**SAS Programming**

**Access Data -> Explore Data -> Prepare Data -> Analyze and Report on Data -> Export Results**

SAS Program Structure:

- Data Step

- Proc Step

- Global statements are defined above the code and they do not require run statement to get executed.

Comments:

- /\*\_\_\_\*/

- \*\_\_\_;

Types of Data:

- Structured-> defined rows and columns

- Unstructured-> no defined columns

SAS Table (Dataset):

Structured data with defined rows (observations) and columns (variables).

.sas7bdat file extension (Descriptor)

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

(Data)

Descriptor portion contains the metadata</> or the properties of the table such as table name, no. of rows, date/time created, column names, column attributes.

Data portion contains the data values stored in columns.

Column Attributes:

- Name

- Type -> Numeric, Characters, and Dates

- Length -> Number of bytes allocated for storing the column values.

* Numeric: 8 bytes (~16 significant digits)
* Character: 1-32,767 bytes (1 byte = 1 character)

SAS-date value is counted in number of days from 01Jan1960.

SAS-time value is counted in seconds from midnight.

SAS-datetime value is counted in seconds from 01Jan1960 midnight.

Contents procedure creates a report of the descriptor portion of the table.

PROC CONTENTS DATA=data-set;

RUN;

PROC CONTENTS DATA=libref.\_ALL\_ (Reads all tables from the library)

NODS; (Supresses the descriptor portion of the table in the report)

RUN;

Creating a SAS library:

LIBNAME libref engine “path/file-name.extension”;

(Keyword) (Library name) (Type of data) (Location);

LIBNAME is a global statement.

Engines:

* Base (default): for reading SAS tables
* XLSX: for reading excel files
* Teradata, Hadoop, etc.

libname mylib base “s: /workshop/data”;

Using library to access tables: libref.table-name;

Note: libref gets deleted every time the SAS session ends but the table is saved permanently. The deleted library can be re-established by simply resubmitting the code i.e. run the LIBNAME statement.

Automatic SAS libraries:

- Work Library (default) -> it is a temporary library.

DATA=work.test or DATA=test

- Sashelp Library -> it contains a collection of sample tables.

DATA=sashelp.cars

Using a library to read Excel files:

OPTIONS VALIDVARNAME=V7; (Force column names to adhere to SAS naming conventions)

LIBNAME libref XLSX “path/filename.xlsx”;

LIBNAME libref CLEAR; (Deletes the libref at the end of program)

Importing Unstructured Data:

PROC IMPORT DATAFILE=”path/filename” DBMS=file-type

OUT=output-table <REPLACE>;

RUN;

<REPLACE> overwrite the SAS output table if it already exists.

Importing a CSV file:

PROC IMPORT DATAFILE=”path/filename” DBMS=CSV

OUT=output-table <REPLACE>;

<GUESSINGROWS=n | MAX;>

RUN;

By default, SAS assumes column names are found in the first row of the file.

By default, SAS scans 20 rows of the data to make a guess of column attributes.

<GUESSINGROWS=n|MAX> provide n or max no. of rows to examine. This can also resolve the issue of truncation of character variables.

Importing an Excel file:

PROC IMPORT DATAFILE=”path/filename” DBMS=XLSX

OUT=output-table <REPLACE>;

SHEET=sheet-name;

RUN;

While importing an excel file, SHEET=sheetname; is added because PROC IMPORT can read only one spreadsheet at a time i.e. the first worksheet by default. PROC IMPORT creates a copy of the excel file.

Exploring Data with Procedures:

- PRINT ->

PROC PRINT DATA=libref.table-name (OBS=n);

VAR col1 col2 col3;

RUN;

OBS=n includes only n rows

NOOBS (Suppress the printing of observation numbers)

VAR statement limits the columns included in the report

- MEANS -> it computes the basic summary statistics of numeric columns i.e. N (Frequency Count), Mean, Std Deviation, Maximum and Minimum.

PROC MEANS DATA=libref.table-name;

VAR col4 col5;

RUN;

MAXDEC=0 (Round the values to the nearest whole number)

- UNIVARIATE -> it gives us the detailed summary statistics along with 5 extreme low and high values.

PROC univariate data=libref.table-name;

VAR col4 col5;

RUN;

- FREQ -> it generates a one-way frequency report of the each column i.e. Frequency, Percent, Cumulative Frequency and Cumulative Percent.

PROC FREQ DATA=libref.table-name

ORDER=FREQ; (Order the report by descending frequency)

TABLES col1 col2 col3;

RUN;

Filtering Rows of a Table:

WHERE expression;

= or ET

< or LT

> or GT

<= or LE

>=or GE

^= or ~= or NE

AND, OR

IN (value1, value2, value3)

NOT IN (value1, value2, value3)

SAS date constant -> “ddmmmyyyy”d

For two consecutive WHERE statements use WHERE ALSO in the second statement.

Filtering Data by Missing Values:

WHERE col=. ; for numeric missing value

WHERE col=” “; for character missing value

WHERE col IS MISSING;

WHERE col IS NOT MISSING;

WHERE col IS NULL;

WHERE col BETWEEN value1 AND value2; -> range where endpoints are inclusive

WHERE col LIKE “value\_”; wildcard for a character or WHERE col LIKE “value%”; wildcard for a string

Macro Variables: They store a text string.

Creating macro variables -> %LET varname=value; (Quotation marks are not included in the text value)

Using macro variables -> “&varname”

Formatting Data Values in Result:

PROC PRINT DATA=input-table;

FORMAT col-name(s) format;

RUN;

format -> <c>format<w>.<d>

<c> is the preceding character

<w> is the total format width

<d> is the no. of decimal places

Common Formats for Numeric Values:

|  |  |  |  |
| --- | --- | --- | --- |
| Format Name | Example Value | Format Applied | Formatted Value |
| w.d | 12345.67 | 5. | 12346 |
| w.d | 12345.67 | 8.1 | 12345.7 |
| COMMAw.d | 12345.67 | COMMA8.1 | 12,345.7 |
| DOLLARw.d | 12345.67 | DOLLAR10.2 | $12,345.67 |
| DOLLARw.d | 12345.67 | DOLLAR10. | $12,345 |

\*The formatted values are rounded off to the nearest tenth.

Common Formats for Date Values:

|  |  |  |
| --- | --- | --- |
| Value | Format Applied | Formatted Values |
| 21199 | DATE7. | 15JAN18 |
| 21199 | DATE9. | 15JAN2018 |
| 21199 | MMDDYY10. | 01/15/2018 |
| 21199 | DDMMYY8. | 15/01/18 |
| 21199 | MONYY7. | JAN2018 |
| 21199 | MONNAME. | January |
| 21199 | WEEKDATE. | Monday, January 15, 2018 |
| 21199 | DOWNAME3. | Mon |
| 21199 | YEAR4. | 2018 |

Sorting Data:

PROC SORT DATA=input-table <OUT=output-table>;

BY <DESCENDING> col-name(s); (The sorting order is ascending by default)

RUN;

It doesn’t generate printed output, so you have to print the sorted table.

Identifying and Removing Duplicates:

PROC SORT DATA=input-table <OUT=output-table>

<DUPOUT=output-table> (Write the removed duplicate rows into the output table)

NODUPRECS (Remove adjacent rows that are entirely duplicated);

BY \_ALL\_;

RUN;

PROC SORT DATA=input-table <OUT=output-table>

<DUPOUT=output-table>

NODUPKEY (Keeps only the first occurrence of each unique value);

BY col-name(s);

RUN;

Making a Copy of a table:

DATA output-table;

SET input-table.libref;

RUN;

DROP col-name(s); (Drops a column from the input table)

KEEP col-name(s); (Keeps a column from the input table)

When you use DROP or KEEP statement in the DATA step, the column is flagged for dropping or keeping in the PDV.

WHERE and FORMAT statements can also be used in the DATA Step. The FORMAT is permanently applied to a column in the properties of the new table.

Creating new columns:

DATA output-table;

SET input-table.libref;

new-column=expression; (ASSIGNMENT statement)

RUN;

Using Functions to create new columns:

function(argument1, argument2…);

|  |  |  |
| --- | --- | --- |
| Common Numeric Functions | Common Character Functions | Common Date Functions |
| SUM(num1, num2,…) | UPCASE(char) | MONTH(SAS-date) |
| MEAN(num1, num2,…) | LOWCASE(char) | YEAR(SAS-date) |
| MEDIAN(num1, num2,…) | PROPCASE(char, <delimiters>) | DAY(SAS-date) |
| RANGE(num1, num2,…) | CATS(char1, char2,…) | WEEKDAY(SAS-date) |
| MIN(num1, num2,…) | SUBSTR(string, start-position, <length>)=”char” | TODAY() |
| MAX(num1, num2,…) | COMPBL(string) | MDY(month, day, year) |
| N(num1, num2,…) | COMPRESS(string, <characters>) | YRDIF(start-date, end-date, ‘AGE’) |
| NMISS(num1, num2,…) | STRIP(string) | DATEPART(SAS-datetime) |
| LARGEST(k, value1, value2,…) | SCAN(string, n, <delimiters>) | TIMEPART(SAS-datetime) |
| RAND(‘distribution’, lower-Limit, upper-limit) | FIND(string, substring, <modifiers>) | INTCK(‘interval’, start-date, end-date, <‘method’>) |
| ROUND(number, rounding-unit) | LENGTH(string) | INTNX(‘interval’, start-date, increment, <’alignment’>) |
| CEIL(num) | ANYDIGIT(string) |  |
| FLOOR(num) | ANYALPHA(string) |  |
| INT(num) | ANYPUNCT(string) |  |
|  | TRANWRD(source, target, replacement) |  |
|  | CAT(string1, string2,…) |  |
|  | CATX(‘delimiter’, string1, string2,…) |  |

Method in INTCK function can be either Discrete (ending each Sunday) which is default or Continuous which requires ‘C’ as the fourth optional parameter.

By default, the INTNX function returns the first date of the interval. To view the last date of the interval ‘end’ must be added as the fourth optional parameter. Similarly, ‘middle’ can be used to view the mid date of the interval and ‘same’ can be used to view the same date of the interval as mentioned in the observation.

In FIND function, modifiers can be ‘I’ that makes the search case insensitive and ‘T’ that trims leading and trailing blanks. It returns a number which indicates the start position of the substring. If the substring is not found, it returns a zero.

LENGTH function returns 1 for a completely blank string.

CATS function removes leading and trailing blanks while CAT function doesn’t. CATX function inserts the delimiter between each string.

In the ROUND function the rounding-unit is .1 for the nearest tenth and .01 for the nearest hundredth.

IF-THEN/ELSE statement:

IF expression1 THEN statement1;

ELSE IF expression2 THEN statement2;

ELSE statement3;

When the DATA step is compiled, the first mention of **the column** determines the column name, type, and length. The length is determined by the value in the assignment statement.

LENGTH char-column $ length; (Defines the length of a character column in Data Step)

IF-THEN/DO statement:

DATA table1 table2;

SET libref.table-name;

IF expression THEN DO;

statement1;

statement2;

OUTPUT table1;

END;

ELSE DO;

statement1;

statement2;

OUTPUT table2;

END;

Titles & Footnotes:

TITLE<n> “title-text”;

FOOTNOTE<n> “footnote-text”;

These are global statements.

TITLE; (Clear titles)

FOOTNOTES; (Clear footnotes)

Labelling columns: label is listed as a column in the result.

LABEL col-name(s) =”label-text”;

(Label option must be included in the PROC PRINT statement)

Labels in PROC step are temporary and Labels in DATA step are permanent.

Segmenting Reports: segment reports based on the unique values of one or more columns.

First SORT DATA using the same BY-column then

BY variable(s); (Used in a reporting procedure)

The ID statement can be used in PROC PRINT statement to specify a column to identify rows and replace the OBS column.

Customizing Frequency Reports:

PROC FREQ DATA=libref.table-name <proc options>;

TABLES col-name(s) </ options>;

RUN;

ORDER=FREQ|FORMATTED|DATA

NLEVELS (Number of variable levels i.e. unique values)

NOPRINT (Supresses the output)

/ NOCUM (Eliminates cumulative columns)

/ NOPERCENT (Eliminates percent columns)

ODS - Output Delivery System

ODS GRAPHICS ON;

/ PLOTS=FREQPLOT(ORIENT=horizontal SCALE=percent);

By default, the orientation will be vertical and scale will be frequency counts.

ODS NOPROCTITLE; (Removes procedure title)

ODS PROCTITLE;

Two-Way Frequency Report:

PROC FREQ DATA=input-table <options>;

TABLES col-name\*col-name </ options>;

(Rows) (Columns)

It shows the Frequency, Percent, Row Pct. and Column Pct. of each combination of values in the two columns.

/ NOROW (Eliminates the row percent)

/ NOCOL (Eliminates the column percent)

/ LIST (Creates a list like frequency report)

/ CROSSLIST (Creates a cross list like frequency report)

/ OUT=output-table

Creating Summary Reports:

PROC MEANS DATA=input-table <stat-list>;

VAR col-name(s);

CLASS col-name(s); (Groups data based on unique values of the columns provided without requiring the data to be sorted first)

WAYS n; (n is the number of classification variables to segment the data)

OUTPUT OUT=output-table <statistic=col-name>; (Creates an output summary table)

RUN;

Exporting Data:

PROC EXPORT DATA=input-table OUTFILE=”output-file”

<DBMS=identifier> <REPLACE>;

RUN; (CSV, TAB, DLM, XLSX)

Exporting Reports:

SAS Procedure -> Output Objects -> ODS Destinations (XLSX, RTF, PPTX, PDF)

ODS <destination> <destination-specifications>;

/\* SAS code that produces output \*/

ODS <destination> CLOSE;

Exporting Results to a CSV file:

ODS CSVALL FILE=”filename.csv”;

/\* SAS code that produces output \*/

ODS CSVALL CLOSE;

Exporting Results to an Excel file:

ODS EXCEL FILE=”filename.xlsx” STYLE=style

OPTIONS(SHEET\_NAME=’label’);

OPTIONS(EMBEDDED\_TITLE=’yes’); (Print the titles above each report in the worksheets)

/\* SAS code that produces output \*/

ODS EXCEL CLOSE;

PROC TEMPLATE; (This procedure allows us to view the different styles that are available)

LIST STYLES;

RUN;

Exporting Results to a PPT file:

ODS POWERPOINT FILE=”filename.pptx” STYLE=style;

/\* SAS code that produces output \*/

ODS POWERPOINT CLOSE;

Exporting Results to a WORD file:

ODS RTF FILE=”filename.rtf” STARTPAGE=NO;

/\* SAS code that produces output \*/

ODS RTF CLOSE;

Exporting Results to a PDF file:

ODS PDF FILE=”filename.pdf” STYLE=style

STARTPAGE=NO (Eliminates the page break)

PDFTOC=n; (Control the levels of bookmarks that are open)

ODS PROCLABEL “label”; (Label the bookmark for the procedure)

/\* SAS code that produces output \*/

ODS PDF CLOSE;

SQL with SAS:

PROC SQL;

/\* SQL statement \*/

QUIT;

DATA Step Processing:

- Compilation phase:

1. Check for syntax errors.

2. Create the program data vector or PDV. It includes each column in the DATA step and its attributes i.e. column name, type and length.

3. Establish rules for processing data in the PDV.

4. Create descriptor portion of output table.

- Execution phase:

1. Initialize PDV.

2. Read a row from the input table into the PDV.

3. Sequentially process statements and update value in the PDV.

4. At end of the step, write the contents of the PDV to the output table.

PDV includes two additional columns \_ERROR\_ (which is assigned a value of 0 if there aren’t any errors and 1 if there is an error) and \_N\_ (the value of this automatic variable in the PDV represents the number of times the DATA step is iterated).

DROP, LENGTH, BY AND WHERE are compile-time statements.

PUTLOG \_ALL\_; (writes all columns and values in the PDV to the log)

PUTLOG column=; (writes selected columns and values in the PDV to the log)

PUTLOG “message”; (writes a text string to the log)

Implicit and Explicit Output:

At the end of DATA step an implicit OUTPUT; statement writes the data in the PDV as a row in the output table. Then an implicit RETURN; statement loops processing back to the top of the DATA step.

An explicit OUTPUT; statement anywhere in the DATA step takes control of the output and there is no implicit output.

OUTPUT table-name; statement also helps to write the output into multiple tables.

OUTPUT table(DROP=col-name(s));

OUTPUT table(KEEP=col-name(s));

Creating an Accumulating column:

RETAIN col-name <initial-value>; This is a compile-time statement that sets a rule for one or more columns to keep their value each time the PDV is reinitialized and establishes an initial value for them in the PDV.

RETAIN statement can also be used to determine the order of the columns in the output-table by using it in a separate DATA step.

Accumulating-col=sum (Accumulating-col, column to add); This assignment statement with sum() function is used to ignore missing values in the column to add.

All this can be achieved automatically using a shortcut statement i.e. Accumulating-col + column to add;

Processing Data in Groups:

Sort table into groups you want to analyse

PROC SORT DATA=input-table

<OUT=sorted-output-table>;

BY <DESCENDING> col-name(s);

RUN;

Processes data in sorted table by groups

DATA output-table;

SET sorted-output-table;

BY <DESCENDING> col-name(s);

RUN;

First.by-col and Last.by-col are added to the PDV. During the execution phase these temporary variables are assigned value of 0 and 1. The First.by-col variable is 1 for the first row within group and 0 for all other rows. The Last.by-col variable is 1 for the last row within group and 0 for all other rows.

Subsetting IF statement:

IF First.by-col=1; (Includes only the first row of each group)

IF Last.by-col=1; (Includes only the last row of each group)

Creating an Accumulating column within Groups:

BY col-name;

IF First.by-col=1 THEN Accumulating-col=0;

Accumulating-col + column to add;

Specifying column list as the Arguments of a Function:

(Quiz1, Quiz2, Quiz3, Quiz4, Quiz5), (of Quiz1--Quiz5), (of Quiz:)

\_NUMERIC\_, \_ALL\_, \_CHARACTER\_

This technique can also be used for formatting multiple columns based on a particular sequence or type.

Call Routines:

They perform a computation or a system manipulation based on the input provided in arguments. However, a call routine does not return a value, instead it alters column values.

CALL routine(col-name(s));

Arguments in a call routine must be supplied as column names since constants and expressions are not valid.

Converting Type of a column:

INPUT(source, informat); Converts a character value to a numeric value.

Converting the Type of an existing column:

Use table (RENAME=(original-column=renamed-column)); in the SET statement.

Then in an assignment statement original-column =input(renamed-column, informat);

Then drop renamed-column;

PUT(source, format); Converts a numeric value to a character value.

The ANYDTEDTEw. informat converts all date formats into raw numeric values.

SAS uses the DATESTYLE= system option to determine the order of month, day and year. The default value of DATESTYLE= option is LOCALE. If LOCALE= is set to English, then DATESTYLE sequence is MDY.

DATESTYLE=MDY;

DATESTYLE=DMY;

Custom Format:

PROC FORMAT;

VALUE format-name value-or-range-1 = ‘formatted-value’

value-or-range-2 = ‘formatted-value’

…;

RUN;

* Character formats - $ followed by letter or underscore
* Numeric formats - begin with letter or underscore
* Up to 32 characters long

Range of values:

X –Y includes value of X and all values up to and including Y

X-<Y includes value of X and all values less than Y

X<-Y includes values greater than X up to and including Y

Keywords can also be used on the VALUE statement for specifying range where **Low** represents the lowest possible value and **High** represents the highest possible value.

**Other** keyword can be used as a single value which includes all values that don’t match any other value or range.

Reading a table of values for a format: (Required for numeric range values)

|  |  |  |  |
| --- | --- | --- | --- |
| FmtName | Start | End | Label |
|  |  |  |  |
|  |  |  |  |

PROC FORMAT CNTLIN=libref.table-name;

RUN;

Storing Custom Formats:

By default, custom formats are stored in temporary WORK library in a catalog named Formats.

To permanently store custom formats:

PROC FORMAT lib=libref.catalog-name; (Formats is the default catalog for any library)

Use OPTIONS FMTSEARCH=(); global statement to specify the location where the custom format has been saved.

To generate a report for all formats in the WORK library:

PROC FORMAT FMTLIB library=work;

RUN;

Concatenating tables:

When we have two or more tables with same columns

DATA output-table;

SET input-table1 input-table2 …;

…additional statements…

RUN;

Note: All columns from both input tables are included in the output table.

Merging tables:

One-to-One merge & One-to-Many merge

DATA output-table;

MERGE input-table1 input-table2 …;

BY col-name(s); (Data must be in sorted order)

RUN;

Note: By default, the DATA step writes both matching and nonmatching rows to the output table.

Identifying Matches and Nonmatching rows:

DATA output-table;

MERGE input-table1(IN=inTable1)

input-table2(IN=inTable2);

BY col-name(s);

IF inTable1 =1 AND inTable2=1; (Include matching rows only)

RUN;

Iterative DO Loops:

DO index-column=start TO stop <BY increment>;

…repetitive code…

END;

Note: If an increment value is not specified, the default increment is 1.

DO index-column=index1, index2, index3…;

…repetitive code…

END;

Conditional DO Loops:

DO WHILE (expression); (condition is checked)

…repetitive code…

END;

DO UNTIL (expression);

…repetitive code…

END; (condition is checked)

Note: The DO UNTIL loop always executes at least once. SAS statements in the DO loop are executed until the specified condition is true.

Structure of a table:

- Wide table (The measures are split into multiple columns)

- Narrow table (The measures are all stacked in a single column)

Note: A Narrow table is perfect for most reporting and analytics, including PROC MEANS, PROC FREQ, or graphics procedure.

Transpose of a table:

PROC TRANSPOSE DATA=input-table <OUT=output-table> <options>;

<VAR col-name(s)>; (Variables to be transposed or which columns from the input-table shall be displayed as rows in the output-table)

<BY col-name(s)>; (To transpose values within groups into rows)

<ID col-name>; (Entries of which column in the input-table shall be used as a list of columns in the output-table)

RUN;

PREFIX=column; (Specifies prefix for the column(s) that contain the transposed data values)

NAME=column; (Name the column containing column-names that were transposed)

Note:

ID column is used when converting a narrow table into a wide table.

If a VAR statement is not used, all numeric columns in the table are transposed.