

DISCUSSION DOCUMENT



Mu Sigma

SAS Workshop

Chicago, IL
Bangalore, India
Month 2005
www.mu-sigma.com

Proprietary Information

"This document and its attachments are confidential. Any unauthorized copying, disclosure or distribution of the material is strictly forbidden"



DISCUSSION DOCUMENT



Mu Sigma

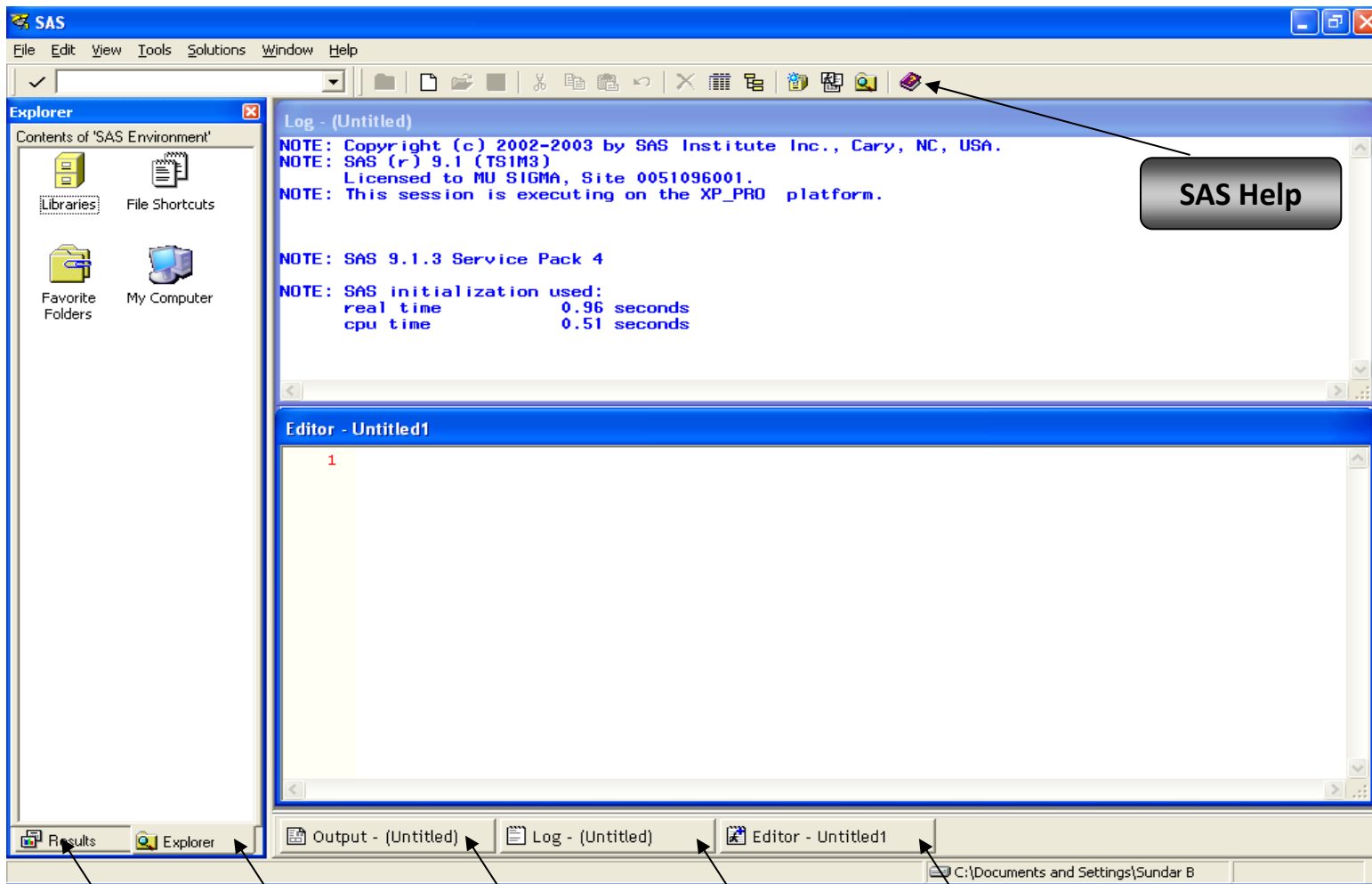
Class room session

Chicago, IL
Bangalore, India
Month 2005
www.mu-sigma.com

Proprietary Information

"This document and its attachments are confidential. Any unauthorized copying, disclosure or distribution of the material is strictly forbidden"

There are five basic windows available in the SAS software – irrespective of whether its Windows SAS or Unix SAS



▶ Editor – Write the SAS program

▶ Log – Check the log after running the SAS code

▶ Output – Check the output, if applicable, post SAS processing

▶ Explorer – Navigate and check libraries and datasets

▶ Results – Stores past results for review

Results

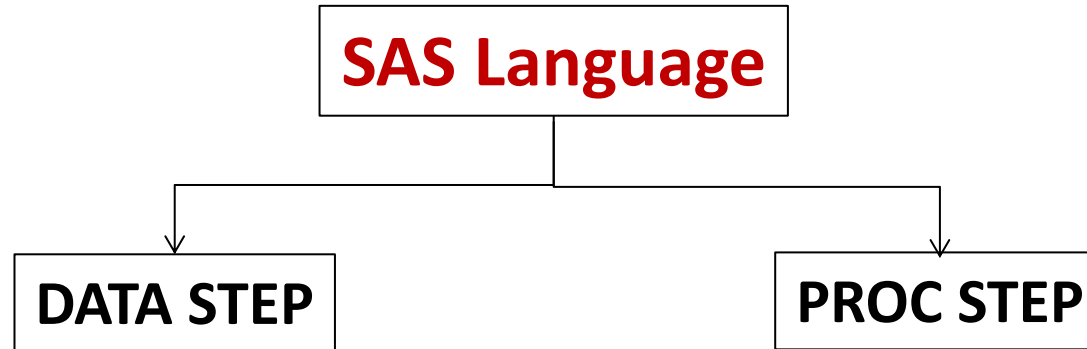
Explorer

Output

LOG

Editor

The SAS programming language consists of mainly two kinds of steps



▶ **DATA steps** typically create or modify SAS datasets. They can also be used to produce custom designed reports. For example you can use DATA steps to:

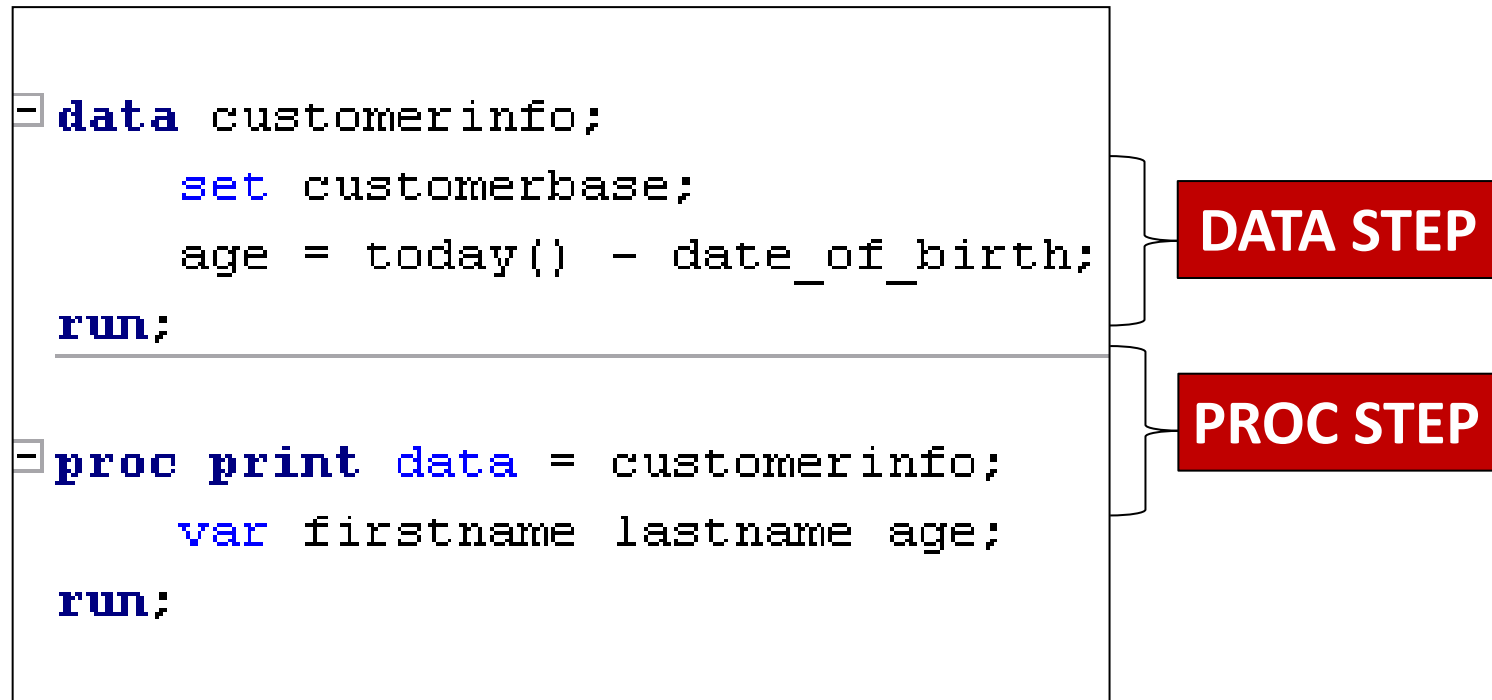
- put your data into a SAS data set
- compute values
- check for and correct errors in your data

produce new SAS data sets by subsetting, merging, and updating existing data sets

▶ **PROC (procedure) steps** are pre written routines that enable you to analyze and process the data in SAS data set. For example you can use PROC steps to:

- create a report that lists the data
- produce descriptive statistics
- create a summary report
- produce plots and charts

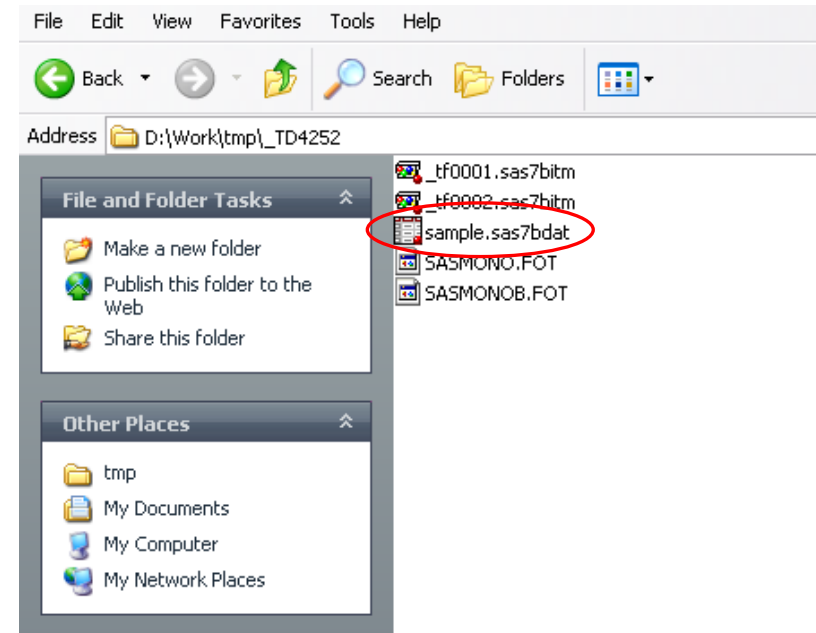
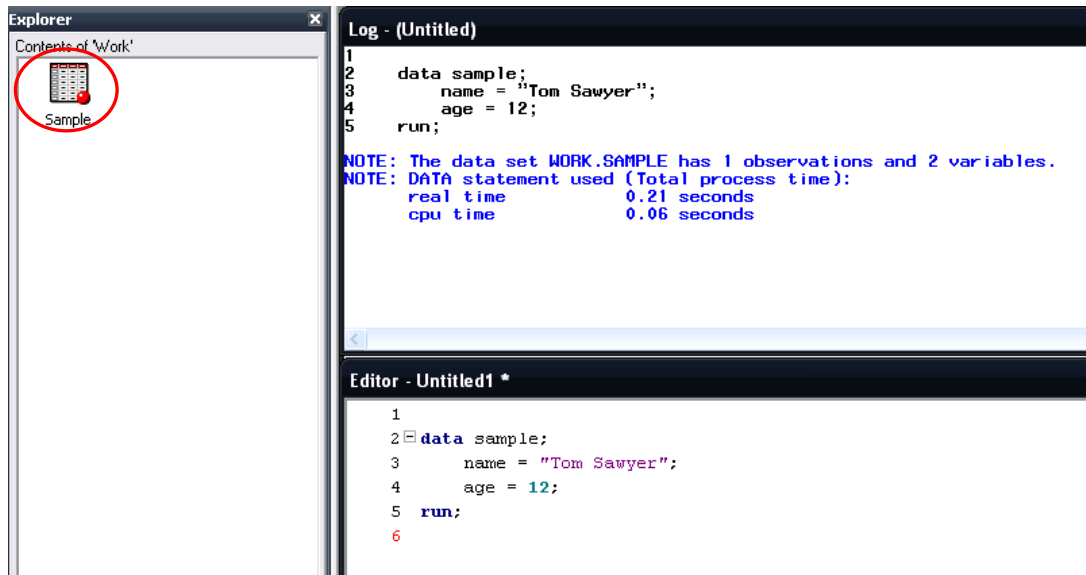
Lets look at a sample SAS program



A normal SAS program can go up to 1000s of lines of code consisting of DATA & PROC steps

A SAS Library is basically an address to a particular location in your hard drive

- ▶ By default all SAS datasets will get created in the WORK library
- ▶ SAS datasets in WORK library are stored temporarily and exist only during the session

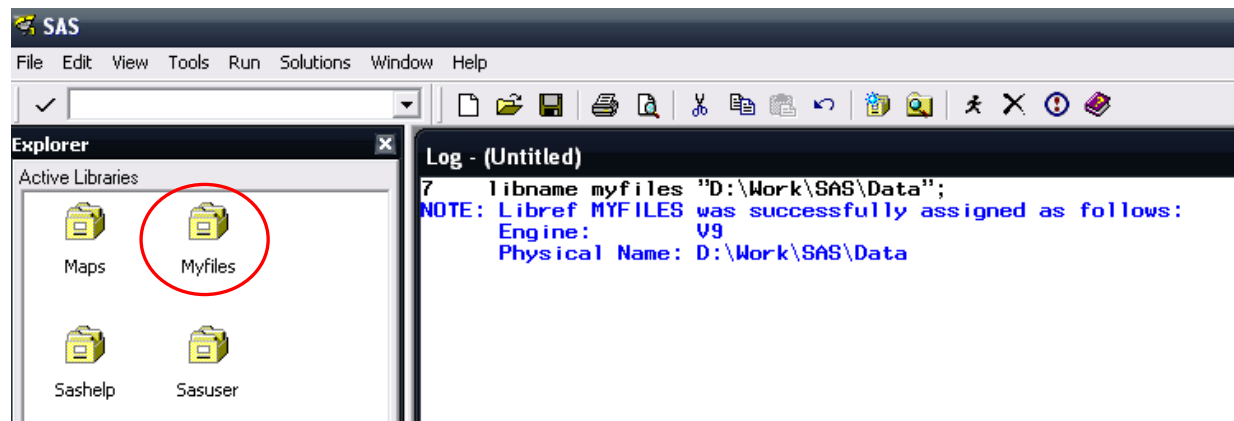


- ▶ Usually the SAS WORK library is located at "*C:\Documents and Settings\<User name>\Local Settings\Temp\SAS Temporary Files*"
- ▶ SAS gives you the ability to change the location of your default WORK library

LIBNAME statement can be used to create libraries at user defined locations

- ▶ A permanent library will store datasets permanently, i.e., they will not get deleted after the SAS session is closed
- ▶ The syntax for LIBNAME is as follows:

```
libname myfiles "D:\Work\SAS\Data";
```



- ▶ A valid library name must start with an alphabet and cannot have more than 8 characters

To store datasets in the permanent library use a two – level naming convention

- ▶ To store datasets in the your permanent library the syntax is *libname.dataset_name*

```
data myfiles.sales;
    set sales;
run;
```

Copies the SALES dataset from WORK library to MYFILES library

- ▶ When the SAS session is closed, the library reference will get deleted but the SAS datasets in that location will be retained
- ▶ Other uses of LIBNAME statements are:
 - Creating ODBC connections to query databases
 - Make connections to SAS transport files
 - Use SPSS engine in SAS to read SPSS portable files

SAS system options control the way SAS performs operations



- ▶ SAS system options differ from SAS data set options and statement options in that once you invoke a system option, it remains in effect for all subsequent DATA and PROC steps unless modified
- ▶ SAS system options can be specified in the following ways:
 - SAS default
 - The configuration file
 - The command line
 - The autoexec file
 - Options Environment Window
 - **The OPTIONS statement**
 - The opLoad procedure

The default SAS system options can be displayed by running:

PROC OPTIONS; RUN;



- ▶ In this session we will cover how to set SAS system options only through the OPTIONS statement

SAS system options... contd.

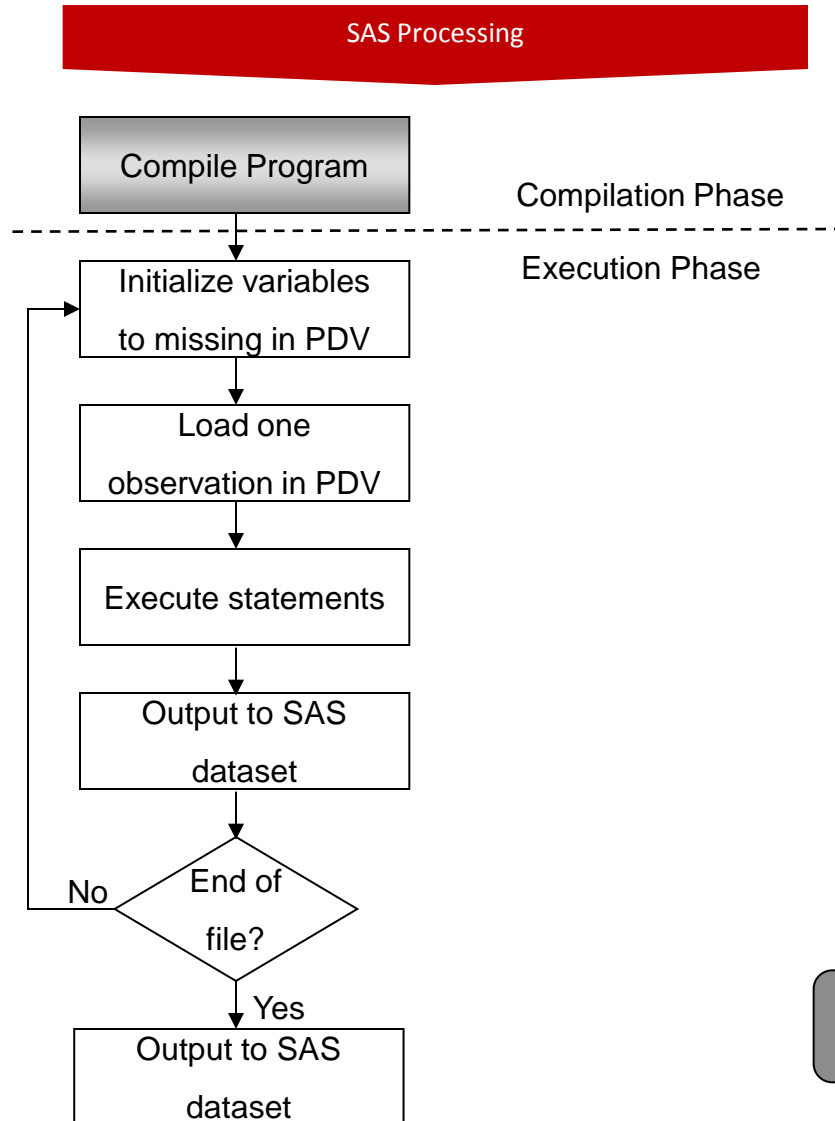
- ▶ The syntax for OPTIONS statement in SAS is as follows:

```
options <option1> <option2> ... ;
```

- ▶ Some of the most commonly specified options are
 - **errors = n** (stops SAS procession after it encounters the *n*th error)
 - **compress = yes** (compresses SAS datasets after they are created)
 - **reuse = yes** (makes efficient use of space inside compressed SAS datasets)
 - **obs = n** (uses only the first *n* observations from a dataset; specifying MAX resets this option)
 - **center / nocenter** (centers / left aligns the SAS output respectively)
 - **nodate** (suppresses printing of Date in the SAS output)
 - **mergenoby = error** (throws error in the log if the *by* statement is missing in a merge step)
 - **msglevel = i** (throws information into log when one variable in a dataset gets overwritten while merging)
 - **mlogic, mprint, symbolgen** (throws additional information in the log window for debugging macros)
 - **fmtsearch = (libname)** (uses the permanent library for storing and searching user defined formats)

How will you use SAS system options to check if the syntax of your code is correct without actually processing any observations?

Data processing using SAS – Concept of Program Data Vector (PDV)



Program Data Vector

N	_ERROR_	Name	Id	DOJ

- ▶ PDV, created during the compilation phase, is a logical area in memory where a dataset is processed one observation at a time
 - ▶ SAS creates two temporary variables
 - ❖ **_N_** – current observation number in process
 - ❖ **_ERROR_** – Binary flag to indicate data processing errors
 - ❖ Other variables – **_TYPE_**, **_FREQ_**, **_STATISTICS_** etc. get created for specific procedures
 - ❖ Each variable is set to missing at the start of an interaction

Understanding the PDV is crucial to writing efficient error free SAS programs

Advantages of PDV

- ▶ In a PDV, SAS is able to look at only one row at a time – hence less memory intensive
- ▶ DATA step conditions which are applied before data is read into PDV reduces processing time and increases efficiency
- ▶ For more detailed explanation of PDV refer to the **PDV Concepts.pdf** document



DISCUSSION DOCUMENT



Mu Sigma

Day 1 – Import, Data step concepts, functions, date time

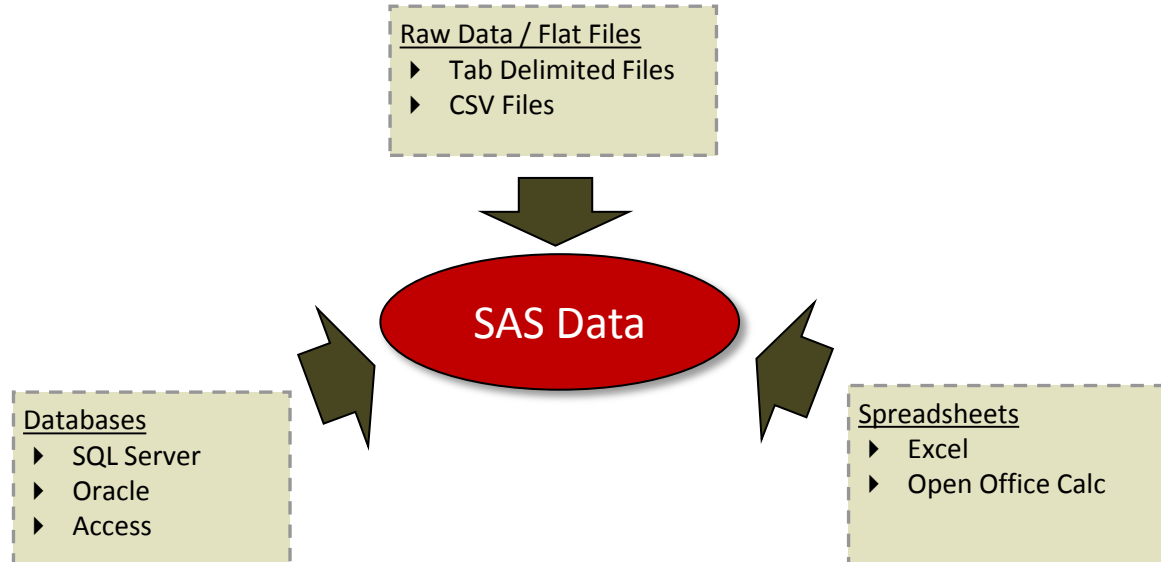
Chicago, IL
Bangalore, India
www.mu-sigma.com

Proprietary Information

"This document and its attachments are confidential. Any unauthorized copying, disclosure or distribution of the material is strictly forbidden"

The first step in working with SAS is to get your data in to it!

- ▶ Data can be obtained from multiple sources for the purpose of analysis



- ▶ Following are some of the ways in which you can get data into SAS
 - Using PROC IMPORT / IMPORT wizard in SAS window
 - Using DATA – INFILE statements
 - Through an ODBC connection to databases
 - Using a DDE connection to a Windows Application

Points to Remember While Importing

- Get the layout details of the raw files from the client before trying to read in the data
- Prepare a data dictionary of all the variables – which will include variable names, labels, formats, length etc.
- Check for errors while importing
- Print observations or open datasets from SAS and do a visual check to see if the data has been read correctly

Types of raw or flat files

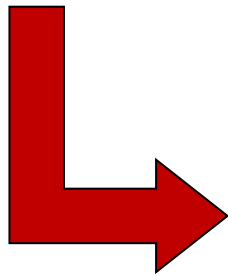
- ▶ Raw flat files are usually of two types
 - **Fixed Width:** Every variable starts from a particular column and ends at a particular column. Hence the width of every field is fixed
 - **Delimited:** Every field is separated by a delimiter. Commonly used delimiters are comma, tab, pipe

Fixed Width

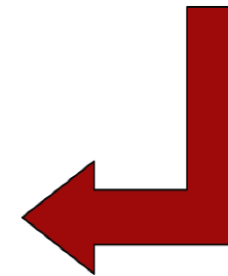
```
0031GOLDENBERG  DESIREE
0040WILLIAMS    ARLENE M.
0071PERRY      ROBERT A.
0082MCGWIER-WATTSCHRISTINA
```

Delimited

```
0031,GOLDENBERG,DESIREE
0040,WILLIAMS,ARLENE M.
0071,PERRY,ROBERT A.
0082,MCGWIER-WATTS,CHRISTINA
```



Emp Id	Last Name	First Name
0031	GOLDENBERG	DESIREE
0040	WILLIAMS	ARLENE M.
0071	PERRY	ROBERT A.
0082	MCGWIER-WATTS	CHRISTINA



A brief overview of informats and formats in SAS

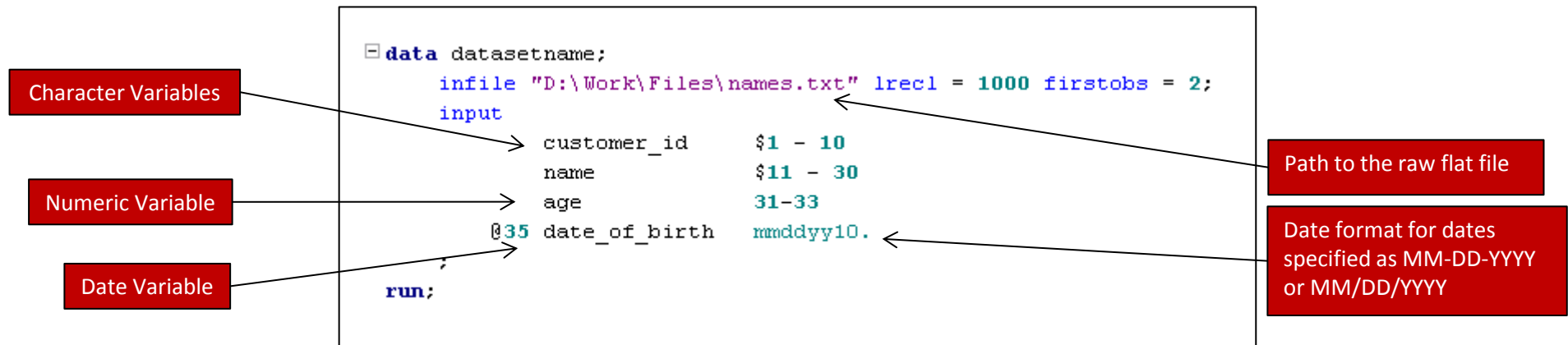
- ▶ **Informat** is an instruction which tells SAS how to read data into SAS variables
- ▶ **Format** instructs SAS how to display a variable
- ▶ There are typically three categories of SAS **informats** and **formats**: character, numeric and data / time
- ▶ The syntax structure of **Informats** and **formats** are as follows:
 - Character informats: **\$INFORMATw.**
 - Numeric informats: **INFORMATw.d**
 - Date / Time informats: **INFORMATw.**
- ▶ **\$** indicates a character informat. The **w** indicates the width and **d** specifies the number of decimal places. The following table gives an example of how SAS can display the same value in different ways using the inbuilt formats library

var1	var2	var3	var4	var5	var6
9,876	\$9,876.00	MMMMMMMMDCCCLXXVI	01/15/1987	01.15.1987	01-15-1987

Reading fixed width files

- ▶ To read fixed width files you would need to know the layout of the raw data completely, i.e., where does each column start and end

Example – Reading fixed width files using DATA INFILE



- ▶ The ***infile*** statement instructs SAS to read from an external file
- ▶ ***lrecl*** option changes the default width of the external file to be read by SAS which is 256 columns
- ▶ ***firstobs*** option instructs SAS to read data from a specific row

Reading Delimited Files

- ▶ Reading delimited files will require the knowledge of the layout of the dataset along with the type of delimiter used in the file

Example – Reading delimited files using DATA INFILE

```

data datasetname;
  infile "D:\Work\Files\sales.dat" lrecl = 1000 firstobs = 2
    dsd dlm = "09"x truncover;
input
  customer_id :$10.
  order_date  :mmddy10.
  product_id  :$4.
  quantity    :5.
  lineval     :dollar10.
;
run;

```

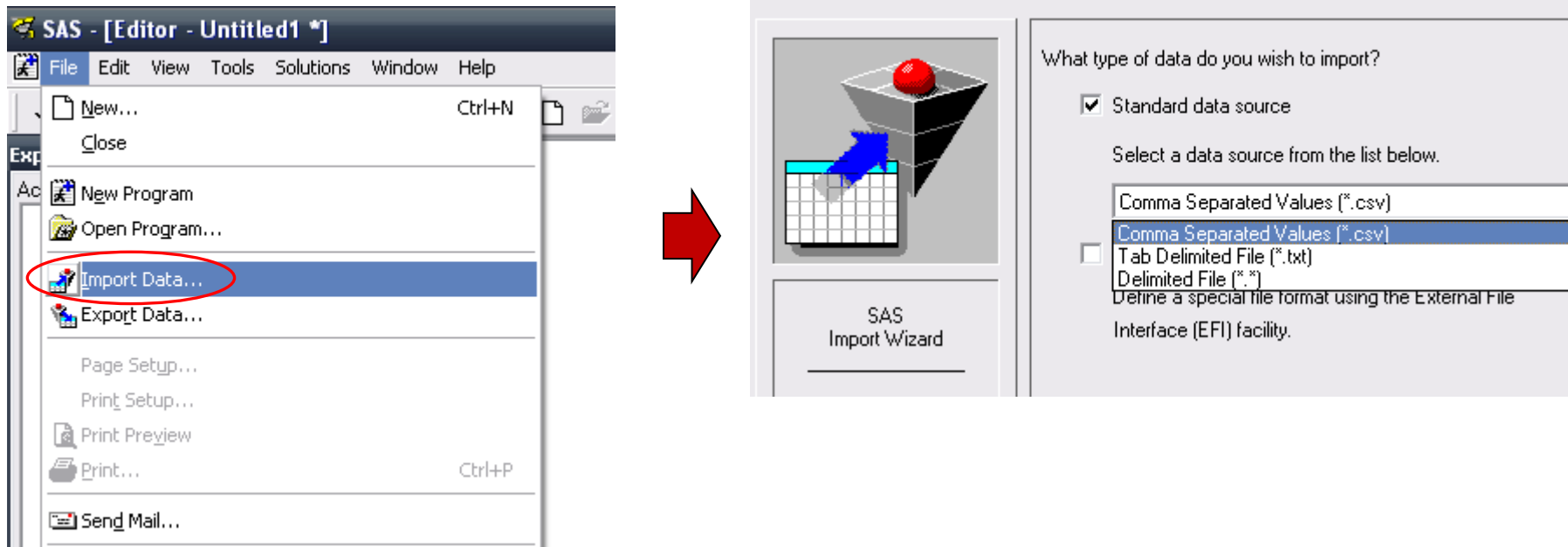
The "09"x indicates it is TAB delimited file

The colon instructs SAS to stop reading a field once it encounters the next delimiter

- ▶ **DSD** option stands for delimiter sensitive data. It has three functions
 - Instructs SAS that the file to be read is a delimited file and sets default delimiter to comma
 - Allows delimiters to be included within quotes
 - Reads two consecutive delimiters as missing
- ▶ The **TRUNCOVER** options prevents SAS from going to next row to read data if any observation is incomplete

Importing data from Data Import Wizard

- ▶ SAS windows gives you the flexibility of reading delimited files using an import wizard

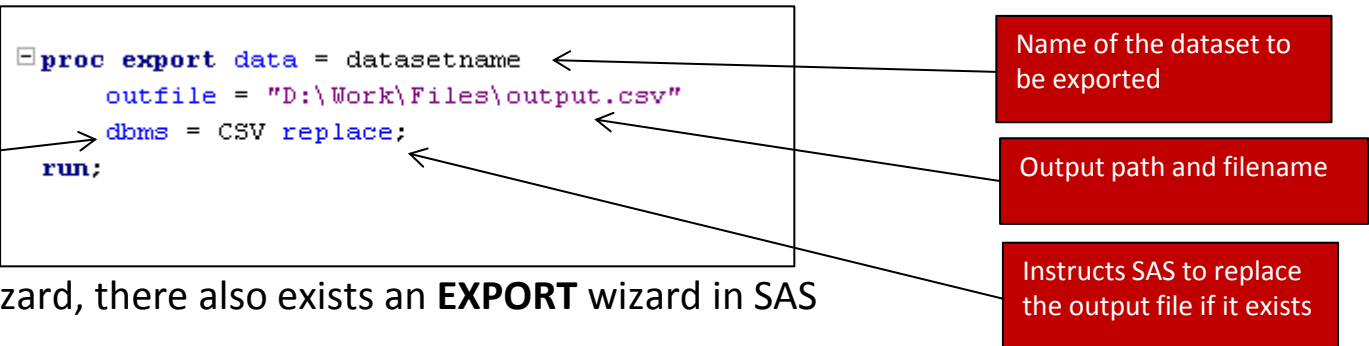


- ▶ SAS will try to intelligently guess the format of each variable while reading and hence may not be perfect!

Data can be exported from SAS either using PROC Step, Export Wizard or using a DATA step

- ▶ The **EXPORT** procedure has the capability create all kinds of files such as delimited files, Excel sheets, Access databases, Lotus spreadsheets, etc. – provided proper license is available

Example – Exporting SAS data set to CSV file



- ▶ Similar to the **IMPORT** wizard, there also exists an **EXPORT** wizard in SAS
- ▶ The **EXPORT** wizard can export SAS datasets to any format provided the license is available
- ▶ Additional methods of exporting SAS data sets are
 - Using DATA – PUT statements
 - SAS Output Delivery System (**ODS**)
 - Using SAS DDE connection



Importing data with a lot of variables

1. Running PROC IMPORT on the file or a small subset of the file.
2. If possible, select the option to use the first line as variable names
3. Copy the SAS log to the Program Editor
4. Delete the non-SAS code.. line number, titles, etc.
5. Modify the informats, formats, lengths as required
6. Run the new code

Data steps are used to manipulate data in SAS

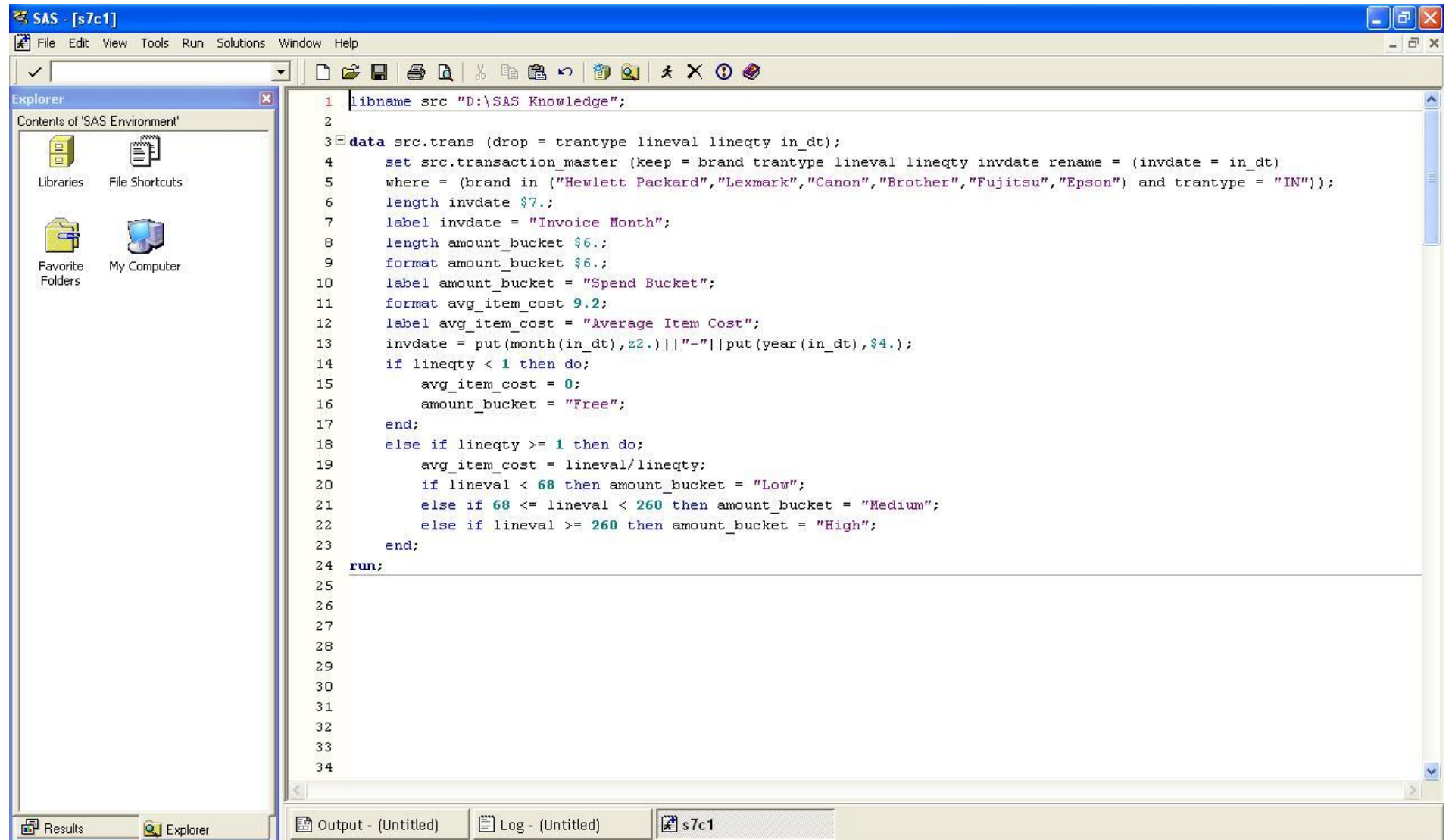
- ▶ Sub-setting variables from datasets
 - Keep option
 - Drop option
 - Keep statement
 - Drop statement

- ▶ creating new/modifying existing variables
 - Using functions
 - By conditionally assigning values using “if-then-else” loops

- ▶ Filtering observations in datasets
 - Where statement
 - Where option

- ▶ Example - The code on the next slide assigns a path on the local machine to the library “src”, creates a subset “trans” of the dataset “transaction_master” in the library “src” with only certain variables from “transaction_master”, conditionally creates new variables in “trans” and with observation matching a set of criteria

This code will be used as an example to explain data step concepts



```

1 libname src "D:\SAS Knowledge";
2
3 data src.trans (drop = trantype lineval lineqty in_dt);
4   set src.transaction_master (keep = brand trantype lineval lineqty invdate rename = (invdate = in_dt)
5   where = (brand in ("Hewlett Packard","Lexmark","Canon","Brother","Fujitsu","Epson") and trantype = "IN"));
6   length invdate $7.;
7   label invdate = "Invoice Month";
8   length amount_bucket $6.;
9   format amount_bucket $6.;
10  label amount_bucket = "Spend Bucket";
11  format avg_item_cost 9.2;
12  label avg_item_cost = "Average Item Cost";
13  invdate = put(month(in_dt),z2.)||"-"||put(year(in_dt),$4.);
14  if lineqty < 1 then do;
15      avg_item_cost = 0;
16      amount_bucket = "Free";
17  end;
18  else if lineqty >= 1 then do;
19      avg_item_cost = lineval/lineqty;
20      if lineval < 68 then amount_bucket = "Low";
21      else if 68 <= lineval < 260 then amount_bucket = "Medium";
22      else if lineval >= 260 then amount_bucket = "High";
23  end;
24 run;
25
26
27
28
29
30
31
32
33
34

```

File saved successfully.

C:\Documents and Settings\krishnaraj.g

Ln 1, Col 1

The basics - Data and set statements and dataset options

- ▶ In the example above, the third line is the data statement and the next two lines are the set statement:
 - The data statement tells SAS to create an output dataset “trans” in the library “src”

```
3 data src.trans (drop = trantype lineval lineqty in_dt);
```

- The set statement tells SAS to read data from the input dataset “transaction_master” in the library “src”

```
4 set src.transaction_master (keep = brand trantype lineval lineqty invdate rename = (invdate = in_dt)
5 where = (brand in ("Hewlett Packard", "Lexmark", "Canon", "Brother", "Fujitsu", "Epson") and trantype = "IN"));
```

- ▶ Within brackets in each statement, dataset options are used. Three commonly used options are the drop, keep and rename options:
 - The drop option in the data statement tells SAS to not write the variables trantype, lineval, lineqty and in_dt to the output dataset
 - The keep option in the set statement tells SAS to read only the variables brand, trantype, lineval, lineqty and invdate from the input dataset
 - The rename option in the set statement tells SAS to read the variable invdate as in_dt from the input dataset
 - The where option in the set statement tells SAS to read only those observations where brand is either of “Hewlett Packard”, “Lexmark”, “Canon”, “Brother”, “Fujitsu”, “Epson” and trantype is “IN” from the input dataset

The drop, keep, rename and where options – usage

- ▶ Drop: When used in the data statement, it specifies which variables ***should not be written to the output dataset***. When used in the set statement, it specifies which variables ***should not be read from the input dataset***
- ▶ Keep: When used in the data statement, it specifies which variables ***should be written to the output dataset***. When used in the set statement, it specifies which variables ***should be read from the input dataset***
- ▶ Rename: When used in the data statement, it specifies which variables ***should be written to the output dataset with different names and what those names should be***. When used in the set statement, it specifies which variable ***should be read from the input dataset with different names and what those names should be***
- ▶ Where: When used in the data statement, it specifies which ***observations should be written to the output dataset***. When used in the set statement it specifies which observations ***should be read from the input dataset***
- ▶ **NOTE** - Using keep/drop in the set statement ***does not affect the input dataset in any way***. It simply specifies which variables should be read/not be read from the input dataset
- ▶ Question: What would happen if the keep option in the set statement were to be changed to:

```
4 set src.transaction_master (keep = brand trantype lineval lineqty in_dt rename = (invdate = in_dt)
```

Creating/modifying variables in data steps – Declaring variables

- ▶ In the example above, the seven lines after the set statement are used to declare new variables and assign their formats

```

6   length invdate $7.;
7   label invdate = "Invoice Month";
8   length amount_bucket $6.;
9   format amount_bucket $6.;
10  label amount_bucket = "Spend Bucket";
11  format avg_item_cost 9.2;
12  label avg_item_cost = "Average Item Cost";

```

- ▶ The three most widely used system defined formats are:
 - Character formats - Declared as **\$n.** where **n** is the *number of characters*
 - Numeric formats - Declared as **k.n** where **k** is the *total number of characters* (including decimal places) and **n** is the *number of characters after the decimal place*
 - Date formats - Declared in various ways, some common date formats are:

Format	Example
date9.	14JUL1983
date11.	14 JUL 1983
ddmmyy8.	14/07/83
ddmmyy10.	14/07/1983

Creating/modifying variable - Variable attributes

- ▶ There are many variable attributes, some important ones being:
 - Length: The length of a variable specifies what format the variable *should be stored as*. Therefore the first line below specifies that the variable `invdate` be stored as a character of length 7
 - Format: The format of a variable specifies what format the variable *should be displayed as*. Therefore the third line below specifies that the variable `amount_bucket` be *stored* as a character of length 6 but the fourth line specifies that it be *displayed* as a character of length 2
 - Label: The label of a variable contains allows the user to assign a 256 length character string as a label for a variable. This is usually used in practice to store a short description/definition of the variable

```

6      length invdate $7.;
7      label invdate = "Invoice Month";
8      length amount_bucket $6.;
9      format amount_bucket $6.;
10     label amount_bucket = "Spend Bucket";
11     format avg_item_cost 9.2;
12     label avg_item_cost = "&Average Item Cost";

```

- ▶ **NOTE** – The informat of a variable is used to specify what format the variable *should be read as* when reading data, as discussed in the previous session
- ▶ **NOTE** –SAS stores date variables as an integer (number of days since 1 Jan, 1960). Thus, date formats *cannot be used* with length statements
- ▶ Question: What would happen if the format of the variable `invdate` were to be changed to **\$2**?

Creating/modifying variables – assigning values

- ▶ In the example above, the 13th, 15th, 16th, 19th, 20th, 21st and 22nd lines are used to assign values to variables
 - In line 13, functions and a concatenation operator is used to assign a value to the variable invdate

```
13      invdate = put(month(in_dt), z2.) || "-" || put(year(in_dt), $4.);
```

- In lines 15, 16, 20, 21, 22, constant values are assigned to the variables avg_item_cost and amount_bucket. The variable avg_item_cost is assigned a constant numeric value. The variable amount_bucket is assigned a constant character value

```
15      avg_item_cost = 0;
16      amount_bucket = "Free";
20      if lineval < 68 then amount_bucket = "Low";
21      else if 68 <= lineval < 260 then amount_bucket = "Medium";
22      else if lineval >= 260 then amount_bucket = "High";
```

- In line 19, a value calculated by using the division operator is assigned to the variable avg_item_cost

```
19      avg_item_cost = lineval/lineqty;
```

- **NOTE** - By using conditional logic, the variable amount_bucket is assigned *different* constant character values for observations meeting different conditions

Conditional logic in SAS – If-then-else, do loops

- ▶ Conditional logic can be used to tell SAS to execute statements only when certain conditions are met
- ▶ In the example above, lines 14 to 23 use conditional logic to assign values to the variables `avg_item_cost` and `amount_bucket`, conditional to values of the variables `lineqty` and `lineval`

```

14     if lineqty < 1 then do;
15         avg_item_cost = 0;
16         amount_bucket = "Free";
17     end;
18     else if lineqty >= 1 then do;
19         avg_item_cost = lineval/lineqty;
20         if lineval < 68 then amount_bucket = "Low";
21         else if 68 <= lineval < 260 then amount_bucket = "Medium";
22         else if lineval >= 260 then amount_bucket = "High";
23     end;

```

- ▶ The general syntax for **if/else if** loops is – **if/else if <logical expression> then <executable statements>;** (lines 20, 21,22)
- ▶ If multiple expressions are to be evaluated after an **if/else if** statement then a **do** is additionally used in conjunction with the **if/else if** statement. The general syntax for this is – **if/else if <logical expression> then do; <executable statements>; end;** (lines14-17, 18-23)
- ▶ Therefore in the example above, if `lineqty` is less than 1 then `avg_item_cost` = 0 and `amount_bucket` = “Free” and if `lineqty` is greater than 1 then `avg_item_cost` = `lineval/lineqty` and `amount_bucket` is conditionally assigned values based upon values of `lineval`

Logical expressions

- ▶ Since logical expressions are essentially binary in nature (either the condition is true or it's false), **not** can be used in an **if/else if** statement to execute statement when a condition is **not met**. The general syntax for this is – **if/else if not <logical expression> then <executable statements>;**
 - Therefore, the following two pieces of code are logically equivalent and will yield the same results:

14	<code>if lineqty < 1 then do;</code>	14	<code>if lineqty < 1 then do;</code>
18	<code>else if lineqty >= 1 then do;</code>	18	<code>else if not (lineqty < 1) then do;</code>

- ▶ Since logical expressions are binary in nature, if/else if statements can also be used to execute different statements for zero and non-zero values of integer variables – the zero values of the variable are evaluated as false and all other values as true. The general syntax for this is – **if/else if <integer variable> then <executable statements>;**
 - Therefore, the following two pieces of code will yield the same results:

14	<code>if lineqty < 1 then do;</code>	14	<code>if lineqty then do;</code>
18	<code>else if not (lineqty < 1) then do;</code>	20	<code>if lineqty < 1 then do;</code>

- But the following two pieces are not logically equivalent and will yield different results:

14	<code>if lineqty then do;</code>	14	<code>if not lineqty then do;</code>
20	<code>if lineqty < 1 then do;</code>	18	<code>else if lineqty then do;</code>

- ▶ Question: Of the first two comparisons, which will be more efficient? How will the output of the two statements in the third comparison differ?

Mnemonic operators

Symbol	Mnemonic	Definition
<	Lt	Less than
<=	Le	Less than or equal to
>	Gt	Greater than
>=	Ge	Greater than or equal to
=	Eq	Equal to
≠	Ne	Not equal to
&	AND	And
	OR	Or
~	NOT	Not

So finally, let us look back at the example and look at the expected output

- ▶ The variables brand, trantype, lineval, lineqty and invdate and observations meeting the following criteria
 - Brand is either “Hewlett Packard”, Lexmark”, “Canon”, “Brother”, “Fujitsu” or “Epson”
 - Trantype is “IN”
- ▶ are read from the dataset “src.transaction_master”
 - The variable invdate is read as in_dt
- ▶ Variables invdate, amount_bucket and avg_item_cost are declared and some of their attributes are defined
- ▶ Invdate is assigned text values of the type “07-1983” by taking the year and month of the in_dt variable and concatenating them
- ▶ For observations where lineqty is lesser than 1, avg_item_amount is set to 0 and amount_bucket is set to “Free”
- ▶ For observations where lineqty is greater than 1, avg_item_amount is set to lineval/lineqty and amount_bucket is set to either “Low”, “Medium” or “High”, based on values of lineval
- ▶ Variables trantype, lineval, lineqty and in_dt are not written to the output dataset “src.trans”

Multiple datasets

- ▶ If we write multiple dataset names in the **set statement**, *all of those datasets are appended*
 - They must all have the same variables
 - The variables must be in the same order on all datasets
 - Common variables in all the datasets must all have the same format
- ▶ If we write multiple dataset names in the **data statement**, *all of those datasets are created*
 - Using the following syntax, we can conditionally write observations meeting different requirements to different output datasets, as seen in the example below

If <condition> then output <dataset>;

```
1 data test1 test2 test3;  
2   set src.source;  
3   first_name = scan(full_name,1," ");  
4   if first_name = "Mr." then output test1;  
5   else if first_name = "Mrs." then output test2;  
6   else if first_name = "Ms." then output test3;  
7 run;
```

Functions in SAS

- ▶ SAS has numerous types of functions. The types that are commonly used are:
 - Character functions
 - Numeric functions
 - Date/date-time functions

- ▶ In this class, we will begin by looking at some commonly used functions falling in the first two categories

- ▶ In general, SAS functions use the following syntax:
 - $X = \text{Function}(\text{argument}_1, \text{argument}_2, \dots, \text{argument}_n)$

- ▶ Arguments can be:
 - Columns (example: $X = \max(Y,Z)$)
 - Expressions (example: $X = \max((Y-Z),(W-Z))$)
 - Other functions (example: $X = \max(\text{sqrt}(Y),\text{sqrt}(Z))$)

- ▶ For a list of commonly used functions refer to the **Important SAS Functions.pdf** document

Where: Statement vs. option

- ▶ The where statement allows the user some more flexibility than the where option, in that it allows for the usage of expressions and functions
- ▶ Thus, the following two codes will filter the same observations:

```
1 data test (where = (first_name eq "Mr."));  
2   set src.source;  
3   first_name = scan(full_name,1," ");  
4 run;
```

```
1 data test;  
2   set src.source;  
3   where scan(full_name,1," ") eq "Mr.";  
4 run;
```

- ▶ Obviously, the second code is more efficient since it allows us to apply our filters without having to create the first_name variable

Date and Time – Format and Informat

► Important Date and Time Formats:

Format	Input	Result
DATEw.	7/25/1984	25Jul84
DAYw.	7/25/1984	25
DDMMYY10.	7/25/1984	25/07/1984
MMDDYY10.	7/25/1984	7/25/1984
DOWNAME.	7/25/1984	Wednesday
MONTH.	7/25/1984	7
MONNAME.	7/25/1984	July
QTR.	7/25/1984	3
TIMEw.	7/25/1984	2:29:32
YYQRw.	7/25/1984	84QIII
DATETIMEw.	7/25/1984	01JAN60:02:29:32

► Important Date and Time Informats:

Informat	Input	Result
DATEw.	25Jul84	8972
DDMMYY10.	25/07/1984	8972
MMDDYY10.	7/25/1984	8972
MONYYw.	Jul-84	8972
TIMEw.	2:29:32	8972
YYQw.	84Q1	8972
DATETIMEw.	01JAN60:02:29:32	8972

Note: “w” stands for the width

Creating SAS Dataset – Date and Time – Format and Informat



▶ Sample SAS Code: Date and Time Variable

```
data datetime;
  %let _EFIERR_ = 0; /* set the ERROR
    detection macro variable */
  retain Name;
  informat DOB mmddy10.;
  informat DOB_time datetime18.;
  format DOB mmddy10.;
  format DOB_time datetime18.;
  input Name $8. DOB DOB_time;

cards;
Pinaki 07/25/1984 25JUL1984:14:45:32
Tapan 07/04/1983 07Jul1983:14:45:32
Sourav 12/12/1982 12Dec1982:14:45:32
;
run;
```

▶ Sample SAS Code: Different Date Formats

```
data pink;
  %let _EFIERR_ = 0; /* set the ERROR detection macro variable
    */
  retain Name;
  informat DOB DOB_1 DOB_2 DOB_3 DOB_4 DOB_5 DOB_6 DOB_7 DOB_8
    DOB_9 best32. ;
  format DOB date9. DOB_1 day2. DOB_2 ddmmyy10. DOB_3 mmddy10.
    DOB_4 DOWNNAME. DOB_5 MONTH. DOB_6 MONNAME.
    DOB_7 qtr. DOB_8 time7. DOB_9 yyqr6.;
  /* format DOB_1 day2.;*/
  input Name $8. DOB DOB_1 DOB_2 DOB_3 DOB_4 DOB_5 DOB_6 DOB_7
    DOB_8 DOB_9;

cards;
Allstate 14690 14690 14690 14690 14690 14690 14690 14690 14690
14690
;
run;
```

Import Date Variable from CSV Itself

▶ Importing Date Functions:

The date variables are stored as numeric values. Its important to import DATE variable directly from CSV file itself. This is because the START DATE in SAS is '1JAN1960', where as in CSV/Excel, the START DATE is '1JAN1900'. Hence, the results will be different if we import the date variable as Numeric Variable and convert the Numeric variable to Date variable in SAS itself.

▶ Sample SAS Code:

```
data impor;
  infile 'C:\Temp\Desktop\Date_Time.csv' dsd
  missover lrecl=1000 dlm=',' firstobs=2;
  retain Name;
  informat Name $6.;
  informat DOB mmddyy10.;
  format Name $6.;
  format DOB mmddyy10.;
  input Name $ DOB DOB_time;
run;
```

```
data impor;
  set impor;
  informat check best32.;
  format check mmddyy10.;
  check=DOB_time;
run;
```

Date and Time – Functions

► Important Date and Time Functions:

► Sample SAS Code:

Function	Input	Result
datepart	25JUL84:14:45:32	7/25/1984
timepart	25JUL84:14:45:32	14:45:32
day	25JUL84:14:45:32	25
Today()		6/1/2009
Hour	25JUL84:14:45:32	14
minute	25JUL84:14:45:32	45
month	25JUL84:14:45:32	7
qtr	25JUL84:14:45:32	3
weekday	25JUL84:14:45:32	4
year	25JUL84:14:45:32	1984
datetime ()		01JUN09:02:06:13

```

data datetime;
  set datetime;
  format Date1 mmddyy10. Time1 time9. Today1 mmddyy10.
  Todaytime datetime18. ;
  Date1= datepart(DOB_time);
  Time1=timepart(DOB_time);
  Day1 = day(Date1);
  Today1=Today();
  Hour1= Hour(DOB_time);
  Minute1=minute(DOB_time);
  month1=month(Date1);
  Qtr1=qtr(Date1);
  Weekday = weekday(Date1);
  Year1=year(Date1);
  Todaytime = datetime();
run;

```

Date and Time – Functions (INTCK/ INTNX)

- ▶ **INTCK function:** This is a popular and powerful SAS function to determine the number of time periods between two SAS dates. The form of this function is: **INTCK('interval',from,to)**
- Where: 'interval' = character constant or variable name representing the time period of interest enclosed in single quotes
 - "from" = SAS date, time or datetime value identifying the start of a time span
 - "to" = SAS date, time or datetime value identifying the end of a time span
- This function will return the number of time periods which have occurred (i.e., have been *crossed*) between the values of the *from* and the *to* variables.

- ▶ **INTNX function:** This is a popular and powerful SAS function to determine a SAS date, time or datetime value for a given number of time intervals from a starting value.. The form of this function is: **INTNX('interval',from,to,position)**
- Where: 'interval' = character constant or variable name representing the time period of interest enclosed in single quotes
 - "from" = SAS date, time or datetime value identifying the start of a time span
 - "to" = SAS date, time or datetime value identifying the end of a time span
 - "Position" = Position of the pointer

▶ Sample SAS Code:

```
data intervals;
set datetime;
format Date1 mmddyy10. Today1 mmddyy10. ;
Today1=Today();
Date1= datepart(DOB_time);
INTCK_Cal=intck('year',date1,Today1);

run;
```

```
data intervals;
set datetime;
format Date1 mmddyy10. Today1 mmddyy10.
INTNX_Cal mmddyy10.;
Today1=Today();
Date1= datepart(DOB_time);
INTNX_Cal=intnx('year',date1,4,'M');

run;
```


Filtering Date and Time Variable

- ▶ Suppose you want to apply a “Subsetting Where” statement in a data step so that only observations with a value of the variable `DOB_time` is that occurred after 4th July, 1984 will remain in the program. All you need to write is:
 - ▶ Declaring a Date Constant:
 - where `date1 > '04Jul1983'd;`
The letter “D” next to the text string with the day, month and year instructs the SAS System to convert the string to the appropriate SAS date value
 - ▶ Declaring a Time Constant:
 - where `time1 > '14:44:32't;`
As with the date constant example given above, the letter “T” next to the text string with the hour, minutes and seconds instructs the SAS System to calculate the appropriate SAS time value (number of seconds from midnight)
 - ▶ Declaring a Date Time Constant:
 - where `DOB_time > '04Jul1983:14:44:32'dt;`
By now you’ve probably figured out that SAS will calculate the appropriate datetime value (number of seconds from midnight on January 1, 1960)



DISCUSSION DOCUMENT



Mu Sigma

Day 2 – Sort and merge, Common procedures, SQL

Chicago, IL
Bangalore, India
www.mu-sigma.com

Proprietary Information

"This document and its attachments are confidential. Any unauthorized copying, disclosure or distribution of the material is strictly forbidden"

PROC SORT

Definition: Sorts the observations in a SAS data set by the values of one or more variables

```
PROC SORT DATA = <Input DSN> OUT = <Output DSN>
  NODUPKEY DUPOUT = <Output DSN for duplicate recs>;
  BY ascending/descending <variable list>;
  RUN;
```

DUPOUT: Store only duplicates in a separate SAS dataset

NODUPKEY: Remove Duplicates

BY: Sort values by one or more variables

- ▶ **SORT** procedure can be used either to modify the original dataset or create a new sorted dataset
- ▶ SAS, by default, sorts the datasets in an ascending order, unless specified otherwise variables should be mentioned in the same order as sorting is required
- ▶ Using NODUPKEY option without using the OUT = statement may destroy your original dataset and duplicate records might not be available for any future analysis



Example:

```
DATA work.sample;                                /*create dataset*/
INPUT F_Name $ L_name $ Age Extn;
CARDS;
Eric Thompson 20 3324
Jack Smith 26 3421
Daniel M 22 2323

Jack Jones 25 4124

Jack Simpson 23 3265
;
PROC SORT DATA = work.sample;                   /*sort dataset*/
BY Age;
RUN;
```

- ▶ Q: How is the dataset sorted if you try to sort by more than one BY variable?



'BY' Statement Processing (In a Datastep)

- ▶ Is a method of processing observations from one or more SAS data sets that are grouped or ordered by values of one or more common variables
- ▶ SAS creates 2 temporary variables internally: 'FIRST.<variable>' and 'LAST.<variable>'

```
DATA work.sample;  
SET work.Sample;  
BY F_Name;  
RUN;
```

- ▶ Their values are set based on whether an observation is
 - the first or last one in a BY group
 - neither the first nor the last one in a BY group
 - both first and last, as is the case when there is only one observation in a BY group

▶ Q: Which other automatic variable gets created by SAS when you run your code

MERGE Statement

MERGE statement is used to joins observations from two or more SAS data sets into single observations

```
DATA <Output DSN>;
MERGE <DSN1> (IN=a) <DSN2> (IN=b);
BY variable-list;
IF a OR b;
RUN;
```

- Match-merging combines observations from two or more SAS data sets into a single observation in a new data set according to the values of a common variable.
- **Sorting is necessary before Match Merging.**

F_Name	L_Name	Age
Bob	Simpson	23
Daniel	M	22
Eric	Thompson	20
Jack	Smith	26
Jack	Jones	25

F_Name	Salary
Daniel	200
Eric	300
Jack	250



F_Name	L_Name	Age	Salary
Bob	Simpson	23	.
Daniel	M	22	200
Eric	Thompson	20	300
Jack	Smith	26	250
Jack	Jones	25	250

▶ Q: There can be variables with the same name (apart from the BY variable) in more than one input data set; What will be the output?

▶ Q: How will you overcome the above situation?

Using 'IF' statement to subset dataset while Merging:

```
DATA Work.Emp_final;
MERGE Work.Emp_Info (IN=a) Work.Emp_Sal (IN=b);
BY F_Name;
IF a AND b;
RUN;
```

F_Name	L_Name	Age
Bob	Simpson	23
Daniel	M	22
Eric	Thompson	20
Jack	Smith	26
Jack	Jones	25

F_Name	Salary
Daniel	200
Eric	300
Jack	250

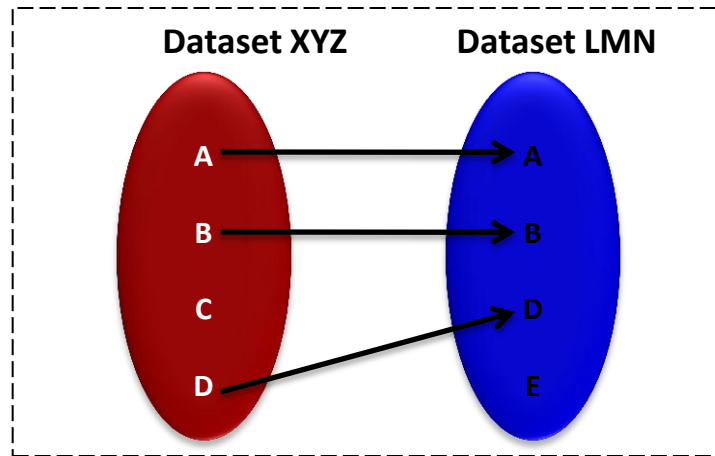


F_Name	L_Name	Age	Salary
Daniel	M	22	200
Eric	Thompson	20	300
Jack	Smith	26	250
Jack	Jones	25	250

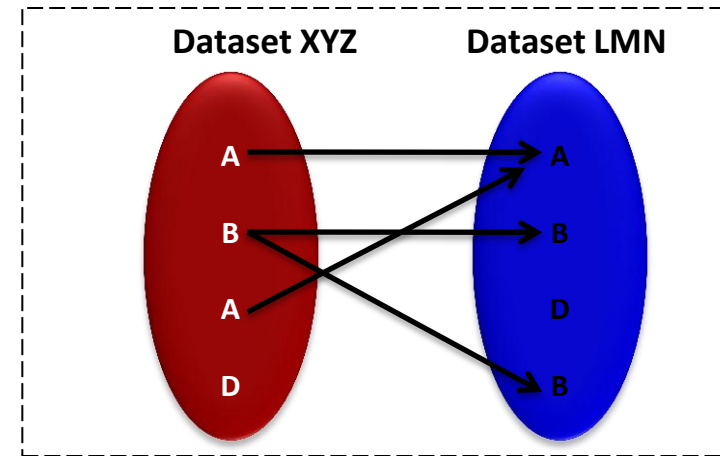
► Q: What happens if you don't use a 'BY' statement in your MERGE?

- While merging if you don't use a 'BY' statement; SAS will merge the 1st observation from dataset A and 1st observation from dataset B to form the 1st observation of the final dataset
- Do not miss out on the 'BY' statement. It might lead to useless results.

Mappings for Merge:



One to One Mapping



Many to Many Mapping

- ▶ If more than one dataset has multiple values for 'BY' variable (many to many mapping); SAS does not pick up values for the variable in a defined order
- ▶ SAS registers a note in the LOG. No error is generated
- ▶ To avoid unknown outputs; remove the duplicate values from one dataset prior to merging

```
NOTE: MERGE statement has more than one data set with repeats of BY values.
NOTE: There were 5 observations read from the data set WORK.EMPLOYEE.
NOTE: There were 4 observations read from the data set WORK.EMP_SALARY_MANY.
NOTE: The data set WORK.EMP_INFO_MANY has 4 observations and 5 variables.
NOTE: DATA statement used (Total process time):
      real time           0.04 seconds
      cpu time            0.04 seconds
```

- ▶ Proc Sort – PDV Concepts – Many to many merge

Before you dare to merge

1. Check if the key variable exists in both the datasets that are to be merged
2. The key variable should have the same format and length in both the datasets
3. Sort the files by the key variable
4. The key variable can repeat in only one of the datasets
5. If there are common variables other than the key variable in the dataset treat them appropriately
6. Always write a BY statement
7. Always write an IF statement
8. Subsetting variables and observations can be done while merging – need not write a separate step

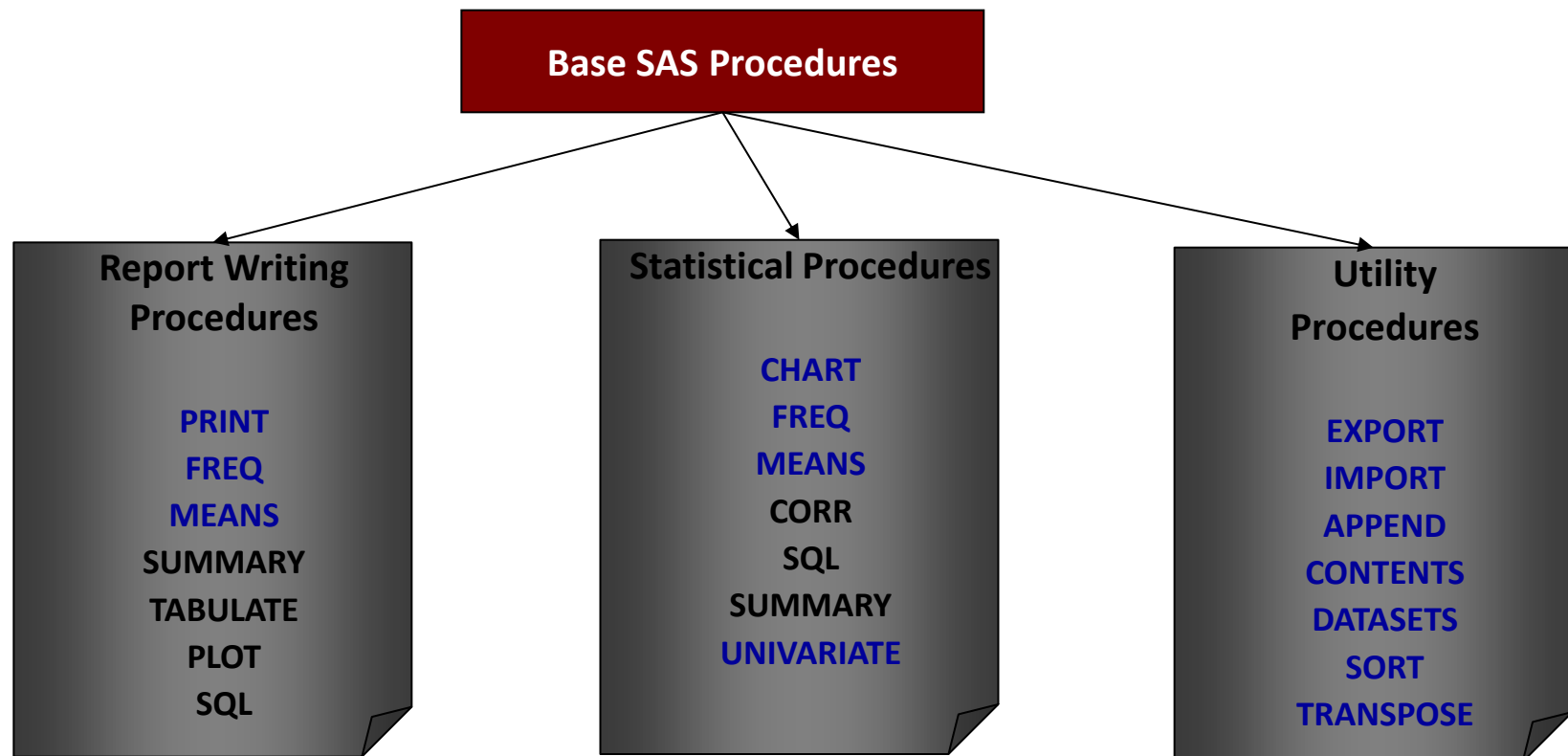


Common SAS log notes and warnings

- ▶ **NOTE:** Missing values were generated as a result of performing an operation on missing values.
- ▶ **NOTE:** MERGE statement has more than one data set with repeats of BY values.
- ▶ **WARNING:** Multiple lengths were specified for the BY variable emp_id by input data sets. This may cause unexpected results.
- ▶ **WARNING:** The variable x in the DROP, KEEP, or RENAME list has never been referenced
- ▶ **NOTE:** Variable var1 is uninitialized.
- ▶ **NOTE:** Numeric values have been converted to character values at the places given by: (Line):(Column). 52:8
- ▶ **NOTE:** Character values have been converted to numeric values at the places given by: (Line):(Column).55:9

SAS Procedures – An Introduction

- ▶ SAS Procedure or a PROC step always starts with the word **PROC**
- ▶ Some commonly used Base SAS procedures are listed below





A typical SAS procedure has a few key words that are a part of the syntax

```
PROC <PROCEDURE NAME> DATA = <DSN Name> OPTIONS;  
  BY <Variable List>;  
  CLASS <Variable List>;  
  VAR ;  
  WHERE ;  
RUN ;
```

- ▶ Most SAS procedures require an input dataset which is specified using the **DATA=** option
- ▶ **VAR** specifies the variables on which the procedure is applicable. If no variables are specified, then SAS will automatically apply the procedure on all the variables.
- ▶ **WHERE** allows usage of a particular filter criteria on the procedure.
- ▶ “**Sales**” is used to refer the SAS dataset to elaborate any SAS procedure going forward
- ▶ Only a few most frequently used SAS procedures are covered in the training
- ▶ Further, not all options available on the SAS procedure is covered in the training

PROC CONTENTS

```
PROC CONTENTS DATA = SALES OUT = VAR_LIST VARNUM;  
RUN;
```

Name of the input data set

Option lists all the variables in the same order as present in the data set

Name of the output data set will contain the list of the variables with their formats

Q: What is the output if you don't use the option "varnum"

PROC Printto

```
FILENAME printout 'c:\demo.txt';  
FILENAME logout 'c:\demo.log';  
PROC PRINTTO  
    print=printout  
    log=logout new;  
RUN;
```

Location to where a log should be saved

Option to print the log in the desired location

Q: How do we return and procedure output to their default destination

PROC DATASETS

PROC DATASETS

```
MEMTYPE = DATA LIB = WORK NOLIST;
APPEND BASE = DATA = ;
CHANGE old_name = new_name ;
COPY IN = libref-1 OUT = libref-2 ;
SELECT sas_files;
DELETE sas_files;
RUN;QUIT;
```

- ▶ PROC DATASETS should ideally be preceded by a lib statement

PROC DATASETS

```
LIB = WORK;
MODIFY DS_Name (LABEL = 'new_label');
RENAME old_var_name = new_var_name;
LABEL new_var_name = label_for_new_var;
FORMAT existing_var_name COMMA11.2;
RUN;QUIT;
```

- ▶ The APPEND command adds observations from one dataset to another

Specifies the kind of files to process

Specifies the library

Option does not print any kind of output in the SAS output window

Specifies the dataset to be renamed

Specifies the library to copy SAS datasets

Only specified datasets will be copied

Specifies SAS datasets to be deleted

The MODIFY command gives you the ability to change a specific library member or attributes of the library member's variables

Rename a variable in the specified data set

Label can be added to a variable

A format can be specified for a variable

Q: How can we delete all the datasets in a library?

PROC TRANSPOSE

```
PROC TRANSPOSE DATA = <DSN Name>
  OUT = <Output DSN Name>;
  BY <Variables>;
  ID <Variables>;
  IDLABEL <Variables>;
  VAR <Variables>;
RUN;
```

Specifies the variable whose formatted values name the transposed variable

Creates labels for the transposed variables

Specifies the variable to transpose

Example 1

Original			
Obs	A	B	C
1	1	2	3
2	4	5	6

```
proc transpose data = d1 out = d2;
run;
```

Transposed			
Obs	_NAME_	COL1	COL2
1	A	1	4
2	B	2	5
3	C	3	6

Original

Obs	model	week1	week2	week3
1	Ford	\$3,400	\$2,400	\$4,500
2	Chevy	\$5,900	\$7,100	\$9,800
3	Honda	\$4,000	\$6,500	\$7,000

```
proc transpose data = car_sales out = transposed_sales;
  id model;
  var week1 week2 week3;
run;
```

Example 2

Transposed				
Obs	_NAME_	Ford	Chevy	Honda
1	week1	\$3,400	\$5,900	\$4,000
2	week2	\$2,400	\$7,100	\$6,500
3	week3	\$4,500	\$9,800	\$7,000

Q: In the above examples, identify the variables and relate them to the syntax

PROC FREQ

```

PROC FREQ DATA = SALES;
WEIGHT <Weight Variable>;
TABLES <Variable List> /
MISSING NOROW NOCOL NOPERCENT ALL NOPRINT;
WHERE <Condition>;
RUN;

```

- ▶ For two-way tables, *PROC FREQ* can compute tests and measures of association

Option: Give different weight to the observations

Option:
Missing: Treats missing values as a separate observation
Norow: Removes row percentages
Nocol: Removes column percentages
Nopercent: Removes cell percentage

Q: What option would you use to generate three way tables?

Q: How would you output the results of Freq procedure to a SAS dataset?

PROC UNIVARIATE

```

PROC UNIVARIATE DATA = <DSN Name> <OPTIONS>;
BY <Variables>;
CLASS <Variables>;
FREQ <Variables>;
HISTOGRAM <Variables> / <options>;
OUTPUT OUT= <DSN Name> <options>;
PROBPLOT <Variables> / <options>;
QQPLOT <Variables> / <options>;
VAR <Variables>;
RUN;

```

- ▶ The *UNIVARIATE* procedure provides a variety of descriptive measures, high-resolution graphical displays, and statistical methods, which you can use to summarize, visualize, analyze, and model the statistical distributions of numeric variables.

E.g. **PLOTS, NORMAL**

Information will be represented in the form of a Histogram in the output

Probability plots for the distribution specified will be displayed

Quantile (Q-Q) plots for the distribution specified will be displayed. It is used to compare the values with quantiles of a specified distribution

Q: For which variables will the histogram be displayed if no variables are mentioned in the "VAR" option of the "Proc Univariate" procedure

PROC MEANS

```

PROC MEANS DATA = SALES;
  CLASS <Variable List>;
  VAR <Variable List>;
  OUTPUT OUT = <DSN Name> <Summary Procedure>;
  RUN;

```

Example:

```

SUM(Variable) = New Variable 1
MEAN(Variable) = New Variable 2
MIN(Variable) = New Variable 3
MAX(Variable) = New Variable 4

```

- ▶ **PROC MEANS** also computes
 - Descriptive statistics based on moments and quantiles
 - Calculates confidence interval for means
 - Performs t – test

Q: Which SAS dataset will contain the results without the use of “output out =” option

Q: What SAS default variables which will be created in the output SAS dataset?

SAS and SQL terminology

SAS Data step	Proc SQL
Dataset	Table
Variable	Column
Observation/ records	Rows/ records
Append	Union
Merge	Join

How about differences in ease of use, Efficiency?

- ▶ Depends on various factors
 - ▶ Size of dataset
 - ▶ Operation performed etc.

Syntax of Proc SQL?

- ▶ Very straightforward and extremely structured syntax
- ▶ Highly scalable, from simple one row queries, to complicated queries with multiple sub-queries
- ▶ Possible to combine with all SAS functions (except the lag function)

Theory

Proc SQL <options> ;

- *Multiple SQL statements (or queries) can be put into a single Proc SQL*
- *You can have a series/collection of SQL queries that are your entire program*
- *Think of the PROC SQL/QUIT; combination as boundaries for a different coding environment*

Quit;

Practical

proc sql;

create table tablename as

select [distinct] column1, column2,

[], ...*

from library.table

where expression

order by column1 etc.;

quit;

** = all columns*

Creating, Updating and deleting tables

The select statement

SAS Data step	Proc SQL
<pre>Proc Print Data= First; Run;</pre>	<pre>Proc SQL; Select * From First; Quit;</pre>
<pre>Proc Print Data=First; Var Patient Age Losses; Run;</pre>	<pre>Proc SQL; Select Patient, Age, Losses From First; Quit;</pre>

How to create dataset using select statement?

- ▶ Through use of a “*Create table tablename as*” clause, can create:
 - datasets/tables

Creating, Updating and deleting tables

Creating Table

SAS Data step	Proc SQL
<pre>Data lib.new; set lib.old; Run;</pre>	<pre>Proc SQL; create table lib.new as select * From lib.old; Quit;</pre>

- ▶ “*Create table tablename as*” clause is used to create a new table called lib.new
- ▶ We have used “*select **” here, which means all variables will be selected in the resulting dataset.

Creating, Updating and deleting tables

Creating new Variables

SAS Data step	Proc SQL
Data Second; Set First; new = old + 1; Run;	Proc SQL; create table second as select *, old+1 as new From First; Quit;

▶ Q: How do you update a variable name? Say renaming a variable.

▶ Q: How do you delete a table/ dataset using proc SQL?

SQL – Filtering, Ordering and Grouping

SELECT statement is used to retrieve data from a database

▶ Clauses in Select

- Where
- Group by
- Having
- Order by

▶ Operators

- Logical Operators
- Arithmetic Operators
- Relational Operators
- Special Operators

▶ Functions

- Aggregate Functions
- String Functions
- Date Functions

SQL – Clauses in select

▶ Where Clause

- Subsets the output based on specified conditions
- When a condition is met (that is, the condition resolves to true), those rows are displayed in the result table; otherwise, no rows are displayed.
- You cannot use summary functions that specify only one column

▶ Group by Clause

- Specifies how to group the data for summarizing.

▶ Having Clause

- Subsets grouped data based on specified conditions

▶ Order By Clause

- Specifies the order in which rows are displayed in a result table.
- ASC in order by clause orders the data in ascending order. This is the default order; if neither ASC nor DESC is specified, the data is ordered in ascending order
- DESC in order by clause orders the data in descending order

SQL – Clauses in select

How clauses are used in SQL and Data steps?

Operation	SAS Data step	Proc SQL
Filtering	<pre>Data lib.new; set lib.old; Where age>30 Run;</pre>	<pre>Proc SQL; create table lib.new as select * From lib.old where age>30; Quit;</pre>
Ordering	<pre>Proc sort data=lib.old out=lib.new; by age descending; Run;</pre>	<pre>Proc SQL; create table lib.new as select * From lib.old Order by age desc; Quit;</pre>
Grouping	<pre>Proc Means Data=First ; Class desig; Var Wealth; Run;</pre>	<pre>Proc SQL; Select desig, mean(Wealth) as Mean From First Group By desig; Quit;</pre>

SQL – Operators

▶ Logical | Boolean Operators

Logical operators, also called Boolean operators, are usually used in expressions to link sequences of comparisons

- AND
- OR
- NOT

▶ Arithmetic Operators

Arithmetic operators indicate that an arithmetic calculation is performed

- Multiplication (*)
- Division (/)
- Addition (+)
- Subtraction (-)

▶ Relational | Comparison Operators

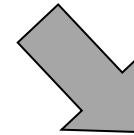
- Comparison operators set up a comparison, operation, or calculation with two variables, constants, or expressions. If the comparison is true, the result is 1. If the comparison is false, the result is 0.
- Equal to (=), not equal to (<>), not equal to (!=), less than (<), greater than (>), less than or equal to (<=), greater than or equal to (>=)

SQL Functions

PROC SQL;

```
SELECT COUNT(*) as count, designation
FROM employees
GROUP BY designation;
```

QUIT;



Count(*): Count Function

*: All Records

Count: Alias name for count

count	Designation
7	Business Analyst
10	Director
10	Manager
13	Senior Business Analyst
5	Senior Manager
3	Senior Vice President
3	Vice President

SQL Functions

- ▶ PROC SQL supports all the functions available to the SAS DATA step that can be used in a proc sql select statement
- ▶ Because of how SQL handles a dataset, these functions work over the entire dataset
- ▶ Common Functions:

- | | |
|------------|----------|
| ▫ COUNT | ▫ NMISS |
| ▫ DISTINCT | ▫ RANGE |
| ▫ MAX | ▫ SUBSTR |
| ▫ MIN | ▫ LENGTH |
| ▫ SUM | ▫ UPPER |
| ▫ AVG | ▫ LOWER |
| ▫ VAR | ▫ CONCAT |
| ▫ STD | ▫ ROUND |
| ▫ STDERR | ▫ MOD |

Notes

1. PROC SQL does not support LAG, DIF and SOUND functions.
2. This is not a comprehensive list of functions. There are thousands of functions for different needs
3. The functions can be used with Macro variables in SAS
4. There are numerous functions for string operations – See Appendix.

- ▶ Q: Subset the second name from the “name” variable in the employees table where the age is between 25 and 50. Also, create a new variable which adds 10 to their age and order by this age.
- ▶ Q: How do you find the location of a particular character in a string in a variable?

PROC SQL

```

PROC SQL;

CREATE TABLE SALES AS
  SELECT
    CustNum, Brand, count(distinct EDC) as No_Items, sum(LineVal) as Total_Sales
  FROM
    S.Transaction_Master
  Where
    TranType = "IN" and LineVal > 0
  GROUP BY
    CustNum, Brand
  HAVING
    No_Items >= 2 or Total_Sales > 50
  ORDER BY
    Custnum, Total_Sales DESC;

QUIT;

```

Subsetting data:

```

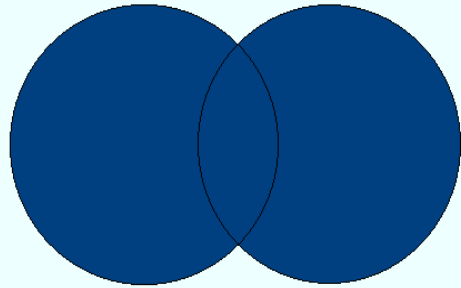
PROC SQL;

CREATE TABLE <TABLE_NAME> AS
  SELECT
    <VARLIST>
  FROM
    <PARENT_TABLE>
  Where
    <VAR_USED_SUBSET> in (SELECT DISTINCT <VAR_USED_SUBSET>
                          FROM <TABLE_W_VAR_USED_SUBSET> );

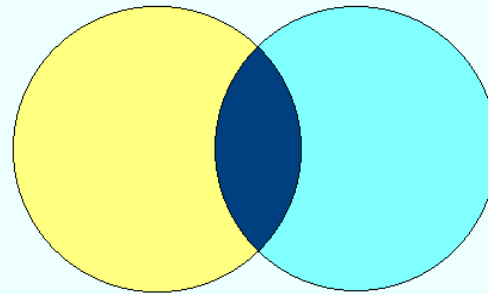
QUIT;

```

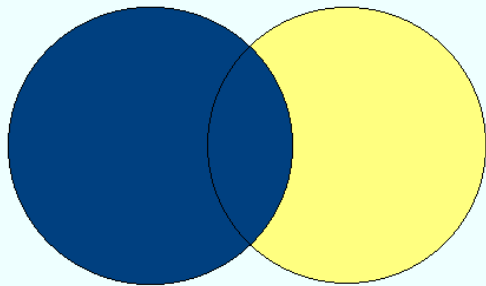
Combining Datasets: Joins



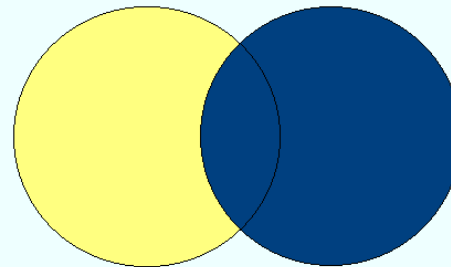
Full Join



Inner Join



Left Join



Right Join

Joins: Full Join

Records From Both datasets That Match + All Others

SAS Data step	Proc SQL
<pre> Proc Sort data = d1; By var; Run; Proc Sort data = d2; By var; Run; Data d3; Merge Claims(in = a) members(in= b); By var; If a OR b; Run; </pre>	<pre> Proc SQL; Create table d3 as Select * From d1 Full join d2 On d1.var= d2.var; Quit; </pre>

Joins: Inner Join

Only Those Records From Both datasets That Match

SAS Data step	Proc SQL
<pre>Proc Sort data = d1; By var; Run; Proc Sort data = d2; By var; Run; Data d3; Merge d1(in = a) d2(in= b); By var; If a and b; Run;</pre>	<pre>Proc SQL; Create table d3 as Select * From d1 Inner join d2 On d1.var= d2.var; Quit;</pre>

Joins: Left Join

Records From Both datasets That Match + All Remaining Records In *First* Dataset

SAS Data step	Proc SQL
<pre> Proc Sort data = d1; By var; Run; Proc Sort data = d2; By var; Run; Data d3; Merge d1(in=a) d2(in= b); By var; If a; Run; </pre>	<pre> Proc SQL; Create table d3 as Select * From d1 Left join d2 On d1.var= d2.var; Quit; </pre>

Joins: Right Join

Records From Both datasets That Match + All Remaining Records In *Second* Dataset

SAS Data step	Proc SQL
<pre>assume 2 datasets sorted... Data d3; Merge d1(in=a) d2(in= b); By var; If b; Run;</pre>	<pre>Proc SQL; Create table d3 as Select * From d1 Right join d2 On d1.var= d2.var; Quit;</pre>