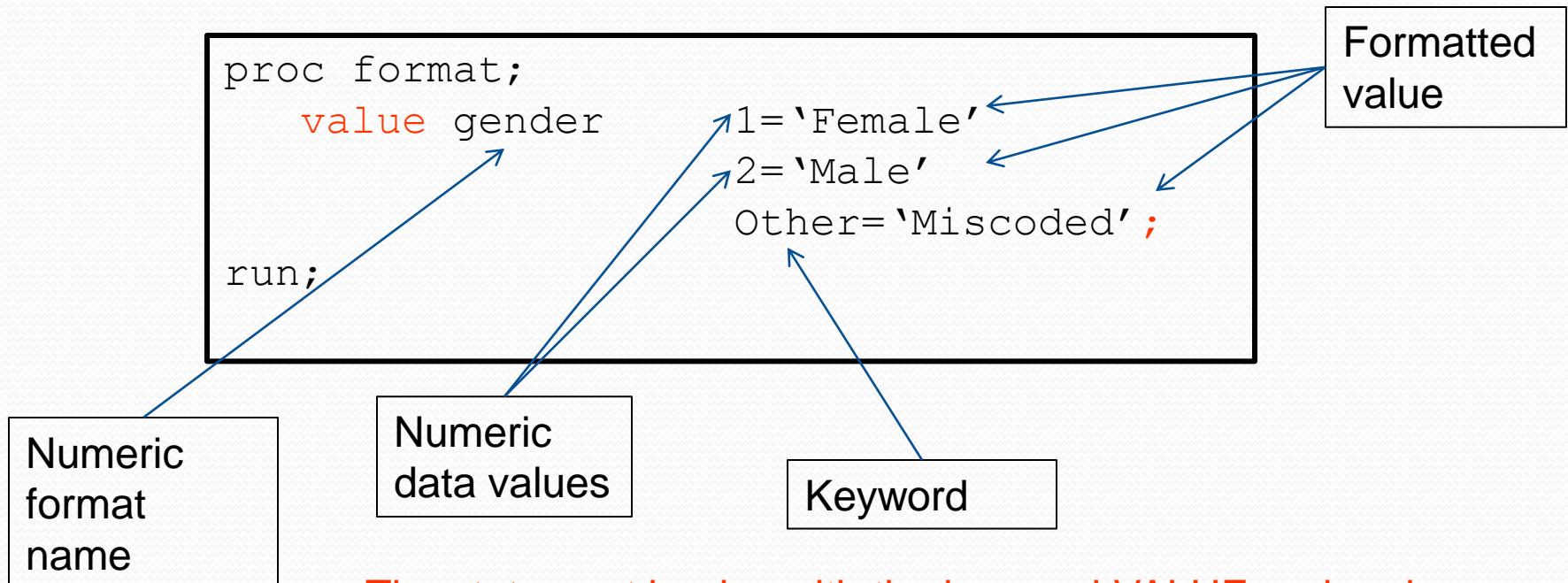
- Can be up to 32.767 characters in length
- Are typically text string enclosed in quotation marks, although it is not required

Range(s): specifies one or more variable values and a character string or an existing format.

- *Can be single values or eg. 24, 'S'...*
- *ranges of values, eg. 0-1500 (numeric), 'A'-'M' (characer)*
- *More examples next page*

Creating User-defined Formats

Assigning labels to **single** numbers



The statement begins with the keyword VALUE and ends with a semicolon after all the labels have been assigned.

Creating User-defined Formats

Assigning labels to a **range of numbers**

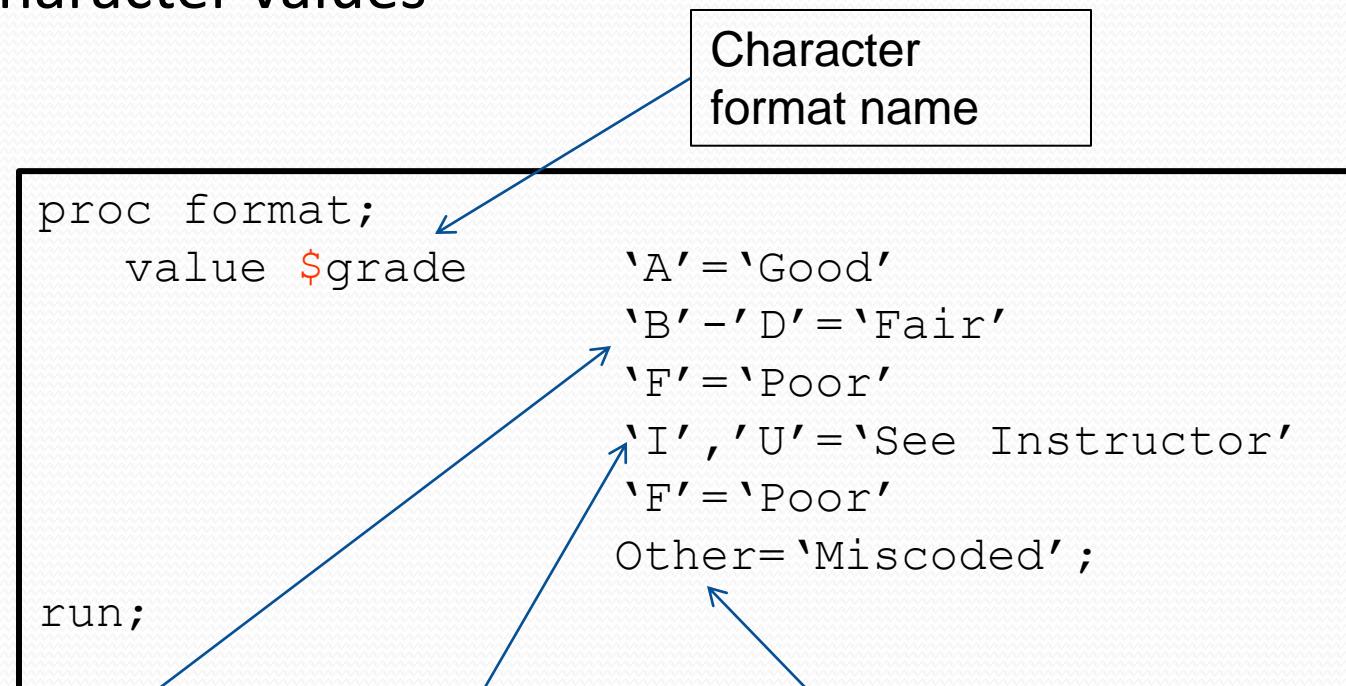
```
proc format;
  value boardfmt
    low-49='Below Average'
    50-99='Average'
    100-high='Above Average';
run;
```

Numeric
data values

Keywords

Creating User-defined Formats

Assigning labels to character and ranges of character values



Character
value range

Discrete
character values

Keyword

Creating User-defined Formats

Step 1. Create the format

```
proc format;
    value $codefmt  'FLTAT'='Flight Attendant'
                    'PILOT'='Pilot';
run;
```

Step 2. Apply the format

```
proc print data=ia.empdata;
    format Jobcode $codefmt.;
run;
```

Creating User-defined Formats

Step 1. Create the format

```
proc format;
    value money      low-<25000='Less than 25,000'
                    25000-50000='25,000 to 50,000'
                    50000-high='More than 50,000';
run;
```

Step 2. Apply the format

```
proc print data=ia.empdata;
    format Salary money.;
run;
```

Creating User-defined Formats

Define multiple formats within the same FORMAT procedure

```
proc format;
    value $codefmt 'FLTAT'='Flight Attendant'
                    'PILOT'='Pilot';
    value money     low-<25000='Less than 25,000'
                    25000-50000='25,000 to 50,000'
                    50000-high='More than 50,000';
run;
```

Applying User-defined Formats

```
proc print data=ia.empdata split=' ' noobs;
  label LastName='Last Name'
        FirstName='First Name'
        Salary='Annual Salary';
  format Salary money. Jobcode $codefmt.;
  title1 'Salary Report in Categories';
run;
```

Salary Report in Categories				
Emp ID	Last Name	First Name	JobCode	Annual Salary
0031	GOLDENBERG	DESIREE	Pilot	More than 50,000
0040	WILLIAMS	ARLENE M.	Flight Attendant	Less than 25,000
0071	PERRY	ROBERT A.	Flight Attendant	Less than 25,000
0082	MCGWIER-WATTS	CHRISTINA	Pilot	More than 50,000
0091	SCOTT	HARVEY F.	Flight Attendant	25,000 to 50,000
0106	THACKER	DAVID S.	Flight Attendant	Less than 25,000
0355	BELL	THOMAS B.	Pilot	More than 50,000
0366	GLENN	MARTHA S.	Pilot	More than 50,000

PSTAT 130

SAS Base Programming

Summer 2017

William Qiu

Department of Statistics and Applied Probability

UCSB

Class 3

Parking Lot Issues

- Homework is due in **your registered section**
- Data file location
- Default SAS directory
- List of formats – SAS Help
- SAS vs. User-defined formats

Objectives

- Generate simple list reports using the PRINT procedure.
- Display Formatted Data Values
- Display selected variables (columns) in a list report.
- Display selected observations (rows) in a list report.
- Display a list report with column totals.

PROC PRINT Default Output

- Use variable names as column headings
- Display all variables contained in data set
- Display all observations contained in data set
- Display variable values in their “native” format

PROC PRINT – Key Options

With PROC PRINT options, you can display

- titles and footnotes
- descriptive column headings
- formatted data values
- specific variables and observations
- column totals and column subtotals
- page breaks for each subgroup

PROC PRINT Syntax

The screenshot shows the SAS Help and Documentation interface. The title bar reads "SAS Help and Documentation". The menu bar includes File, Edit, View, Go, and Help. The toolbar contains icons for Hide, Locate, Back, Forward, Stop, Refresh, Print, and Options. Below the menu is a navigation bar with Contents, Index, Search, and Favorites. A search bar says "Type in the keyword to find:" followed by "proc print". To the right, the main content area has a title "Producing Detail Reports with the PRINT Procedure" and a section titled "Review of SAS Tools". Under "Review of SAS Tools", there is a section titled "PROC PRINT Statements". The text lists the following statements:

- PROC PRINT** <DATA=SAS-data-set> <option(s)>;
- BY variable(s);
- FOOTNOTE<n> <'footnote'>;
- FORMAT variable(s) format-name;
- ID variable(s);
- LABEL variable='label';
- PAGEBY variable;
- SUM variable(s);
- SUMBY variable;
- TITLE<n> <'title'>;
- VAR variable(s);
- WHERE where-expression;

Defining Titles and Footnotes

- General form of the TITLE statement

TITLEn 'text'

- General form of the FOOTNOTE statement

FOOTNOTEn 'text';

- Examples

- title1 'PSTAT 130 Homework #1';
- footnote2 'Confidential';

Defining Titles and Footnote

Features of titles:

- Titles appear at the top of the page of output
- The default title is “The SAS System”
- You can have more than title line
- The value of n can be from 1 to 10 and refers to title line
- An unnumbered TITLE is equivalent to TITLE1
- Titles remain in effect until they are changed
- The null TITLE statement , title; cancels all titles

Defining Titles and Footnote

Features of footnotes:

- Footnotes appear at the bottom of the page of output
- There is no default footnote
- You can have more than footnote
- The value of n can be from 1 to 10 and refers to footnote line
- An unnumbered FOOTNOTE is equivalent to FOOTNOTE1
- Footnotes remain in effect until they are changed
- The null FOOTNOTE statement, footnote; cancels all footnote

Changing Titles and Footnotes

TITLEn or FOOTNOTEⁿ

- replaces a previous title or footnote with the same number
- Cancels all titles or footnotes with higher numbers

Defining Titles and Footnotes

PROC PRINT Code	Resultant Title(s)
<pre>proc print data=work.march; title1 'The First Line'; title2 'The Second Line'; run;</pre>	
<pre>proc print data=work.april; title2 'The Next Line'; run;</pre>	
<pre>proc print data=work.may; title 'The Top Line'; run;</pre>	
<pre>proc print data=work.june; title3 'The Third Line'; run;</pre>	
<pre>proc print data=work.july; title; run;</pre>	

Assigning Column Labels

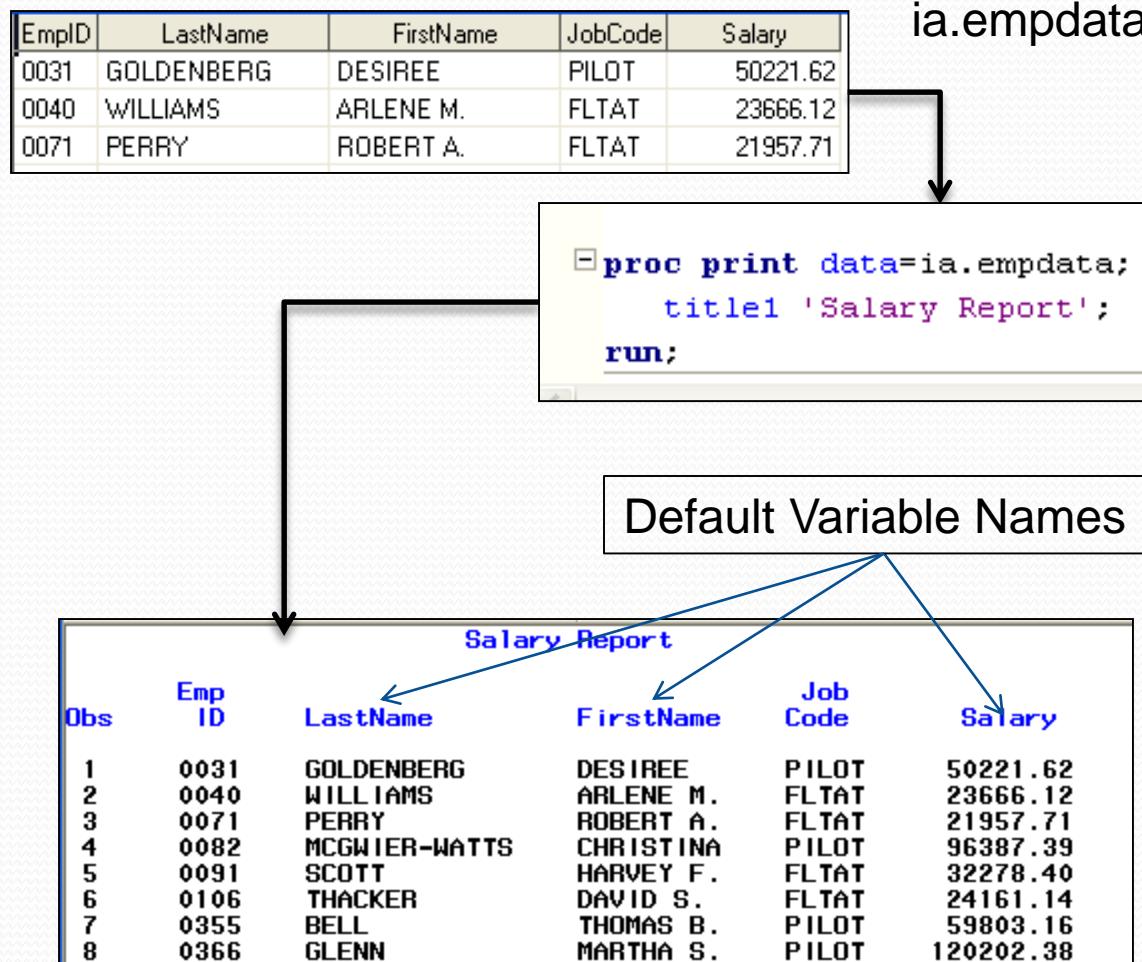
- General form of the LABEL statement

```
LABEL variable='label'  
        variable='label';
```

- Examples:

```
LABEL Units='Number of Units'  
        DOB='Date of Birth';
```

Assigning Column Labels



Assigning Column Labels

EmplID	LastName	FirstName	JobCode	Salary
0031	GOLDENBERG	DESIREE	PILOT	50221.62
0040	WILLIAMS	ARLENE M.	FLTAT	23666.12
0071	PERRY	ROBERT A.	FLTAT	21957.71

ia.empdata

```
proc print data=ia.empdata label;
  label LastName='Last Name'
        FirstName='First Name'
        Salary='Annual Salary';
  title1 'Salary Report';
run;
```

Obs	Emp ID	Last Name	First Name	Job Code	Annual Salary
1	0031	GOLDENBERG	DESIREE	PILOT	50221.62
2	0040	WILLIAMS	ARLENE M.	FLTAT	23666.12
3	0071	PERRY	ROBERT A.	FLTAT	21957.71
4	0082	MCGWIER-WATTS	CHRISTINA	PILOT	96387.39
5	0091	SCOTT	HARVEY F.	FLTAT	32278.40
6	0106	THACKER	DAVID S.	FLTAT	24161.14

Assigning Column Labels

EmplID	LastName	FirstName	JobCode	Salary
0031	GOLDENBERG	DESIREE	PILOT	50221.62
0040	WILLIAMS	ARLENE M.	FLTAT	23666.12
0071	PERRY	ROBERT A.	FLTAT	21957.71

ia.empdata

```
proc print data=ia.empdata split=' '|;
  label LastName='Last Name'
        FirstName='First Name'
        Salary='Annual Salary';
  title1 'Salary Report';
```

Obs	Emp ID	Last Name	First Name	Job Code	Annual Salary
1	0031	GOLDENBERG	DESIREE	PILOT	50221.62
2	0040	WILLIAMS	ARLENE M.	FLTAT	23666.12
3	0071	PERRY	ROBERT A.	FLTAT	21957.71
4	0082	MCGWIER-WATTS	CHRISTINA	PILOT	96387.39
5	0091	SCOTT	HARVEY F.	FLTAT	32278.40
6	0106	THACKER	DAVID S.	FLTAT	24161.14
7	0355	BELL	THOMAS B.	PILOT	59803.16
8	0366	GLENN	MARTHA S.	PILOT	120202.38

Using SAS Options

- General form of the OPTIONS statement:

```
OPTIONS option...;
```

- Run the following program to generate a list of all options
(the list will appear in the LOG window):

```
PROC OPTIONS;
```

```
RUN;
```

Using SAS System Options

Selected SAS system options:

- DATE vs. NODATE
 - Determines whether SAS displays the system date on output
- LINESIZE=width (LS=width)
 - Determines size of output line before line break
- PAGESIZE=n (PS=n)
 - Determines number of lines on page before page break
- NUMBER vs. NONUMBER
 - Determines whether SAS displays page numbers
- PAGENO=n
 - Determines starting page number
- Example:
Options nodate nonumber ls=72;

Formatting Data Values

Objectives

- Display formatted values using SAS formats in a list report
- Create user-defined formats using the FORMAT procedure
- Apply user-defined formats to variables in a list report

Using SAS Formats

Enhance the readability of reports by formatting the data values

Salary Report					
Obs	Emp ID	Last Name	First Name	Job Code	Annual Salary
1	0031	GOLDENBERG	DESIREE	PILOT	\$50,221.62
2	0040	WILLIAMS	ARLENE M.	FLTAT	\$23,666.12
3	0071	PERRY	ROBERT A.	FLTAT	\$21,957.71
4	0082	MCGWIER-WATTS	CHRISTINA	PILOT	\$96,387.39
5	0091	SCOTT	HARVEY F.	FLTAT	\$32,278.40
6	0106	THACKER	DAVID S.	FLTAT	\$24,161.14
7	0355	BELL	THOMAS B.	PILOT	\$59,803.16
8	0366	GLENN	MARTHA S.	PILOT	\$120,202.38

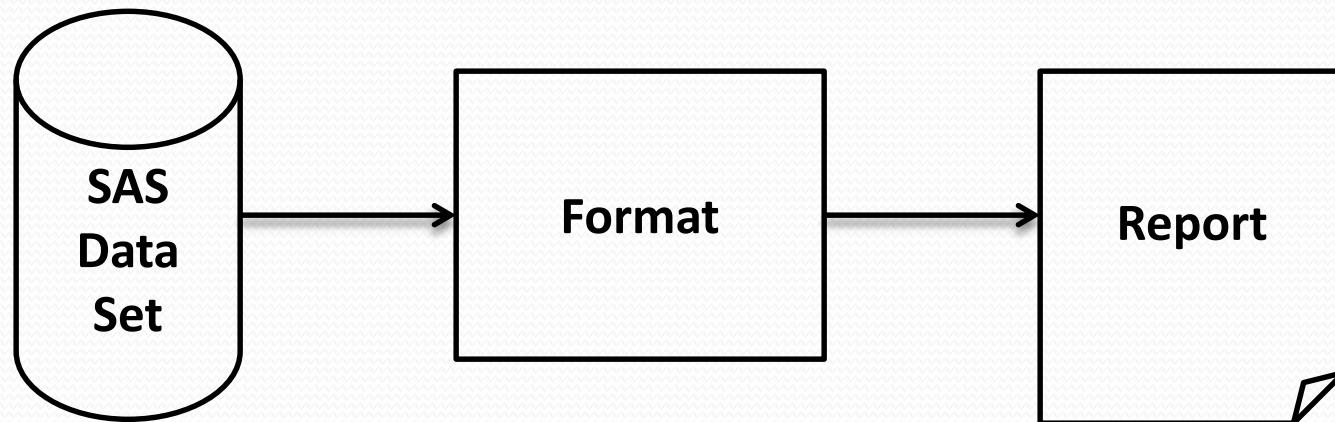
Using SAS Formats

Create custom formats to recode data values in a report

Salary Report in Categories					
Emp ID	Last Name	First Name	JobCode	Annual Salary	
0031	GOLDENBERG	DESIREE	Pilot	More than	50,000
0040	WILLIAMS	ARLENE M.	Flight Attendant	Less than	25,000
0071	PERRY	ROBERT A.	Flight Attendant	Less than	25,000
0082	MCGWIER-WATTS	CHRISTINA	Pilot	More than	50,000
0091	SCOTT	HARVEY F.	Flight Attendant	25,000 to	50,000
0106	THACKER	DAVID S.	Flight Attendant	Less than	25,000
0355	BELL	THOMAS B.	Pilot	More than	50,000
0366	GLENN	MARTHA S.	Pilot	More than	50,000

For a list of formats, see Formats, By Category in SAS Help

Formatting Data Values



Formatting Data Values

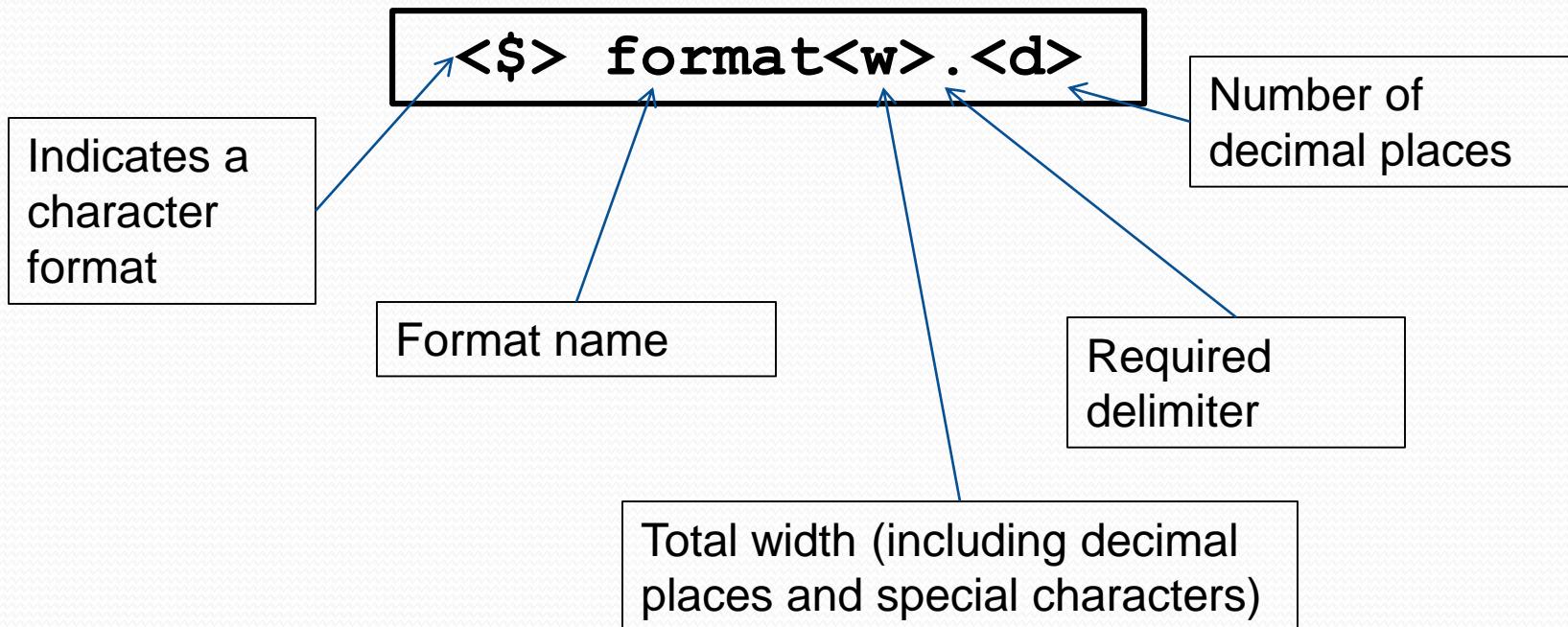
- General form of the FORMAT statement

FORMAT variable(s) format;

- Example:

```
proc print data=ia.empdata split=' ';
    format Salary dollar11.2;
run;
```

What is a SAS Format?



SAS Formats

- Selected SAS formats

Format/ Example	Result	Description
w.d 8 . 2	12234 . 21	standard numeric format Width=8, 2 decimal places
\$w. \$5 .	KATHY	standard character format width=5
COMMAw.d COMMA9 . 2	12 , 234 . 21	Commas in a number Width=9, 2 decimal places
DOLLARw.d DOLLAR10 . 2	\$12 , 234 . 21	Dollar signs and commas in a number Width=10, 2 decimal places

SAS Formats

Stored Value	Format	Displayed Value
27134.2864	COMMA12.2	27,134.29
27134.2864	12.2	27134.29
27134.2864	DOLLAR12.2	\$27,134.29
27134.2864	DOLLAR9.2	\$27134.29
27134.2864	DOLLAR8.2	27134.29
27134.2864	DOLLAR5.2	27134
27134.2864	DOLLAR4.2	27E3

Formatting Data Values

EmplID	LastName	FirstName	JobCode	Salary
0031	GOLDENBERG	DESIREE	PILOT	50221.62
0040	WILLIAMS	ARLENE M.	FLTAT	23666.12
0071	PERRY	ROBERT A.	FLTAT	21957.71

ia.empdata

```
proc print data=ia.empdata split=' '|;
  label LastName='Last Name'
        FirstName='First Name'
        Salary='Annual Salary';
  format Salary dollar11.2;
  title1 'Salary Report';
run;
```

Obs	Emp ID	Last Name	First Name	Job Code	Annual Salary
1	0031	GOLDENBERG	DESIREE	PILOT	\$50,221.62
2	0040	WILLIAMS	ARLENE M.	FLTAT	\$23,666.12
3	0071	PERRY	ROBERT A.	FLTAT	\$21,957.71
4	0082	MCGWIER-WATTS	CHRISTINA	PILOT	\$96,387.39
5	0091	SCOTT	HARVEY F.	FLTAT	\$32,278.40
6	0106	THACKER	DAVID S.	FLTAT	\$24,161.14
7	0355	BELL	THOMAS B.	PILOT	\$59,803.16
8	0366	GLENN	MARTHA S.	PILOT	\$120,202.38

Date Formats

MMDDYYw.

Format	Displayed Value
MMDDYY6.	101601
MMDDYY8.	10/16/01
MMDDYY10.	10/16/2001

DATEw.

Format	Displayed Value
DATE7.	16OCT01
DATE9.	16OCT2001

DATE Formats and Serial Date Values

Stored Value	Format	Displayed Value
0	MMDDYY8.	01/01/60
0	MMDDYY10.	01/01/1960
1	DATE9.	02JAN1960
-1	WORDDATE.	December 31, 1959
365	DDMMYY10.	31/12/1960
366	WEEKDATE.	Sunday, January 1, 1961

Creating User-defined Formats

To create and use your own formats,

1. Use the FORMAT procedure to create the format
2. Apply the format to specific variables by using a FORMAT statement

Create User-defined Formats

General form of a PROC FORMAT step:

```
PROC FORMAT;  
  VALUE format-name  
    range1='label'  
    range2='label'  
    ...;  
RUN;
```

Creating User-defined Formats

Format-name

- Names the format you are creating (i.e., DATE7.)
- Cannot be more than 8 characters
- For character values, must have a dollar \$ as the first character , a letter or underscore as the second character, and no more than 6 additional characters, number and underscores
- For numeric values, must have a letter or underscore as the first character, and no more than 7 additional characters, number and underscores
- Cannot end in a number
- Cannot be the same as a SAS format
- Does not end with a period in the VALUE statement

Creating User-define Formats

Labels

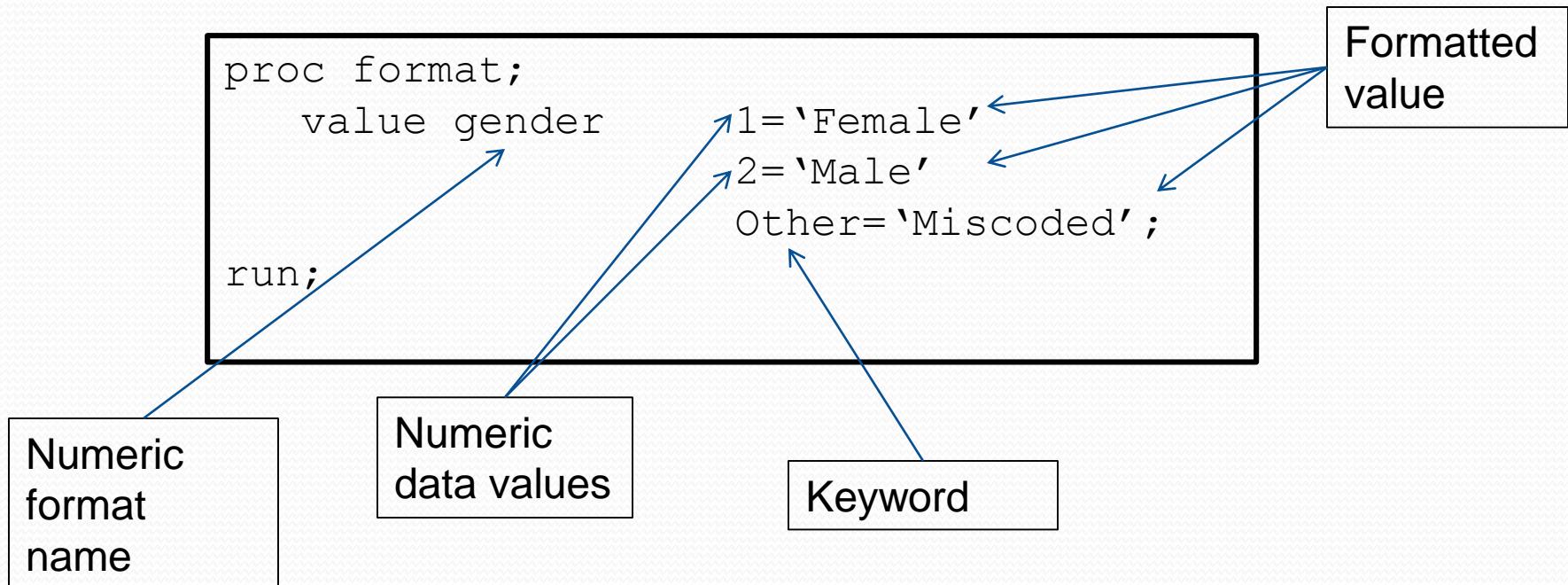
- Can be up to 32,767 characters in length
- Are typically enclosed in quotes, although it is not required

Range(s)

- *Can be single values or*
- *ranges of values*

Creating User-defined Formats

Assigning labels to single numbers



Creating User-defined Formats

Assigning labels to a range of numbers

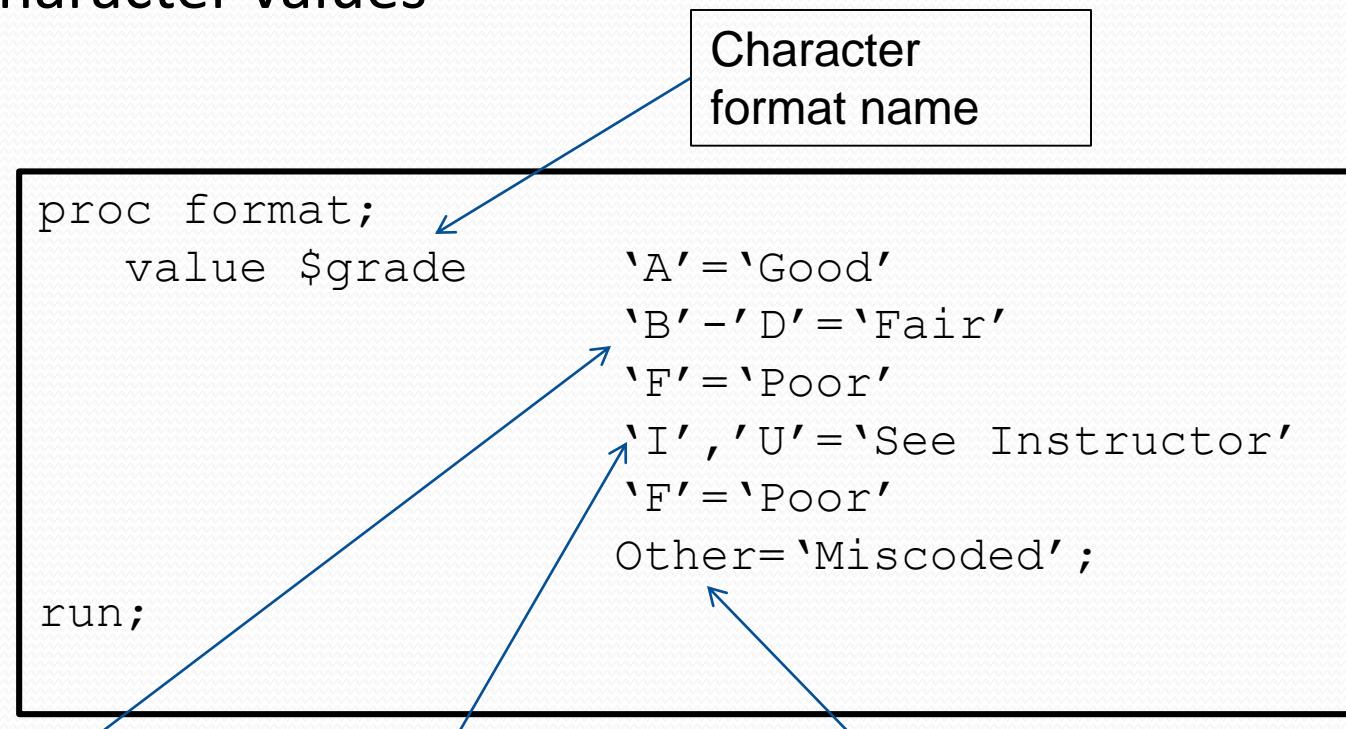
```
proc format;
  value boardfmt
    low-49='Below Average'
    50-99='Average'
    100-high='Above Average';
run;
```

Numeric
data values

Keywords

Creating User-defined Formats

Assigning labels to character and ranges of character values



Character
value range

Discrete
character values

Keyword

Creating User-defined Formats

Step 1. Create the format

```
proc format;
    value $codefmt "FLTAT"="Flight Attendant"
                  "PILOT"="Pilot";
run;
```

Step 2. Apply the format

```
proc print data=ia.empdata;
    format Jobcode $codefmt.;
run;
```

Creating User-defined Formats

Step 1. Create the format

```
proc format;
    value money      low-25000='Less than 25,000'
                    25000-50000='25,000 to 50,000'
                    50000-high='More than 50,000';
run;
```

Step 2. Apply the format

```
proc print data=ia.empdata;
    format Salary money.;
run;
```

Creating User-defined Formats

Define multiple formats within the same FORMAT procedure

```
proc format;
    value $codefmt 'FLTAT'='Flight Attendant'
                    'PILOT'='Pilot';
    value money     low-25000='Less than 25,000'
                    25000-50000='25,000 to 50,000'
                    50000-high='More than 50,000';
run;
```

Applying User-defined Formats

```
proc print data=ia.empdata split=' ' noobs;
  label LastName='Last Name'
        FirstName='First Name'
        Salary='Annual Salary';
  format Salary money. Jobcode $codefmt.;
  title1 'Salary Report in Categories';
run;
```

Salary Report in Categories				
Emp ID	Last Name	First Name	JobCode	Annual Salary
0031	GOLDENBERG	DESIREE	Pilot	More than 50,000
0040	WILLIAMS	ARLENE M.	Flight Attendant	Less than 25,000
0071	PERRY	ROBERT A.	Flight Attendant	Less than 25,000
0082	MCGWIER-WATTS	CHRISTINA	Pilot	More than 50,000
0091	SCOTT	HARVEY F.	Flight Attendant	25,000 to 50,000
0106	THACKER	DAVID S.	Flight Attendant	Less than 25,000
0355	BELL	THOMAS B.	Pilot	More than 50,000
0366	GLENN	MARTHA S.	Pilot	More than 50,000

Printing Selected Variables

- By default, PROC PRINT prints all variables in the data set

The **VAR** statement enables you to

- select variables to include in the report
- define the order of the variables in the report.

General form of the VAR statement:

```
VAR variable(s);
```

Select and order variables to print:

```
proc print data=ia.empdata;
  var JobCode EmpID Salary;
run;
```

Suppressing the Obs Column

The NOOBS option suppresses the row numbers on the left side of the report.

General form of the NOOBS option:

```
PROC PRINT DATA=SAS-data-set NOOBS;  
RUN;
```

Suppress the Obs column:

```
proc print data=ia.empdata noobs;  
  var JobCode EmpID Salary;  
run;
```

Subsetting Data: WHERE Statement

Produce a listing report that displays information for pilots only.

The WHERE statement

- enables you to **select observations** that meet a certain condition
- can be used with most SAS procedures.

Subsetting Data: WHERE Statement

General form of the WHERE statement:

```
WHERE where-expression;
```

where-expression is a sequence of **operands** and **operators**.

Operands include

- variables or constants

Operators include

- comparison operators
- logical operators
- special operators
- functions

Printing Selected Observations

Use the WHERE statement to control which observations are processed.

```
proc print data=ia.empdata noobs;
  var JobCode EmpID Salary;
  where JobCode='PILOT';
run;
```

WHERE Subsetting Example

- Print only employees who are Flight Attendants

```
proc print data=ia.empdata;
  var JobCode EmpID Salary;
  WHERE JobCode = 'FLTAT';
run;
```

- The WHERE statement selects employees whose JobCode is “FLTAT”

Comparison Operators

Mnemonic Equivalent	Symbol	Definition	Example
EQ	=	equal to	where empnum eq 3374;
NE	^=	not equal to	where status ne fulltime;
	~=		
	<>		
GT	>	greater than	where hiredate gt '01jun1982'd;
LT	<	less than	where empnum < 2000;
GE	>=	greater than or equal to	where empnum >= 3374;
LE	<=	less than or equal to	where empnum <= 3374;
IN		equal to one from a list of values	where state in ('NC','TX');

- Character comparisons are case-sensitive.
- The IN operator allows commas or spaces to separate arguments.

WHERE Examples

- Print only people under 40

- **Where Age < 40 ;**
 - **Where Age LT 40 ;**
 - **Where Age <= 39 ;**
 - **Where Age LE 39 ;**

- Print only adults

- **Where Age >= 18 ;**
 - **Where Age GE 18 ;**
 - **Where Age > 17 ;**
 - **Where Age GT 17 ;**

- Print only women

- **Where sex = 'F' ;**
 - **Where sex EQ 'F' ;**
 - **Where sex <> 'M' ;**
 - **Where sex NE 'M' ;**

Logical Operators

- AND (&)
 - if both expressions are true, then the compound expression is true

```
where JobCode='FLTAT' and Salary>50000;
```

```
where JobCode='FLTAT' & Salary>50000;
```

- OR (|)
 - if either expression is true, then the compound expression is true

```
where JobCode='PILOT' or JobCode='FLTAT' ;
```

```
where JobCode='PILOT' | JobCode='FLTAT' ;
```

- NOT
 - can be combined with other operators to reverse the logic of a comparison.

```
where not JobCode in('PILOT','FLTAT') ;
```

Special Operators

Additional special operators supported by the WHERE statement are

- BETWEEN-AND
- CONTAINS (?)
- LIKE
- sounds like
- IS MISSING (or IS NULL)

Special Operators

Special operators include

- BETWEEN-AND
 - selects observations in which the value of the variable falls within a range of values, inclusively.

`where Salary between 50000 and 70000;`

Same as

`where 50000 <= Salary <= 70000;`

- CONTAINS (?)
 - selects observations that include the specified substring.

`where LastName ? 'LAM' ;`

(LAMBERT, BELLAMY, and ELAM are selected.)

Contrast In and ?

Special Operators

The following are special operators :

- **LIKE** selects observations by comparing character values to specified **patterns**.
- LIKE uses two “wildcard” symbols:
 - A percent sign (%) replaces **any number** of characters.
- An underscore (_) replaces **one** character.

```
where Code like 'E_U%';
```

- Selects observations where the value of Code begins with an E, followed by a **single character**, followed by a U, followed by **any number** of characters.

Special Operators

- The sounds like (=*) operator selects observations that contain **spelling variations** of the word or words specified.

```
where Name=*&'SMITH';
```

Selects names like **SMYTHE** or **SMITT**

- IS NULL** or **IS MISSING** selects observations in which the value of the variable is missing.

```
where Flight is missing;  
where Flight is null;
```

- You can use the NOT logical operator to select non-missing values.

Adjusting Column Width

The **WIDTH** option controls the column width of each variable on the PRINT report

General form of the WIDTH Option:

```
PROC PRINT DATA=SAS-data-set WIDTH=options;  
RUN;
```

Example:

```
proc print data=ia.empdata width=uniform;  
run;
```

Adjusting Column Width

By default, PROC PRINT uses the **formatted column width** for each variable. To set the column width to the same size for each variable, use **WIDTH=UNIFORM**.

Note: Columns will not display smaller than the width of their column headings.

Requesting Column Totals

The SUM statement produces **column totals** for numeric variables.

General form of the SUM statement:

```
SUM variable(s);
```

Example:

```
proc print data=ia.empdata noobs;
  var JobCode EmpID Salary;
  sum Salary;
run;
```

(Note: the SUM statement also produces subtotals if you print the data in groups.)

Sequencing and Grouping Observations

Objectives

- Sequence (sort) observations in a SAS data set.
- Group observations in a list report.
- Print column subtotals in a list report.
- Control page breaks for subgroups

Sorting a SAS Data Set

To request subgroup totals in PROC PRINT, the observations in the data set must be grouped.

The SORT procedure

- rearranges the observations in a SAS data set
- can create a new SAS data set containing the rearranged observations
- can sort on multiple variables
- can sort in ascending (default) or descending order
- does not generate printed output
- treats missing values as the smallest possible value.

Sorting a SAS Data Set

General form of the PROC SORT step:

```
PROC SORT DATA=input-SAS-data-set
            <OUT=output-SAS-data-set>;
    BY <DESCENDING> by-variable(s);
RUN;
```

Examples (sorts the data by Salary):

```
proc sort data=ia.empdata;
    by Salary;
run;
```

CAUTION!

Overwrites the original dataset with the **sorted** data set

```
proc sort data=ia.empdata OUT=work.jobsal;
    by Salary;
run;
```

Saves the sorted data in a **new** data set

Sorting a SAS Data Set

When you include more than one Sort variable in the BY statement:

- SAS sorts the data set by the first variable listed
- then sorts by the second variable WITHIN the values of the first variable
- then sorts by the third variable WITHIN the values of the second variable, etc

By default, SAS sorts in **ascending** order. The keyword **descending** applies to the **following** variable.

Sorting Exercise

- The **admit** dataset in Classes\Pstat130\data1 contains hospital admitting information on patients
- Print the **admit** dataset
- Sort the **admit** dataset INTO A NEW dataset called work.temp_admit
 - Sort it by Sex Actlevel
 - Print the new dataset
- Now sort by Actlevel Sex
 - Print the new dataset

Create listings for subgroups

The BY statement enables you to

- Create a separation between listings of several subgroups
- Identify each subgroup of the top of its listing
- Data set must be sorted on BY variable first

General form of the BY statement:

BY variable(s);

Example:

```
proc print data=data1.diabetes;  
  BY Sex;  
run;
```

Sex=F

Obs	ID	Name	Age	Date	Height	Weight	Level	Fee
1	2462	Almers, C	34	3	66	152	HIGH	124.80
2	2571	Nunnelly, A	44	19	66	140	HIGH	149.75
3	2575	Quigley, M	40	8	69	163	HIGH	124.80
4	2589	Wilcox, E	41	16	67	141	HIGH	149.75
5	2501	Bonaventure, T	31	17	61	123	LOW	149.75
6	2568	Eberhardt, S	49	27	64	172	LOW	124.80
7	2572	Oberon, M	28	17	62	118	LOW	85.20
8	2588	Ivan, H	22	20	63	139	LOW	85.20
9	2523	Johnson, R	43	31	63	137	MOD	149.75
10	2552	Reberson, P	32	9	67	151	MOD	149.75
11	2584	Takahashi, Y	43	29	65	123	MOD	124.80

Sex=M

Obs	ID	Name	Age	Date	Height	Weight	Level	Fee
12	2458	Murray, W	27	1	72	168	HIGH	85.20
13	2544	Jones, M	29	6	76	193	HIGH	124.80
14	2586	Derber, B	25	23	75	188	HIGH	85.20
15	2539	LaMance, K	51	4	71	158	LOW	124.80
16	2563	Pitts, D	34	22	73	154	LOW	124.80
17	2579	Underwood, K	60	22	71	191	LOW	149.75
18	2555	King, E	35	13	70	173	MOD	149.75
19	2574	Peterson, V	30	6	69	147	MOD	149.75
20	2578	Cameron, L	47	5	72	173	MOD	124.80
21	2595	Warren, C	54	7	71	183	MOD	149.75

Printing Subtotals and Grand Totals

Print the data set grouped by **JobCode** with a subtotal for the **Salary** column for each JobCode.

```
proc sort data=ia.empdata  
    out=work.empdata;  
    by JobCode;  
run;  
proc print data=work.empdata;  
    by JobCode;  
    sum Salary;  
run;
```

Using a **BY statement** and a **SUM statement** together in a PROC PRINT step produces subtotals and grand totals.

----- ActLevel=LOW -----

Obs	ID	Name	Sex	Age	Date	Height	Weight	Fee
8	2501	Bonaventure, T	F	31	17	61	123	149.75
9	2568	Eberhardt, S	F	49	27	64	172	124.80
10	2572	Oberon, M	F	28	17	62	118	85.20
11	2588	Ivan, H	F	22	20	63	139	85.20
12	2539	LaMance, K	M	51	4	71	158	124.80
13	2563	Pitts, D	M	34	22	73	154	124.80
14	2579	Underwood, K	M	60	22	71	191	149.75

ActLevel

844.30

----- ActLevel=MOD -----

Obs	ID	Name	Sex	Age	Date	Height	Weight	Fee
15	2523	Johnson, R	F	43	31	63	137	149.75
16	2552	Reberson, P	F	32	9	67	151	149.75
17	2584	Takahashi, Y	F	43	29	65	123	124.80
18	2555	King, E	M	35	13	70	173	149.75
19	2574	Peterson, V	M	30	6	69	147	149.75
20	2578	Cameron, L	M	47	5	72	173	124.80
21	2595	Warren, C	M	54	7	71	183	149.75

ActLevel

998.35
=====
2687.0

Page Breaks

Use the PAGEBY statement to put each subgroup on a separate page.

General form of the PAGEBY statement:

PAGEBY by-variable;

Example:

```
proc print data=work.empdata;
  by JobCode;
  pageby JobCode;
  sum Salary;
run;
```

The PAGEBY statement **must** be used with a BY statement.

Printing Sorted Observations

If you attempt to use the **BY** statement on an un-sorted data set, you will get an **error**.

ERROR: Data set IA.EMPDATA is not sorted in ascending sequence. The current by-group has JobCode = PILOT and the next by-group has JobCode = FLTAT.

NOTE: The SAS System stopped processing this step because of errors.

NOTE: There were 2 observations read from the data set IA.EMPDATA.

Identifying Observations

Objectives

- Use the ID statement to identify observations.
- Combine the BY and ID statements to produce special formatting.

Identifying Observations

The ID statement enables you to

- suppress the Obs column in the report
- specify which variable(s) should replace the Obs column.

General form of the ID statement:

ID variable(s);

Example:

```
proc print data=ia.empdata;
  id JobCode;
  var EmpID Salary;
run;
```

Special BY-Group Formatting

When the ID and BY statements specify the same variable,

- the Obs column is suppressed
- the BY line is suppressed
- the ID/BY variable prints in the leftmost column
- each ID/BY value only prints at the start of each BY group
(and on the subtotal line, if a SUM statement is used).

Special BY-Group Formatting

- Specify **JobCode** in the BY and ID statements to change the report format.

```
proc sort data=ia.empdata out=work.empdata;
  by JobCode;
run;
proc print data=work.empdata;
  by JobCode;
  id JobCode;
  sum Salary;
run;
```

Special BY-Group Formatting

The SAS System				
Job	Emp			
Code	ID	LastName	FirstName	Salary
FLTAT	0040	WILLIAMS	ARLENE M.	23666.12
	0071	PERRY	ROBERT A.	21957.71
	0091	SCOTT	HARVEY F.	32278.40
	0106	THACKER	DAVID S.	24161.14

FLTAT				102063.37
PILOT	0031	GOLDENBERG	DESIREE	50221.62
	0082	MCGWIERN-WATTS	CHRISTINA	96387.39
	0355	BELL	THOMAS B.	59803.16
	0366	GLENN	MARTHA S.	120202.38

PILOT				326614.55
=====				
				428677.92

PSTAT 130

SAS Base Programming

Summer 2017

William Qiu

Department of Statistics and Applied Probability

UCSB

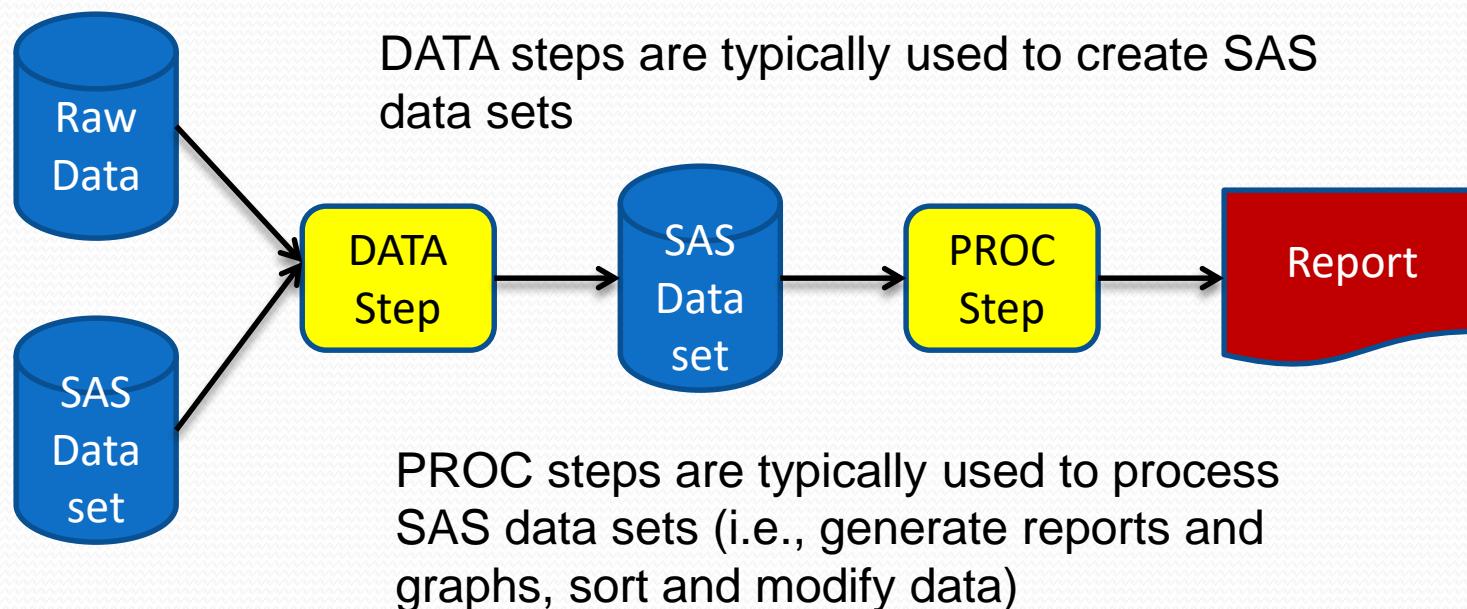
Class 4

Objectives

- State the components of a SAS program
- State the modes in which you can run SAS

SAS Programs

- A SAS Program is a collection of statements, grouped into a sequence of steps, that the programmer submits for execution



SAS Programs

```
data work.staff;
  infile 'X:\PStat 130\data1\emplist.dat';
  input LastName $ 1-20 FirstName $ 21-30
        JobTitle $ 36-43 Salary 54-59;
run;

proc print data=work.staff;
run;

proc means data=work.staff;
  class JobTitle;
  var Salary;
run;
```

DATA Step

PROC Steps

Step Boundaries

SAS steps begin with a

- DATA statement
- PROC statement

SAS detects the end of a step when it encounters

- A RUN statement (for most steps)
- A QUIT statement (for some procedures)
- The beginning of another step (DATA statement or PROC statement)

Step Boundaries

```
data work.staff;
  infile 'raw-data-file';
  input LastName $ 1-20 FirstName $ 21-30
    JobTitle $ 36-43 Salary 54-59;
run;

proc print data=work.staff;
run;

proc means data=work.staff;
  class JobTitle;
  var Salary;
run;
```

Running a SAS Program

You can invoke SAS in

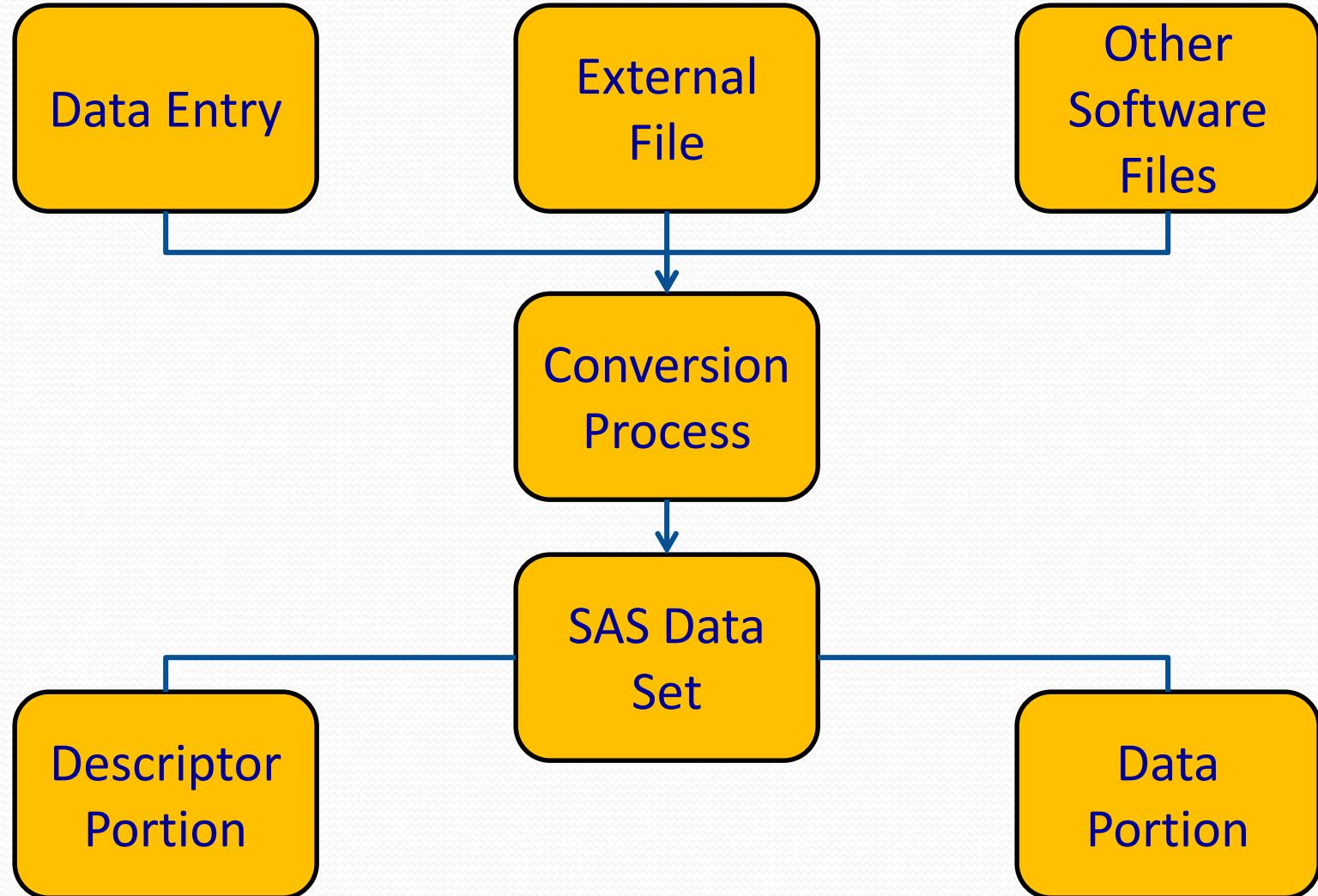
- Interactive windowing mode (SAS windowing environment)
- Interactive menu-driven mode (Enterprise Guide, SAS/ASSIST , SAS/AF, or SAS/EIS software)
- Batch mode
- Noninteractive mode

Fundamental Concepts

Objectives

- Define the components of a SAS data set
- Define a SAS variable
- Identify a missing value and a SAS date value
- State the naming conventions for SAS data sets and variables
- Explain SAS syntax rules
- Investigate a SAS data set using the CONTENTS and PRINT procedures

SAS Data Sets



SAS Data Sets

SAS data sets have a descriptor portion and a data portion.

Descriptor Portion	<p>General data set information</p> <ul style="list-style-type: none">* data set name * data set label* date/time created * storage information* number of observations <p>Information for each variable</p> <ul style="list-style-type: none">* Name * Type * Length * Position* Format * Informat * Label																
Data Portion	<p>Observations for each variable</p> <table><thead><tr><th>Name</th><th>Age</th><th>Height</th><th>Weight</th></tr></thead><tbody><tr><td>John</td><td>19</td><td>69</td><td>180</td></tr><tr><td>Mary</td><td>22</td><td>63</td><td>130</td></tr><tr><td>John</td><td>21</td><td>67</td><td>165</td></tr></tbody></table>	Name	Age	Height	Weight	John	19	69	180	Mary	22	63	130	John	21	67	165
Name	Age	Height	Weight														
John	19	69	180														
Mary	22	63	130														
John	21	67	165														

Browsing the Descriptor Portion

The descriptor portion of a SAS data set contains

- general information about the SAS data set (such as)
- data set name and number of observations)
- variable attributes (name, type, length, position, informat, format, label).

The CONTENTS procedure (PROC CONTENTS) displays the descriptor portion of a SAS data set.

Browsing the Descriptor Portion

General form of the CONTENTS procedure:

```
PROC CONTENTS DATA=<SAS-data-set>;  
RUN;
```

Example:

```
proc contents data=data1.empdata;  
run;
```

Sample CONTENTS Output

The SAS System

The CONTENTS Procedure

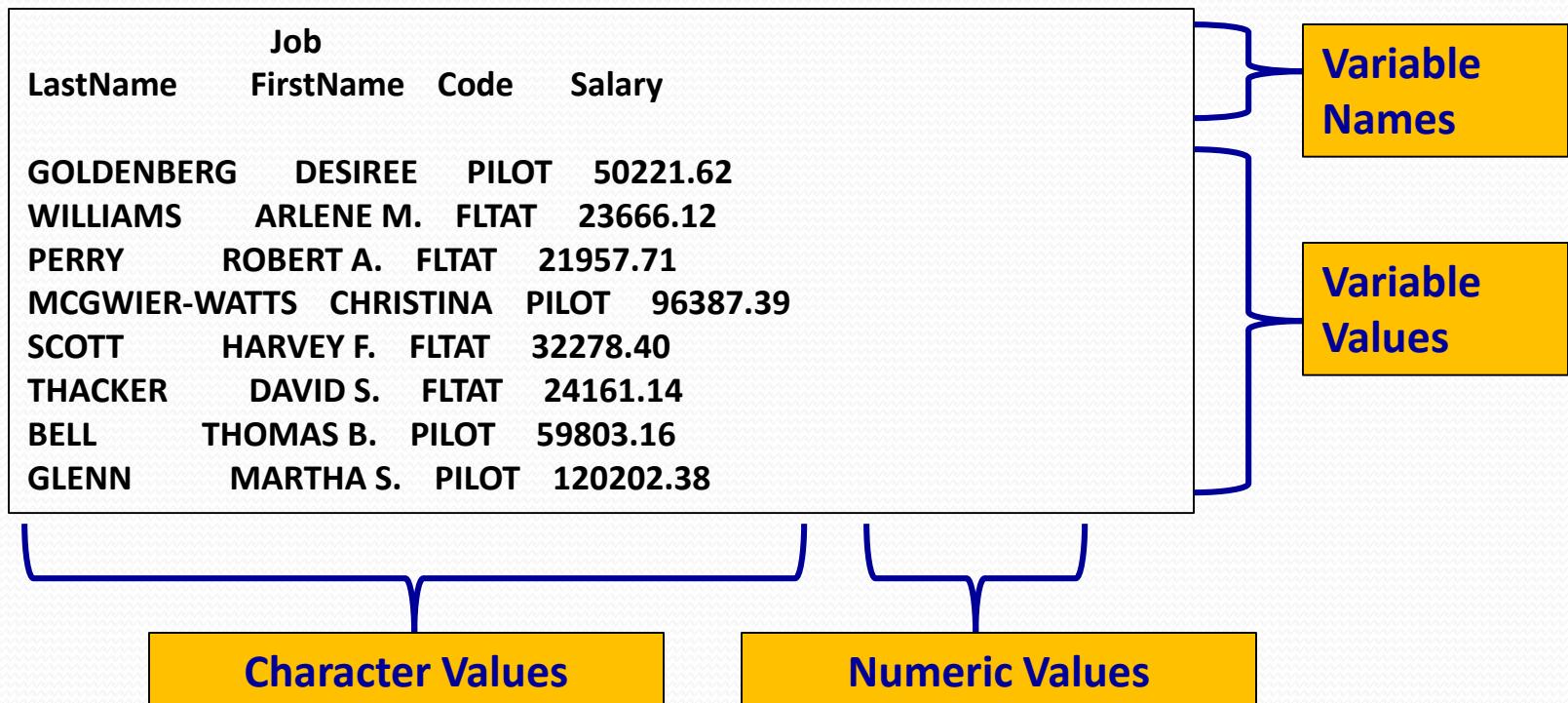
Data Set Name	WORK.EMPDATA	Observations	8
Member Type	DATA	Variables	5
Engine	V9	Indexes	0
Created	Sunday, October 03, 2010 04:12:45 PM	Observation Length	48
Last Modified	Sunday, October 03, 2010 04:12:45 PM	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	WINDOWS_32		
Encoding	wlatin1 Western (Windows)		

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
1	EmpID	Char	4
3	FirstName	Char	13
4	JobCode	Char	5
2	LastName	Char	13
5	Salary	Num	8

SAS Data Sets: Data Portion

The data portion of a SAS data set is a rectangular table of character and/or numeric data values.



SAS Variable Values

There are two types of variables:

Character

- contains any value: letters, numbers, special characters, and blanks. Character values are stored with a length of 1 to 32,767 bytes. One byte equals one character.

Numeric

- stored as floating point numbers in 8 bytes of storage by default. Eight bytes of floating point storage provide space for 16 or 17 significant digits. You are not restricted to 8 digits.

SAS Data Set and Variable Names

SAS names:

- can be 32 characters long.
- can be uppercase, lowercase, or mixed-case.
- must start with a letter or underscore. Subsequent characters can be letters, underscores, or numeric digits.

Valid SAS Names

Which of the following are valid SAS names?

- A. data5mon
- B. 5monthsdata
- C. five months data
- D. five_month_sdata
- E. data#5

SAS Date Values

- SAS stores date values as numeric values.
- A SAS date value is stored as the number of days between January 1, 1960, and a specific date.

Missing Data Values

A value must exist for every variable for each observation.
Missing values are valid values.

LastName	FirstName	Job Code	Salary
GOLDENBERG	DESIREE	PILOT	50221.62
WILLIAMS	ARLENE M.	FLTAT	23666.12
PERRY	ROBERT A.	FLTAT	.
MCGWIER-WATTS	CHRISTINA	PILOT	96387.39
SCOTT	HARVEY F.		32278.40

A character missing value is displayed as a blank.

A numeric missing value is displayed as a period.

Browsing the Data Portion

The PRINT procedure displays the data portion of a SAS data set.

By default, PROC PRINT displays

- all variables
- all observations
- an Obs column on the left side.

Browsing the Data Portion

General form of the PRINT procedure:

```
PROC PRINT DATA=<SAS-data-set>;  
RUN;
```

Example:

```
proc print data=work.empdata;  
run;
```

SAS Data Set Terminology

- SAS documentation and text in the SAS windowing environment use the following terms interchangeably:
 - SAS Data Set = SAS Table
 - Variable=Column
 - Observation=Row

SAS Syntax Rules

- SAS statements usually begin with a keyword and always end with a semi-colon.
- SAS statements are free-format.
- One or more blanks or special characters can be used to separate words.
- Statements can begin and end in any column.
- A single statement can span multiple lines.
- Several statements can be on the same line.

Unconventional Spacing, Poor Readability

```
libname mydrive 'W:\Classes\Pstat  
130\data1'; options nonumber nodate ls=80;  
data staff; set mydrive.empdata; keep  
lastname firstname jobcode salary; run;  
proc contents data=work.staff; run; proc  
print noobs; run;
```

Conventional Spacing, Better Readability

```
libname mydrive 'E:\UCSB\Pstat 130\data1';
options nonumber nodate ls=80;

data staff;
  set mydrive.empdata;
  keep lastname firstname jobcode salary;
run;

proc contents data=work.staff;
run;

proc print noobs;
run;
```

SAS Libraries - Details

A SAS library is a collection of SAS data files

- Use the LIBNAME statement to assign a “libref” (library reference) to a SAS data library

General form:

```
LIBNAME libref 'SAS-data-library' <options>;
```

- Rules for Naming a Libref :
 - must be 8 characters or less
 - must begin with a letter or underscore
 - remaining characters are letters, numbers, or underscores.

Two-level SAS Data Filenames

Every SAS data file has a two-level name:

General Form

Libref.filename

Examples:

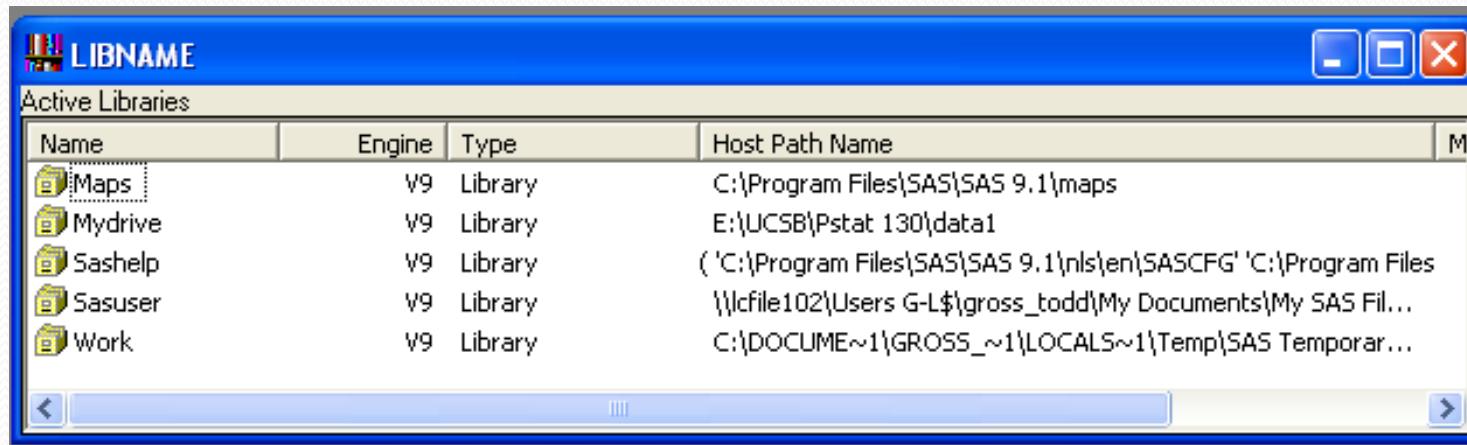
- **ia.empdata**
- **work.staff**

Browsing SAS Libraries

During an interactive SAS session, the LIBNAME window enables you to investigate the contents of a SAS data library.

In the LIBNAME window, you can

- view a list of all the libraries available during your current SAS session
- drill down to see all members of a specific library
- display the descriptor portion of a SAS data set



Browsing a SAS Data Library

Use the `_ALL_` keyword to list all the SAS files in the library and the `NODS` option to suppress the descriptor portions of the data sets.

- General form of the `NODS` option:

```
PROC CONTENTS DATA=libref._ALL_ NODS;  
RUN;
```

- `NODS` must be used in conjunction with the keyword `_ALL_`.

```
proc contents data=ia._all_ nods;  
run;
```

Browsing a SAS Library – Partial Output

The SAS System

The CONTENTS Procedure

Directory

Libref IA
Engine V9
Physical Name E:\UCSB\pstat 130\data1
File Name E:\UCSB\pstat 130\data1

#	Name	Member	File	Size	Last Modified
		Type			
1	ADMIT	DATA	9216	12Apr04:10:16:08	
2	ADMITJUNE	DATA	9216	12Apr04:10:16:08	
3	ALLGOALS	DATA	5120	12Apr04:10:16:08	
4	ALLGOALS2	DATA	5120	12Apr04:10:16:08	
5	ALLSALES	DATA	5120	12Apr04:10:16:08	
6	ALLSALES2	DATA	5120	12Apr04:10:16:08	
7	APRTARGET	DATA	17408	12Apr04:10:16:08	
8	CHICAGO	DATA	17408	12Apr04:10:16:14	

PSTAT 130

SAS Base Programming

Summer 2017

William Qiu

Department of Statistics and Applied Probability

UCSB

Class 5

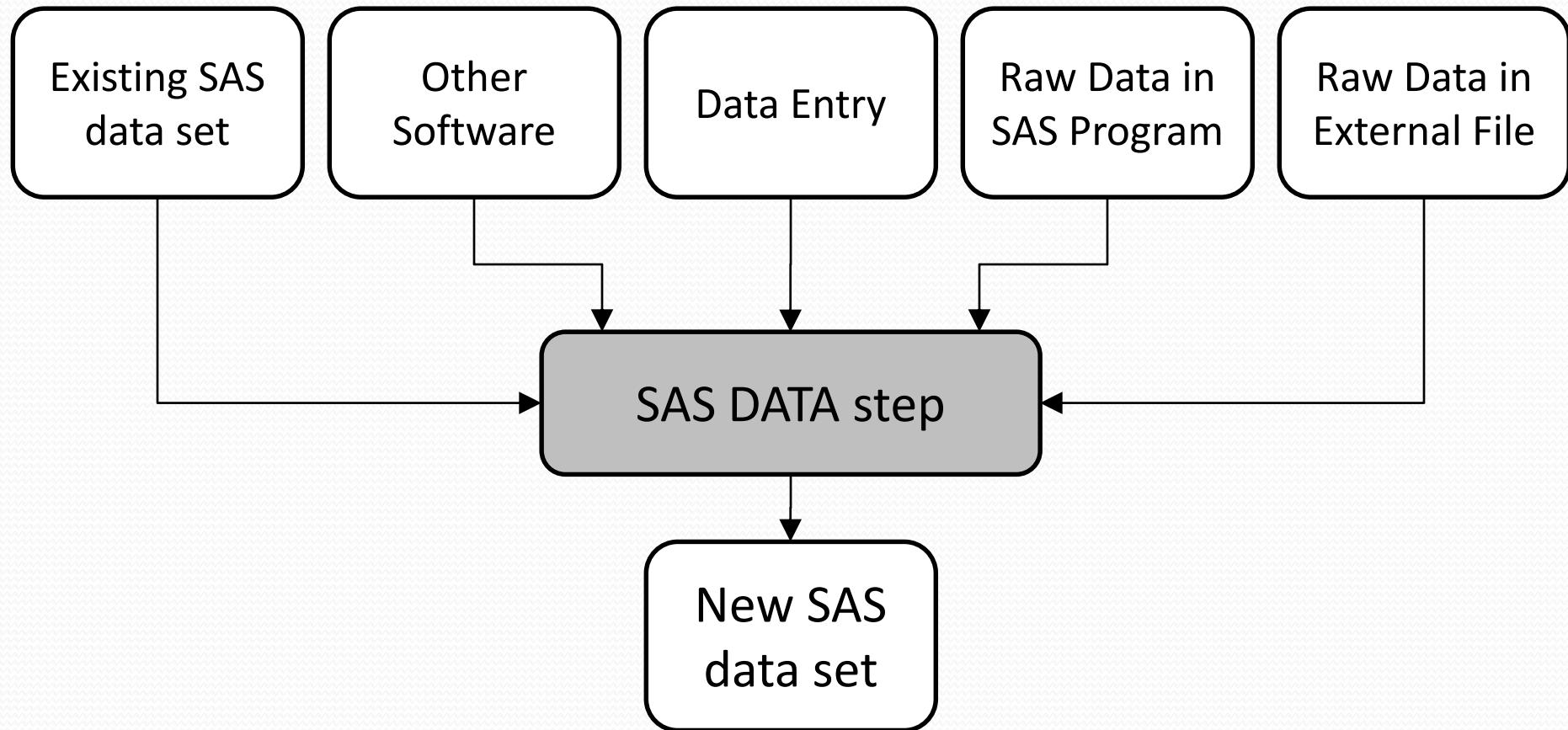
Objectives

- Identify the five sources of SAS data
- Create temporary and permanent SAS data sets
- Use list, column and formatted input to read raw data
- Explain how the DATA step processes data

Fundamental Principles

- The **source, structure and format** of your data determines the methods used to bring it into SAS
 - **Source** – where the data are stored (e.g., paper, text file, another program file)
 - **Structure** – what variables are present in the data (e.g., last name, hire date, salary, flight number, destination)
 - **Format** – how are values displayed (e.g., in neat columns, separated by spaces, 10/12/2010 vs. 12OCT2010)

Five Data Sources



Creating a SAS data set

Data Source	KEYWORD/Method
Data in a permanent SAS dataset or created by another SAS program	SET
Data created in another software program (e.g., Excel)	Import (interactive)
Data Entry	Table Editor
<u>Raw Data</u> in a SAS Program	DATALINES
<u>Raw Data</u> in an external text file	INFILE

Using an Existing SAS Data Set

If your data are in an existing SAS data set:

- Use the **SET** statement to read the **existing SAS data set**.
The data step creates a new SAS data set.

Example:

```
data work.employees;  
  set ia.empdata;  
run;
```

SET can refer to any existing SAS data set, temporary or permanent

Importing Data from Other Software

If your data were created by other software (e.g., Excel)

- Use the IMPORT option on the File menu to read it in and create a new SAS data set
- You will need to tell SAS
 - what format the data are in
 - where the file is stored
 - what character is used to separate data values, (i.e., the “delimiter ”)
- Save your SAS data set in an existing library or create a new library

Data Entry in SAS Table Editor

- Start SAS
- Go to the Tools Menu and Open the Table Editor
- Enter data values into the table
- Set the name, type, and width for each variable/column using Data Menu - Column Attributes (or right-click on each column heading)
- Save the data set into a library (you can create a library at this time, if desired)
- You can now use this SAS data set in your program

Create SAS Data Set from Raw Data

To create a SAS data set from raw data you need to:

1. Use a **DATA** statement to name the SAS data set
2. Use an **INFILE** or **DATALINES** statement to identify where to find the raw data
3. Use an **INPUT** statement to identify the variables in the raw data file

Naming a SAS Data Set

General form of the DATA statement;

- This DATA statement creates a temporary SAS data set

```
data work.dfwlax;  
    set ia.dfwlax;  
run;
```

- This DATA statement creates a permanent SAS data set

```
libname mydrive 'D:\';  
  
data mydrive.dfwlax;  
    set ia.dfwlax;  
run;
```

Reading Raw Data within a SAS Program

If the raw data are contained in the SAS Program:

- Use the **DATALINES** keyword, followed by the **raw data lines**.

Example:

```
data work.sample;
    input firstname $ gender $ age;
    datalines;
        John Male 22
        Jane Female 19
    ;
run;
```

This method works well for small amounts of data.

Pointing to a Raw Data File

If the raw data are contained in a separate text file:

- Use the **INFILE** statement, followed by a **file specification**.

Example:

```
data work.sample;
    infile 'D:\UCSB\sample.txt';
    input name $ gender $ age;
run;
```

- This method is much more common than DATALINES.
- The file specification does NOT use a library reference.
- The INFILE statement comes before the INPUT statement.

DATALINES versus INFILE

- The **DATALINES** method:
 - Allows you to see your data directly
 - Is good for small amounts of data
 - Is often used to create “test” data sets
- The **INFILE** method
 - Is more common
 - Is necessary if your data comes from an outside source (i.e., client, download, website, etc)
 - Is preferred for large data sets
 - Allows you to easily re-run your programs on updated data

Reading Data Fields

The **INPUT** statement tells SAS the **name** and **type** of each variable in the data set, and how to read the data.

Simple form of the INPUT statement

```
INPUT variable <$> <options>
```

General form of the INPUT statement:

```
INPUT <pointer-control> variable <$> <&> <@ | @@>;
```

```
INPUT variable <$> start-column <- end-column> <.decimals> <@ | @@>;
```

```
INPUT <pointer-control> variable <:|&|~> <informat.> <@ | @@>;
```

INPUT Statement - Example

```
data work.students;
    input firstname $ gender $ age;
datalines;
John Male 19
Wendy Female 22
;
run;
```

Types of Raw Data Input

- **List Input** - each data value is separated by a space as a “delimiter”
 - John Male 22
 - Wendy Female 19
- **Column Input** – each data value is in a fixed location
 - John Male 22
 - Wendy Female 19
- **Formatted Input** – uses SAS formats (called informats)
 - John Male 4/12/91
 - Elizabeth Female 8/24/90

Simple List Input

If your data values are separated by a single space, use list input

Example:

```
data work.students;
    input Name $ Team $ Age;
    datalines;
David Male 19
Amelia Female 23
Ravi Male 17
Ashley Female 20
Jim Male 26
;
run;
```

Column Input

If your data values are in fixed columns, and consist of “standard” character and numeric values, use **column input**

Example:

```
data work.students;
input Name $ 1-6 Gender $ 9-14 Age 18-20;
datalines;
David   Male      19
Amelia  Female    23
Ravi    Male      17
Ashley  Female    20
Jim     Male      26
;
run;
```

Formatted Input

- If your data contains “non-standard” values, use formatted input, called “informats”

```
data students;  
input Name $ Gender $ Age Enroll mmddyy8.;  
datalines;  
David Male 19 06/18/10  
Amelia Female 23 08/02/10  
Ravi Male 17 07/22/10  
Ashley Female . 09/14/10  
Jim Male 26 08/26/10  
;  
run;
```

informats

non-standard data
(dates)

Format versus Informat

- A **Format** controls the way data look when you print them.
- An **Informat** controls the way SAS reads data in.

PSTAT 130

SAS Base Programming

Summer 2017

William Qiu

Department of Statistics and Applied Probability

UCSB

Class 6

Review – Reading Data into SAS

There are three steps to reading raw data

1. DATA statement
2. INFILE or DATALINES statement
3. INPUT statement

There are three basic types of input

1. List input
Input Name \$ GPA Score
2. Column input
Input Name \$ 1-8 GPA 10-12 Score 14-16
3. Formatted input
Input Name \$ 8. GPA 3.1 Score 3.

Class Exercise

The data set below contains the 2010 population estimates (in millions) of several U.S. states. Which type of input statement would be used for the data set below?

Calif 36.9

Texas 24.8

NewYork 19.5

Florida 18.5

Illinois 12.9

Write the Input statement.

Attributes of List Input

- All data values must be separate by a single space
- All variables must be in standard format
 - Character and numeric values cannot contain spaces
 - Character values cannot be longer than 8 characters
 - Numeric values cannot contain commas or dollar signs
 - Dates will be read as characters rather than date values

Class Exercise (con't)

The data set below contains the 2010 population estimates (in millions) of several U.S. states. Which type of input statement would be used for the data set below?

California	36.9
Texas	24.8
New York	19.5
Florida	18.5
Illinois	12.9

Write the Input statement.

Attributes of Column Input

- The data values must occupy the same columns within each observation
 - This is called “fixed” or “aligned”
- Character variables can
 - be longer than 8 characters
 - contain spaces
- You can skip some data fields, if desired
- The data must be in “standard” format
 - Numbers may not contain commas or dollar signs
 - Dates will be read as character, instead of numeric, variables

Class Exercise (con't)

The data set below contains the 2010 population estimates of several U.S. states. Which type of input statement would be used for the data set below?

California	36,961,664
Texas	24,782,302
New York	19,541,453
Florida	18,537,969
Illinois	12,910,409

Write the Input statement.

Attributes of Formatted Input

- Data can be in “non-standard” format
 - Numbers can contain commas and dollar signs
 - Dates can be read as numeric variables
- Data can be free-form or fixed text files

Reading Excel Spreadsheets

- Create a SAS data set from an Excel spreadsheet using the Import Wizard.
- Create a SAS data set from an Excel spreadsheet using PROC IMPORT.

The IMPORT Procedure

General form of the IMPORT procedure

```
PROC IMPORT OUT=SAS-data-set
      DATAFILE='external-file-name'
      DBMS=file-type;
      GETNAMES=YES;
RUN;
```

Excel Import Example

```
PROC IMPORT OUT= WORK.tdfwlax  
            DATAFILE= "D:\Pstat 130\data1\DallasLA.xls"  
            DBMS=XLS REPLACE;  
            SHEET="DFWLAX";  
            GETNAMES=YES;  
            MIXED=NO;  
            SCANTEXT=YES;  
            USEDATE=YES;  
            SCANTIME=YES;  
  
RUN;
```

Assigning Variable Attributes

SAS allows you to:

- Assign permanent attributes to SAS variables.
- Change or override permanent variable attributes.

Default Variable Attributes

When a variable **is created** in a DATA step, the

- name, type, and length of the variable are automatically assigned
- remaining attributes such as label and format are not automatically assigned.

When the variable is used in a later PROC step, the output uses:

- the **variable name**
- a **system-determined format**.

Specifying Variable Attributes

Use LABEL and FORMAT statements in the

- DATA step to permanently assign the attributes (stored in the data set descriptor portion).
- PROC step to temporarily assign the attributes (for the duration of the step only)

Comparison of assignment in DATA and PROC steps

```
DATA work.bonus;  
  Set pstatlib.empdata;  
  Bonus = Salary * .1;  
  Label Bonus = 'Annual Bonus';  
  Format Bonus Dollar12.2;  
Run;
```

Permanent Attributes

```
PROC PRINT data=work.bonus;  
  Label Bonus = 'Incentive Bonus';  
  Format Bonus Dollar12.;  
Run;
```

Temporary Attributes

The DATASETS Procedure

You can use the DATASETS procedure to modify a variable's

- name
- label
- format
- informat.

The DATASETS Procedure

General form of PROC DATASETS for changing variable attributes:

```
PROC DATASETS LIBRARY=libref;  
  MODIFY SAS-data-set ;  
  RENAME old-name-1=new-name-1  
         <... old-name-n=new-name-n>;  
  LABEL variable-1='label-1'  
        <... variable-n='label-n'>;  
  FORMAT variable-list-1 format-1  
        <... variable-list-n format-n>;  
  INFORMAT variable-list-1 informat-1  
        <... variable-list-n informat-n>;  
RUN;
```

Selecting Variables

Use the **KEEP** or **DROP** statement to select variables from the data set.

- The **KEEP** statement identifies the variables that will remain in the data set. Only the variables in the KEEP statement will remain. All others are eliminated.
- The **DROP** statement identifies variables to be eliminated from the data set. All variables except those in the DROP statement will remain.

KEEP vs KEEP= and DROP vs. DROP=

Use the KEEP or DROP statement to eliminate variables from the **output** data set. These variables can still be used in SAS expressions.

```
Data libref.new-data-set;
KEEP variables;
or
DROP variables;
run;
```

Use the KEEP= or DROP= option in a SET statement to eliminate variables from the **input** statement. The eliminated variables cannot be used in expressions.

```
DATA libref.new-data-set;
SET SAS-data-set(KEEP=variables)
or
SET SAS-data-set(DROP=variables)
run;
```

DROP & KEEP Examples

- Use DROP if you want to keep most of the variables

```
data work.tempdata;
  Set ia.empdata;
  Drop EmpID Hire;
run;
```

- Use KEEP if you only want to keep a few variables

```
data work.tempdata;
  Set ia.empdata;
  Keep Firstname Lastname;
run;
```

Creating New Variables

Use **variable assignment** statements in the DATA step to create new variables

An assignment statement:

- evaluates an expression
- assigns the resulting value to a variable.

General form of an assignment statement

```
DATA output-SAS-data-set;  
  SET input-SAS-data-set;  
  variable = expression;  
RUN;
```

SAS Expressions

An **expression** contains **operands** and **operators** that form a set of instructions that produce a value.

Operands are

- Variable names
- Constants

Operators are

- Arithmetic symbols (+, -, /, *, etc)
- SAS functions

Using Operators

Selected **operators** for basic arithmetic calculations in an assignment statement:

Operator	Action	Example	Priority
+	Addition	Sum=x+y;	III
-	Subtraction	Diff=x-y;	III
*	Multiplication	Product=x*y;	II
/	Division	Divide=x/y;	II
**	Exponent	Raise=x**y;	I
-	Negative	Negative = -x;	I

Variable Assignment

Here are some examples of creating new variables using arithmetic operators:

- **TotalComp = Salary + Bonus ;**
- **NetPay = GrossPay - Tax**
- **NewPay = Salary * (1 + Raise) ;**
- **Percent = Score/Maximum**

Using SAS Functions

A SAS function is a routine that returns a value that is determined from specified arguments.

General form of a SAS function:

Function-name(argument1, argument2, ...);

Example:

Total=sum (Salary, Bonus) ;

Using SAS Functions

SAS functions

- perform **arithmetic operations**
- compute **sample statistics** (for example: sum, mean, and standard deviation)
- **manipulate** SAS dates and process character values
- perform many other tasks.

Sample statistics functions **ignore** missing values.

Numeric Functions

- **MIN(x,y,z)**- Returns the smallest value
- **MAX(x,y,z)**- Returns the largest value
- **MEDIAN(x,y,z)**- Calculates the median
- **MEAN(x,y,z)** – Calculates the average value
- **STD(x,y,z)** – Returns the standard deviation of the values
- **ABS(x,y,z)** – Returns the absolute value
- **FACT(x)** – Calculates the factorial, $x!$
- **COS(x)** – Returns the cosign of x
- **TAN(x)** – Returns the tangent of x

Character Functions

- **LOWCASE(x)** – Converts all letters in x to lower case
- **UPCASE(x)** – Converts all letters in x to upper case
- **TRIM(x)** – Removes all trailing blanks
- **SUBSTR(string,position,length)** – Returns a subset of given length within the characters in the string, starting at the given position:
 - $\text{SUBSTR}(\text{'Statistics'}, 1, 4) = \text{'Stat'}$
- **REVERSE(string)** – reverse the order of letters
 - $\text{REVERSE}(\text{'STATISTICS'}) = \text{'SCITSITATS'}$

Date Functions

TODAY() obtains the date value from the system clock.

MDY(month,day,year) uses numeric month, day, and year values to return the corresponding SAS date value.

YEAR(SAS-date) extracts the year from a SAS date and returns a four-digit value for year.

QTR(SAS-date) extracts the quarter from a SAS date and returns a number from 1 to 4.

MONTH(SAS-date) extracts the month from a SAS date and returns a number from 1 to 12.

WEEKDAY(SAS-date) extracts the day of the week from a SAS date and returns a number from 1 to 7, where 1 represents Sunday, and so on.

Using SAS Date Constants

Use a Date Constant to return a SAS date value for a specific date.

Example:

```
evaldate = '14FEB2009'd;
```

sets the value of **evaldate** to 17942, which is the SAS date value (number of days since 1960) corresponding to Feb 14, 2009. The text portion can be in the form of '**ddmmmyyyy**' or '**ddmmmyy**' .

DateTime Values and DATEPART

A *SAS datetime value* is interpreted as the **number of seconds** between midnight, January 1, 1960, and a specific date and time.

Example: 12/01/2000 9:15 am is stored as 1,291,281,300 seconds since 01/01/1960 12:00 am.

The DATEPART function will return just the Date portion, 14945, which is the number of days since 01/01/1960.

Example:

```
Birthdate = datepart(Birthdatetime) ;
```

Conditional Processing

- Execute statements conditionally using IF-THEN logic.
- Control the length of character variables explicitly with the LENGTH statement.
- Select rows to include in a SAS data set.

Conditional Execution

General form of IF-THEN and ELSE statements:

```
IF expression THEN statement;  
ELSE statement;
```

Expression contains operands and operators that form a set of instructions that produce a value.

Examples:

```
If Hours < 40 then Status = 'Parttime';  
If Jobcode = 'PILOT' then Bonus = Salary * 0.1;  
else Bonus = 0;
```

Only one executable statement is allowed per IF-THEN or ELSE statement.

Conditional Execution

To allow more than one statement, use DO and END statements.

```
IF expression THEN DO;
  executable statements
END;
ELSE DO;
  executable statements
END;
```

Subset Dataset

In a DATA step, you can subset the rows (observations) in a SAS data set with a:

- WHERE statement
- DELETE statement
- subsetting IF statement.

The WHERE statement in a DATA step is the same as the WHERE statement you saw in a PROC step.

Selecting and Deleting Rows

Use an IF statement to include only those rows that meet the criteria

IF *expression*;

Use an IF-THEN DELETE statement to exclude rows that meet the criteria

IF *expression* THEN *DELETE*;

WHERE versus IF

If the variable already exists in the input data set, you can use a WHERE statement.

```
data work.tempdata;  
  set ia.empdata;  
  where jobcode = 'Pilot';  
run;
```

If you are evaluating a calculated variable, use IF.

```
data work.tempdata;  
  set ia.empdata;  
  Bonus = Salary * 0.1;  
  if Bonus > $5000;  
run;
```

WHERE or Subsetting IF?

Step and Usage	WHERE	IF
PROC Step	Yes	No
DATA Step - Source of Variable		
INPUT statement (i.e., existing variable)	No	Yes
assignment statement (i.e., created variable)	No	Yes
SET/MERGE (multiple data sets)		
Variable in ALL data sets	Yes	Yes
Variable not in ALL data sets	No	Yes

PSTAT 130

SAS Base Programming

Summer 2017

William Qiu

Department of Statistics and Applied Probability

UCSB

Class 7

Overview of 2nd Half of Quarter

- Combining Data Sets
 - SET and MERGE
- Summarizing Data Sets
 - PROC MEANS, FREQ, TABULATE, REPORT
- Graphing Data
 - PROC GCHART, GPLOT
- Controlling Data Step Processing
- Do Loops – A Powerful Programming Tool
- Arrays and Macros

Looking Behind the Scenes

The DATA step is processed in two phases:

- compilation
- execution

```
data work.dfwlax;
infile 'raw-data-file';
input Flight $ 1-3 Date $ 4-11
      Dest $ 12-14 FirstClass 15-17
      Economy 18-20;
run;
```

Looking Behind the Scenes

At compile time, SAS creates

- an **input buffer** to hold the current raw data file record that is being processed

1	2	3
1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0
0 1 5 1 0 / 2 5 / 1 2 L A X 1 4 1 6 3		

- a **program data vector (PDV)** to hold the current SAS observation

Flight \$3	Date \$8	Dest \$3	FirstClass N8	Economy N8
015	10/25/12	LAX	14	163

- the **descriptor portion** of the output data set.

Compiling the Data Step

SAS creates the data set placeholder

SAS opens the data source

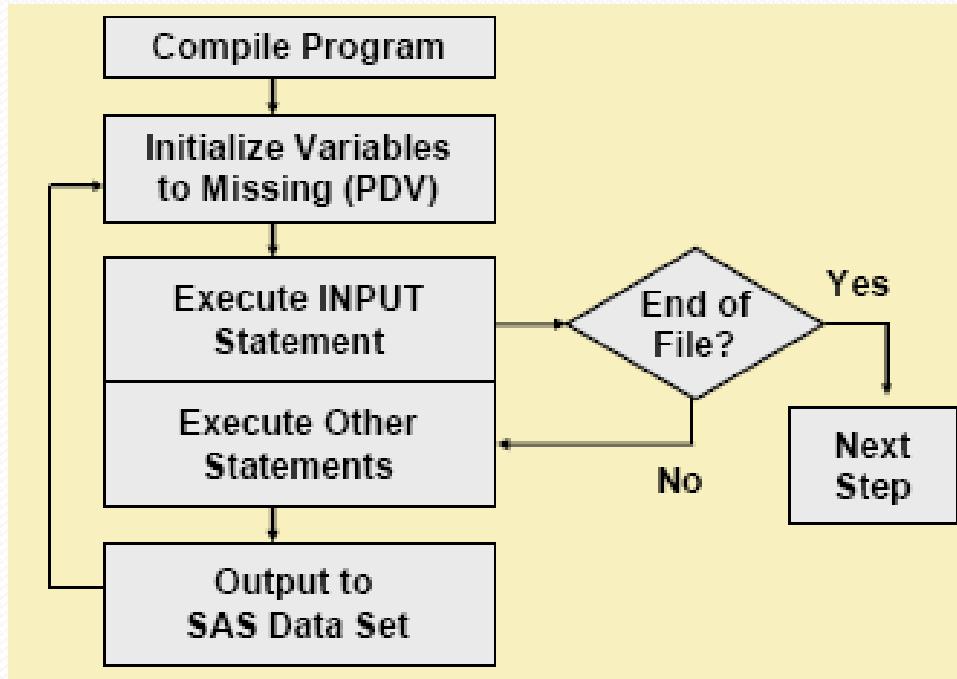
```
data work.dfwlax;
  infile 'raw-data-file';
  input Flight $ 1-3 Date $ 4-11
        Dest $ 12-14 FirstClass 15-17
        Economy 18-20;
run;
```

SAS prepares the input buffer

Lecture Overview

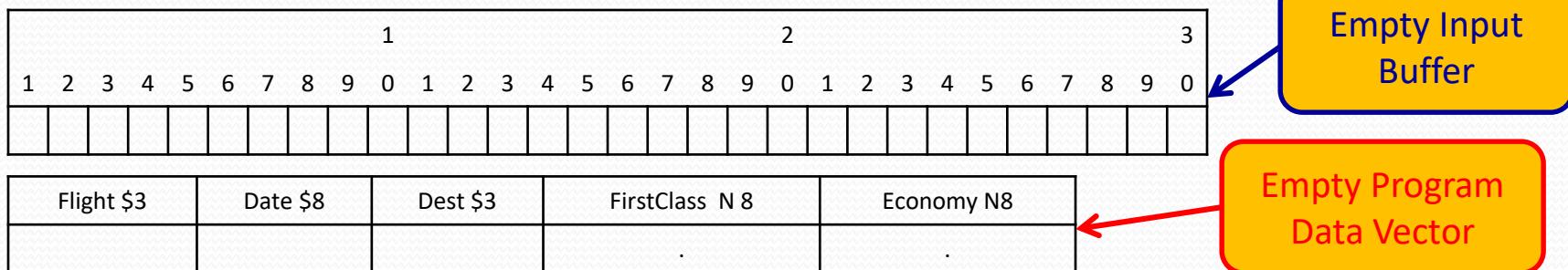
- A Look Behind Data Step Processing
 - Compilation and Execution
- More on INPUT statements
- Examining Errors
 - Data and Programming Errors

DATA Step Execution: Summary

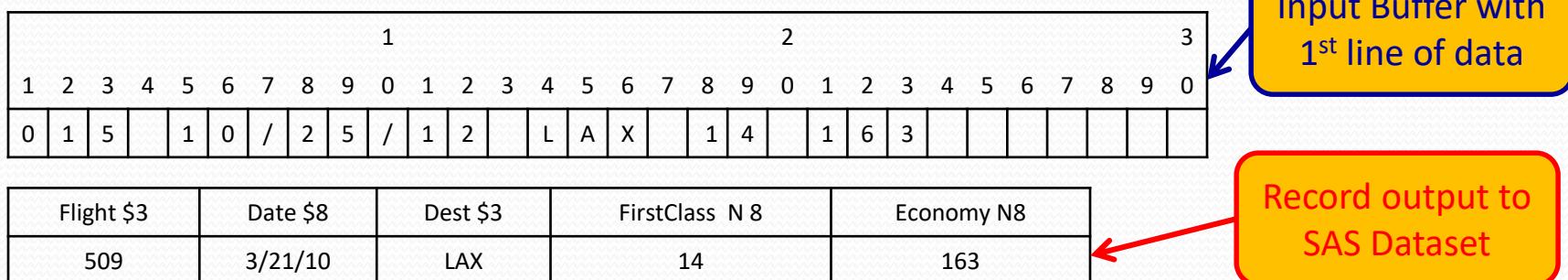


Data Step Execution: Details

- At compilation, SAS creates an empty Input Buffer, and an empty Program Data Vector, to store the incoming data



- It then loads the first line of data into the input buffer, parses it into variables, and outputs those values to the SAS dataset



Reading Data Using Formatted Input

Pointer controls:

- $@n$ moves the pointer to column n .
- $+n$ moves the pointer n positions.

An *informat* specifies

- the width of the input field
- how to read the data values that are stored in the field.

Pointer Control - Absolute

- With **formatted** input, you can “point” at the first column of each variable, instead of using start and end columns.
 - The @ symbol tells SAS at what column to start reading the value

```
data students;
input @1 Name $8. @9 Gender $6. @18 Age 2. @22
Enroll mmddyy8. ;
datalines;
David   Male      19  06/18/10
Amelia  Female    23  08/02/10
Ashley  Female    20  09/14/10
Jim     Male      26  08/26/10
;
run;
```

Pointer Control - Relative

- You can also move the pointer forward a specific number of spaces forward, using the + symbol

```
data students;
input Name $6. +2 Gender $6. +3 Age 2. +2 Enroll
      mmddyy8. ;
datalines;
David    Male      19  06/18/10
Amelia   Female    23  08/02/10
Ashley   Female    20  09/14/10
Jim      Male      26  08/26/10
;
run;
```

Class Exercise

- Write an input statement to read in the following data, which consists of five variables: instructor name, academic rank, annual salary, course name, and first class date.

John Tukey	Asst	\$56,000	PSTAT130	09/23/10
Sigmund Freud	Prof	\$92,000	PSYCH118	09/24/10
Karl Marx	Assoc	\$78,000	POLI125	09/27/10

Steps for Creating an Input Statement

1. How many variables are there in your data?
2. How many character variables? How many numeric variables?
3. Are the data values separated by a single space (List input) or are they presented in fixed columns (Column input)?
4. Are there any non-standard variables such as dates, commas or dollar signs (Formatted input)?
5. Write an input statement with a name for each variable, and a trailing dollar sign to indicate character variables.
6. For column input, add start and stop columns, or absolute/relative pointers for each variable.
7. For formatted input, add appropriate character, numeric or other formats to each variable.

What Are Data Errors?

SAS detects data errors when

- the INPUT statement encounters invalid data in a field
- illegal arguments are used in functions
- impossible mathematical operations are requested.

Examining Data Errors

When SAS encounters a data error,

1. a **note** that describes the error is printed in the SAS log
2. the **input record** being read is displayed in the SAS log (contents of the input buffer)
3. the **values in the SAS observation** being created are displayed in the SAS log (contents of the PDV)
4. a **missing value** is assigned to the appropriate SAS variable
5. execution **continues**.

Programming Errors - Tips

- Use the Enhanced Editor – it color-codes keywords and highlights errors in red.
- Write your program in small parts and test each part.
- Clear the log and output before running your program
- Review the log, looking for red and green text.
- Confirm the number of records and variables in each data set using the Log.
- Keep all variables in your interim data sets.
- Inspect the data sets you create in Table Editor, or using Proc Print.

PSTAT 130

SAS Base Programming

Summer 2017

William Qiu

Department of Statistics and Applied Probability

UCSB

Class 8

Lecture Outline

- Combining Datasets
 - Concatenating Datasets (Appending)
 - The SET statement
 - Merging Datasets
 - The MERGE and BY statements
 - Types of Merging
 - Parent-Child Relationships

COMBINING SAS DATA SETS

- Append (or concatenate) – **SET** statement
 - Two data sets each with the SAME variables
 - Data set A has m records and k variables
 - Data set B has n records and k variables
 - Combined data set has $(m + n)$ records and k variables
- Merge (or link) – use **MERGE & BY** statements
 - Two data sets with at least one common variable and other unique variables
 - Data set A has m records and k UNIQUE variables
 - Data set B has n records and j UNIQUE variables
 - Combined data set (typically) has $\max(m,n)$ records and $k+j+1$ variables

Concatenating SAS Data Sets

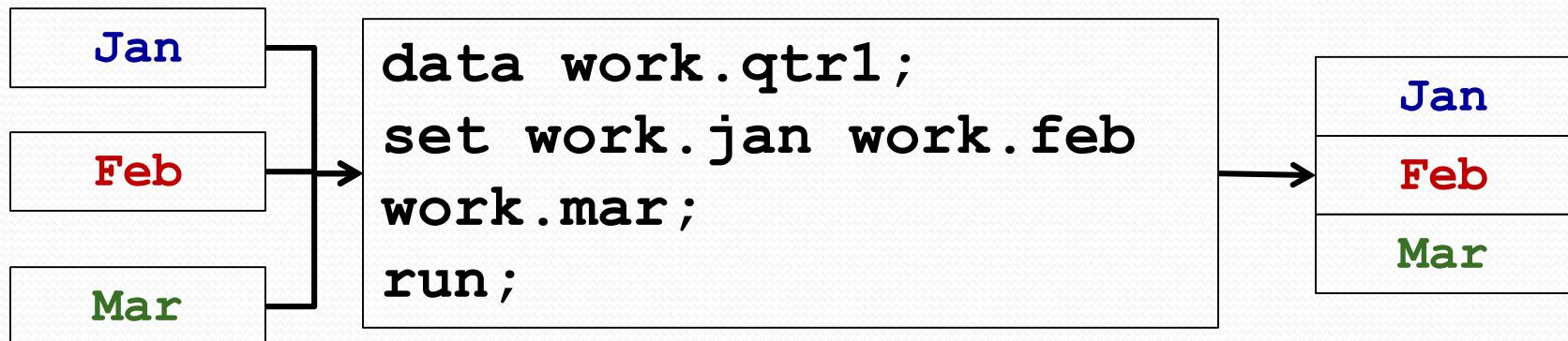
General form of a DATA step concatenation:

```
DATA SAS-data-set ;
    SET SAS-data-set1 SAS-data-set2 . . .;
<other SAS statements>
RUN;
```

Concatenating SAS Data Sets

SAS data sets

work.qtr1



Concatenating Data Sets – Example

I have two sections of my class, Morning and Afternoon, and want to combine their scores

```
Data allsections;  
Set morning afternoon;  
run;
```

Morning

Name	Score
Mary	75
Mark	82
Mike	68

allsections

Name	Score
Mary	75
Mark	82
Mike	68
Andy	78
Alice	85
Art	62

Afternoon

Name	Score
Andy	78
Alice	85
Art	62

Concatenating Data Sets - Details

To append (or concatenate) data sets:

- The variable names and data types should be the same in both data sets.
- You can append multiple data sets in a single SET statement.
- You may want to create a variable that identifies the source of each set of data. Do this in a separate data step prior to appending the data sets.

Creating an identifier variable

```
data morning;
input name $ score;
class = 'M';
datalines; ...
run;

data afternoon;
input name $ score;
class = 'A';
datalines; ...
run;

data allselections;
    set morning afternoon;
run;
```



Name	Score	Class
Alice	85	A
Art	62	A
Barry	82	M
Mary	75	M
Will	78	A
Zach	68	M

Merging Data Sets

General form of a DATA step match-merge:

```
DATA SAS-data-set;  
    MERGE SAS-data-sets;  
    BY BY-variable(s);  
    <other SAS statements>  
RUN;
```

Note: Data sets must be sorted on BY variable prior to merging.

Merging Data Sets - Example

I have midterm scores in one data set, and final scores in another and want to combine them

```
Data allscores;  
    Merge midterm final;  
    by Name;  
run;
```

Midterm

Name	Midscore
Wendy	32
Andy	38
John	27

allsections

Name	Midscore	Finalscore
Andy	38	82
John	27	91
Wendy	32	73

Final

Name	Finalscore
John	91
Wendy	73
Andy	82

Why Have Separate Datasets?

- Efficiency of Storage
 - Keep only the data you need in each data set
 - Example: Customer information is kept separate from order information
- Efficiency of Processing
 - Smaller datasets can be processed faster than larger datasets
 - Example: Reading in fewer variables, or sorting fewer records
- Different Data Sources
 - Appending and Merging allow you to combine information from separate sources
 - Example: Information on the same topic from two different researchers or two different websites

The RENAME= Data Set Option

General form of the RENAME= data set option:

```
SAS-data-set(RENAME=(old-name-1=new-name-1  
              old-name-2=new-name-2  
              .  
              .  
              .  
              old-name-n=new-name-n))
```

- When appending data sets, use the RENAME= option to create common variable names.
- When merging data sets, use the RENAME= option to create unique variable names.

RENAME: Appending Datasets

- Goal: append two datasets
- Problem: the variable names are different in the two datasets (e.g., test scores are called “score” in Morning section, and called “testscore” in Afternoon section).
- Solution: RENAME the testscore variable to score in the Afternoon section

```
data allsections;
    set morning
        afternoon (RENAME=(testscore=score)) ;
run;
```

RENAME: Merging Datasets

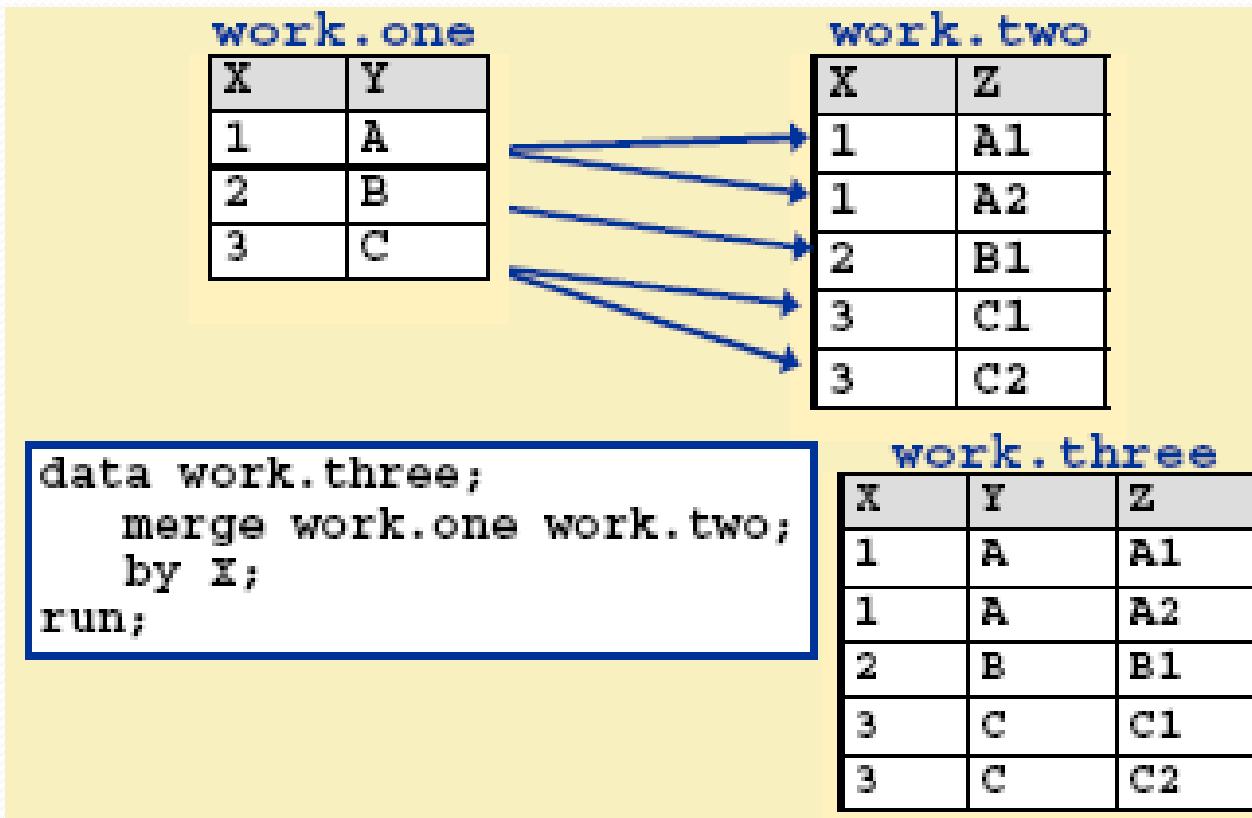
- Goal: merge two datasets
- Problem: the variable names are the same in the two datasets (e.g., midterm is called “score” in the Midterm dataset, and Final is called “score” in Final data).
- Solution: RENAME score variable

```
Data allscores;  
    Merge midterm (RENAME=(score=midterm))  
            final (RENAME=(score=final));  
    by Name;  
run;
```

Types of Merges

- Match Merging – MERGE and BY statements
 - The records from each data set with the **same** value of the (unique) BY variable are **linked**, and output as one record
 - If you omit the BY statement, the **first** record from each data set are output together as one record **without being linked** by a common variable
- One-to-many
 - unique BY values are in one data set and duplicate matching BY values are in the other data set.
- Many-to-many
 - duplicate matching BY values are in both data sets.

One-to-Many Merging



One-to-Many Merging

ia.allsales		
Month	Region	Sales
1	Europe	2118222
1	North America	3135765
2	Europe	1960034
2	North America	2926929

ia.allgoals	
Month	Goal
1	2127742
2	1920751
3	2125112

```
data ia.allcompare;
    merge ia.allsales ia.allgoals;
    by Month;
    Difference=Sales-Goal;
run;
```

ia.allcompare				
Month	Region	Sales	Goal	Difference
1	Europe	2118222	2127742	-9520
1	North America	3135765	2127742	1008023
2	Europe	1960034	1920751	39283
2	North America	2926929	1920751	1006178

Many-to-Many Merging

work.one

X	Y
1	A1
1	A2
2	B1
2	B2

work.two

X	Z
1	AA1
1	AA2
1	AA3
2	BB1
2	BB2

```
data work.three;
    merge work.one work.two;
    by X;
run;
```

work.three

X	Y	Z
1	A1	AA1
1	A2	AA2
1	A2	AA3
2	B1	BB1
2	B2	BB2

The IN= Data Set Option

Use the IN= option to create variables identifying which data sets contained the observation.

General form of the IN= data set option:

SAS-data-set(*IN=variable*)

Example:

```
Data allscores;
  Merge midterm (IN=InMidterm)
            final (IN=InFinal);
  by Name;
  if InMidterm and InFinal;
run;
```

Lookup Tables

- Data set variable contains ‘codes’
 - Males are coded as 1
 - Females are coded as 2
- Lookup table contains ‘labels’ that can be merged with ‘codes’

GenderLookup

GenderCode	GenderLabel
1	Male
2	Female

Parent – Child Tables

- Sales Order Form
 - Parent Table contains Order-level information
 - Order #, Order Date, Name, Phone Number, Shipping Address, Billing Address, Payment Method, etc
 - Child Table contains list of items ordered, quantity, unit cost
 - Order # appears on every record in Parent and Child tables
 - Tables are linked through One to Many merging using a key variable such as Order #

Scheduling Example

- The University maintains multiple data sets to schedule classes
 - A list of instructors and the courses they teach.
 - A list of students taking each course.
 - A list of classrooms and the courses that meet in them.

Students

- Variables:
 - StudentName
- Three data sets are provided
 - PSTAT130.txt
 - PSYCH118.txt
 - POLI125.txt

1	2
12345678901234567890	
John Thomas	
Elizabeth Smith	
Rajesh Krish	
Lily Yang	
Robert Williams	
Tracy Jones	
Cheryl Smith	
Alex Shepard	
Trinh Phan	
Lee Barrett	
Clark Johnson	
Jenny Page	
Mary Marcus	
Curt Forrest	
Andy Potts	

Instructors

- Variables:
 - InstructorName
 - AcademicRank
 - Salary
 - CourseName
 - FirstClassDate
- File saved as Instructors.txt

1	2	3	4	5
12345678901234567890123456789012345678901234567890	12345678901234567890123456789012345678901234567890	12345678901234567890123456789012345678901234567890	12345678901234567890123456789012345678901234567890	12345678901234567890123456789012345678901234567890
John Tukey	Assoc	\$56,000	PSTAT130	09/23/10
Sigmund Freud	Assoc	\$92,000	PSYCH118	09/24/10
Karl Marx	Asst	\$78,000	POLI125	09/27/10

Classrooms

- Variables
 - BldgName
 - RoomNumber
 - CourseName
 - Days
 - Time (read as a character variable)
- File saved as Classrooms.txt

1	2	3	4
123456789012345678901234567890123456789012345			
Phelps Hall	222	PSYCH118	T/TH 10:00 am
South Hall	518	PSTAT130	M/W/F 5:00 pm
Phelps Hall	126	POLI125	M/W 2:00 pm

Task - Input/Data Preparation

- Create three data steps to read in each list of students
 - Create a variable to store the CourseName for each dataset because only the file name indicates which class it is for
- Create a data step to read in Instructors data
- Create a data step to read in the Classrooms data
- Combine the student lists into a single data set of all students - call it AllStudents
- Combine the Instructor, Classrooms and AllStudent data sets so that each student is paired with his or her instructor and the room information for that class

Task – Create Roster

- Create a ‘Roster’ for each class showing the names of the students taking that class.
 - Separate the lists onto a page for each class.
 - Assign appropriate variable labels for CourseName and StudentName
 - Show only the CourseName at the top of each page, and the list of students names below.

----- Course Name=POLI125 -----

Student Name

Alex Shepard
Andy Potts
Cheryl Smith
Curt Forrest

Task – Output Class List

- Create a Class List for each Student showing details of the classes each is taking.
 - Include the variables below
 - Assign appropriate variable labels
 - Use an appropriate format for FirstClassDate
- Example on next slide

Class List Example

----- Student Name=Alex Shepard -----

Course Name	First Class Date	Instructor Name	Building Name	Room Number	Class Days	Class Time
PSTAT130	09/23/10	John Tukey	South Hall	518	M/W/F	5:00 pm
POLI125	09/27/10	Karl Marx	Phelps Hall	126	M/W	2:00 pm

----- Student Name=Andy Potts -----

Course Name	First Class Date	Instructor Name	Building Name	Room Number	Class Days	Class Time
PSTAT130	09/23/10	John Tukey	South Hall	518	M/W/F	5:00 pm
PSYCH118	09/24/10	Sigmund Freud	Phelps Hall	222	T/TH	10:00 am
POLI125	09/27/10	Karl Marx	Phelps Hall	126	M/W	2:00 pm

Tasks – Output

- Create a Master List of all Students whose instructor is an Associate Professor.
 - Include the variables in the example on the next page
 - Assign appropriate variable labels
 - Create a user-defined format and apply it to academic rank, assigning the label “Assistant Professor” to “Asst” and “Associate Professor” to “Assoc”
 - Use an appropriate format for Salary

Associate Professor Example

List of Students for Associate Professors

Student Name	Course Name	Instructor Name	Academic Rank	Salary
Alex Shepard	PSTAT130	John Tukey	Associate Professor	\$56,000
Andy Potts	PSTAT130	John Tukey	Associate Professor	\$56,000
Andy Potts	PSYCH118	Sigmund Freud	Associate Professor	\$92,000
Cheryl Smith	PSTAT130	John Tukey	Associate Professor	\$56,000
Clark Johnson	PSTAT130	John Tukey	Associate Professor	\$56,000
Clark Johnson	PSYCH118	Sigmund Freud	Associate Professor	\$92,000
Curt Forrest	PSTAT130	John Tukey	Associate Professor	\$56,000
Curt Forrest	PSYCH118	Sigmund Freud	Associate Professor	\$92,000
Elizabeth Smith	PSTAT130	John Tukey	Associate Professor	\$56,000
Elizabeth Smith	PSYCH118	Sigmund Freud	Associate Professor	\$92,000
Jenny Page	PSTAT130	John Tukey	Associate Professor	\$56,000
Jenny Page	PSYCH118	Sigmund Freud	Associate Professor	\$92,000
John Thomas	PSTAT130	John Tukey	Associate Professor	\$56,000
John Thomas	PSYCH118	Sigmund Freud	Associate Professor	\$92,000
Lee Barrett	PSTAT130	John Tukey	Associate Professor	\$56,000
Lee Barrett	PSYCH118	Sigmund Freud	Associate Professor	\$92,000

PSTAT 130

SAS Base Programming

Summer 2017

William Qiu

Department of Statistics and Applied Probability

UCSB

Class 9

Lecture Outline

- Summarizing Your Data
 - PROC MEANS
 - PROC FREQ
 - PROC TABULATE
 - PROC REPORT

Summarizing Data

In order to summarize a data set, what three things do you need to know?

Summarizing Data

In order to summarize a data set, what three things do you need to know?

- The Shape (Frequency Distribution, Outliers)
- The Center (Mean, Median, Mode)
- The Spread (Standard Deviation)

Procedures for Summarizing Data

- PROC MEANS – calculate and display simple summary statistics
- PROC FREQ – calculate and display frequency counts
- PROC TABULATE – flexible tool to calculate and display multi-dimensional tables with summary statistics
- PROC REPORT – flexible tool for creating listings and summary displays

PROC MEANS Output

Salary by Job Code						
The MEANS Procedure						
Analysis Variable : Salary						
Job Code	N Obs	N	Mean	Std Dev	Minimum	Maximum
FLTAT1	14	14	25642.86	2951.07	21000.00	30000.00
FLTAT2	18	18	35111.11	1906.30	32000.00	38000.00
FLTAT3	12	12	44250.00	2301.19	41000.00	48000.00
PILOT1	8	8	69500.00	2976.10	65000.00	73000.00
PILOT2	9	9	80111.11	3756.48	75000.00	86000.00
PILOT3	8	8	99875.00	7623.98	92000.00	112000.00

PROC FREQ Output

Distribution of Job Code Values

The FREQ Procedure

Job Code	Frequency	Percent	Cumulative Frequency	Cumulative Percent
FLTAT1	14	20.29	14	20.29
FLTAT2	18	26.09	32	46.38
FLTAT3	12	17.39	44	63.77
PILOT1	8	11.59	52	75.36
PILOT2	9	13.04	61	88.41
PILOT3	8	11.59	69	100.00

PROC TABULATE Output

Average Salary for Cary and Frankfurt

JobCode	Location		All
	CARY	FRANKFURT	
	Salary	Salary	
	Mean	Mean	
FLTAT1	\$26,200	\$25,000	\$25,667
FLTAT2	\$35,000	\$36,200	\$35,500
FLTAT3	\$43,400	\$44,667	\$43,875
All	\$34,882	\$34,583	\$34,759

PROC REPORT Output

Salary Analysis

Job Code	Home Base	Salary
FLTAT1	CARY	\$181,000
	FRANKFURT	\$100,000
	LONDON	\$128,000
FLTAT2	CARY	\$245,000
	FRANKFURT	\$181,000
	LONDON	\$206,000
FLTAT3	CARY	\$217,000
	FRANKFURT	\$134,000
	LONDON	\$180,000
PILOT1	CARY	\$211,000
	FRANKFURT	\$185,000
	LONDON	\$210,000
PILOT2	CARY	\$323,000
	FRANKFURT	\$240,000
	LONDON	\$158,000
PILOT3	CARY	\$300,000
	FRANKFURT	\$205,000
	LONDON	\$294,000
		=====
		\$3,598,000

Calculating Summary Statistics

PROC MEANS:

- calculates common **summary statistics**
- summarizes **numeric variables**
- **BY** and **CLASS** statements can be used to create summaries for **sub-groups**
- can create an **output data set** of summary statistics

Calculating Summary Statistics

- General form of a simple PROC MEANS step:

```
PROC MEANS DATA=SAS-data-set;
RUN;
```

- Example:

```
proc means data=ia.admit;
run;
```

PROC MEANS – Default Output

The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
Age	21	38.0476190	10.3124982	22.0000000	60.0000000
Date	21	14.5238095	9.1630729	1.0000000	31.0000000
Height	21	68.2380952	4.3116674	61.0000000	76.0000000
Weight	21	156.5238095	22.6398301	118.0000000	193.0000000
Fee	21	127.9500000	24.1524222	85.2000000	149.7500000

Calculating Summary Statistics

By default, PROC MEANS

- analyzes every **numeric variable** in the SAS data set
- prints the statistics **N**, **MEAN**, **STD**, **MIN**, and **MAX**
- excludes missing values before calculating statistics.

PROC MEANS – Options

- **VAR <variable list>** - selects variables to be summarized
- **BY <variable list>** - creates separate summaries for each BY group
- **CLASS <variable list>** - creates separate summaries for each CLASS group
- **OUTPUT out=sas data set** – creates an output data set containing summary statistics
- Statistical Keywords – option on PROC MEANS statement

Specifying Which Variables to Include

Use the VAR statement to select specific variables

```
proc means data=ia.admit;  
var age height weight;  
run;
```

PROC MEAN - Common Statistics

- MIN
- MAX
- RANGE
- MEAN
- MEDIAN
- STDDEV
- SUM
- N
- NMISS – the number of observations missing a value on each variable
- Confidence intervals, percentiles, and probability functions can also be requested – see MEANS procedure, statistic keywords in SAS Help

Specifying Summary Statistics

List Keywords for statistics as Options to the PROC MEANS statement

```
proc means data=ia.admit n mean stddev;  
var age height weight;  
run;
```

Analyzing Subgroups – BY Statement

Use the BY statement to request summaries for sub-groups

```
proc means data=ia.admit n mean stddev;  
var age height weight;  
by actlevel;  
output out=meansout;  
run;
```

Data MUST be sorted on the BY variable(s) first

BY Statement - Partial Output

The MEANS Procedure

ActLevel=HIGH -----

Variable	N	Mean	Std Dev
Age	7	34.2857143	7.5213980
Height	7	70.1428571	4.2201332
Weight	7	163.5714286	21.1412483
fffff	fffff	fffff	fffff

ActLevel=LOW -----

Variable	N	Mean	Std Dev
fffff	fffff	fffff	fffff
Age	7	39.2857143	14.0441481
Height	7	66.4285714	5.0284903
Weight	7	150.7142857	26.1897835

Analyzing Subgroups – CLASS Statement

Use the CLASS statement to request summaries for subgroups

```
proc means data=ia.admit n mean stddev;  
class actlevel;  
output out=meansout;  
run;
```

Data DO NOT need to be sorted on the CLASS variable(s) first

CLASS Statement - Output

PROC MEANS

With a CLASS Variable

The MEANS Procedure

Act	N				
Level	Obs	Variable	N	Mean	Std Dev
			fffff	fffff	fffff
HIGH	7	Age	7	34.2857143	7.5213980
		Height	7	70.1428571	4.2201332
		Weight	7	163.5714286	21.1412483
LOW	7	Age	7	39.2857143	14.0441481
		Height	7	66.4285714	5.0284903
		Weight	7	150.7142857	26.1897835
MOD	7	Age	7	40.5714286	8.6575043
		Height	7	68.1428571	3.2877840
		Weight	7	155.2857143	21.8305160
			fffff	fffff	fffff

Limit Number of Decimals

- Use the MAXDEC option in the PROC MEANS Statement to limit the number of decimal places in the summary statistics

```
proc means data=ia.admit n mean stddev maxdec=2;  
var age height weight;  
run;
```

The MEANS Procedure				
Variable	N	Mean	Std Dev	
Age	21	38.05	10.31	
Height	21	68.24	4.31	
Weight	21	156.52	22.64	

Displays 2 decimal places

Class Exercise

- Open the **heart** dataset
- Inspect the dataset using PROC CONTENTS and PROC PRINT
- Run PROC MEANS using default options
- Now limit the statistics to one decimal place
- Run PROC MEANS for the **Arterial** and **Cardiac** variables
- Run PROC MEANS using **Sex** as a Class variable

Creating a Frequency Report

The FREQ procedure can be used to summarize the frequency of values within the data set. In other words, it creates frequency distributions.

Creating a Frequency Report

- General form of a simple PROC FREQ step:

```
PROC FREQ DATA=SAS-data-set;  
RUN;
```

- Example:

```
proc freq data=ia.admit;  
run;
```

Creating a Frequency Report

By default, PROC FREQ

- analyzes **every variable** in the SAS data set
- displays each distinct data value
- calculates the **number of observations** in which each data value appears (and the corresponding **relative and cumulative percentages**)
- indicates for each variable how many observations have missing values.

PROC FREQ – TABLES Statement

Use the TABLES statement to select variable and to specify the type of frequency report.

General form of a PROC FREQ step with a TABLES statement:

```
PROC FREQ DATA=SAS-data-set;
    TABLES variable list;
RUN;
```

Creating a Frequency Report

```
proc freq data=ia.crew;
  tables JobCode;
  title 'Distribution of Job Code Values';
run;
```

Displays a frequency table
for the variable, Jobcode

Distribution of Job Code Values

The FREQ Procedure

Job Code	Frequency	Percent	Cumulative Frequency	Cumulative Percent
FLTAT1	14	20.29	14	20.29
FLTAT2	18	26.09	32	46.38
FLTAT3	12	17.39	44	63.77
PILOT1	8	11.59	52	75.36
PILOT2	9	13.04	61	88.41
PILOT3	8	11.59	69	100.00

The table lists each value of JobCode,
and its frequency

Analyzing Categories of Values

Use the FORMAT Statement to analyze the frequency of observations within User-Defined Categories

```
proc format;  
value $codefmt  
    'FLTAT1'-'FLTAT3'='Flight Attendant'  
    'PILOT1'-'PILOT3'='Pilot';  
run;  
proc freq data = ia.crew;  
    format JobCode $codefmt.;  
    tables JobCode;  
run;
```

Analyzing Categories of Values

Distribution of Job Code Values

The FREQ Procedure

JobCode	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Flight Attendant	44	63.77	44	63.77
Pilot	25	36.23	69	100.00

Crosstabular Frequency Reports

General Form of two-way tables, known as crosstabs:

```
PROC FREQ DATA=SAS-data-set;  
    TABLES variable1 * variable 2;  
RUN;
```

- Two-way tables categorize observations on the combination of two sets of categories (e.g., Male Pilots, Female Pilots, Male Flight Attendants and Female Flight Attendants)

```
proc format;
value $codefmt
  'FLTAT1'-'FLTAT3'='Flight Attendant'
  'PILOT1'-'PILOT3'='Pilot';
value money
  low-<25000 ='Less than 25,000'
  25000-50000='25,000 to 50,000'
  50000<-high='More than 50,000';
run;
proc freq data=ia.crew;
  tables JobCode*Salary;
  format JobCode $codefmt. Salary money. ;
  title 'Salary Distribution by Job Codes';
run;
```

Crosstabular Output

Salary Distribution by Job Codes

The FREQ Procedure

Table of JobCode by Salary

JobCode	Salary
Frequency	,
Percent	,
Row Pct	,
Col Pct	,Less tha,25,000 t,More tha, Total
	,n 25,000,o 50,000,n 50,000,
	ffffffffff^fffff^ffff^ffff^ffff^ffff^
Flight Attendant	, 5 , 39 , 0 , 44
	, 7.25 , 56.52 , 0.00 , 63.77
	, 11.36 , 88.64 , 0.00 ,
	, 100.00 , 100.00 , 0.00 ,
	ffffffffff^fffff^ffff^ffff^ffff^
Pilot	, 0 , 0 , 25 , 25
	, 0.00 , 0.00 , 36.23 , 36.23
	, 0.00 , 0.00 , 100.00 ,
	, 0.00 , 0.00 , 100.00 ,
	ffffffffff^fffff^ffff^ffff^ffff^
Total	5 39 25 69
	7.25 56.52 36.23 100.00

PSTAT 130

SAS Base Programming

Summer 2017

William Qiu

Department of Statistics and Applied Probability

UCSB

Lecture 10

Lecture Outline

- Summarizing Your Data – con't
 - PROC MEANS
 - PROC FREQ
 - PROC TABULATE
 - PROC REPORT

Procedures for Summarizing Data

- PROC MEANS – calculate and display simple summary statistics
- PROC FREQ – calculate and display frequency counts
- PROC TABULATE – flexible tool to calculate and display multi-dimensional tables with summary statistics
- PROC REPORT – flexible tool for creating listings and summary displays

PROC MEANS Output

Salary by Job Code						
The MEANS Procedure						
Analysis Variable : Salary						
Job Code	N Obs	N	Mean	Std Dev	Minimum	Maximum
FLTAT1	14	14	25642.86	2951.07	21000.00	30000.00
FLTAT2	18	18	35111.11	1906.30	32000.00	38000.00
FLTAT3	12	12	44250.00	2301.19	41000.00	48000.00
PILOT1	8	8	69500.00	2976.10	65000.00	73000.00
PILOT2	9	9	80111.11	3756.48	75000.00	86000.00
PILOT3	8	8	99875.00	7623.98	92000.00	112000.00

PROC FREQ Output

Distribution of Job Code Values

The FREQ Procedure

Job Code	Frequency	Percent	Cumulative Frequency	Cumulative Percent
FLTAT1	14	20.29	14	20.29
FLTAT2	18	26.09	32	46.38
FLTAT3	12	17.39	44	63.77
PILOT1	8	11.59	52	75.36
PILOT2	9	13.04	61	88.41
PILOT3	8	11.59	69	100.00

PROC TABULATE Output

Average Salary for Cary and Frankfurt

JobCode	Location		All
	CARY	FRANKFURT	
	Salary	Salary	
	Mean	Mean	
FLTAT1	\$26,200	\$25,000	\$25,667
FLTAT2	\$35,000	\$36,200	\$35,500
FLTAT3	\$43,400	\$44,667	\$43,875
All	\$34,882	\$34,583	\$34,759

PROC REPORT Output

Salary Analysis

Job Code	Home Base	Salary
FLTAT1	CARY	\$181,000
	FRANKFURT	\$100,000
	LONDON	\$128,000
FLTAT2	CARY	\$245,000
	FRANKFURT	\$181,000
	LONDON	\$206,000
FLTAT3	CARY	\$217,000
	FRANKFURT	\$134,000
	LONDON	\$180,000
PILOT1	CARY	\$211,000
	FRANKFURT	\$185,000
	LONDON	\$210,000
PILOT2	CARY	\$323,000
	FRANKFURT	\$240,000
	LONDON	\$158,000
PILOT3	CARY	\$300,000
	FRANKFURT	\$205,000
	LONDON	\$294,000
		=====
		\$3,598,000

The TABULATE Procedure

The report writing features of PROC TABULATE include

- control of table construction
- differentiating between **classification variables** and **analysis variables**
- specifying **statistics**
- **formatting** of values
- labeling variables and statistics.

PROC TABULATE syntax

General form of a PROC TABULATE step

```
PROC TABULATE DATA=SAS-data-set <options>;
  CLASS class-variables;
  VAR analysis-variables;
  TABLE page-expression,
         row-expression,
         column-expression </ option(s)>;
RUN;
```

Specifying Classification Variables

```
PROC TABULATE DATA=SAS-data-set <options>;
  CLASS class-variables;
  VAR analysis-variables;
  TABLE page-expression,
        row-expression,
        column-expression </ option(s)>;
RUN;
```

- Class variables are used to define subgroups on one or more dimensions

Specifying Analysis Variables

```
PROC TABULATE DATA=SAS-data-set <options>;
  CLASS class variables;
  VAR analysis-variables;
  TABLE page-expression,
        row-expression,
        column-expression </ option(s)>;
RUN;
```

- Summary statistics (e.g., means) are calculated for the VAR variables

Specifying Table Structure

```
PROC TABULATE DATA=SAS-data-set <options>;
  CLASS class-variables;
  VAR analysis-variables;
  TABLE page-expression,
         row-expression,
         column-expression </ option(s)>;
RUN;
```

- The TABLE statement specifies the format of the table

Using Only Class Variables

```
title 'Flight Attendant Counts by Location';
proc tabulate data=ia.fltat;
    class Location;
    table Location;
run;
```

Flight Attendant Counts by Location

Obtaining a Total

```
proc tabulate data=ia.flflat;
    class Location;
    table Location All
run;
```

Blank Operator between Location and All concatenates information

Flight Attendant Counts by Location

„ffffffffff...ffffffffff...ffffffffff...ffffffffff...ffffffffff...ffffffffff...ffffffffff+
, Location , ,
#ffffffffff...ffffffffff...ffffffffff%„ ,
, CARY , FRANKFURT , LONDON , All
#ffffffffff^ffffffffff^ffffffffff^ffffffffff^ffffffffff%„
, N , N , N , N ,
#ffffffffff^ffffffffff^ffffffffff^ffffffffff%„
, 17.00, 12.00, 15.00, 44.00,
Šffffffffff<ffffffffff<ffffffffff<ffffffffffŒ

Two-Dimensional Tables

```
title2 'by JobCode';
proc tabulate data=ia.fltat;
  class Location JobCode;
  table JobCode, Location;
run;
```

Row Dimension

Comma operator
moves to a new
dimension

Column Dimension

Two-Dimensional Tables

by JobCode

Row Dimension

Flight Attendant Counts by Location

Column Dimension

JobCode	CITY	LOC	FLTAT1	FLTAT2	FLTAT3
,	,	,	5.00	7.00	5.00
,	CARY	, FRANKFURT	4.00	5.00	3.00
,					
,	N	, N	5.00	6.00	4.00
,					
JobCode	,	,	,	,	,
#fffff	fffff	fffff	fffff	fffff	fffff
,FLTAT1	,	5.00	4.00	5.00	
#fffff	fffff	fffff	fffff	fffff	fffff
,FLTAT2	,	7.00	5.00	6.00	
#fffff	fffff	fffff	fffff	fffff	fffff
,FLTAT3	,	5.00	3.00	4.00	
Šfffff	fffff	fffff	fffff	fffff	fffff

Subsetting the Data

```
title 'Counts for Cary and Frankfurt';
proc tabulate data=ia.fltat;
  where Location in ('CARY', 'FRANKFURT');
  class Location JobCode;
  table JobCode, Location;
run;
```

Subsetting the Data

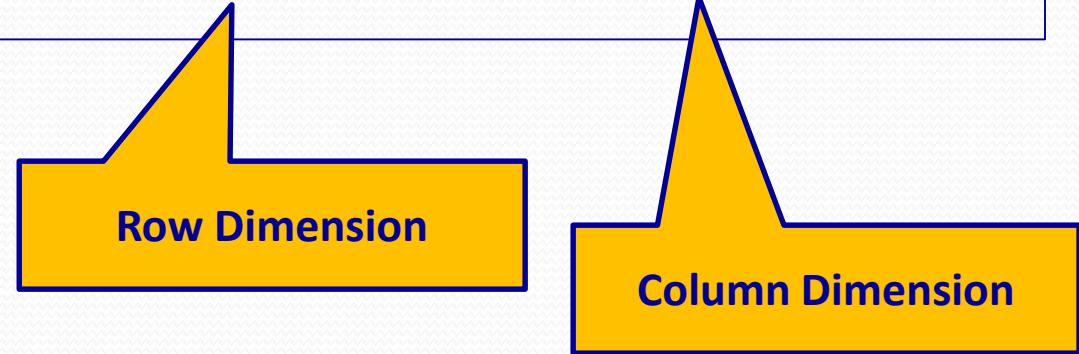
Flight Attendant Counts by Location

by JobCode

```
,fffff...fffff+  
,      , Location ,  
,      #fffff...fff%o  
,      , CARY , FRANKFURT ,  
,      #fffff...fff%o  
,      , N , N ,  
#fffff...fff%o  
,JobCode      , , ,  
#fffff...fff%o      , , ,  
,FLTAT1      , 5.00, 4.00,  
#fffff...fff%o  
,FLTAT2      , 7.00, 5.00,  
#fffff...fff%o  
,FLTAT3      , 5.00, 3.00,  
$fffff...fffŒ
```

Two-Dimensional Tables

```
proc tabulate data=ia.fltat;  
    where Location in ('CARY', 'FRANKFURT');  
    class Location JobCode;  
    table JobCode all, Location all;  
run;
```



Two-Dimensional Table

Total Salary for Cary and Frankfurt

All in Row Dimension

```
,,ffffffffff...ffff...ffff...ffff...ffff...ffff...ffff...ffff...ffff+  
, , Location , ,  
#ffff...ffff...ffff...ffff%o ,  
, CARY , FRANKFURT , All ,  
#ffff...ffff...ffff^ffff...ffff...ffff%o  
, N , N , N ,  
#ffff...ffff...ffff...ffff^ffff...ffff...ffff...ffff...ffff%o  
,JobCode , , , ,  
#ffff...ffff...ffff%o , , , ,  
,FLTAT1 , 5.00, 4.00, 9.00,  
#ffff...ffff...ffff...ffff^ffff...ffff...ffff...ffff%o  
,FLTAT2 , 7.00, 5.00, 12.00,  
#ffff...ffff...ffff...ffff^ffff...ffff...ffff...ffff%o  
,FLTAT3 , 5.00, 3.00, 8.00,  
#ffff...ffff...ffff...ffff^ffff...ffff...ffff...ffff%o  
,All , 17.00, 12.00, 29.00,  
Šffff...ffff...ffff...ffff<ffff...ffff...ffff<ffff...ffff...ffff<ffff...ffff...ffffŒ
```

All in Column Dimension

Using Analysis Variables

```
title 'Total Salary for Cary and Frankfurt';
proc tabulate data=ia.fltat;
    where Location in ('CARY', 'FRANKFURT');
    class Location JobCode;
    var Salary;
    table JobCode, Location*Salary;
run;
```

Using Analysis Variables

Total Salary for Cary and Frankfurt

```
„fffff...fffff...fffff+  
,          , Location ,  
,          #fffff...fffff%o  
,          , CARY , FRANKFURT ,  
,          #fffff...fffff^fffff%o  
,          , Salary , Salary ,  
,          #fffff...fffff^fffff%o  
,          , Sum , Sum ,  
#fffff...fffff^fffff^fffff%o  
,JobCode      , , ,  
#fffff...fffff%o      , ,  
,FLTAT1      , 131000.00, 100000.00,  
#fffff...fffff^fffff^fffff%o  
,FLTAT2      , 245000.00, 181000.00,  
#fffff...fffff^fffff^fffff%o  
,FLTAT3      , 217000.00, 134000.00,  
$fffff...fffff<fffff...fffff<fffff...fffffŒ
```

Salary within
each Location



Formatting the Statistic

```
proc tabulate data=ia.flat format=dollar12.
    where Location in ('CARY', 'FRANKFURT');
    class Location JobCode;
    var Salary;
    table JobCode, Location*Salary;
run;
```

Formatting a Statistic

Total Salary for Cary and Frankfurt

```
„fffff...fffff+  
,      , Location ,  
,      , #fffff...fffff%  
,      , CARY , FRANKFURT ,  
,      , #fffff...fffff%  
,      , Salary , Salary ,  
,      , #fffff...fffff%  
,      , Sum , Sum ,  
#fffff...fffff%  
,JobCode , , ,  
#fffff...fffff%  
,FLTAT1 , $131,000, $100,000,  
#fffff...fffff%  
,FLTAT2 , $245,000, $181,000,  
#fffff...fffff%  
,FLTAT3 , $217,000, $134,000,  
$fffff...fffffŒ
```

FORMAT= Option
changes default
format for all cells

Specifying a Statistic

```
title 'Average Salary for Cary and Frankfurt';
proc tabulate data=ia.flflat format=dollar12. ;
  where Location in ('CARY', 'FRANKFURT') ;
  class Location JobCode;
  var Salary;
  table JobCode, Location*Salary*mean;
run;
```

Specifying a Statistic

Average Salary for Cary and Frankfurt

, , Location ,
, , CARY , FRANKFURT ,
, , Salary , Salary ,
, , Mean , Mean ,
#fffffffffffff^ffffffffffff%o
,JobCode , , ,
#fffffffffffff^ffffffffffff%o
,FLTAT1 , \$26,200, \$25,000,
#fffffffffffff^ffffffffffff%o
,FLTAT2 , \$35,000, \$36,200,
#fffffffffffff^ffffffffffff%o
,FLTAT3 , \$43,400, \$44,667,
Šfffffffffffff^ffffffffffffCE

MEAN statistic

ALL with Analysis Variable

General form for generating overall information when using an analysis variable:

ALL**analysis-variable**statistic keyword

```
proc tabulate data=ia.fltat format=dollar12. ;
  where Location in ('CARY', 'FRANKFURT') ;
  class Location JobCode ;
  var Salary ;
  table JobCode all,
        Location*Salary*mean all*Salary*mean ;
run ;
```

```

proc tabulate data=ia.fltat format=dollar12. ;
    where Location in ('CARY', 'FRANKFURT') ;
    class Location JobCode ;
    var Salary ;
    table JobCode all ,
        Location*Salary*mean all*Salary*mean ;
run;

```

„fffff...fffff...fffff...fffff...fffff...fffff...fffff+
 , , Location , ,
 , #fffff...fffff...fffff%o , ,
 , , CARY , FRANKFURT , All ,
 , #fffff...fffff^fffff...fffff%o
 , , Salary , Salary , Salary ,
 , #fffff...fffff^fffff...fffff^fffff...fffff%o
 , , Mean , Mean , Mean ,
 #fffff...fffff^fffff...fffff^fffff...fffff^fffff...fffff%o
 ,JobCode , , , ,
 #fffff...fffff...fffff%o , , , ,
 ,FLTAT1 , \$26,200, \$25,000, \$25,667,
 #fffff...fffff...fffff^fffff...fffff^fffff...fffff%o
 ,FLTAT2 , \$35,000, \$36,200, \$35,500,
 #fffff...fffff...fffff^fffff...fffff^fffff...fffff%o
 ,FLTAT3 , \$43,400, \$44,667, \$43,875,
 #fffff...fffff^fffff...fffff^fffff...fffff^fffff%o
 All , \$34,882, \$34,583, \$34,759,
 \$fffff...fffff<fffff...fffff<fffff...fffff<fffff...fffffŒ



```
proc tabulate data=ia.fltat format=dollar12. ;
    where Location in ('CARY', 'FRANKFURT');
    class Location JobCode;
    var Salary;
    table JobCode all,
        Location*Salary*mean all*Salary*mean;
run;
```

Row Dimension		Location				Salary within each Location				All	
,	,	#fffff				Salary	Salary	Salary	Salary	All	,
,	,	#fffff				#fffff	fffff	fffff	fffff	fffff	%oo
,	,					, Salary	, Salary	, Salary	, Salary		
,	,	#fffff				#fffff	fffff	fffff	fffff	#fffff	%oo
,	,									Mean	,
,	,	#fffff				#fffff	fffff	fffff	fffff	#fffff	%oo
,	,									Mean	,
,	,	#fffff				#fffff	fffff	fffff	fffff	#fffff	%oo
JobCode	,		,	,	,	,	,	,	,		
#fffff											
,FLTAT1	,	\$26,200,	\$25,000,	\$25,667,							
#fffff						#fffff	fffff	fffff	fffff	#fffff	%oo
,FLTAT2	,	\$35,000,	\$36,200,	\$35,500,							
#fffff						#fffff	fffff	fffff	fffff	#fffff	%oo
,FLTAT3	,	\$43,400,	\$44,667,	\$43,875,							
#fffff						#fffff	fffff	fffff	fffff	#fffff	%oo
All	,	\$34,882,	\$34,583,	\$34,759,							
#fffff						#fffff	fffff	fffff	fffff	#fffff	C

All in Row Dimension

Column Dimension

umn
Dimension

FORMAT= option

The REPORT Procedure

PROC REPORT enables you to

- create **listing** reports
- create **summary** reports
- enhance reports
- request separate **subtotals** and **grand totals**
- generate reports in an **interactive** point-and-click or programming environments.

PROC REPORT versus PROC PRINT

FEATURE	REPORT	PRINT
Detail Report	Yes	Yes
Summary Report	Yes	No
Crosstabular Report	Yes	No
Grand Totals	Yes	Yes
Subtotals	Yes	Yes, but not without Grand Total
Labels used automatically	Yes	No
Sort data for report	Yes	No

Creating a Listing Report

General form of a simple PROC REPORT step:

```
PROC REPORT DATA=SAS-data-set <options>;
RUN;
```

Example:

```
proc report data=ia.admit nowindow;
run;
```

Note: the NoWindow option suppresses the interactive report writing function.

PROC REPORT Default Output

The SAS System 10:23 Monday, November 7, 2011 3

Act								
ID	Name	Sex	Age	Date	Height	Weight	Level	Fee
2458	Murray, W	M	27	1	72	168	HIGH	85.20
2462	Almers, C	F	34	3	66	152	HIGH	124.80
2501	Bonaventure, T	F	31	17	61	123	LOW	149.75
2523	Johnson, R	F	43	31	63	137	MOD	149.75
2539	LaMance, K	M	51	4	71	158	LOW	124.80
2544	Jones, M	M	29	6	76	193	HIGH	124.80
2552	Reberson, P	F	32	9	67	151	MOD	149.75
2555	King, E	M	35	13	70	173	MOD	149.75
2563	Pitts, D	M	34	22	73	154	LOW	124.80
2568	Eberhardt, S	F	49	27	64	172	LOW	124.80
2571	Nunnelly, A	F	44	19	66	140	HIGH	149.75
2572	Oberon, M	F	28	17	62	118	LOW	85.20
2574	Peterson, V	M	30	6	69	147	MOD	149.75
2575	Quigley, M	F	40	8	69	163	HIGH	124.80
2578	Cameron, L	M	47	5	72	173	MOD	124.80
2579	Underwood, K	M	60	22	71	191	LOW	149.75
2584	Takahashi, Y	F	43	29	65	123	MOD	124.80
2586	Derber, B	M	25	23	75	188	HIGH	85.20
2588	Ivan, H	F	22	20	63	139	LOW	85.20
2589	Wilcox, E	F	41	16	67	141	HIGH	149.75
2595	Warren, C	M	54	7	71	183	MOD	149.75

The REPORT Procedure

The default listing displays

- each **data value** as it is stored in the data set, or **formatted value** if a format is stored with the data
- **variable names** or **labels** as report column headings
- a **default width** for the report columns
- character values left-justified
- numeric values right-justified
- observations in the **order in which they are stored** in the data set.

Printing Selected Variables

General form of the COLUMN statement:

```
COLUMN SAS-variables;
```

Sample Listing Report

```
title 'Salary Analysis';
proc report data=ia.crew nowd;
    column JobCode Location Salary;
run;
```

Partial
SAS
Output

JobCode	Location	Salary
PILOT1	LONDON	72000
FLTAT3	CARY	41000
PILOT2	FRANKFURT	81000
PILOT2	FRANKFURT	83000
FLTAT2	LONDON	36000
PILOT1	LONDON	65000
FLTAT2	FRANKFURT	35000
FLTAT2	FRANKFURT	38000
FLTAT1	LONDON	28000
FLTAT3	LONDON	44000
FLTAT2	CARY	37000

The DEFINE Statement

You can enhance the report by using **DEFINE** statements to

- define how each **variable** is used in the report
- assign **formats** to variables
- specify report **column headers** and **column widths**
- change the **order** of the **rows** in the report.

The DEFINE Statement - Details

- Character variables are used as Display variables
- Numeric variabeles are used as Analysis variables
- **FORMAT=** option assigns format to a variable
- **WIDTH=** option controls width of report column
- **'report-column-header'** – defines the column header for a variable

Enhanced Listing Report

```
proc report data=ia.crew nowd;
    column JobCode Location Salary;
    define JobCode / width=8 'Job Code';
    define Location / 'Home Base';
    define Salary / format=dollar10. ;
run;
```

Partial
SAS
Output

Job Code	Home Base	Salary
PILOT1	LONDON	\$72,000
FLTAT3	CARY	\$41,000
PILOT2	FRANKFURT	\$81,000
PILOT2	FRANKFURT	\$83,000
FLTAT2	LONDON	\$36,000
PILOT1	LONDON	\$65,000
FLTAT2	FRANKFURT	\$35,000
FLTAT2	FRANKFURT	\$38,000
FLTAT1	LONDON	\$28,000
FLTAT3	LONDON	\$44,000

ORDER Usage Type

- **ORDER** keyword identifies variable used to order rows of report

```
proc report data=ia.crew nowd;
  column JobCode Location Salary;
  define JobCode / order width=8 'Job Code';
  define Location / 'Home Base';
  define Salary / format=dollar10. ;
run;
```

ORDER Usage Type

Partial
SAS
Output

Job Code	Home Base	Salary
FLTAT1	LONDON	\$28,000
	FRANKFURT	\$25,000
	CARY	\$23,000
	CARY	\$21,000
	CARY	\$28,000
	FRANKFURT	\$22,000
	LONDON	\$29,000
FLTAT2	LONDON	\$36,000
	FRANKFURT	\$35,000
	FRANKFURT	\$38,000
	CARY	\$37,000
	CARY	\$34,000
	LONDON	\$34,000
	CARY	\$36,000
FLTAT3	CARY	\$41,000
	LONDON	\$44,000
	FRANKFURT	\$48,000
	FRANKFURT	\$45,000
	CARY	\$44,000

Individual Values
(not Summary Stats)

Defining Group Variables

Use the REPORT procedure to create a **summary** report by defining variables as **group** variables.

All observations whose group variables have the same values are **collapsed** into a single row in the report.

Defining Group Variables

If you have a group variable, there must be no display or order variables.

- **Group** variables produce **summary** reports (observations collapsed into groups).
- **Display** and **order** variables produce **listing** reports (one row for each observation).

Defining Analysis Variables

Default usage for **numeric** variables is **ANALYSIS** with a default statistic of **SUM**.

- If the report contains group variables, the report displays the **sum** of the **numeric variables' values** for each **group**.
- If the report contains at least one display or order variable and no group variables, the report lists **all of the values** of the numeric variable.
- If the report contains only numeric variables, the report displays **grand totals** for the numeric variables.

Defining Analysis Variables

- Selected statistics include:

SUM	sum (default)
N	number of non-missing values
MEAN	Average
MAX	Maximum Value
MIN	Minumum Value

- Example:

```
define Salary / mean format=dollar10. ;
```

Summarizing the Data

Use the GROUP usage in the DEFINE statement to specify the variables that define groups.

```
proc report data=ia.crew nowd;
  column JobCode Location Salary;
  define JobCode / group width=8 'Job Code';
  define Location / group 'Home Base';
  define Salary / analysis mean format=dollar10. ;
run;
```

Summarizing the Data

Salary Analysis		
Job Code	Home Base	Salary
FLTAT1	CARY	\$26,200
	FRANKFURT	\$25,000
	LONDON	\$25,600
FLTAT2	CARY	\$35,000
	FRANKFURT	\$36,200
	LONDON	\$34,333
FLTAT3	CARY	\$43,400
	FRANKFURT	\$44,667
	LONDON	\$45,000
PILOT1	CARY	\$70,333
	FRANKFURT	\$67,500
	LONDON	\$70,000
PILOT2	CARY	\$80,750
	FRANKFURT	\$80,000
	LONDON	\$79,000
PILOT3	CARY	\$100,000
	FRANKFURT	\$102,500
	LONDON	\$98,000

Mean Salary for Flight Attendant 1's at London Base

Printing Grand Totals

General form of the RBREAK statement:

```
RBREAK BEFORE|AFTER <options>;
```

Selected options:

SUMMARIZE	Prints the total.
OL	Prints a single line above the total.
DOL	Prints a double line above the total.
UL	Prints a single line below the total.
DUL	Prints a double line below the total.

The BREAK and RBREAK Statements

Use the BREAK statement to display a total at the end of a Group. Use the RBREAK statement to display the grand total at the bottom of the report.

```
proc report data=ia.crew nowd;
    column JobCode Location Salary;
    define JobCode / group width=8 'Job Code';
    define Location / group 'Home Base';
    define Salary / format=dollar10. ;
    rbreak after / summarize dol;
run;
```

The RBREAK Statement

Job Code	Home Base	Salary
FLTAT1	CARY	\$131,000
	FRANKFURT	\$100,000
	LONDON	\$128,000
FLTAT2	CARY	\$245,000
	FRANKFURT	\$181,000
	LONDON	\$206,000
FLTAT3	CARY	\$217,000
	FRANKFURT	\$134,000
	LONDON	\$180,000
PILOT1	CARY	\$211,000
	FRANKFURT	\$135,000
	LONDON	\$210,000
PILOT2	CARY	\$323,000
	FRANKFURT	\$240,000
	LONDON	\$158,000
PILOT3	CARY	\$300,000
	FRANKFURT	\$205,000
	LONDON	\$294,000

DOL - Double Overline

RBREAK (Grand Total)

Enhancing the Report

```
proc report data=ia.crew nowd headline headskip;  
    column JobCode Location Salary;  
    define JobCode / group width=8 'Job Code';  
    define Location / group 'Home Base';  
    define Salary / format=dollar10.;  
    rbreak after / summarize dol;  
run;
```

Enhancing the Report

Job Code	Home Base	Salary
fffff	fffff	fffff
FLTAT1	CARY	\$131,000
	FRANKFURT	\$100,000
	LONDON	\$128,000
FLTAT2	CARY	\$245,000
	FRANKFURT	\$181,000
	LONDON	\$206,000
FLTAT3	CARY	\$217,000
	FRANKFURT	\$134,000
	LONDON	\$180,000
PILOT1	CARY	\$211,000
	FRANKFURT	\$135,000
	LONDON	\$210,000
PILOT2	CARY	\$323,000
	FRANKFURT	\$240,000
	LONDON	\$158,000
PILOT3	CARY	\$300,000
	FRANKFURT	\$205,000
	LONDON	\$294,000
=====		
		\$3,598,000

Head Line

Head Skip
(Blank Line)

PSTAT 130

SAS Base Programming

Summer 2017

William Qiu

Department of Statistics and Applied Probability

UCSB

Lecture 11

PROC TABULATE vs PROC REPORT

FEATURE	REPORT	TABULATE
Detail Report	Yes	No
Summary Report	Yes	Yes
Crosstabular Report	Yes	Yes
Grand Totals	Yes	Yes
Dividing Lines	Yes	Yes
Labels used automatically	Yes	Yes
Ability to create computed columns	Yes	No

Lecture Outline

- Producing HTML output using the Output Deliver System
- Summarizing data with graphs
 - Bar Charts – vertical and horizontal
 - Pie Charts
 - Graphing averages and sums
 - Graphing options

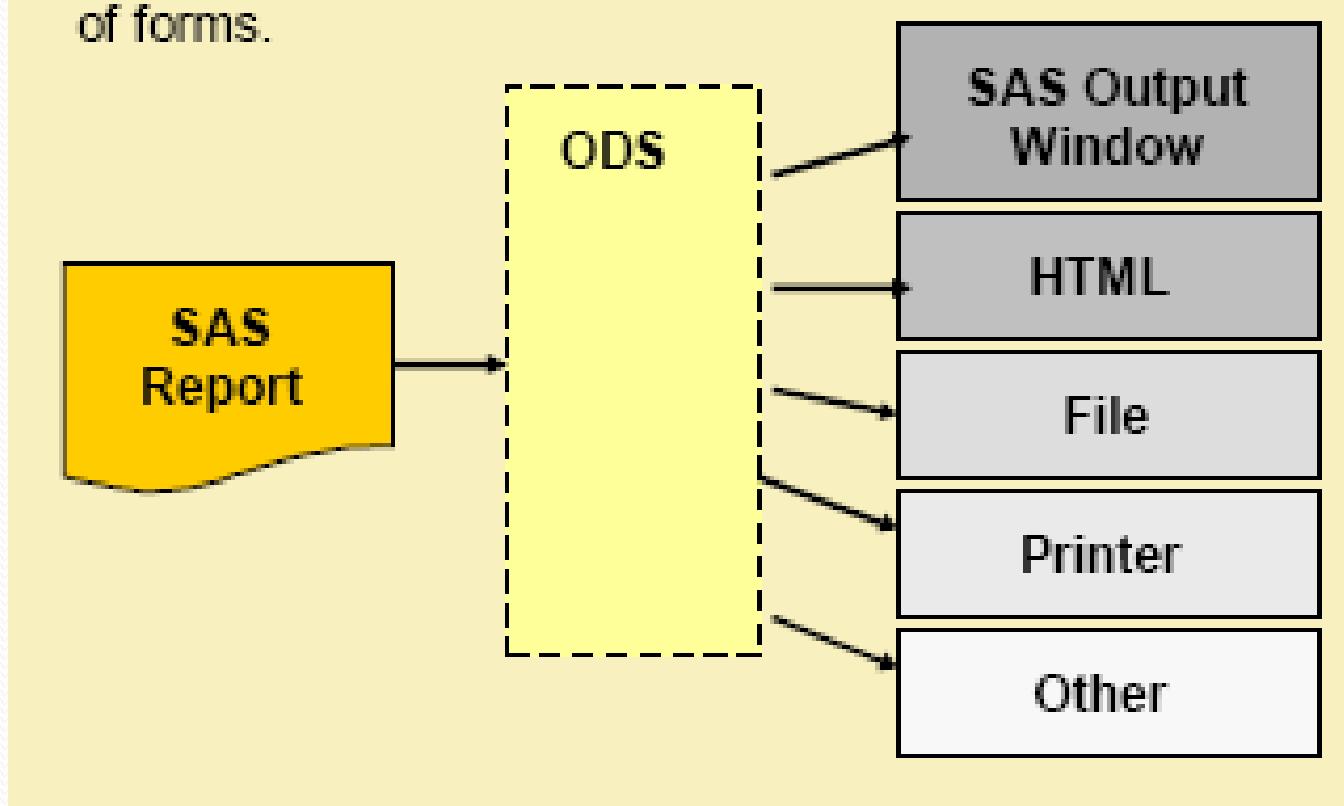
HTML Output

New to SAS Version 9.3

- Output can be directed to
 - the Output Window (in text format) or
 - the Results Viewer (in HTML format) or both
- Select using the menu command:
Tools | Options | Preferences | Results

The Output Delivery System

ODS statements enable you to create output in a variety of forms.



Generating an HTML File

The ODS **HTML** statement **opens**, **closes**, and **manages** the HTML destination.

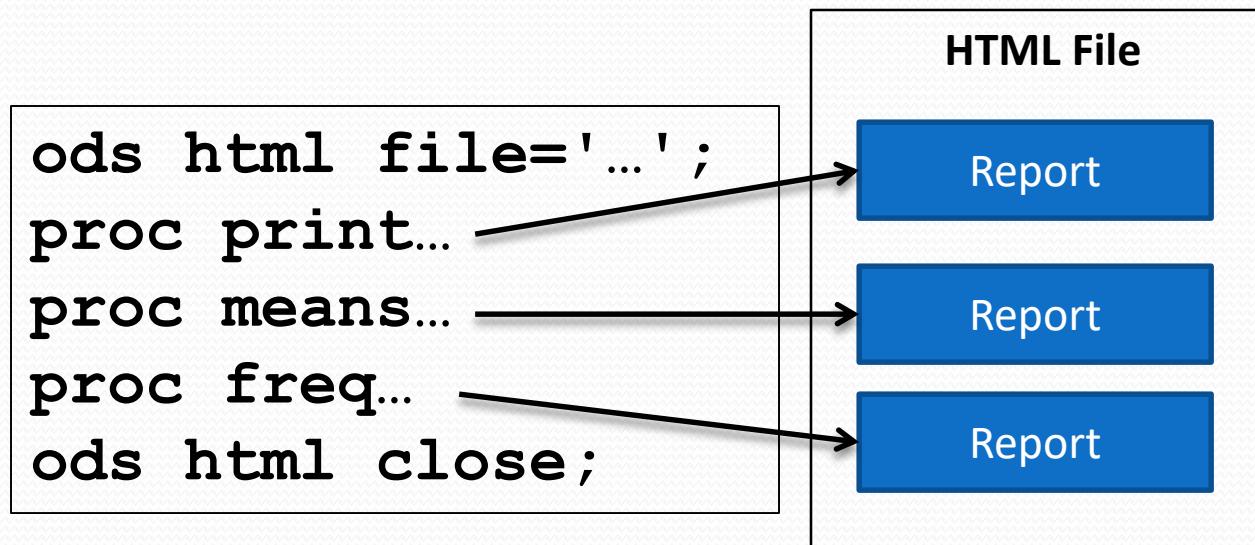
General form of the ODS HTML statement:

```
ODS HTML FILE='HTML-file-specification' <options>;
SAS code that generates output
ODS HTML CLOSE;
```

Generating an HTML File

Output is directed to the specified HTML file until you

- close the HTML destination
- specify another destination file.



Creating an HTML Report

1. Open an HTML destination for the output.
2. Generate the output.
3. Close the HTML destination.

```
ods html file='Salary.html';
proc print data=ia.empdata label noobs;
  label Salary='Annual Salary';
  format Salary money. Jobcode $codefmt. ;
  title1 'Salary Report';
run;
ods html close;
```

Creating an HTML Report

Salary Report

EmpID	LastName	FirstName	JobCode	Annual Salary
0031	GOLDENBERG	DESIREE	Pilot	More than 50,000
0040	WILLIAMS	ARLENE M.	Flight Attendant	Less than 25,000
0071	PERRY	ROBERT A.	Flight Attendant	Less than 25,000
0082	MCGWIER-WATTS	CHRISTINA	Pilot	More than 50,000
0091	SCOTT	HARVEY F.	Flight Attendant	25,000 to 50,000
0106	THACKER	DAVID S.	Flight Attendant	Less than 25,000
0355	BELL	THOMAS B.	Pilot	More than 50,000
0366	GLENN	MARTHA S.	Pilot	More than 50,000

Apply an ODS Style

ODS Styles are pre-defined formats for output.

- Example:

```
ods html file='output.html'  
      style=analysis;
```

- Complete List of Styles:

```
proc template;  
  list styles;  
run;  
quit;
```

ODS File Formats

With ODS you can create file formats:

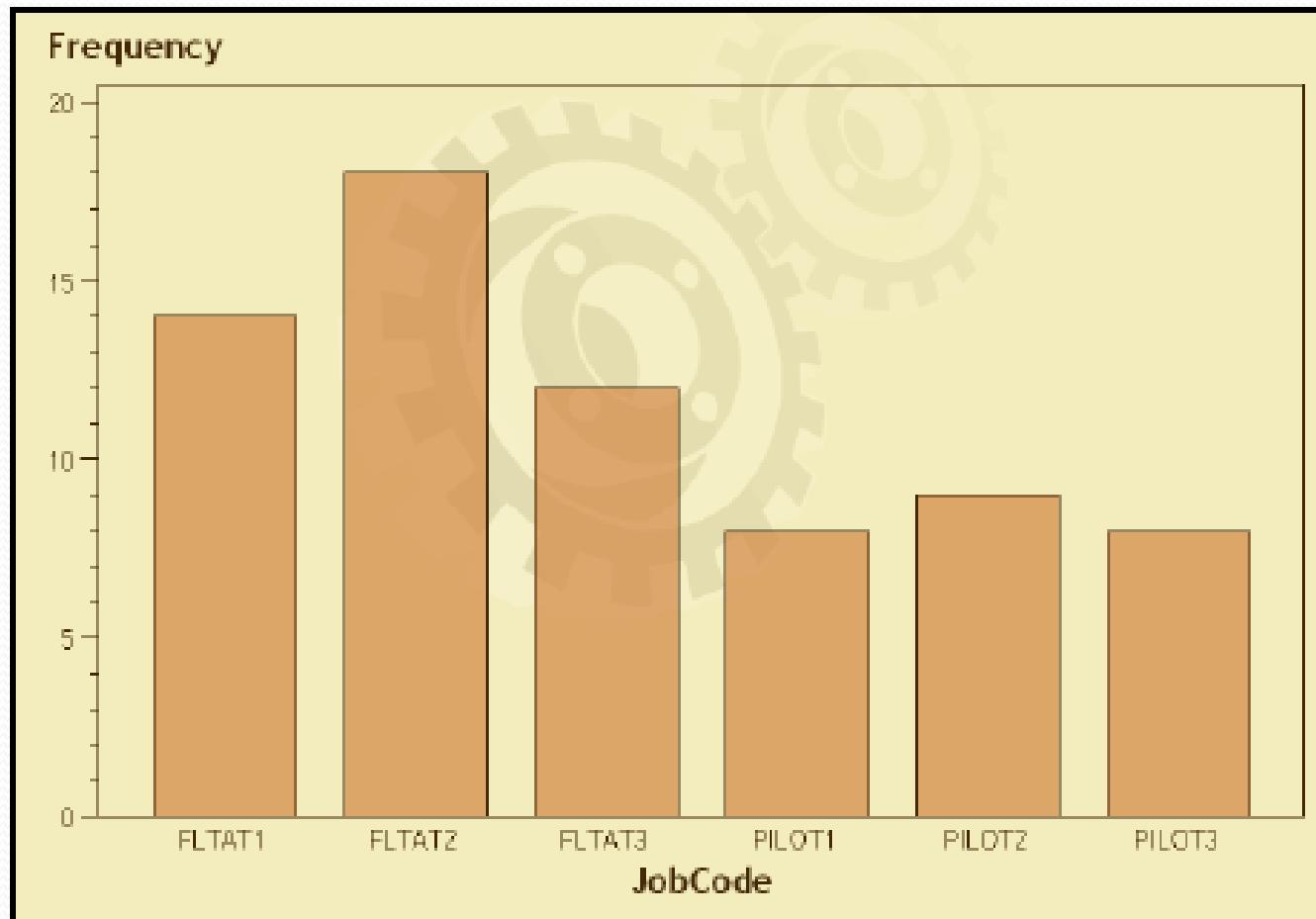
- HTML – HyperText Markup Language – for web pages
- RTF – Rich Text Format – for Word
- PDF – Portable Document Format – for Adobe
- PS – Post-Script – for printers
- Excel
- XML
- and many others

Summarizing Data with Graphs

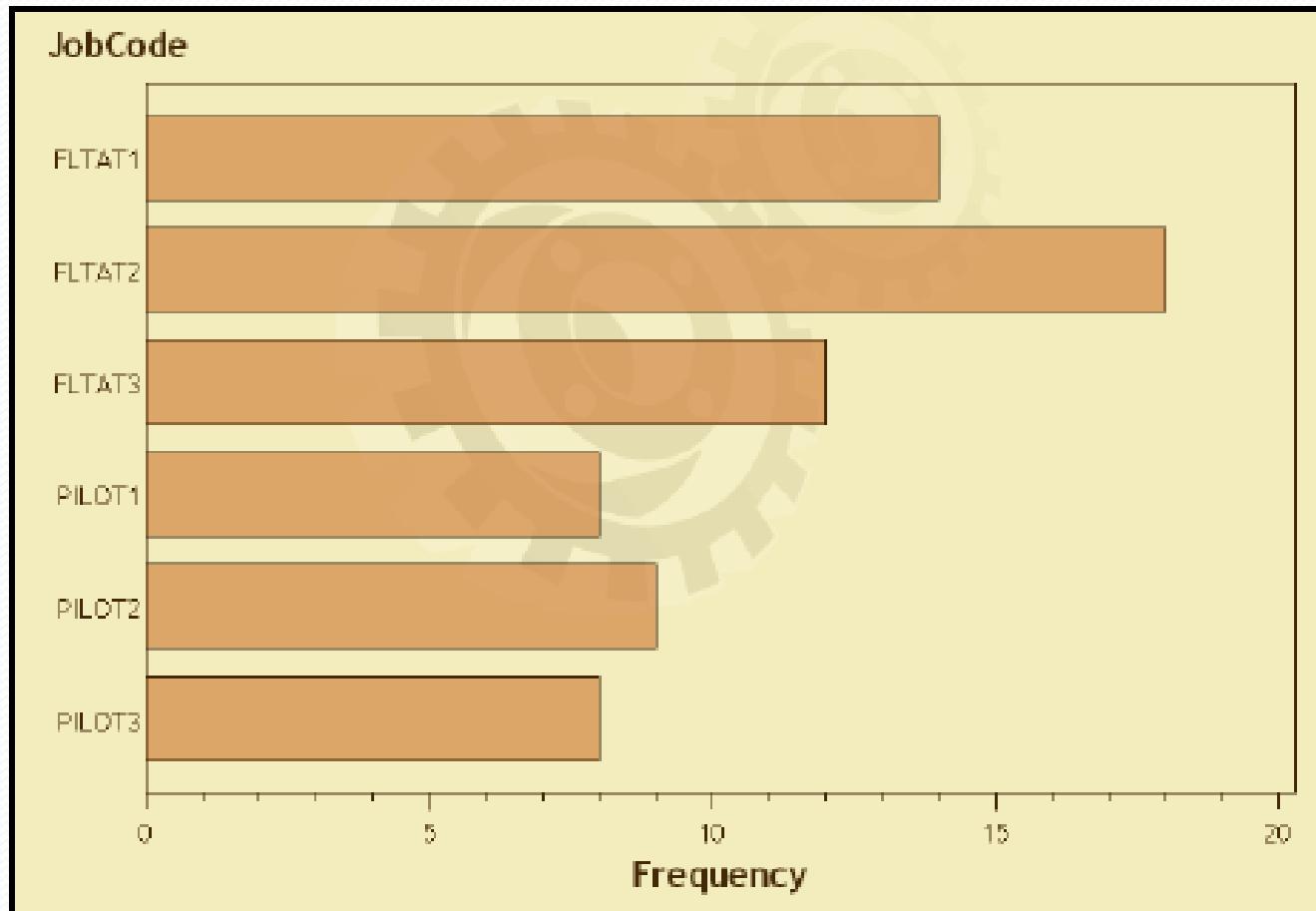
You can use bar or pie charts to graphically display the following:

- distribution of a variable's values
- average value of a variable for different categories
- total value of a variable for different categories

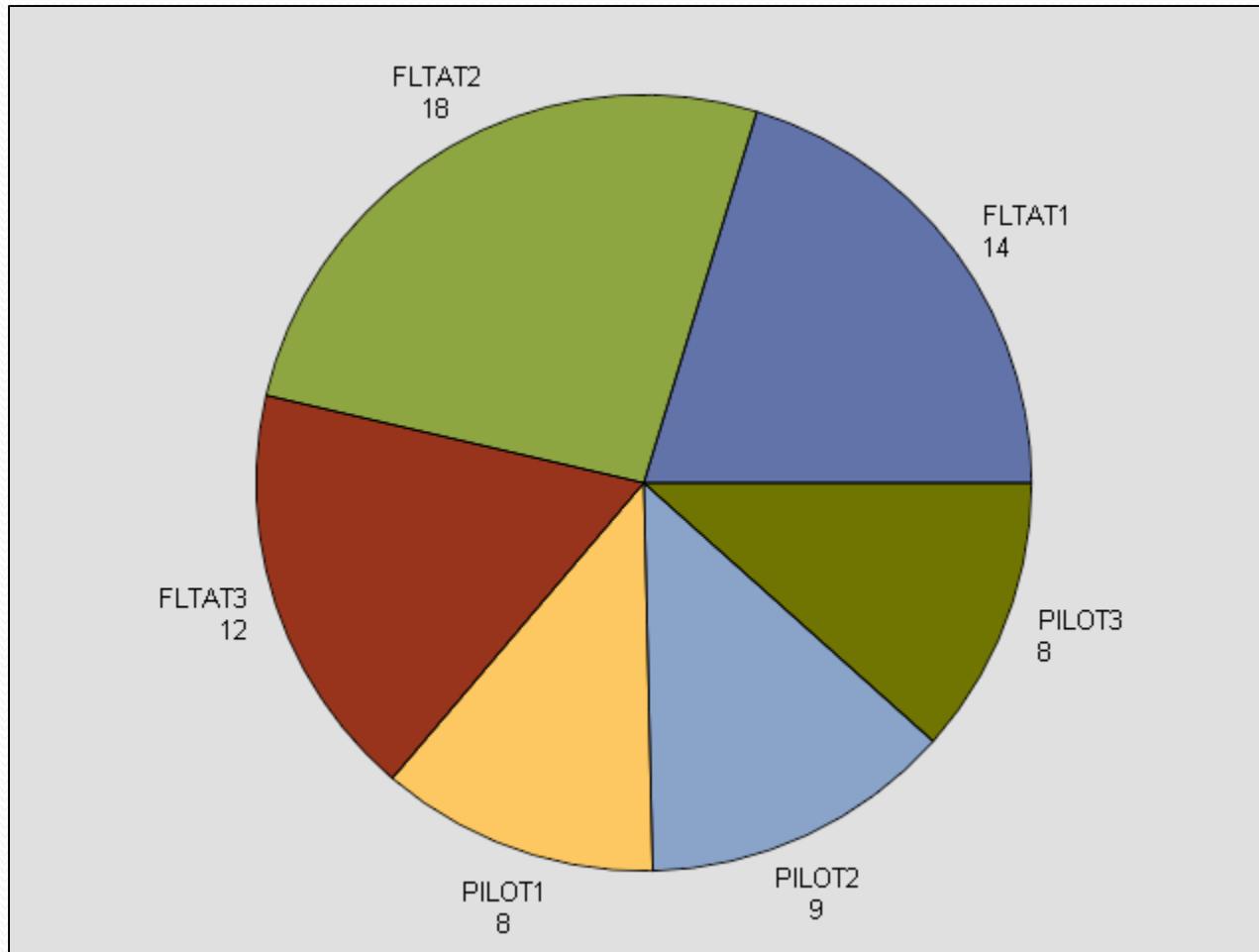
Vertical Bar Chart



Horizontal Bar Chart



Pie Chart



Specifying a Chart

Use the GCHART procedure to do the following:

- Specify the physical form of the chart.
- Identify a chart variable that determines the number of bars or pie slices to create.
- Optionally identify an analysis variable to use for calculating statistics that determine the height (or length) of the bar or the size of the slice.

The GCHART Procedure

- General form of the PROC GCHART statement:

```
PROC GCHART DATA=SAS-data-set;
```

- Use one of these statements to specify the desired type of chart:

```
HBAR chart-variable . . . </options>;
```

```
VBAR chart-variable . . . </options>;
```

```
PIE chart-variable . . . </options>;
```

Chart Variable

The chart variable

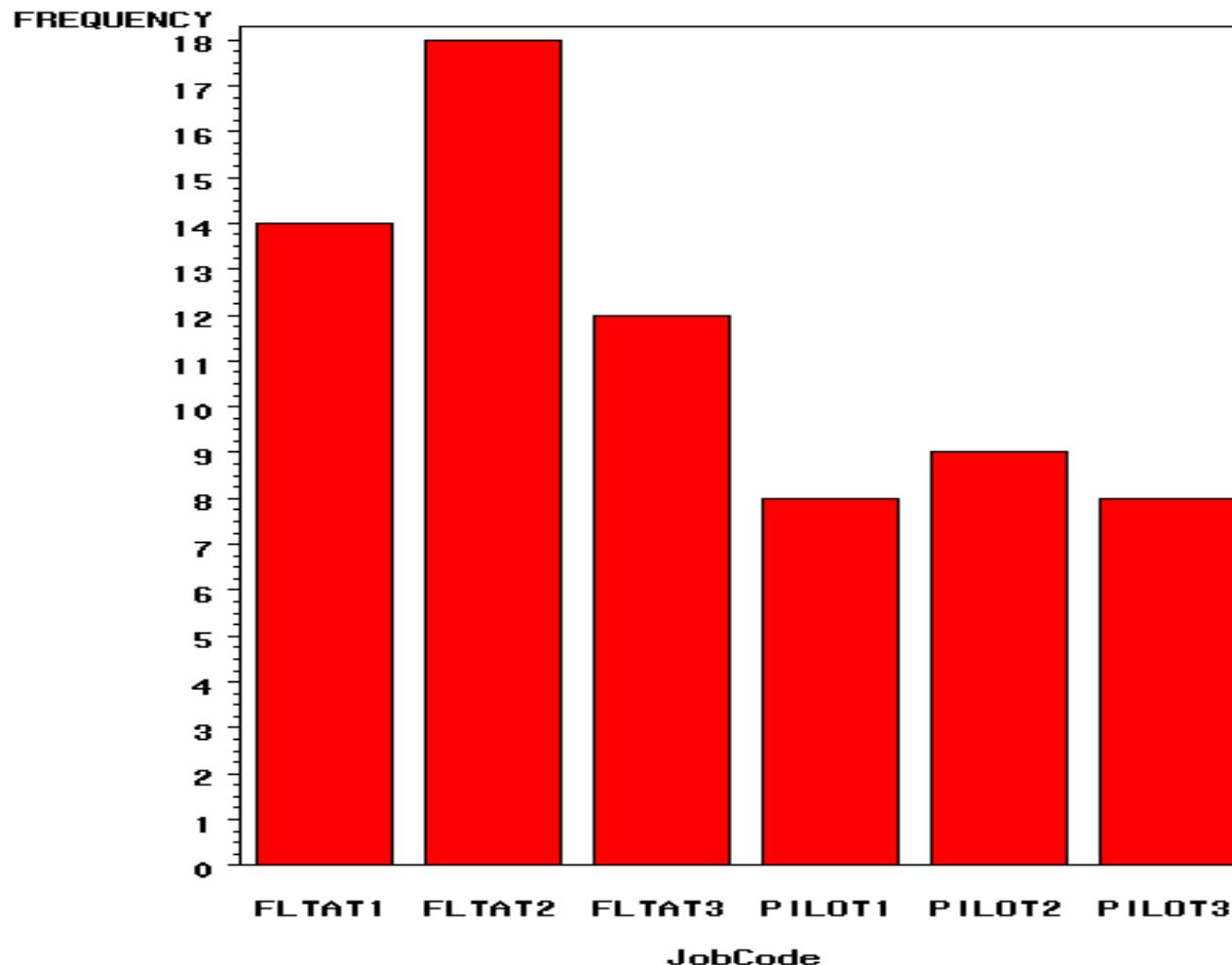
- determines the number of bars or slices produced within a graph
- can be character or numeric.

Basic Vertical Bar Chart

Produce a vertical bar chart that displays the number of employees in each job code.

```
proc gchart data=ia.crew;
  vbar JobCode;
run;
```

Basic Vertical Bar Chart



Defining the Graphics Device

General form of the GOPTIONS statement:

```
GOPTIONS graphics-options;
```

Devices control the exact format of the graph and can be either printers or application drivers

Graphics Device	GOPTIONS Statement
HP Deskjet Printer	<code>goptions dev=HPD;</code>
HP LaserJet Driver	<code>goptions dev=HPL;</code>
GIF Driver	<code>goptions dev=GIF;</code>
Tektronix Driver	<code>goptions dev=TK1;</code>
Portable Document	<code>goptions dev=PDF;</code>
PostScript Driver	<code>goptions dev=PSL;</code>
Windows Metafile Driver	<code>goptions dev=WMF;</code>

Graphics Device	GOPTIONS Statement
ActiveX Control	<code>goptions dev=ACTIVEVEX;</code>
ActiveX Image	<code>goptions dev=ACTXIMG;</code>
JAVA Applet	<code>goptions dev=JAVA;</code>
JAVA Image	<code>goptions dev=JAVAIMG;</code>
JPEG Driver	<code>goptions device=JPEG;</code>

Vertical Bar Chart with ODS HTML

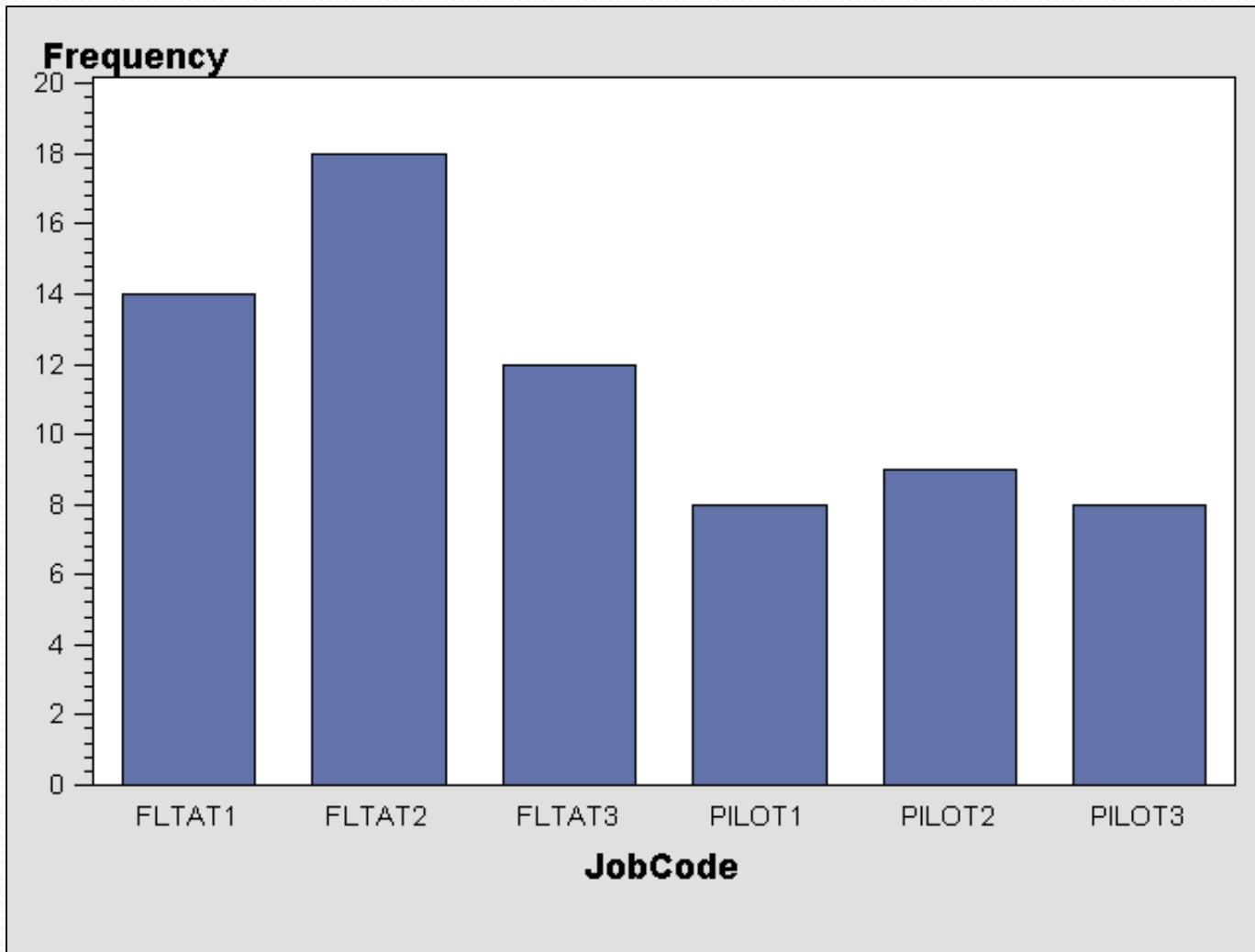
Modify the vertical bar chart to route the output to a Java image within an HTML document.

```
ods html file='vbar.html';
goptions dev=activex;

proc gchart data=ia.crew;
    vbar JobCode;
run;

ods html close;
goptions reset=all;
```

Vertical Bar Chart with ODS HTML

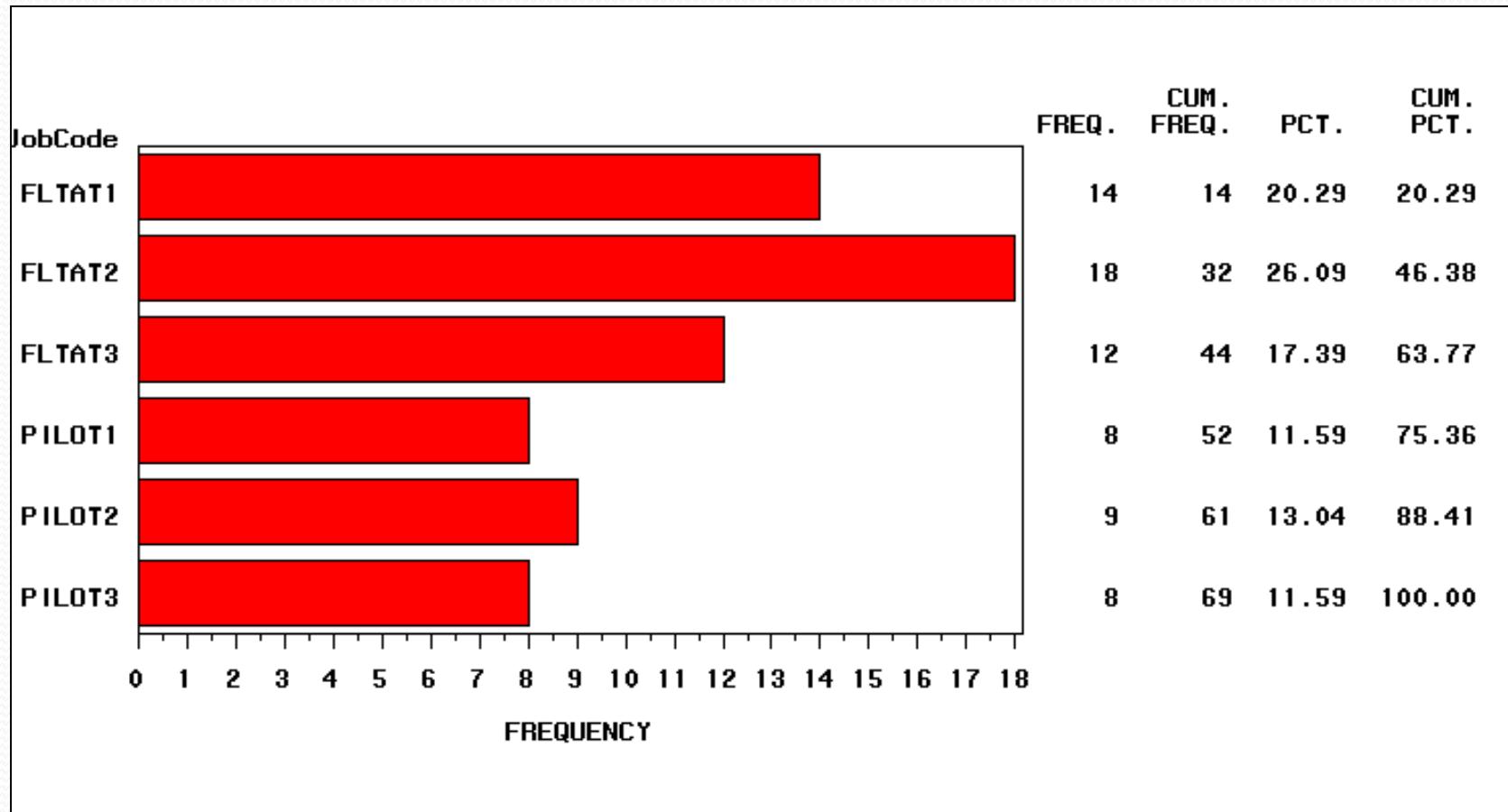


Horizontal Bar Chart

Produce a horizontal bar chart that displays the number of employees in each job code.

```
proc gchart data=ia.crew;
  hbar JobCode;
run;
```

Horizontal Bar Chart



Pie Chart

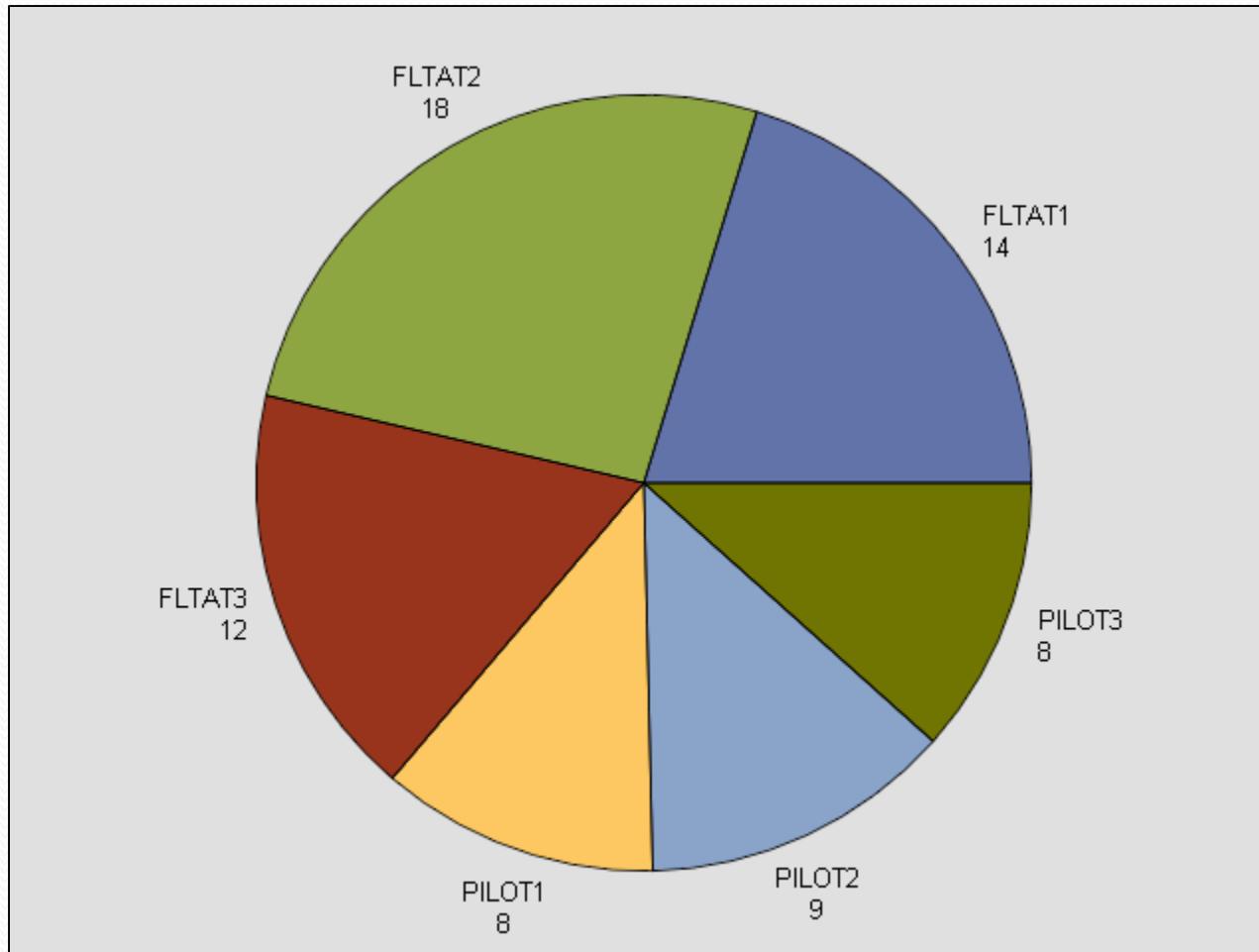
Produce a pie chart that displays the number of employees in each job code. Route the output to an ActiveX image in an HTML document.

```
ods html file='piechart.html';
goptions dev=activex;

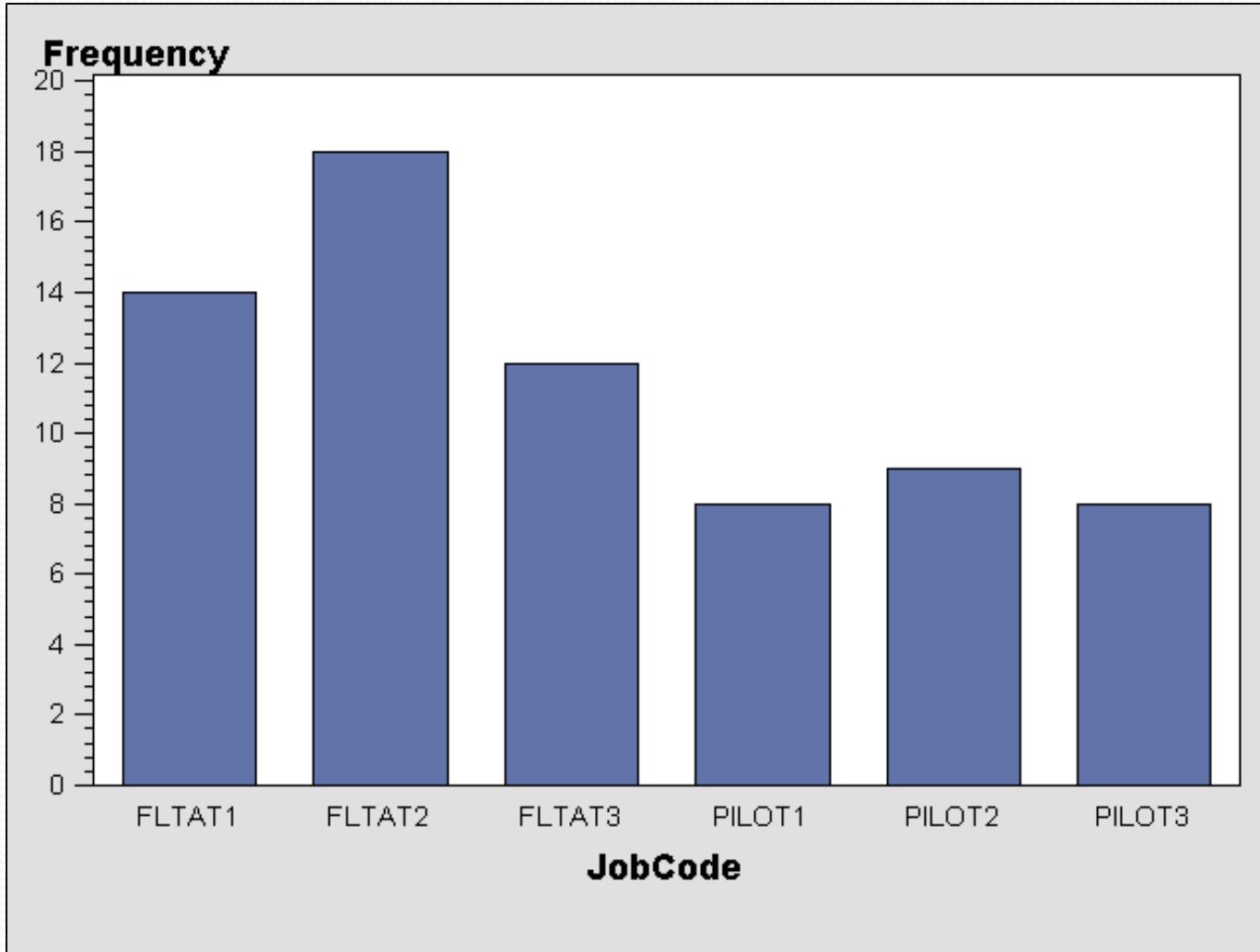
proc gchart data=ia.crew;
    pie JobCode;
run;

ods html close;
goptions reset=all;
```

Pie Chart



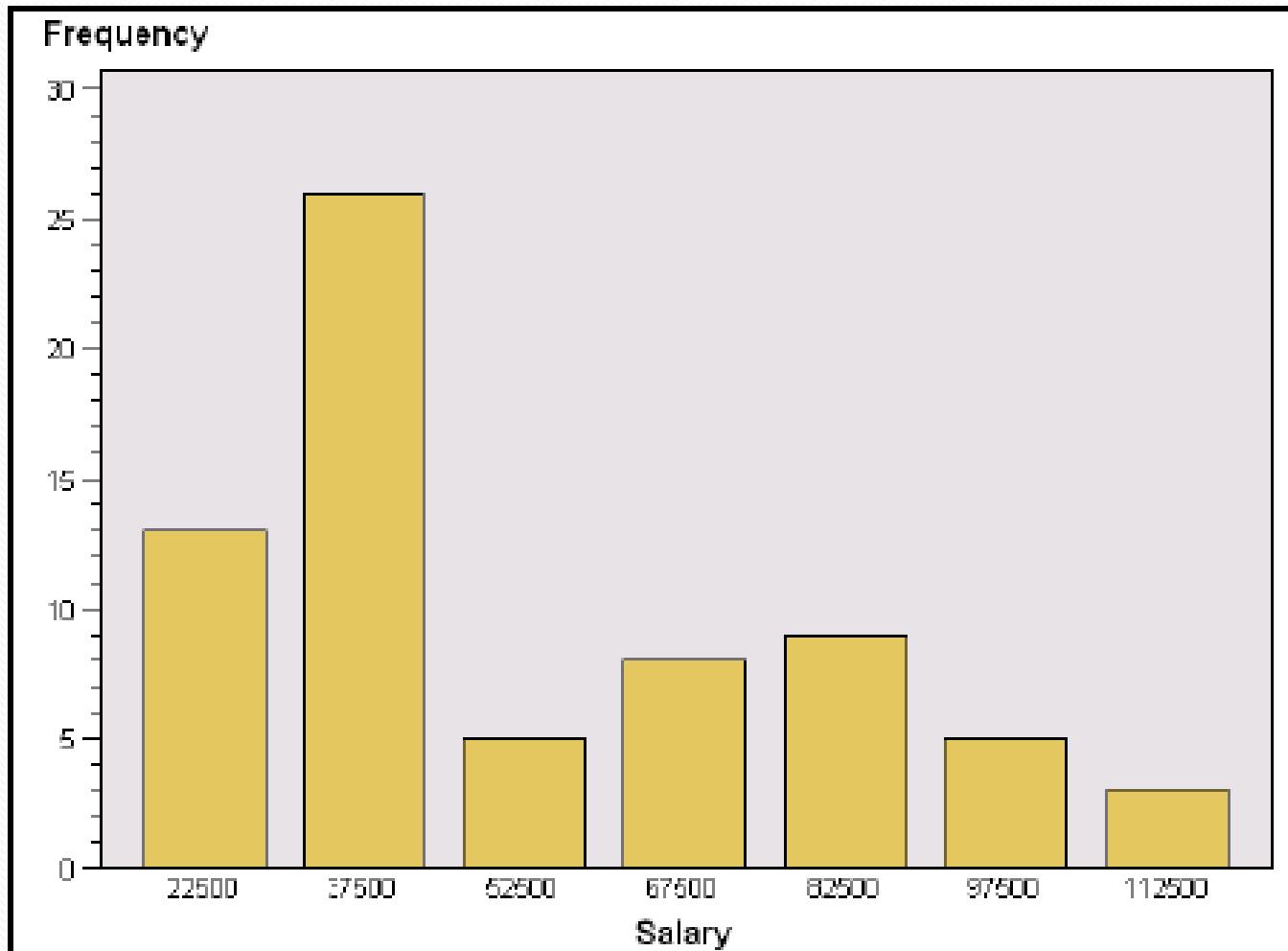
Charting Character Variables



Charting Numeric Variables

- GCHART can graph character or numeric variables
- When graphing numeric variables, by default, the values are grouped together to create a smaller number of lines
- To override the default behavior for numeric chart variables, use the DISCRETE option in the HBAR, VBAR, or PIE statement.

Numeric Chart – Default Grouping

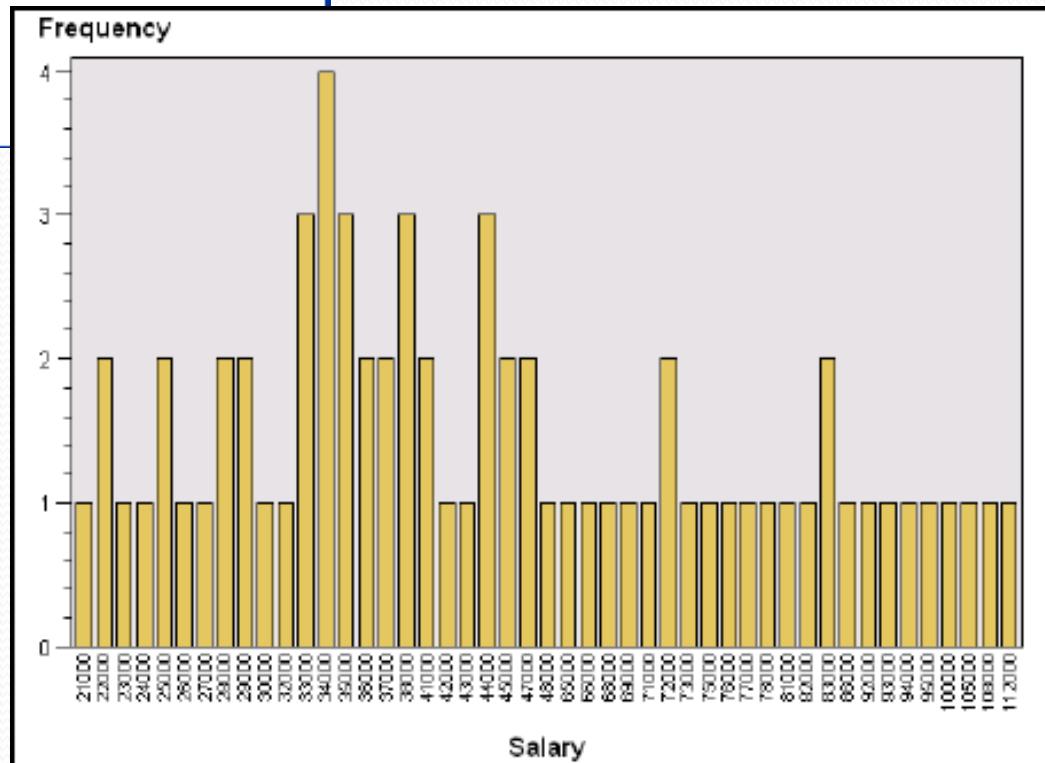


Numeric Chart – DISCRETE option

```
ods html file='Salary2.html';
goptions dev=actximg;

proc gchart data=ia.crew;
  vbar Salary / discrete;
run;

ods html close;
```



Charting Summary Statistics

- You can chart a summary statistic (e.g., mean) using the SUMVAR= and TYPE= options
- General form:

```
SUMVAR=analysis-variable  
TYPE=MEAN | SUM
```

- Description:

SUMVAR =	identifies the analysis variable to use for the sum or mean calculation.
TYPE=	specifies that the height or length of the bar or size of the slice represents a mean or sum of the <i>analysis-variable values</i> .

Using an Analysis Variable

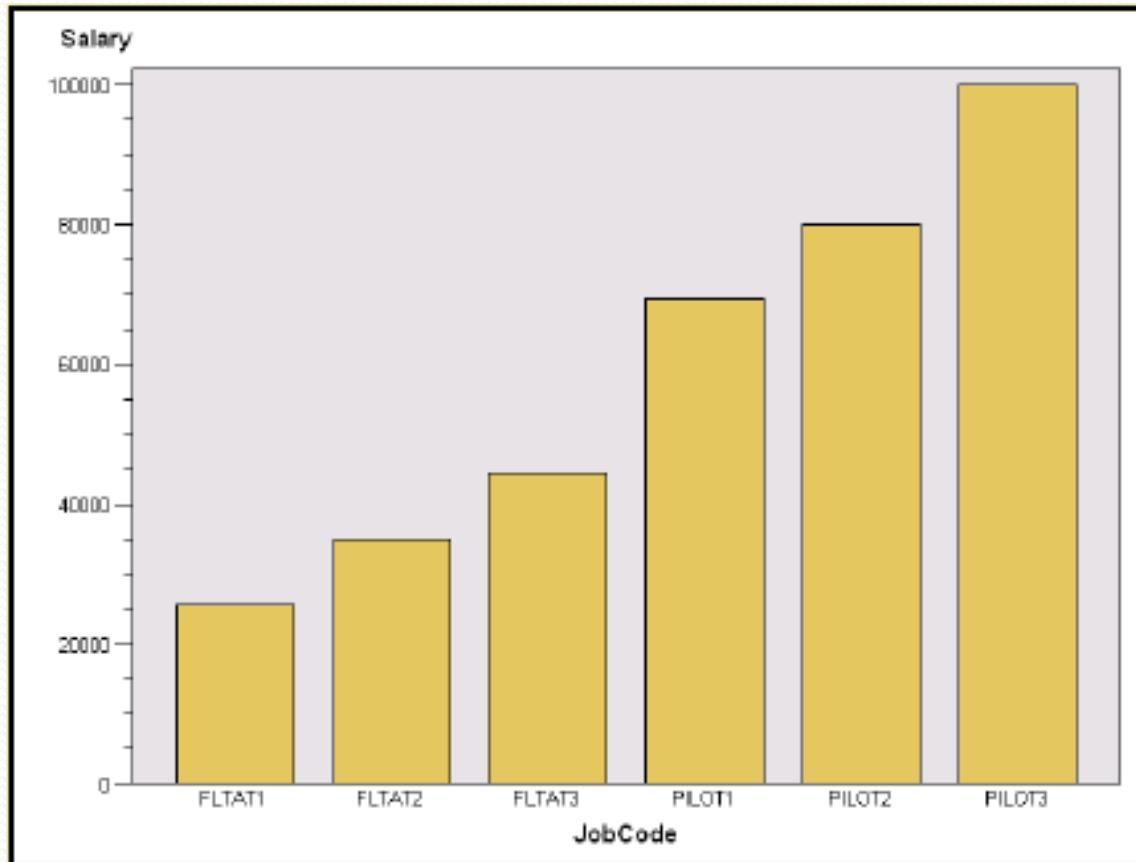
Produce a vertical bar chart that displays the average salary of employees in each job code.

```
ods html file='vbar2.html';
options dev=actximg;

proc gchart data=ia.crew;
    vbar JobCode / sumvar=Salary
                  type=mean;
run;

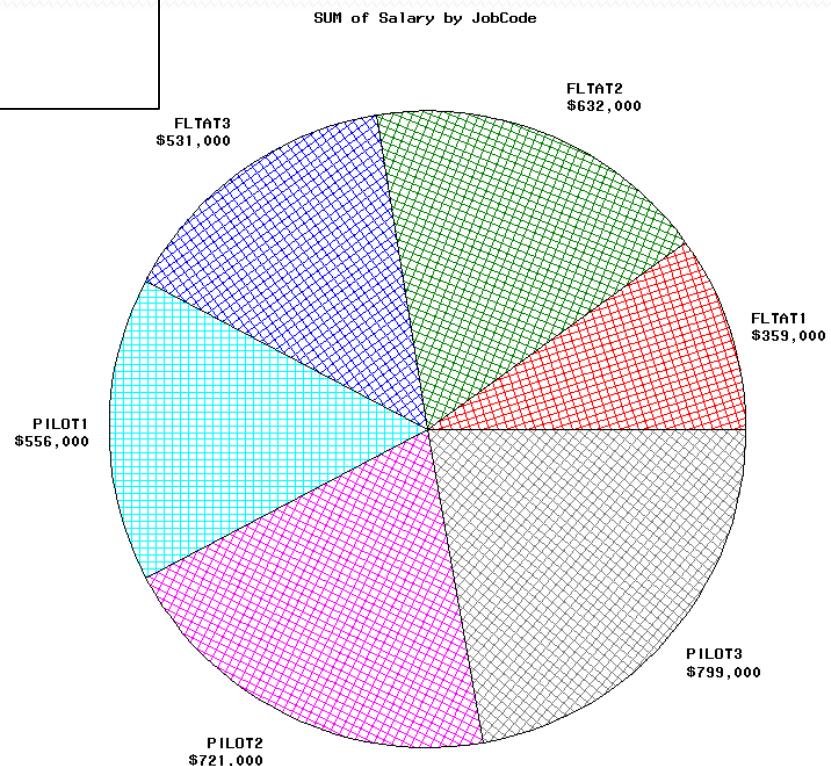
ods html close;
Options reset=all;
```

Using an Analysis Variable



FILL Option

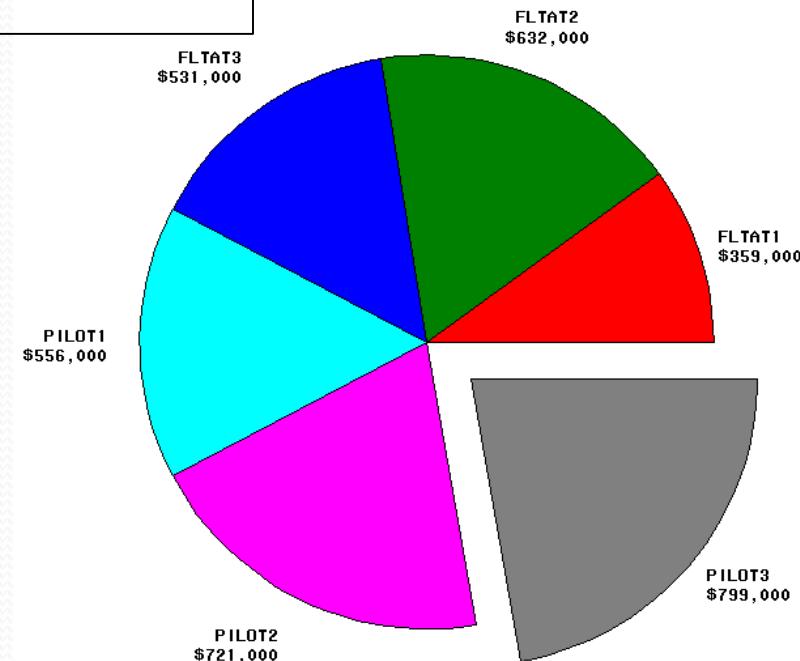
```
goptions dev=gif;
proc gchart data=ia.crew;
    pie JobCode / sumvar=Salary type=sum
        fill=x;
format Salary dollar8.;
run;
goptions reset=all;
```



EXPLODE Option

```
options dev=gif;
proc gchart data=ia.crew;
    pie JobCode / sumvar=Salary type=sum
        explode = 'PILOT3';
format Salary dollar8.;
run;
options reset=all;
```

SUM of Salary by JobCode



Adding Titles and Footnote

- Selected options

COLOR=*color* / **C**=*color*

FONT=*type-font* / **F**=*type-font*

HEIGHT=*n* / **H**=*n*

Title and Footnote Options

```
title color=green 'Number of Pilots by Job Level';  
title font=brush color=red 'March Flights';  
title height=3 in font=duplex 'Flights to RDU';  
footnote height=3 "IA's Gross Revenue by Region";  
footnote height=3 cm 'Average Salary by Job Level';  
footnote height=3 pct 'Total Flights by Model';
```

Producing Plots

Use the GPLOT procedure to produce scatterplots and line graphs

- General form:

```
PROC GPLOT DATA=SAS-data-set;
    PLOT vertical-variable*horizontal-variable </options>;
RUN;
QUIT;
```

- Example:

```
proc gplot data=data1.admit;
    plot weight * height;
run;
```

Producing Line Graph

Produce a plot of the number of passengers by date for flight number 114 over a one-week period.

```
ods html file='plot.html';
ods graphics;

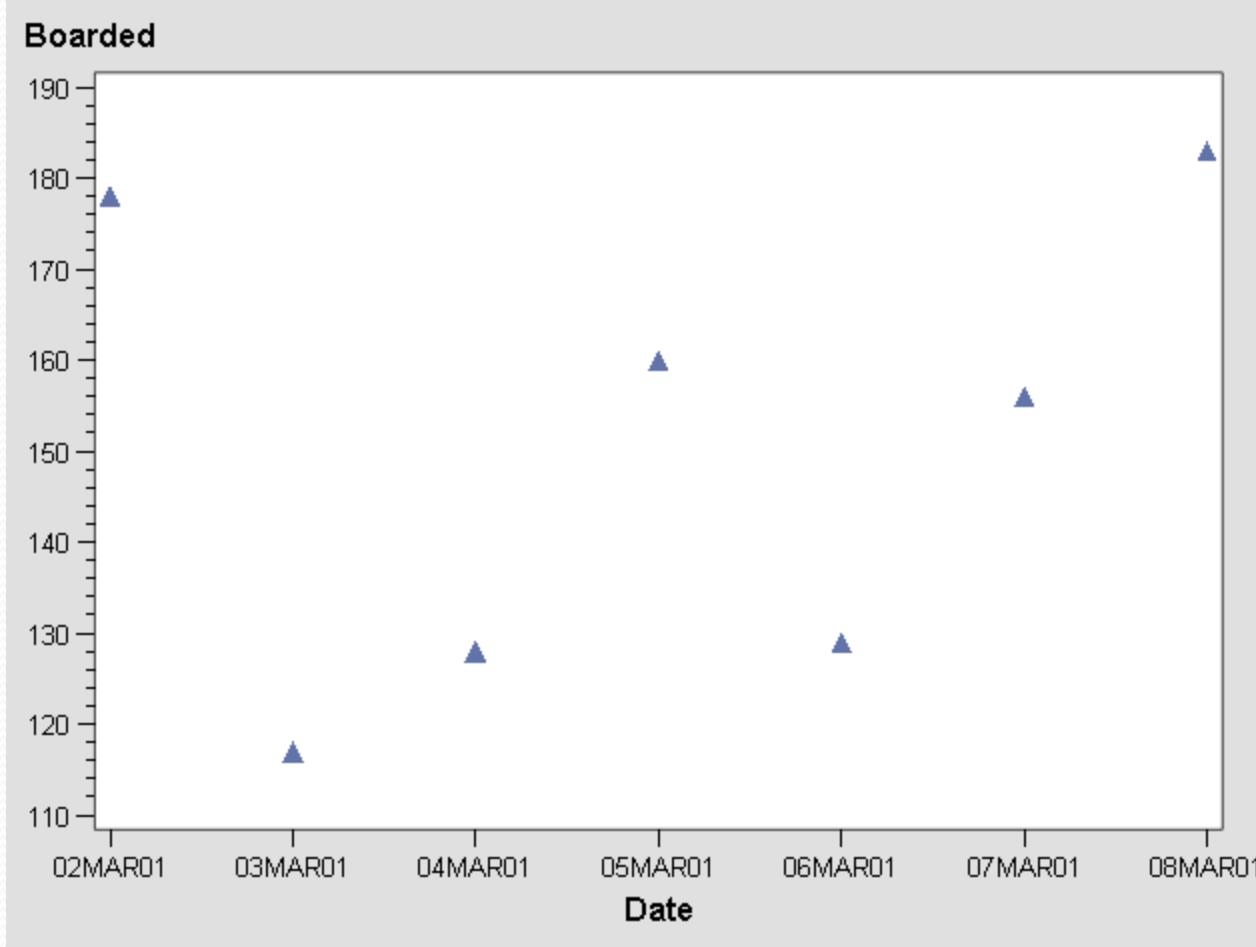
proc gplot data=data1.flight114;
    where date between '02mar2001'd and
        '08mar2001'd; *this selects one week of flights;

    plot Boarded*Date;

run;

goptions reset=all;
```

Producing Plots – G PLOT Output



SYMBOL Statement

You can use the SYMBOL statement to do the following:

- define plotting symbols
- draw lines through the data points
- specify the color of the plotting symbols and lines
- General Form

SYMBOL*n* options;

Symbol Statement Options

Use a square as the plotting symbol and join the points with straight lines.

```
proc gplot data=ia.flight114;
  where date between '02mar2001'd and
    '08mar2001'd;
  plot Boarded*Date;
    symbol value=square i=join;
run;
```

SYMBOL Statement

SYMBOL statements have the following characteristics:

global	After they are defined, they remain in effect until changed or until the end of the SAS session.
additive	Specifying the value of one option does not affect the values of other options.

Symbol Statement Options

- Options for shape of symbol

VALUE=*symbol* / V=*symbol*

- Selected *symbol* values include the following:

PLUS	DIAMOND
STAR	TRIANGLE
SQUARE	NONE (no plotting symbol)

Symbol Statement Options

- Interpolation

I=*interpolation*

- Selected *interpolation* values:

JOIN	joins the points with straight lines.
SPLINE	joins the points with a smooth line.
NEEDLE	draws vertical lines from the points to the horizontal axes.

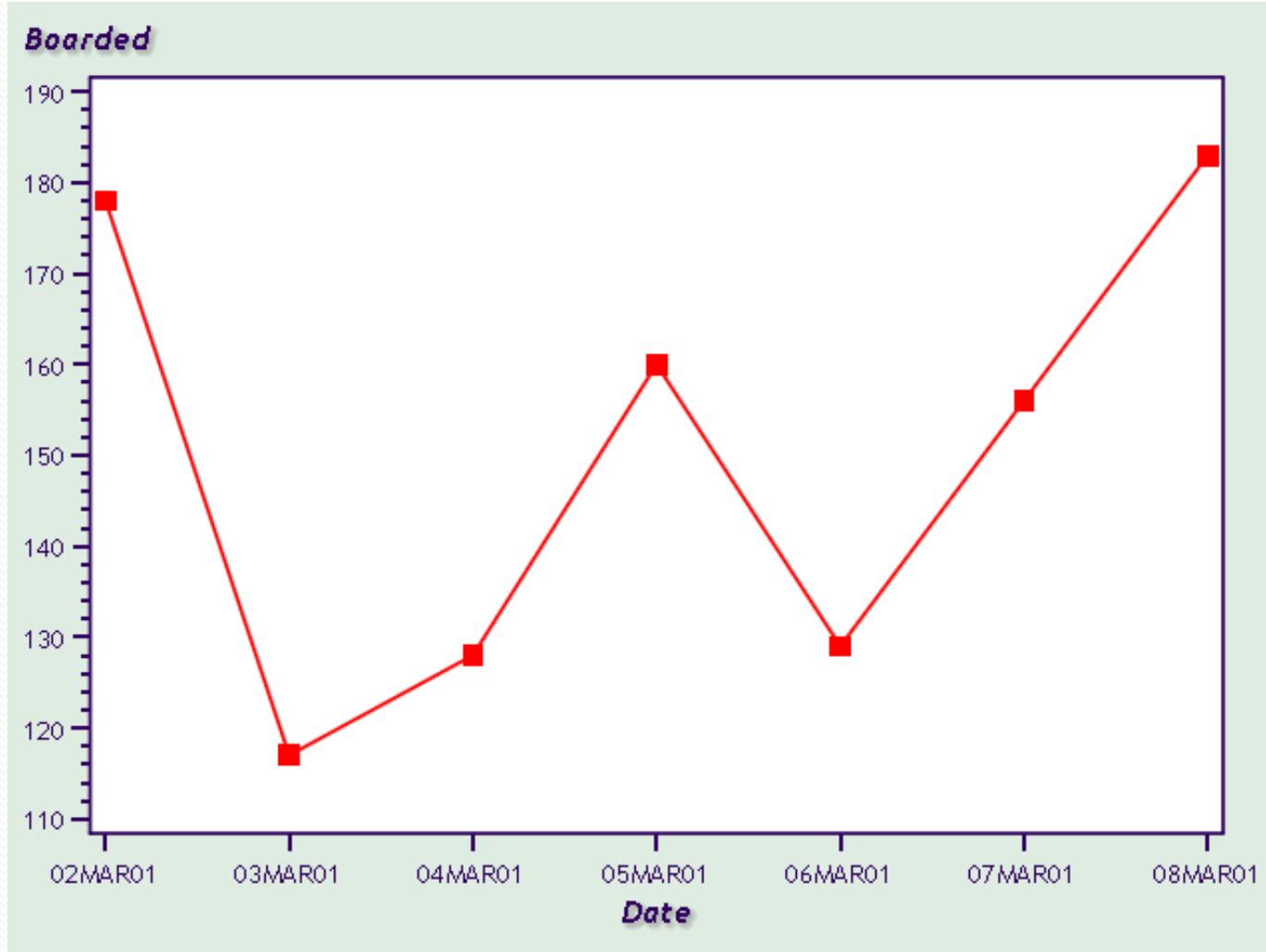
- Combining symbol value=none with interpolation produces a line-only plot

Additional SYMBOL Statement Options

WIDTH= <i>width</i> W= <i>width</i>	specifies the thickness of the line.
COLOR= <i>color</i> C= <i>color</i>	specifies the color of the line.

```
proc gplot data=ia.flight114;
    where date between '02mar2001'd and
        '08mar2001'd;
    plot Boarded*Date;
        symbol color=red width=2;
run;
```

Additional SYMBOL Statement Options



Modifying the SYMBOL Statement

- Set the attributes only for SYMBOL1.

```
symbol1 c=blue v=diamond;
```

- Modify only the color of SYMBOL1, not the V= option setting.

```
symbol1 c=green;
```

- To cancel SYMBOL Statements:

```
symbol1;
```

-OR-

```
goptions reset=symbol;
```

Controlling the Axis Appearance

You can modify the appearance of the axes that PROC GPLOT produces with the following:

- PLOT statement options
- the LABEL statement
- the FORMAT statement

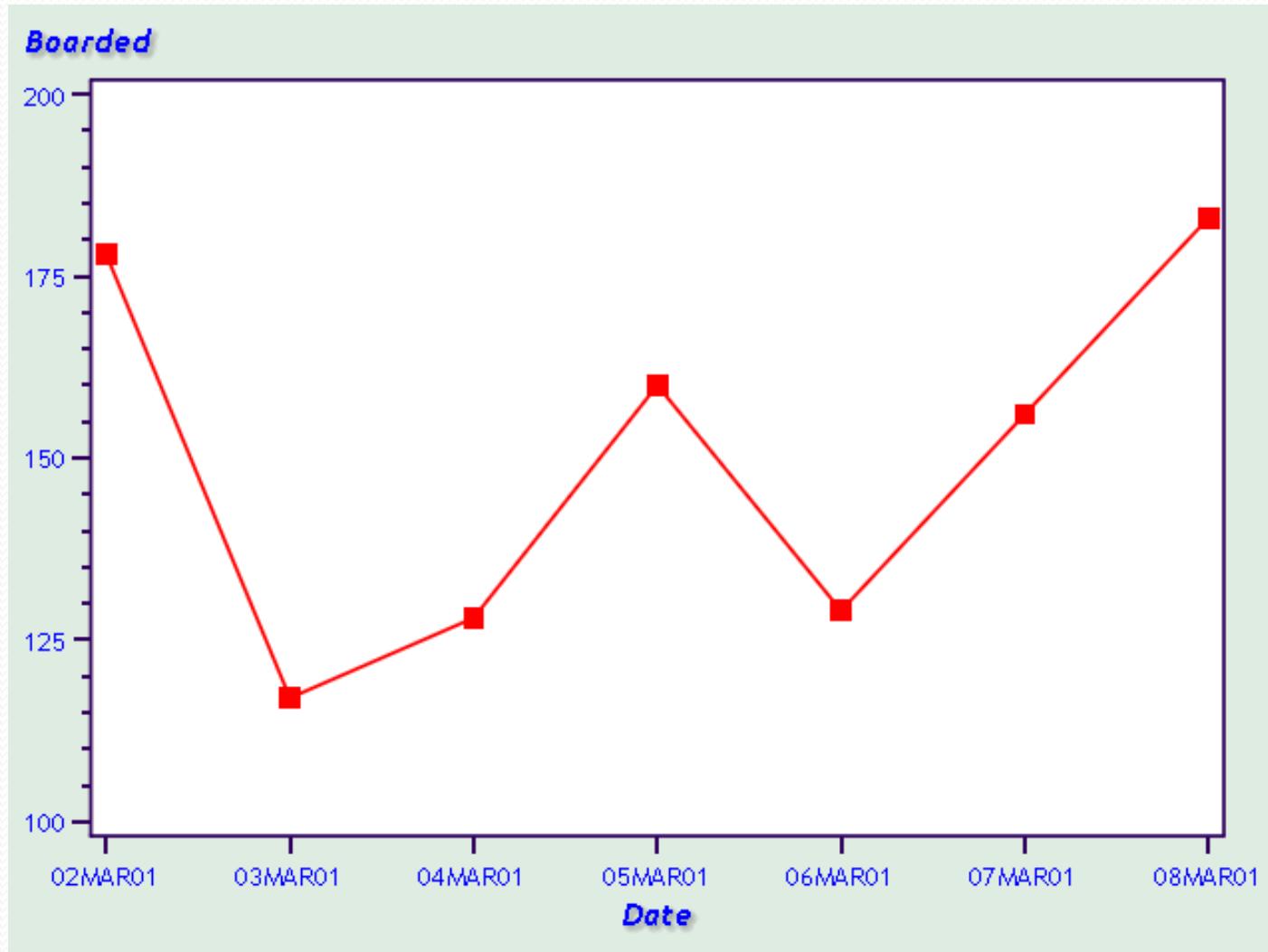
HAXIS= <i>values</i>	scales the horizontal axis.
VAXIS= <i>values</i>	scales the vertical axis.
CAXIS= <i>color</i>	specifies the color of both axes.
CTEXT= <i>color</i>	specifies the color of the text on both axes.

PLOT Statement Options

- Define the scale on the vertical axis and display the axis text in blue.

```
proc gplot data=ia.flight114;
    where date between '02mar2001'd and
        '08mar2001'd;
    plot Boarded*Date / vaxis=100 to 200 by 25
        ctext=blue;
        symbol value=square i=join;
run;
```

Axis Options - Output



Modifying Axis Labels

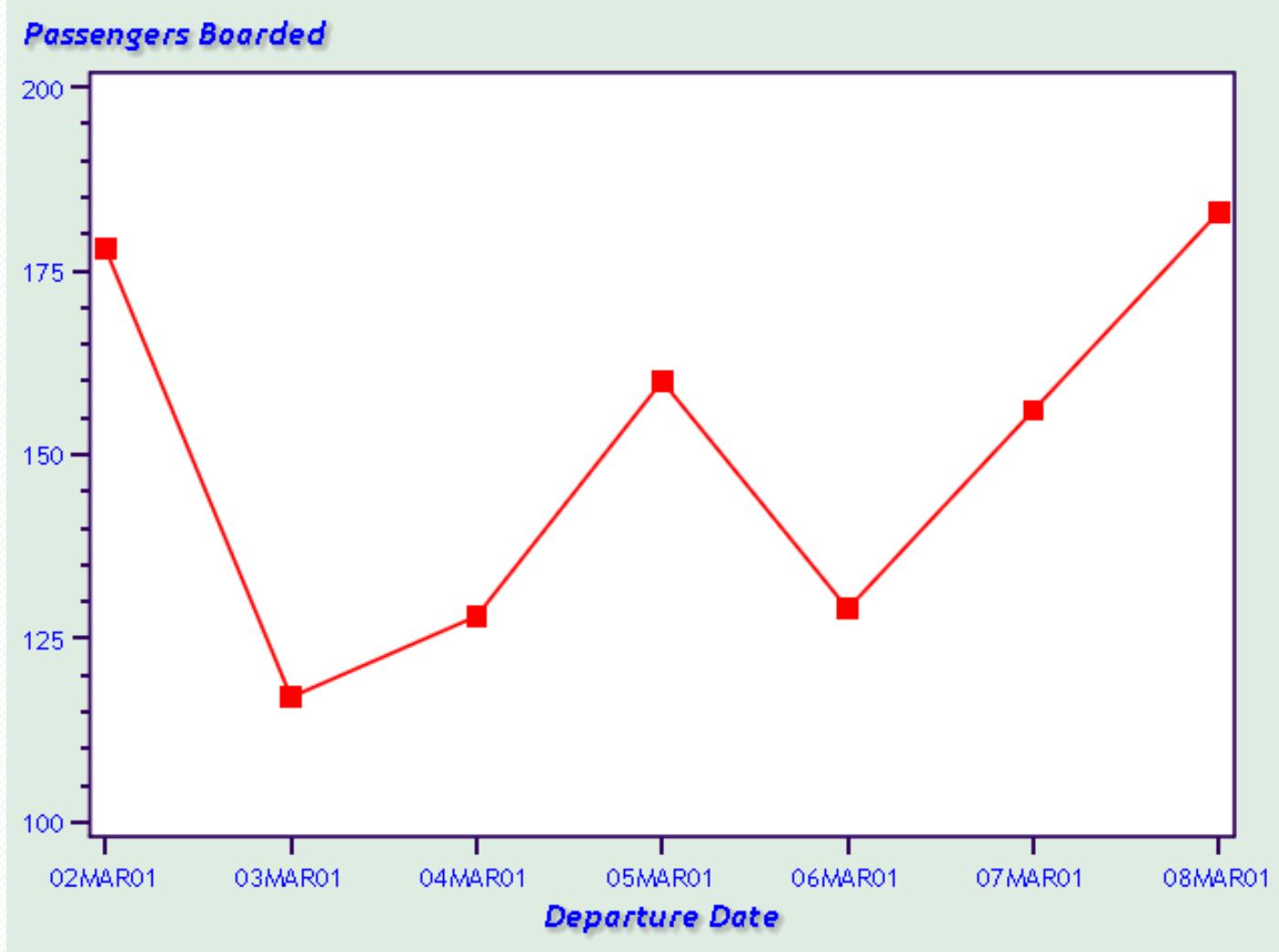
To place labels on the axes:

```
proc gplot data=ia.flight114;
    where date between '02mar2001'd and
        '08mar2001'd;

    plot Boarded*Date / vaxis=100 to 200 by 25
        ctext=blue;
    symbol value=square i=join color=red width=2;
    label Boarded='Passengers Boarded'
        Date='Departure Date';
run;

ods html close;
```

Modifying Axis Labels

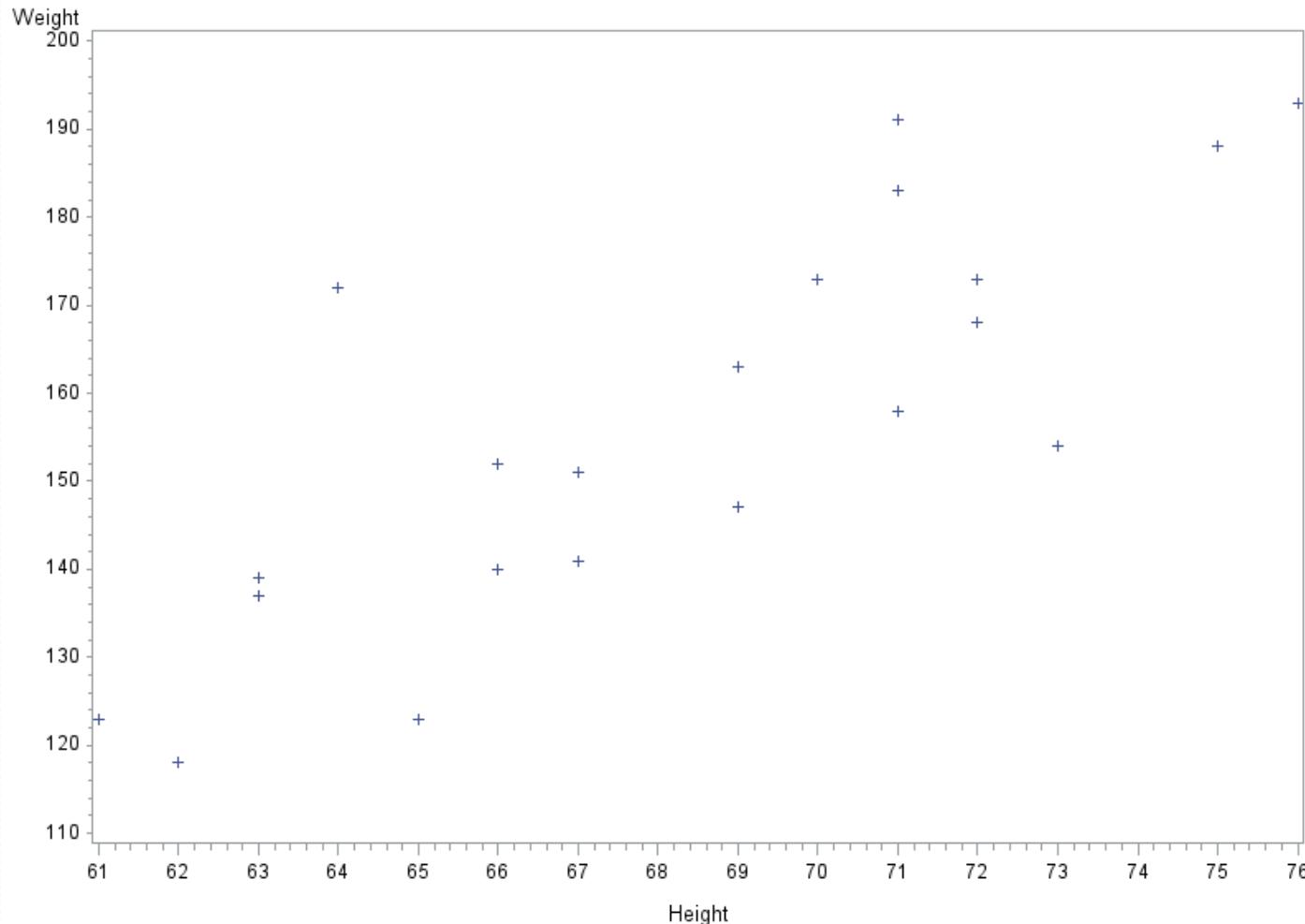


Producing a Scatterplot

- A scatterplot (usually) plots two continuous variables

```
proc gplot data=data1.admit;
  plot weight*height;
run;
quit;
```

ScatterPlot



Adding Options

- You can modify the symbol, axis labels and axis tick marks
- You usually do not connect the dots in a scatterplot

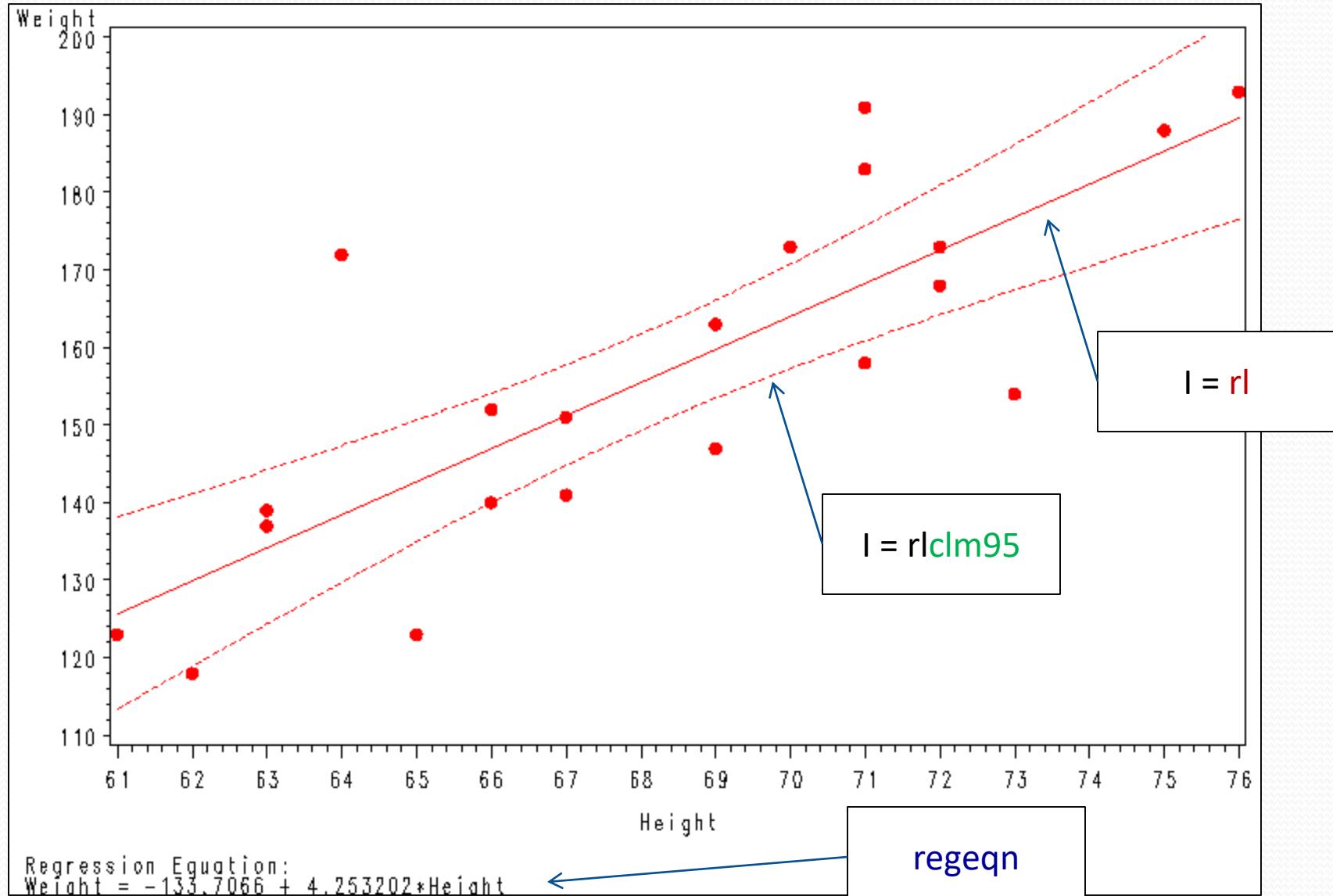
```
proc gplot data=data1.admit;
  plot weight*height;
  symbol v=dot color=blue;
run;
quit;
```

Scatterplot with Regression

- You can also add a regression line, regression equation and prediction confidence interval

```
proc gplot data=data1.admit;
  plot weight*height / regeqn ;
  symbol v=dot i=r1CLM95;
run;
quit;
```

Scatterplot with Regression



PSTAT 130

SAS Base Programming

Summer 2017

William Qiu

Department of Statistics and Applied Probability

UCSB

Lecture 12

Lecture Outline

- Controlling When Records are Written to a SAS Dataset
 - Creating Multiple Records from a Single Observations (OUPUT)
 - Writing to Multiple SAS Data Sets (OUTPUT)
 - Creating a Single Record from Multiple Observations (RETAIN)
 - Creating a running total – RETAIN Statement and Sum Expression
- Limiting the Variables Read from or Written to a SAS Data Set (DROP= and KEEP=)
- Limiting the Number of Observations Read from a Data Set (FIRSTOBS and OBS statements)

Controlling When Records are Written to a SAS Dataset

Automatic Output (Review)

2. Automatic
Return

```
data forecast;  
  set prog2.growth;  
  <additional SAS statements>;  
run;
```

1. Automatic Output

The OUTPUT Statement

General Form

```
OUTPUT <SAS-data-set-1.....SAS-data—set-n>;
```

OUTPUT Statement

- By default, SAS writes a data record to the output data set when it reaches the RUN statement at the end of the Data Step.
- If you include an OUTPUT statement in the Data Step
 - SAS writes a record immediately
 - and DOES NOT write automatically at the end of the data step
- With OUTPUT you can
 - Create multiple records from one observation
 - Write to multiple output data sets
 - Combine information from multiple observations into a single record (when combined with RETAIN statement)

Creating Multiple Observations

PROCESS: Take each patient's weight at admission and forecast forward for the next five years

- Assume 5% weight gain per year
- Example: John Jones currently weighs 175 lbs. We project he will weigh 5% more each year:

Year	Projected Weight
1	183.8
2	192.9
3	202.6
4	212.7
5	223.3

- Use the OUTPUT statement to write forecasted weight to the output data set for each of the next five years

Input Data Set – ia.admit

Name	Level	Sex	Age	Height	Weight
Murray, W	HIGH	M	27	72	168
Almers, C	HIGH	F	34	66	152
Bonaventure, T	LOW	F	31	61	123
Johnson, R	MOD	F	43	63	137
LaMance, K	LOW	M	51	71	158
Jones, M	HIGH	M	29	76	193
Reberson, P	MOD	F	32	67	151
King, E	MOD	M	35	70	173
Pitts, D	LOW	M	34	73	154
Eberhardt, S	LOW	F	49	64	172
Nunnally, A	HIGH	F	44	66	140
Oberon, M	LOW	F	28	62	118
Peterson, V	MOD	M	30	69	147
Quigley, M	HIGH	F	40	69	163
Cameron, L	MOD	M	47	72	173
Underwood, K	LOW	M	60	71	191
Takahashi, Y	MOD	F	43	65	123

Create Multiple Observations

```
data work.weightproj;  
    set ia.admit;  
    increase = 1.05;  
    year = 1;  
    projweight = weight * increase;  
    output;  
    year = 2;  
    projweight = projweight * increase;  
    output;  
    year = 3;  
    projweight = projweight * increase;  
    output;  
    year = 4;  
    projweight = projweight * increase;  
    output;  
    year = 5;  
    projweight = projweight * increase;  
    output;  
run;
```

Writes Year 1
Projection to SAS
Dataset

Calculates 5%
weight increase for
Year 1

Log Output

**NOTE: There were 21 observations read
from the data set IA.ADMIT.**

**NOTE: The data set WORK.WEIGHTPROJ has
105 observations and 12 variables.**

Output Data Set – work.weightproj

Patient Name	Projected Year	Weight
Murray, W	1	176.400
Murray, W	2	185.220
Murray, W	3	194.481
Murray, W	4	204.205
Murray, W	5	214.415
Almers, C	1	159.600
Almers, C	2	167.580
Almers, C	3	175.959
Almers, C	4	184.757
Almers, C	5	193.995
Bonaventure, T	1	129.150
Bonaventure, T	2	135.608
Bonaventure, T	3	142.388
Bonaventure, T	4	149.507
Bonaventure, T	5	156.983

Writing to Multiple SAS Data Sets

- GOAL: Use the ADMIT data set to create three separate output data sets:
 - Patients with Low activity level
 - Patients with Moderate activity level
 - Patients with High activity level
- This is the reverse of Concatenating (Appending) data sets

The DATA Statement (Review)

- General Form

```
DATA <data-set-name-1> <...data-set-name-n>;
```



Name of the FIRST
dataset being created by
this data step



Name of the SECOND
dataset being created by
this data step

Input Data Set – ia.admit

Name	Level	Sex	Age	Height	Weight
Murray, W	HIGH	M	27	72	168
Almers, C	HIGH	F	34	66	152
Bonaventure, T	LOW	F	31	61	123
Johnson, R	MOD	F	43	63	137
LaMance, K	LOW	M	51	71	158
Jones, M	HIGH	M	29	76	193
Reberson, P	MOD	F	32	67	151
King, E	MOD	M	35	70	173
Pitts, D	LOW	M	34	73	154
Eberhardt, S	LOW	F	49	64	172
Nunnelly, A	HIGH	F	44	66	140
Oberon, M	LOW	F	28	62	118
Peterson, V	MOD	M	30	69	147
Quigley, M	HIGH	F	40	69	163
Cameron, L	MOD	M	47	72	173
Underwood, K	LOW	M	60	71	191
Takahashi, Y	MOD	F	43	65	123

Create Multiple SAS Data Sets in a Single DATA Step

```
data work.low work.mod work.high;  
    set ia.admit;  
    if Actlevel = 'LOW' then output low;  
    if Actlevel = 'MOD' then output mod;  
    if Actlevel = 'HIGH' then output high;  
run;
```

Output Data Set – work.low

Name	Level	Sex	Age	Height	Weight
Bonaventure, T	LOW	F	31	61	123
LaMance, K	LOW	M	51	71	158
Pitts, D	LOW	M	34	73	154
Eberhardt, S	LOW	F	49	64	172
Oberon, M	LOW	F	28	62	118
Underwood, K	LOW	M	60	71	191
Ivan, H	LOW	F	22	63	139

Creating One Record from Multiple Observations

Use the RETAIN and OUTPUT statements, and First./Last. variables, to combine information from multiple data records, and write only one “summary” record

- Example: Patients sign up for a six-month weight loss program. They are weighed at the start of the program, and then at their Month 3 and Month 6 visits. The data set records the weight (and the date of the visit) on a separate record for each visit.
 - We need to calculate how much weight the subject has lost since the last visit.

Weightloss Data Set

Obs	name	visit	weight
1	john	01/01/08	210
2	john	04/04/08	199
3	john	07/03/08	183
4	mary	02/14/08	175
5	mary	05/16/08	167
6	mary	08/04/08	153
7	dave	09/23/08	223
8	dave	12/28/08	215
9	dave	03/24/09	206

RETAIN Statement

By default, SAS sets the values of all variables to missing at the start of each pass through the Data Step. As a result, you can't save the values of a variable from one record to the next.

Use the RETAIN statement to keep the value of a calculated variable available for use with the next observation.

```
RETAIN variable-1 < . . . variable-n>;
```

- General Form:

Calculating variables using consecutive records - RETAIN

```
data work.weightchange;
  set weightloss;
  weightchange = lastweight - weight;
  lastweight = weight;
  retain;
run;
```

Weight Change – Version 1 – RETAIN

Subject Name	Visit Date	Change			
		Visit Weight	in Weight	Last Weight	
john	01/01/08	210	.	210	
john	04/04/08	199	11	199	
john	07/03/08	183	16	183	
mary	02/14/08	175	8	175	
mary	05/16/08	167	8	167	
mary	08/04/08	153	14	153	
dave	09/23/08	223	-70	223	
dave	12/28/08	215	8	215	
dave	03/24/09	206	9	206	

First.ByVariable & Last.ByVariable

When PROC SORT is used to sort a data set, it creates two variables that can be used to identify the First record in the By Group and the Last record in the By Group.

- General Form

First.BY-variable
Last.BY-variable

Calculating Weight Change

```
data weightchange;
  set weightloss;
  by name visit;
  if first.name then do;
    baseweight=weight;
    weightchange = .;
  end;
  else weightchange = baseweight - weight;
  retain;
run;
```

Weight Change – Version 2

RETAIN with First.By

Subject Name	Visit Date	Visit Weight	Change	
			Baseline	in
			Weight	Weight
dave	09/23/08	223	223	.
dave	12/28/08	215	223	8
dave	03/24/09	206	223	17
john	01/01/08	210	210	.
john	04/04/08	199	210	11
John	07/03/08	183	210	27
mary	02/14/08	175	175	.
mary	05/16/08	167	175	8
mary	08/04/08	153	175	22



First.Name



Last.Name

Outputting Only Summary Records

```
data weightchange;
    set weightloss;
    by name visit;
    if first.name then do;
        baseweight=weight;
        weightchange = .;
    end;
    if last.name then do;
        weightchange = baseweight - weight;
        output;
    end;
    retain;
run;
```

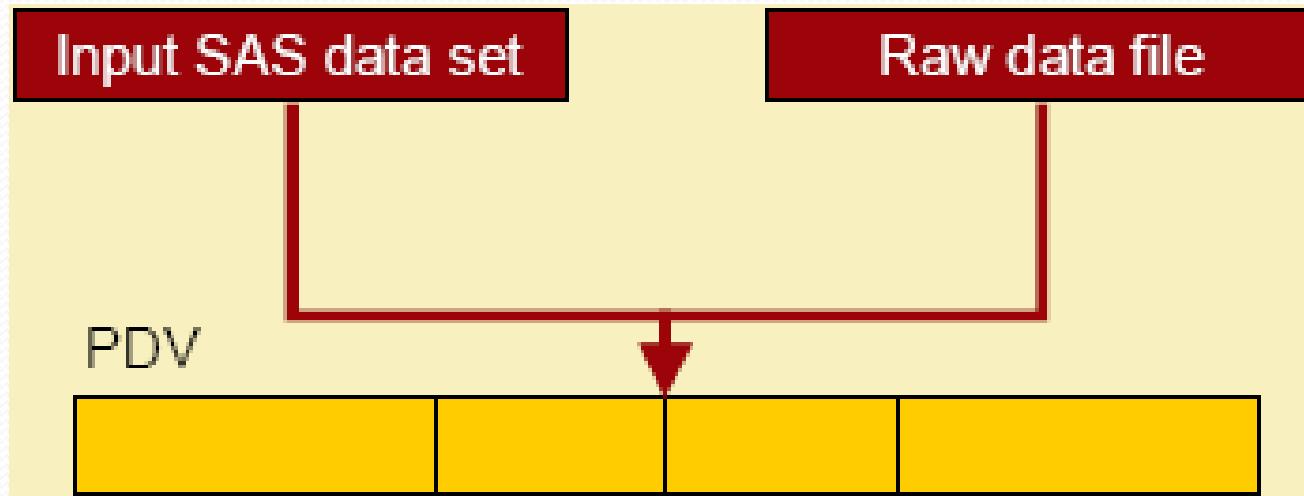
Weight Change – Summary

Subject Name	Visit Date	Visit Weight	Change	
			Baseline	in Weight
dave	03/24/09	206	223	17
john	07/03/08	183	210	27
mary	08/04/08	153	175	22

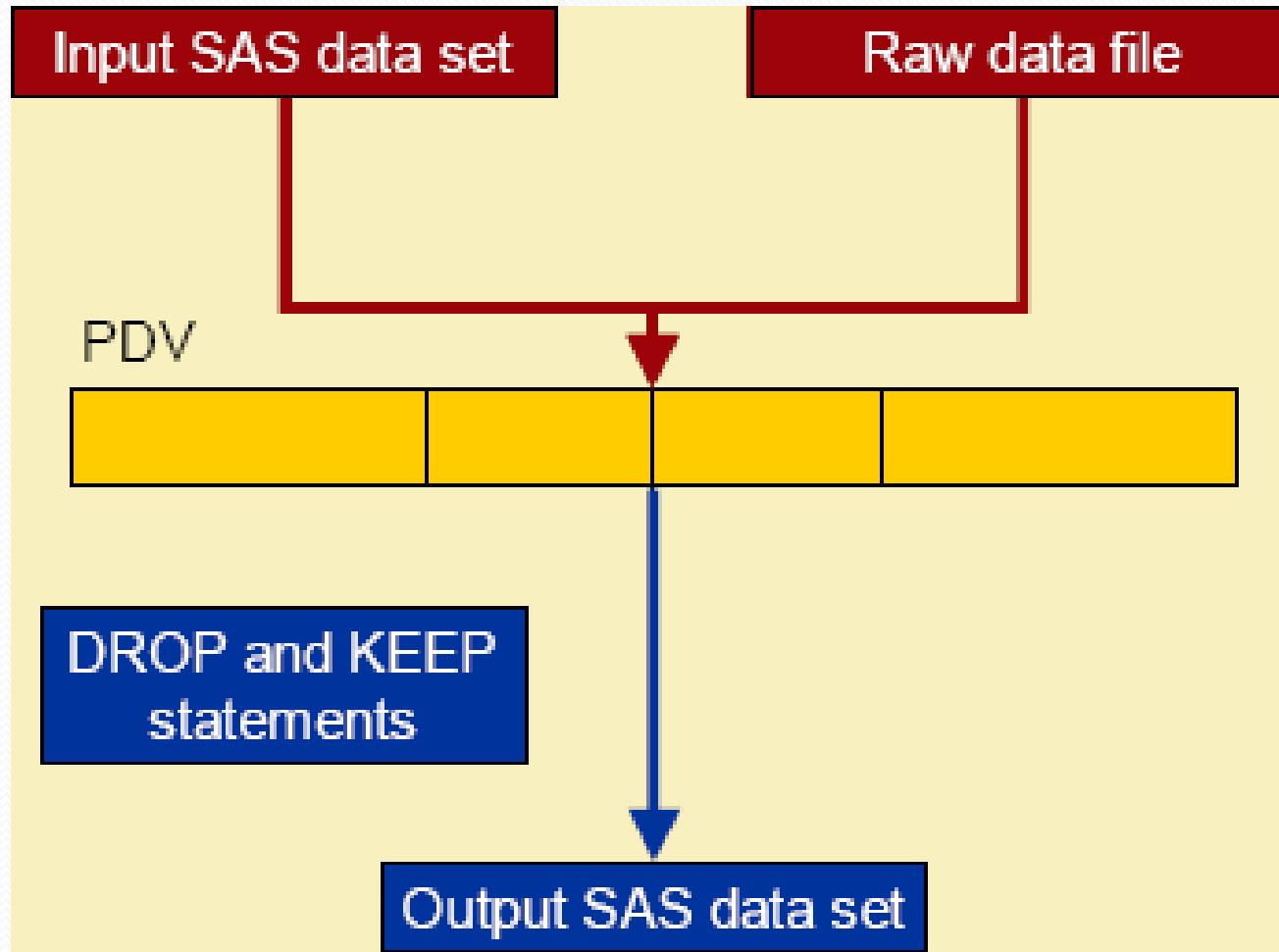
Controlling Variable Output

By default, the SAS System writes all variables from every input data set to every output data set. In the DATA step, the DROP and KEEP statements can be used to control which variables are written to output data sets.

The DROP and KEEP Statements (Review)



The DROP and KEEP Statements (Review)



DROP= and KEEP= Data Set Options

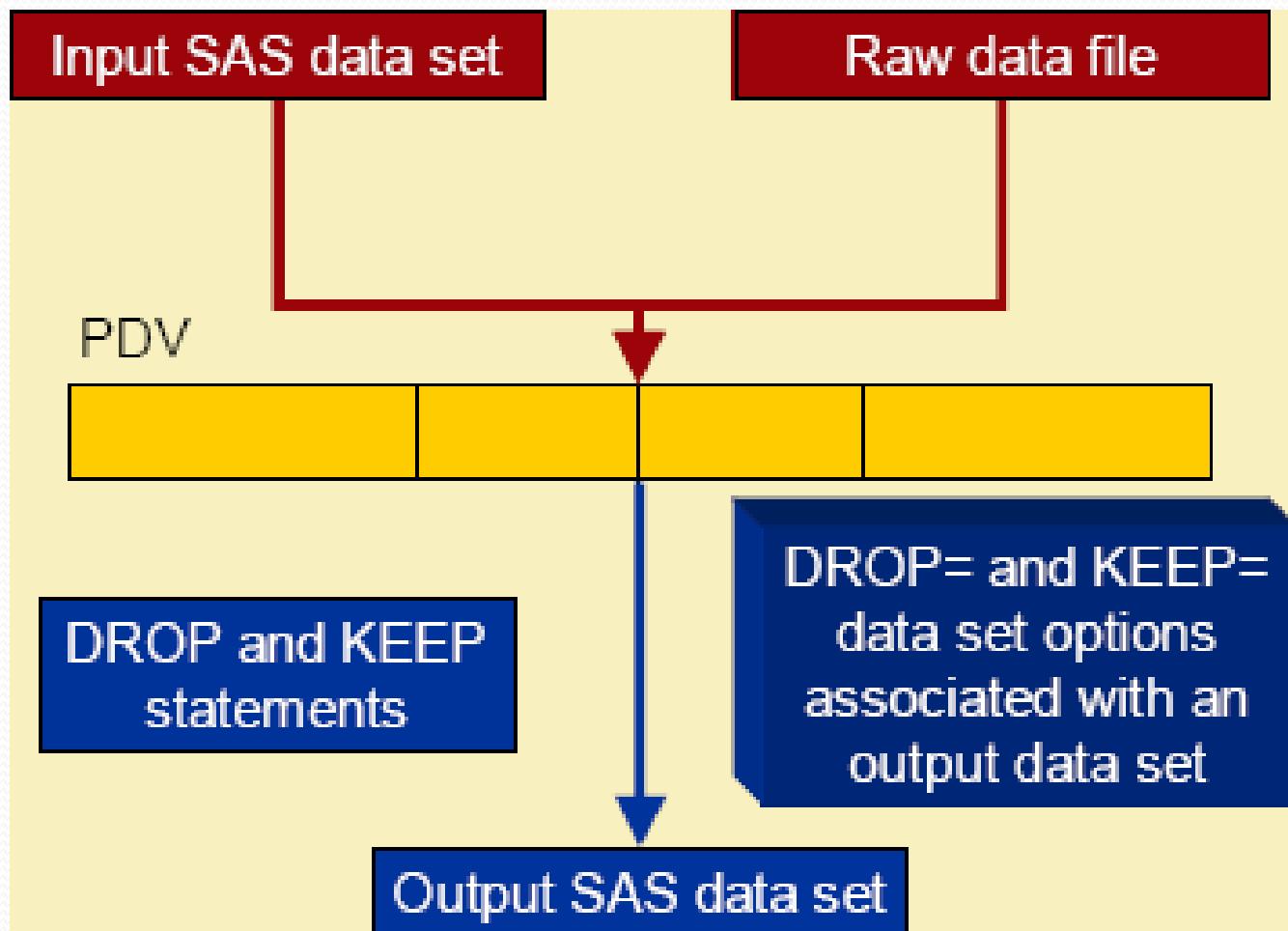
- DROP=

SAS-data-set(DROP=variable-1 variable-2 ...variable-n)

- KEEP=

SAS-data-set(KEEP=variable-1 variable-2 ...variable-n)

DROP= and KEEP= Data Set Options



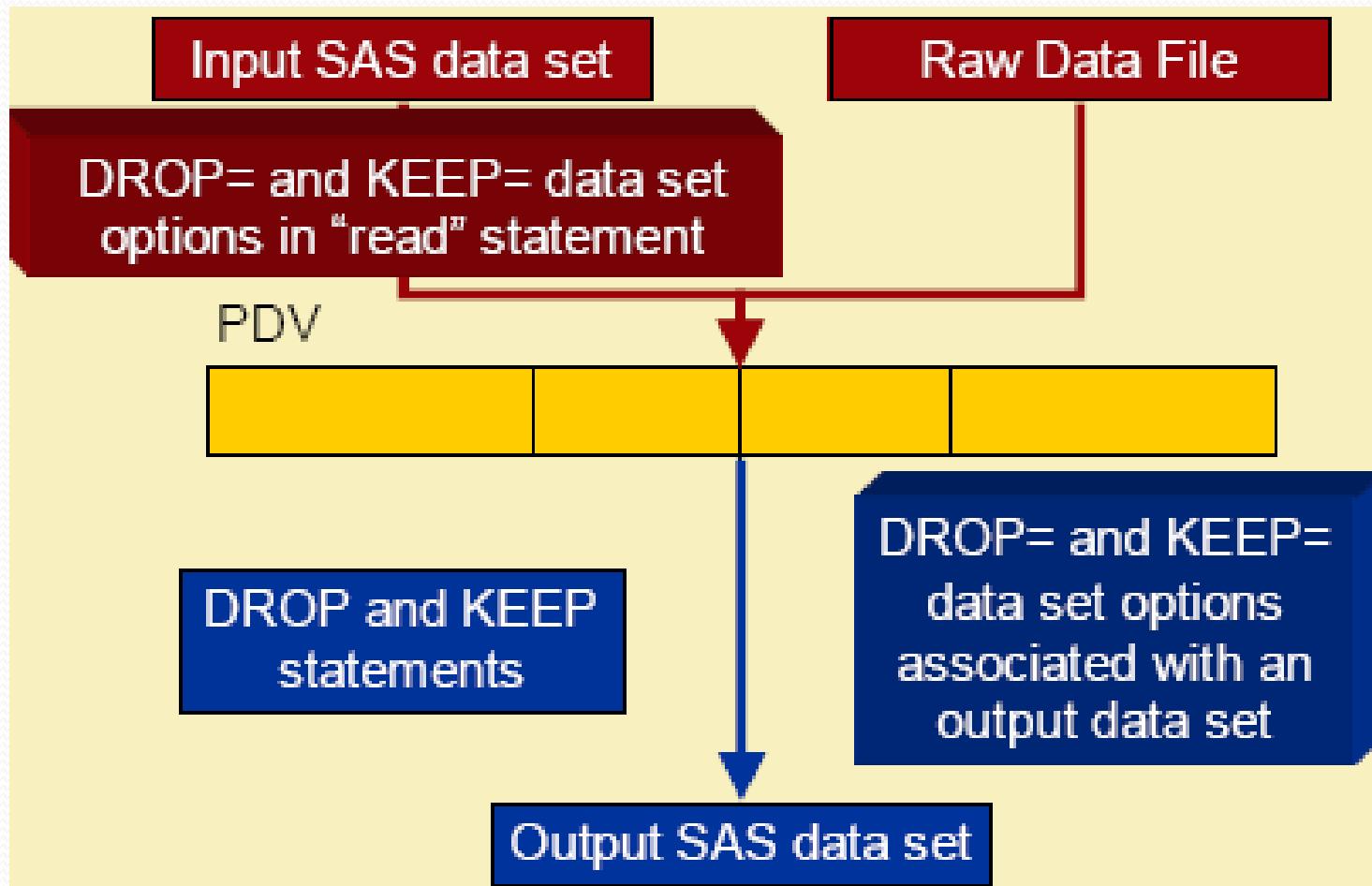
Data2.Military – Partial Proc Print

Type	Code	City	State	Country	Airport
Air Force	NSF	Camp Springs	MD	USA	Andrews Air Force Base
Army	EDG	Edgewood Arsenal	MD	USA	Weide Army Air Field
Army	FME	Fort Meade (Odenton)	MD	USA	Tipton Army Air Field
Naval	NHZ	Brunswick	ME	USA	Brunswick Naval Air
Air Force	LIZ	Limestone	ME	USA	Loring Air Force Base
Air Force	SAW	Gwinn/Marquette	MI	USA	K. I. Sawyer Air Force
Marine	76G	Marine City	MI	USA	Marine City Airport
Naval	NFB	Mount Clemens	MI	USA	Detroit Naval Air
Air Force	SZL	Knob Noster	MO	USA	Whiteman Air Force Base
Air Force	BIX	Biloxi	MS	USA	Keesler Air Force Base
Air Force	CBM	Columbus	MS	USA	Columbus Air Force Base
Naval	NJW	Moscow	MS	USA	Joe Williams Naval
Air Force	GFA	Great Falls	MT	USA	Malmstrom Air Force Base
Air Force	POB	Fayetteville	NC	USA	Pope Air Force Base
Army	FBG	Fort Bragg	NC	USA	Simmons Army Air Field
Air Force	GSB	Goldsboro	NC	USA	Seymour Johnson Air Force
Army	HFF	Hoffmann/Camp Mackal	NC	USA	Mackall Army Air Field

Controlling Variable Output

```
data airforce(drop=Code Type)
    army(drop=City State Country Type)
    navy(drop=Type)
    marines;
set prog2.military;
if Type eq 'Air Force' then
    output airforce;
else if Type eq 'Army' then
    output army;
else if Type eq 'Naval' then
    output navy;
else if Type eq 'Marine' then
    output marines;
run;
```

Controlling Variable Output



Controlling Variable Output

```
data army(keep=Code Airport) ;  
  set prog2.military(drop=City State  
                     Country) ;  
  if Type eq 'Army' then output;  
run;
```

Compiling the Data Step

Variables from prog2.military

Type	Code	City	State	Country	Airport
------	------	------	-------	---------	---------

```
data army(keep=Code Airport);
  set prog2.military(drop=City State
                      Country);
  if Type eq 'Army' then output;
run;
```

PDV

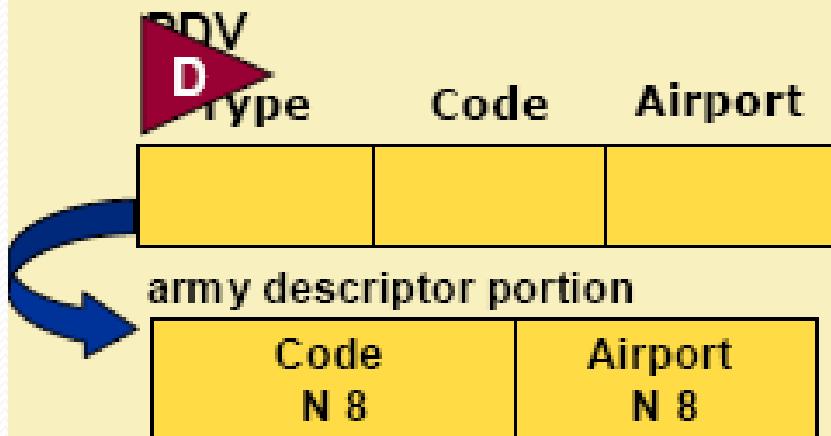
Type	Code	Airport
------	------	---------

--	--	--

Compiling the Data Step

Type	Code	City	State	Country	Airport
------	------	------	-------	---------	---------

```
data army(keep=Code Airport);  
  set prog2.military(drop=City State  
                     Country);  
  if Type eq 'Army' then output;  
run;
```



Controlling Which Observations are Read

The OBS= Data Set Option determines how many observations are read from the input data set

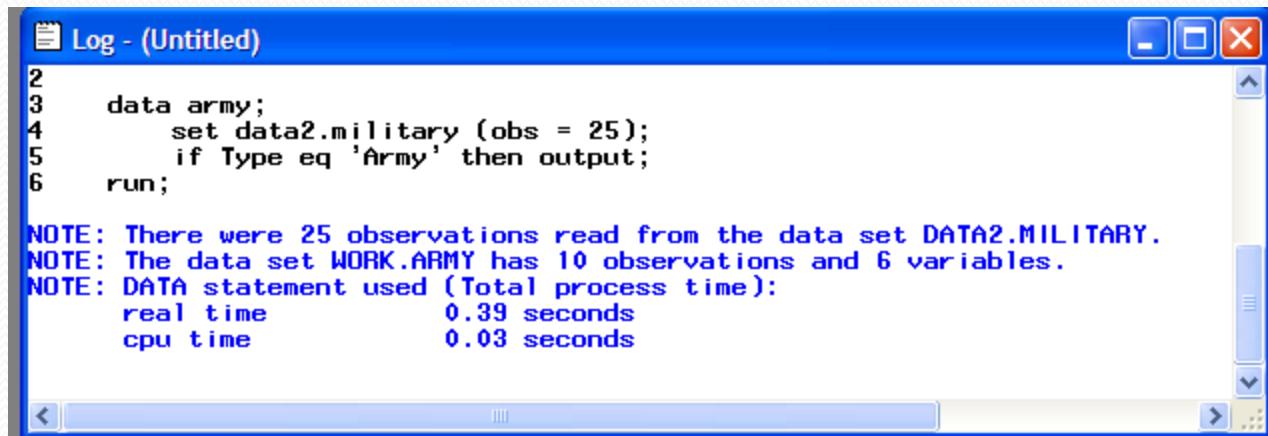
- General Form:

SAS-data-set(OBS=n)

- Example:

```
data army;
    set prog2.military(obs=25);
    if Type eq 'Army' then output;
run;
```

Controlling Which Observations are Read



The screenshot shows a Windows-style application window titled "Log - (Untitled)". The window contains the following content:

```
2
3   data army;
4     set data2.military (obs = 25);
5     if Type eq 'Army' then output;
6   run;

NOTE: There were 25 observations read from the data set DATA2.MILITARY.
NOTE: The data set WORK.ARMY has 10 observations and 6 variables.
NOTE: DATA statement used (Total process time):
      real time          0.39 seconds
      cpu time           0.03 seconds
```

Controlling Which Observations are Read

The FIRSTOBS= Data Set Option allows you to skip records at the top of the input file. Can be useful to eliminate “header” information.

- General Form:

SAS-data-set(FIRSTOBS=n)

- Example (starts reading at 2nd record):

Controlling Which Observations are Read

```
8  
9  data army;  
10     set data2.military (firstobs = 11 obs = 25);  
11     if Type eq 'Army' then output;  
12  run;  
  
NOTE: There were 15 observations read from the data set DATA2.MILITARY.  
NOTE: The data set WORK.ARMY has 5 observations and 6 variables.  
NOTE: DATA statement used (Total process time):  
      real time            0.06 seconds  
      cpu time             0.01 seconds
```

PSTAT 130

SAS Base Programming

Summer 2017

William Qiu

Department of Statistics and Applied Probability

UCSB

Lecture 13

Lecture Outline

- Learn another way to create a running total
 - Sum statement
- Clarify First.Obs and Last.Obs variables
 - With Single and Multiple BY variables
- Writing to external files
 - Creating a comma-separated (or Excel) data file - ODS CSVALL
 - Creating a custom report – DATA _NULL_
- Reading Non-standard Data files
 - Delimiters other than spaces
 - Mixed Format Records
 - Multiple records per line

The SUM Statement

The SUM statement (General Form)

Variable + expression;

- creates the variable on the left side of the plus sign if it does not already exist
- initializes the variable to zero before the first iteration of the DATA step
- automatically retains the variable
- adds the value of the *expression to the variable at execution*
- ignores missing values.
- is an alternative to using the RETAIN statement

Accumulating Totals

```
data mnthtot2;  
  set prog2.daysales2;  
  Mth2Dte+SaleAmt;  
run;
```

Accumulating
Variable

SaleDate	SaleAmt	Mth2Dte
01APR2001	498.49	498.49
02APR2001	946.50	1444.99
03APR2001	994.97	2439.96
04APR2001	564.59	3004.55
05APR2001	783.01	3787.56
06APR2001	228.82	4016.38
07APR2001	930.57	4946.95
08APR2001	211.47	5158.42
09APR2001	156.23	5314.65

Accumulating totals for BY Groups

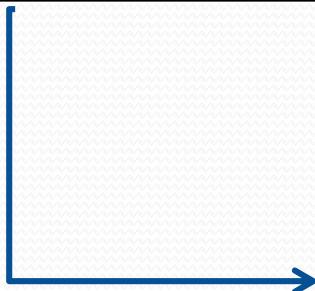
After sorting the data on the BY variables, there is a three-step process for accumulating totals.

1. Set the Accumulator Variable to zero at the start of each BY group.
2. Increment the accumulating variable with a sum statement (automatically RETAINS).
3. Output only the last observation of each BY group.

First. & Last. Example

Div	Salary	DivSal
APTOPS	20000	20000
APTOPS	100000	120000
APTOPS	50000	170000
FINACE	25000	25000
FINACE	20000	45000
FINACE	23000	68000
FINACE	27000	95000
SALES	10000	10000
SALES	12000	22000

```
data divsals(keep=Div DivSal);  
  set salsort;  
  by Div;  
  if First.Div then DivSal=0;  
  DivSal+Salary;  
  if Last.Div;  
run;
```



Div	DivSal
APTOPS	410000
FINACE	163000
FLTOPS	318000
HUMRES	181000
SALES	373000

Sorting using Multiple BY variables

```
proc sort data=prog2.regsals out=regsort;  
    by Region Div;  
run;
```



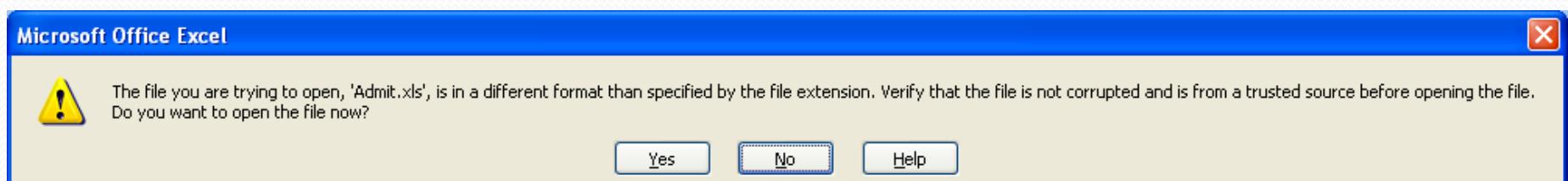
Region	Div	First. Region	Last. Region	First. Div	Last.Div
C	APTOPS	1	0	1	0
C	APTOPS	0	1	0	1
E	APTOPS	1	0	1	0
E	APTOPS	0	0	0	0
E	APTOPS	0	0	0	1
E	FINACE	0	0	1	0

Writing Data to an Excel file

- You can use the ODS HTML and PROC PRINT statements to convert a SAS data set into an Excel file

```
ods html file='Admit.xls';
title;
proc print data=ia.admit noobs;
run;
ods html close;
```

- Note: SAS will warn you that this file is not of .xls type. Just ignore this warning.



Writing a Comma-Separated File

- Use the ODS CSVALL statement to convert a SAS data set to a comma-separated values (CSV) file
- Most other programs can read in a CSV file, including Excel

```
ods csvall file='admit.csv';
title;
proc print data=ia.admit noobs ;
run;
ods csvall close;
```

- You can use titles, footnotes, labels and formats to change the appearance of the data

CSVALL Output

```
"ID","Name","Sex","Age","Date","Height","Weight","ActLevel","Fee"  
2458,"Murray, W","M",27,1,72,168,"HIGH",85.20  
2462,"Almers, C","F",34,3,66,152,"HIGH",124.80  
2501,"Bonaventure, T","F",31,17,61,123,"LOW",149.75  
2523,"Johnson, R","F",43,31,63,137,"MOD",149.75  
2539,"LaMance, K","M",51,4,71,158,"LOW",124.80  
2544,"Jones, M","M",29,6,76,193,"HIGH",124.80  
2552,"Reberson, P","F",32,9,67,151,"MOD",149.75  
2555,"King, E","M",35,13,70,173,"MOD",149.75  
2563,"Pitts, D","M",34,22,73,154,"LOW",124.80  
2568,"Eberhardt, S","F",49,27,64,172,"LOW",124.80  
2571,"Nunnelly, A","F",44,19,66,140,"HIGH",149.75  
2572,"Oberon, M","F",28,17,62,118,"LOW",85.20  
2574,"Peterson, V","M",30,6,69,147,"MOD",149.75  
2575,"Quigley, M","F",40,8,69,163,"HIGH",124.80  
2578,"Cameron, L","M",47,5,72,173,"MOD",124.80  
2579,"Underwood, K","M",60,22,71,191,"LOW",149.75  
2584,"Takahashi, Y","F",43,29,65,123,"MOD",124.80  
2586,"Derber, B","M",25,23,75,188,"HIGH",85.20  
2588,"Ivan, H","F",22,20,63,139,"LOW",85.20  
2589,"Wilcox, E","F",41,16,67,141,"HIGH",149.75  
2595,"Warren, C","M",54,7,71,183,"MOD",149.75
```

More about the Data Step

READING FROM AN EXTERNAL FILE

The **DATA** statement begins the DATA step.

The **INFILE** statement identifies an external file to read with an **INPUT** statement.

The **INPUT** statement describes the arrangement of values in the input data record.

WRITING TO AN EXTERNAL FILE

The **DATA** statement begins the DATA step.

The **FILE** statement identifies an external file to write with a **PUT** statement.

The **PUT** statement describes the arrangement of values in the output data record.

- Note: INFILE vs. FILE, INFORMAT vs. FORMAT, INPUT vs. PUT
 - One controls data coming into SAS, the other controls data leaving SAS

Creating Custom Reports

- You can use the PUT statement with the DATA _NULL_ method to write output in any format you choose.
 - DATA _NULL_ uses the Data Step, but does NOT create a data set.
 - Instead, it writes output to a specified file using PUT statements
 - The PUT statements control the exact location and format of information in the Output file

DATA _NULL_ Program and Output

```
data _null_;
  set ia.admit;
  if Sex = 'M' then Salutation = 'Mr.'; Else Salutation = 'Ms.';
  file 'admitreport.doc' PRINT;
  title;
  put @5 'Dear ' Salutation ' ' Name ':'
    //@5 'Your weight at admission was ' Weight 3. ' pounds.'
    //@5 'Your height at admission was ' Height 2. ' inches.'
    //@10 'Thank you for your ' fee dollar8.2 ' payment!';
  put _page_;
run;
```

Dear Mr. Murray, W :

Your weight at admission was 168 pounds.

Your height at admission was 72 inches.

Thank you for your \$85.20 payment!

Colon Modifier – “Short” Variables

Use the Colon Modifier to read each value only as far as the next delimiter

- Allows you to use Informats with List input, but handle non-standard data values (i.e., values that do not match the format)

50001 4feb1989 132 530
50002 11nov1989 152 540
50003 22oct1991 90 530
50004 4feb1993 172 550
50005 24jun1993 170 510
50006 20dec1994 180 520

```
data airplanes;
    infile 'airdata.txt';
    input ID $  
          InService : date9.  
          PassCap CargoCap;  
run;
```

Colon Modifier – Character Example

The default length for character variables is 8. Use the Colon Modifier to read character values longer than 8 characters. The ‘:’ tells SAS to read in that variable until it reaches a space:

```
data students;
input Name : $20. Gender : $6. Age 2. ;
datalines;
elizabeth female 23
jonathon male 22
zack male 19
;
run;
```

INFILE Statement Options

Problem	Option
Non-blank delimiters	DLM=' <i>delimiter(s)</i> '
Missing data at end of row	MISSOVER
Missing data represented by consecutive delimiters and/or Embedded delimiters where values are surrounded by double quotes	DSD

Using Other Delimiters

- A delimiter is used to separate values of adjacent variables
- In SAS, the default delimiter is a “space”, but many other programs use commas or tabs

Space delimited

**John 22 M
Betty 19 F
Dave 18 M**

Comma delimited

**John,22,M
Betty,19,F
Dave,18,M**

vs.

The DLM= option

- Use the DLM= option to specify a delimiter
 - The delimiter can be any character
- You can use the DLM option with
 - INFILE statements (when reading in data) or
 - with FILE statements (when writing out data)

```
data new;  
    infile 'students.dat' dlm=',';  
    input Name $ Age Gender $;  
run;
```



```
Mary,21,Female  
John,22,Male  
David,18,Male
```

Missing Data at the End of a Row

By default, when there is missing data at the end of a row,

1. SAS loads the next record to finish the observation
2. a note is written to the log
3. SAS loads a new record at the top of the DATA step and continues processing.

Missing Data at the End of a Row

Raw Data File

```
50001 , 4feb1989,132  
50002, 11nov1989,152, 540  
50003, 22oct1991,90, 530  
50004, 4feb1993,172  
50005, 24jun1993, 170, 510  
50006, 20dec1994, 180, 520
```

```
data airplanes3;  
length ID $ 5;  
infile 'raw-data-file'  
      dlm=',';  
input ID $  
      InService : date9.  
      PassCap CargoCap;  
run;
```

Input Buffer



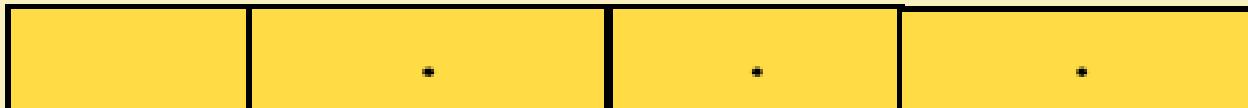
PDV

ID
\$ 5

InService
N 8

PassCap
N 8

CargoCap
N 8



Missing Data at the End of a Row

```
proc print data=airplanes3 noobs;  
run;
```

PROC PRINT Output

ID	In Service	Pass Cap	Cargo Cap
50001	10627	132	50002
50003	11617	90	530
50004	12088	172	50005
50006	12772	180	520

Solution – MISSOVER option

- The MISSOVER option tells SAS NOT to continue reading from the next line if not all variables have values

```
data airplanes3;
    infile 'airdata.txt' dlm=' ', ' missover';
    input ID $  
InService : date9.  
PassCap CargoCap;
run;
```

In	Pass	Carg	
ID	Service	Cap	Cap
50001	10627	132	.
50002	10907	152	540
50003	11617	90	530
50004	12088	172	.
50005	12228	170	510
50006	12772	180	520

The DSD Option

The DSD option:

- sets the default delimiter to a comma
- treats consecutive delimiters as missing values
- enables SAS to read values with embedded delimiters if the value is surrounded by double quotes.

5 0 0 0 1 , 4feb1989	,,	5 3 0
----------------------	----	-------

Mixed Record Types

What if not all records have the same format?

Data file

```
101 USA 1-20-1999 3295.50
3034 EUR 30JAN1999 1876,30
101 USA 1-30-1999 2938.00
128 USA 2-5-1999 2908.74
1345 EUR 6FEB1999 3145,60
109 USA 3-17-1999 2789.10
```

Desired Output

Sales ID	Location	Sale Date	Amount
101	USA	14264	3295.50
3034	EUR	14274	1876.30
101	USA	14274	2938.00
128	USA	14280	2908.74
1345	EUR	14281	3145.60
109	USA	14320	2789.10

The Trailing @ modifier

- The Trailing @ modifier tells SAS to hold the current input line for further processing:

```
input SalesID $ Location $ @;
  if location='USA' then
    input SaleDate : mmddyy10.
      Amount;
  else if Location='EUR' then
    input SaleDate : date9.
      Amount : commax8.;
```

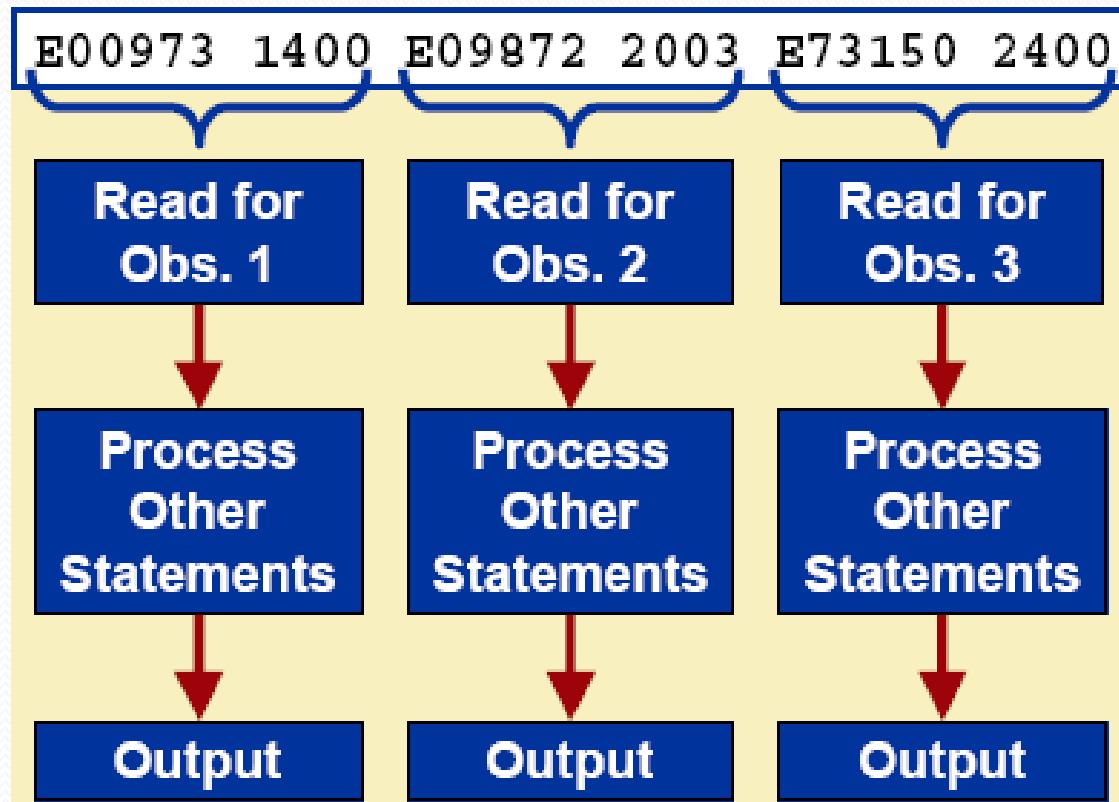
Mixed Record Types

```
proc print data=sales noobs;  
run;
```

PROC PRINT Output

Sales ID	Location	Sale Date	Amount
101	USA	14264	3295.50
3034	EUR	14274	1876.30
101	USA	14274	2938.00
128	USA	14280	2908.74
1345	EUR	14281	3145.60
109	USA	14320	2789.10

Multiple Observations Per Line



The Double Trailing @@ modifier

```
data work.retire;  
  length EmpID $ 6;  
  infile 'raw-data-file';  
  input EmpID $ Contrib @@;  
run;
```

Hold until end
of record.

Trailing @ versus Double Trailing @@

Option	Effect
Trailing @ INPUT var-1... @;	Holds raw data record until 1) an INPUT statement with no trailing @ 2) the bottom of the DATA step.
Double trailing @ INPUT var-1 ... @@;	Holds raw data records in the input buffer until SAS reads past the end of the line.

PSTAT 130

SAS Base Programming

Summer 2017

William Qiu

Department of Statistics and Applied Probability

UCSB

Lecture 14

Lecture Outline

- Variable Lists
- Using SAS functions
 - Parse text data
 - Truncate numeric data
- Converting data
 - numeric to character
 - character to numeric
- Do loops
 - Non-iterative
 - Iterative
 - Condition

SAS Variable Lists

SAS enables you to use the following variable lists:

- numbered range lists
- name range lists
- name prefix lists
- special SAS name lists.

Numbered Range List

- Allows you to refer to a set variables that all start with the same variable name, and each end with a number
- Example:

```
Input Salesman $ Week1 Week2 Week3 Week4...Week52;
```

- A range list refers to all the weeks as follows:

```
Proc Print;  
  Var Week1-Week52;  
Run;
```

Name Range List

- Allows you to refer to a series of variables that appear in consecutive order in the dataset
- Example:

```
Input Salesman $ Mon Tues Wed Thur Fri Sat Sun Region;
```

- A range list refers to all the weeks as follows:

```
Proc Print;  
Var Mon--Sun;  
Run;
```

- Note: **Salesman-Numeric-Region** includes only Numeric variables in the range, and **Salesman-Character-Region** includes on character variables in the range

Name Prefix List

- Some SAS functions and statements allow you to refer to all variables that begin with a specified character string:
- Example:

```
sum(of SALES:)
```

tells SAS to calculate the sum of all the variables that begin with "SALES," such as SALES_JAN, SALES_FEB, and SALES_MAR.

Special SAS Names List

- Special SAS name lists include

NUMERIC specifies all numeric variables that are already defined in the current DATA step.

CHARACTER specifies all character variables that are currently defined in the current DATA step.

ALL specifies all variables that are currently defined in the current DATA step.

SAS Functions

A SAS function is often categorized by the type of data manipulation performed:

- truncation
- character
- date and time
- mathematical
- trigonometric
- special
- sample statistics
- arithmetic
- financial
- random number
- state and ZIP code.

Example: Making Mailing Labels

The **data2.freqflyers** data set contains information about frequent flyers.

ID	Name	Address1	Address2
F31351	Farr, Sue	15 Harvey Rd.	Macon, Bibb, GA, 31298
F161	Cox, Kay B.	163 McNeil Pl.	Kern, Pond, CA, 93280
F212	Mason, Ron	442 Glen Ave.	Miami, Dade, FL, 33054
F25122	Ruth, G. H.	2491 Brady St.	Munger, Bay, MI, 48747

Use this data set to create another data set suitable for mailing labels.

FullName	Address1	Address2
Ms. Sue Farr	15 Harvey Rd.	Macon, GA 31298
Ms. Kay B. Cox	163 McNeil Pl.	Kern, CA 93280
Mr. Ron Mason	442 Glen Ave.	Miami, FL 33054
Mr. G. H. Ruth	2491 Brady St.	Munger, MI 48747

The LENGTH Function

- The LENGTH function returns the number of characters in a string

```
NewVar = LENGTH(string);
```

- Example
 - $\text{LENGTH}(\text{"SMITH, JOHN"}) = 11$

The INDEX Function

- The INDEX function returns the position of specific character (or characters) within a string

```
NewVar = INDEX(string,target);
```

- Example
 - $\text{INDEX}(\text{'SMITH-JOHN'}, \text{'-'}) = 6$
- Returns ZERO if target isn't in string

The SUBSTR Function (Right Side)

- The SUBSTR Function extracts a portion of a character variable:

NewVar=SUBSTR(string,start<,length>);

- Example:
 - $\text{SUBSTR}(\text{"PSTAT130"}, 6, 3) = \text{"130"}$

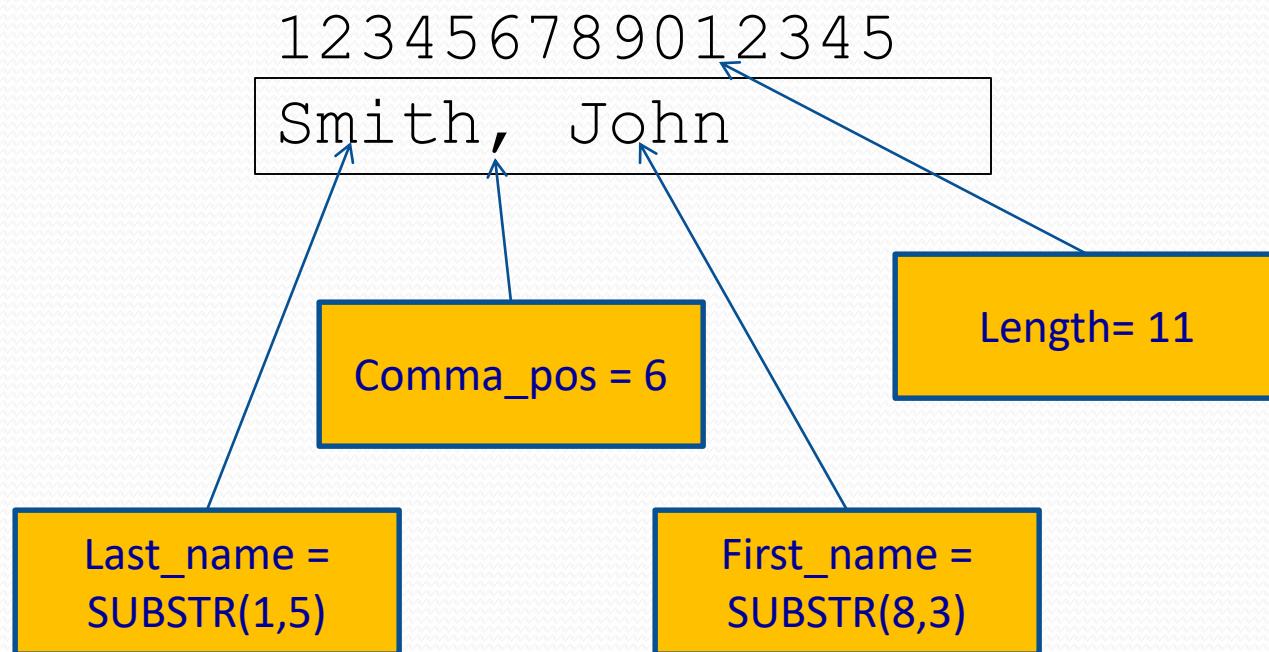
Parsing a Text String

- How can we turn “SMITH, JOHN” into “JOHN SMITH”?
 - Find the location of the comma
 - Last Name = text before the comma
 - First Name = Text after the comma

Putting it all together

```
data mailabels;
    input name $25. ;
    name_len = length(name) ;
    comma_pos = index(name,',') ;
    last_name = substr(name,1,comma_pos-1) ;
    first_name = substr(name,comma_pos+2,name_len-comma_pos) ;
datalines;
Smith, John
Johnson, Davy
Quincy, Elizabeth
;
run;
```

Name Example



Results

name	name_len	comma_pos	last_name	first_name
Smith, John	11	6	Smith	John
Johnson, Davy	13	8	Johnson	Davy
Quincy, Elizabeth	17	7	Quincy	Elizabeth

The SCAN Function

- The SCAN function “parses” a character string into a set of “words” using a delimiter.

```
NewVar=SCAN(string,n<,delimiters>)
```

- Example:
 - $\text{SCAN}(\text{"Smith, John"}, 1) = \text{"Smith"}$
 - $\text{SCAN}(\text{"Smith, John"}, 2) = \text{"John"}$

First “word”

Second “word”

When the SCAN function is used,

- The default delimiters are:
blank . < (+ | & ! \$ *) ; - / , % | ¢
- delimiters before the first word have no effect
- any character or set of characters can serve as delimiters
- two or more contiguous delimiters are treated as a single delimiter
- a missing value is returned if there are fewer than n words in *string*
- if n is negative, *SCAN* selects the word in the character string starting from the end of *string*.

Concatenation Operator

- Use the `||` operator to “concatenate” or join two strings together
- Examples:
 - `“John” || “Smith” = “JohnSmith”`
 - `“John” || “ “ || “Smith” = “John Smith”`

The TRIM Function

A Better Mailing Label Program

```
data test;  
    input name $25.;  
    last_name = scan(name,1);  
    first_name = scan(name,2);  
datalines;  
Smith, John  
Johnson, Davy  
Quincy, Elizabeth  
;  
run;
```



name	last_name	first_name
Smith, John	Smith	John
Johnson, Davy	Johnson	Davy
Quincy, Elizabeth	Quincy	Elizabeth

A Search Application

The **data2.ffhistory** data set contains information about the history of each frequent flyer.

- This history information consists of
- each membership level the flyer has attained (bronze, silver, or gold)
- the year the flier attained each level.

Create a report that shows all frequent flyers who have attained silver membership status and the year each of them became silver members.

A Search Application

prog2.ffhistory			
ID	Status		Seat Pref
F31351	Silver 1998,Gold 2000		AISLE
F161	Bronze 1999		WINDOW
F212	Bronze 1992,silver 1995		WINDOW
F25122	Bronze 1994,Gold 1996,Silver 1998		AISLE

- How would you
 - Parse the membership levels?
 - Parse the year each level was obtained?
 - Select flyers that have achieved Silver status?

Numeric Truncation Functions

Selected functions that truncate numeric values include:

- ROUND function
- CEIL function
- FLOOR function
- INT function.

The ROUND Function

- The ROUND function performs a traditional Round Up/Round Down operation:

```
NewVar=ROUND(argument<,round-off-unit>);
```

- Examples:

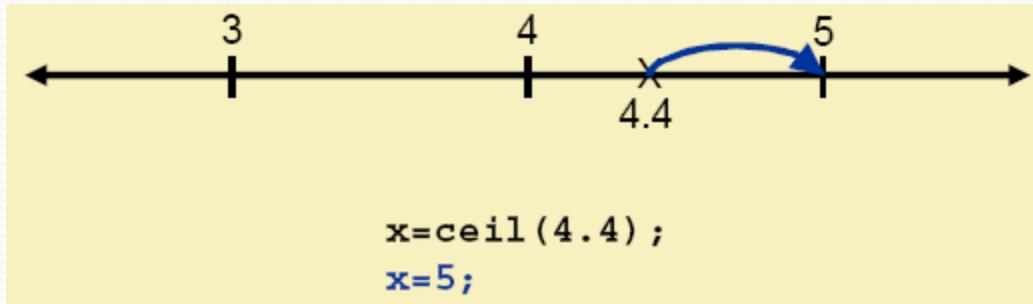
```
data truncate;
  NewVar1=round(12.12);
  NewVar2=round(42.65,.1);
  NewVar3=round(6.478,.01);
  NewVar4=round(96.47,10);
run;
```

NEWVAR1	NEWVAR2	NEWVAR3	NEWVAR4
12	42.7	6.48	100

The CEIL Function

- The CEIL Function performs a Round Up operation only:

```
NewVar = CEIL(argument);
```

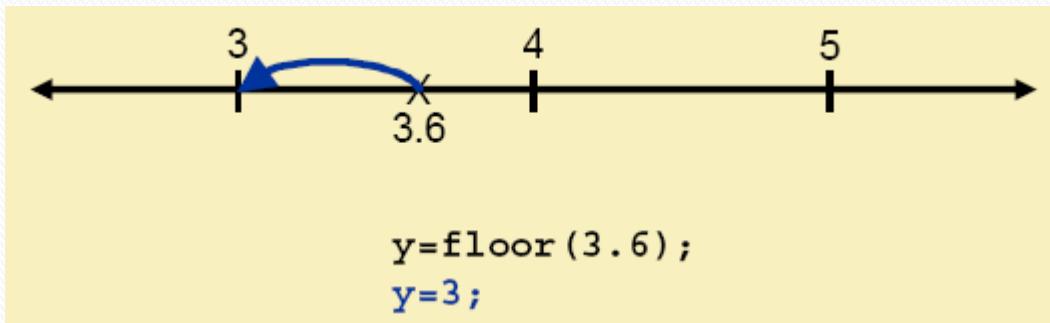


- Note: $\text{CEIL}(4) = 4$

The FLOOR Function

- The FLOOR Function performs a Round Down operation only:

```
NewVar = FLOOR(argument);
```



- Note: $\text{FLOOR}(4) = 4$

The INT Function

- The INT function removes any decimals and returns the next smaller integer:

```
NewVar = INT(argument);
```

- Examples:

- $\text{INT}(3.2) = 3$
- $\text{INT}(-4.8) = -5$
- For positive numbers, INT = FLOOR
- For negative numbers, INT = CEIL

```
data truncate;
  Var1=-6.478;
  NewVar1=ceil(Var1);
  NewVar2=floor(Var1);
  NewVar3=int(Var1);
run;
```

VAR1	NEWVAR1	NEWVAR2	NEWVAR3
-6.478	-6	-7	-6

Data Conversion

In many applications, you may need to convert one data type to another.

- You may need to read in digits in character form and convert them to a numeric value.
- You may need to read in a numeric value and write it out as a character string.

You can convert data types

- implicitly by allowing the SAS System to do it for you
- explicitly with these functions:
 - INPUT character-to-numeric conversion
 - PUT numeric-to-character conversion.

Automatic Character-to-Numeric Conversion

The **data2.salary1** data set contains a character variable **Grosspay**. Compute a 10 percent bonus for each employee.

What will happen when the character values of **Grosspay** are used in an arithmetic expression?

prog2.salary1	
ID	GrossPay
\$11	\$5
201-92-2498	52000
482-87-7945	32000
330-40-7172	49000

```
data bonuses;
  set prog2.salary1;
  Bonus=.10*GrossPay;
run;
```

Automatic Character-to-Numeric Conversion

SAS automatically converts a character value to a numeric value when the character value is used in a numeric context, such as

- assignment to a numeric variable
- an arithmetic operation
- logical comparison with a numeric value
- a function that takes numeric arguments.

The INPUT Function

- The INPUT function uses a SAS format (informat) to convert a character string into a number:

```
NumVar=INPUT(source,informat);
```

```
data conversion;  
  CVar1='32000';  
  CVar2='32,000';  
  CVar3='03may2008';  
  CVar4='050308';  
  NVar1=input(CVar1,5.);  
  NVar2=input(CVar2,comma6.);  
  NVar3=input(CVar3,date9.);  
  NVar4=input(CVar4,mmdyy6.);  
run;
```



CVar1	CVar2	CVar3	CVar4	NVar1
32000	32,000	03may2008	050308	32000
NVar2	NVar3	NVar4		
32000	17655	17655		

Failed Character-to-Numeric Conversion

- This approach will not work because Grosspay has non-numeric characters (commas).

```
data bonuses;  
  set prog2.salary2;  
  Bonus=.10*GrossPay;  
run;  
  
proc print data=bonuses;  
run;
```

PROC PRINT Output

ID	GrossPay	Bonus
201-92-2498	52,000	.
482-87-7945	32,000	.
330-40-7172	49,000	.

Explicit Character-to-Numeric Conversion

```
data bonuses;
  set prog2.salary2;
  Bonus=.10*input(GrossPay,comma6.);
run;

proc print data=bonuses;
run;
```

PROC PRINT Output

ID	GrossPay	Bonus
201-92-2498	52,000	5200
482-87-7945	32,000	3200
330-40-7172	49,000	4900

- Note: You cannot convert data by assigning the converted variable value to a variable with the same name.

```
GrossPay=input(GrossPay,comma6.);
```

Automatic Numeric-to-Character Conversion

SAS automatically converts a numeric value to a character value when the numeric value is used in a character context, such as

- assignment to a character variable
- a concatenation operation
- a function that accepts character arguments.

The PUT Function

- The PUT function uses a SAS format to convert a number into a character string:

CharVar=PUT(source,format);

- Example
 - Areacode = 805
 - AreaChar = PUT(Areacode,3.) = “805”

PUT vs. INPUT Functions

- These are NOT the same as the PUT and INPUT statements used in a Data Step
- Functions
 - The INPUT function converts character data into numeric data
 - The PUT function converts numeric data into character data
- Statements in Data Step
 - The INPUT statement controls how data are read into a data set
 - The PUT statement controls how data are written out of a data step

Do Loop Processing

There are four kinds of DO Loops in SAS:

- DO-END
 - executes statements as a unit, usually as a part of IF-THEN/ELSE statements.
- Iterative DO
 - executes a group of statements repetitively based on the value of an index variable.
- DO WHILE
 - executes a group of statements repetitively as long as the condition that you specify remains true. The condition is checked before each iteration of the loop.
- DO UNTIL
 - executes a group of statements repetitively until the condition that you specify is true. The condition is checked after each iteration of the loop.

DO-END

The simple IF-THEN statement does NOT require a DO-END statement:

```
IF sex = 'Male' THEN Abbreviation = 'Mr.' ;
```

BUT if you want to do **more than one** thing, you need a DO-END statement

```
IF sex = 'Male' THEN  
DO;  
    Abbreviation = 'Mr.' ;  
    Salutation = 'Sir' ;  
END ;
```

Performing Repetitive Calculations

On January 1 of each year, \$5,000 is invested in an account.

Determine the value of the account after three years
based on a constant annual interest rate of 7.5 percent.

```
data invest;  
    do Year=2001 to 2003;  
        Capital+5000;  
        Capital+(Capital*.075);  
    end;  
run;
```



Year	Capital
2004	17364.61

Performing Repetitive Calculations

Generate the list of yearly increases

```
data invest;  
    do Year=2001 to 2003;  
        Capital+5000;  
        Capital+(Capital*.075);  
        output;  
    end;  
run;
```



Year	Capital
2001	5375.00
2002	11153.13
2003	17364.61

Performing Repetitive Calculations

Use the iterative DO to repeat an operation a fixed number of times:

```
data iterate;
    Summation = 0;
    Factorial = 1;
    n = 10;
    do i = 1 to n;
        Summation = Summation + i;
        Factorial = Factorial * i;
        output;
        retain;
    end;
run;
```

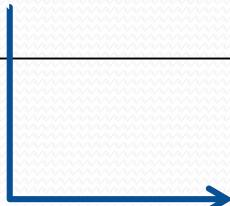
The Iterative DO Statement- Output

Obs	i	Summation	Factorial
1	1	1	1
2	2	3	2
3	3	6	6
4	4	10	24
5	5	15	120
6	6	21	720
7	7	28	5040
8	8	36	40320
9	9	45	362880
10	10	55	3628800

A Forecasting Application (Review)

```
data forecast;
set data2.growth(rename=(NumEmps>NewTotal));
Increase = .1;
Year=1;
NewTotal=NewTotal*(1+Increase);
output;
Year=2;
NewTotal=NewTotal*(1+Increase);
output;
Year=3;
NewTotal=NewTotal*(1+Increase);
output;
run;
```

```
data forecast;
  set prog2.growth(rename=(NumEmps>NewTotal));
  Increase = .1;
  do Year=1 to 3;
    NewTotal=NewTotal*(1+Increase);
    output;
  end;
run;
```



The Iterative DO Statement

- The Index Variable can start or stop on any number

```
do i = 5 to 20;  
    <SAS Statements>  
end;
```

- You can iterate in multiples using the BY option

```
data odd numbers;  
    do i = 1 to 100 by 2;  
        output;  
    end;  
run;
```

- Index variables can be User-Defined lists:

```
do Day = 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun';  
    output;  
end;
```

The Iterative DO Statement

How many times will each DO loop execute?

```
do Month='JAN', 'FEB', 'MAR';
   3 times.
do Fib=1,2,3,5,8,13,21;
   7 times.
do i=Var1,Var2,Var3;
   3 times.
do j=BeginDate to Today() by 7;
   Unknown. The number of iterations depends
   on the values of BeginDate and Today().
do k=Test1-Test50;
   1 time. A single value of k is determined
   by subtracting Test50 from Test1.
```

DO WHILE and DO UNIT Loops

A DO-WHILE Loop executes a group of statements repetitively as long as the condition that you specify remains true. The condition is checked before each iteration of the loop.

A DO-UNTIL Loop executes a group of statements repetitively until the condition that you specify is true. The condition is checked after each iteration of the loop.

Conditional Iterative Processing

Determine the number of years it would take for an account to exceed \$1,000,000 if \$5,000 is invested annually at 7.5 percent interest.

```
data invest;
    do until(Capital>1000000) ;
        Year+1;
        Capital+5000;
        Capital+(Capital*.075);
    end;
run;
```



Capital	Year
1047355.91	38

Iterative DO with Conditional Clause

Determine the return of the account again.

Stop the loop if 25 years is reached or more than \$250,000 is accumulated.

```
data invest;
  do Year=1 to 25 until(Capital>250000);
    Capital+5000;
    Capital+(Capital*.075);
  end;
run;

proc print data=invest noobs;
run;
```



Year	Capital
21	255594.86