

SAS Base Cheat Sheet

esuriddick

Data Creation

```
data DATASET_NAME;  
input <variable-01>$ (if character) <variable-02>...  
datalines;  
<variable-01-value-01> <variable-02-value-01>  
<variable-01-value-02> <variable-02-value-02>  
...;  
run;
```

Data Import

From existing dataset

```
data DATASET_NAME;  
set <library-name>.ORIGINAL_DATASET;  
run;
```

File extensions: .txt and .csv

```
data DATASET_NAME;  
infile 'FILE_LOC/FILE_NAME.FILE_EXTENSION'  
dlim = ',';  
input <variable-01> $ (if character) <variable-02>...  
run;
```

```
data DATASET_NAME;  
infile 'FILE_LOC/FILE_NAME.FILE_EXTENSION'  
input <variable-01> $ (if character) <first-column>-  
|last-column>  
<variable-02> $ (if character) <first-column>-|last-  
column>  
...  
run;
```

```
data DATASET_NAME;  
infile 'FILE_LOC/FILE_NAME.FILE_EXTENSION'  
input  
@<first-column><variable-01> $ (if character) <infor-  
mat>  
@<first-column><variable-02> $ (if character) <infor-  
mat>  
...  
run;
```

File extensions: .xls and .xlsx

```
proc import datafile='FILE_LOC/FILE_NAME'  
DBMS=FILE_EXTENSION out=OUTPUT_NAME  
replace;  
getnames=no; (classifies first row as data instead of  
headers)  
sheet='SHEET_NAME'; (specifies the sheet to import  
from)  
range='SHEET_NAME$A1:B6'; (specify ranges to im-  
port)  
run;
```

Library assignment

```
libname <library-name> '<directory-location>'  
access=readonly (optional);
```

Variable creation, length, label & format

<variable> = <expression>;

```
length <variable-01> <variable-02> $ (if charac-  
ter) <length>  
<variable-03> $ (if character) <length>  
...; (prior to creating the variable in the data step)
```

```
label <variable-01> = '<variable-01-label>'  
<variable-02> = '<variable-02-label>'  
...;
```

```
format <variable-01> <variable-02> <format-name>  
<variable-03><format-name>  
...;
```

Variable formats

Format	Specifies that...	values	Example
COMMAw.d	contains commas and de- cimal places		comma8.2 (5,678.90)
DOLLARw.d	contain dollar signs, commas and decimal places		dollar6.2 (\$23.12)

Format	Specifies that...	values	Example
w.	is rounded to the nearest integer in w spaces		4. (1347)
w.d	is rounded to d decimal places in w spaces		5.2 (12.67)
\$w.	has character values in w spaces		\$12.
MMDDYYw.	has date values of the form 09/12/97 (MMDDYY8.) or 09/12/1997 (MMDDYY10.)		mmddyy10. (09/12/1997)
DATEw.	has date values of the form 16OCT99 (DATE7.) or 16OCT1999 (DATE9.)		date9. (16OCT1999)

Conditional Processing

Single action

```
data DATASET_NAME;  
set <library-name>.ORIGINAL_DATASET;  
if <condition-01> then <action>;  
else if <condition-02> then <action>;  
else <action>;  
run;
```

Multiple actions

```
data DATASET_NAME;  
set <library-name>.ORIGINAL_DATASET;  
if <condition-01> then do;  
    <action-01>;  
    <action-02>;  
    ...;  
end;  
else if <condition-02> then do;  
    ...;  
end;  
else do;  
    ...;  
end;  
run;
```

Logical Operators

Symbolic	Mnemonic	Meaning
=	EQ	Equals
^=, ^=, ~=	NE	Not equal
>	GT	Greater than
<	LT	Lower than
>=	GE	Greater than or equal
<=	LE	Lower than or equal
	(NOT) IN	Compares the value of a variable to a list of values
&	AND	All comparisons must be true
—, !,	OR	Only one comparison must be true

Looping

do <variable> = <start-number> **to** <stop-number>
by <increment>;

SAS statements

output; (optional: to create an observation for each iteration)
end;

do <variable> = <value-1>, <value-2>, <value-3> ...;

SAS statements

output; (optional)

end;

do until (<expression>;

SAS statements

output; (optional)

end;

do while (<expression>;

SAS statements

output; (optional)

end;

do <variable> = <start-number> **to** <stop-number>
by <increment> **until while** (<expression>;

SAS statements

output; (optional)

end;

Arrays

Variable	Form	Example
Numbered range of variables	Var_1-Var_n	array sales(4) qtr1-qtr4;
All numeric variables	_NUMERIC_	array sales(*) _numeric_;
All character variables	_CHARACTER_	array sales(*) _character_;
All character or all numeric variables	_ALL_	array sales(*) _all_;

Create an array

array <array-name>(<dimension>) **_temporary_** <variable-01> <variable-02> ... (<value-01> <value-02> ...);

Notes: **i)** you do not need to specify the individual variable names if you specify the number of elements in the array; **ii)** you may optionally assign initial values to the variables; and **iii)** temporary arrays do not appear in the resulting data set.

dim(<array-name>); (to know the size of the array)

Data Subsetting

If vs. Where

data DATASET_NAME;

set <library-name>.ORIGINAL_DATASET;

if/where <expression>; (e.g., if gender = 'female')

run;

Notes: **i) where** can be used inside procedures; **ii) if** can be applied on newly created variables or automatic ones (e.g., _N_, first.<variable>, last.<variable>); and **iii) where** is more efficient.

Delete

data DATASET_NAME;

set <library-name>.ORIGINAL_DATASET;

if <expression> **then delete**; (e.g., if gender = 'female')

run;

Keep or drop variables

data DATASET_NAME (**keep** = <variable-01> <variable-02> ... **drop** = <variable-03> <variable-04> ...);

set <library-name>.ORIGINAL_DATASET;

run;

Data Combination

Type	Description
One-to-One	Contains all variables from all the datasets, and the number of observations is equal to the number of observations in the smallest dataset. It is worth noting that observations are combined based on their relative position in each dataset (that is, first observation is joined together with first observation, and so on). If the datasets contain variables with same names, the values from the last dataset overwrite the values from earlier datasets.
Concatenate	Contains all of the observations and variables from of the datasets.
Append	Same as concatenating, but the base dataset is modified instead of created and the base dataset is not read during the process.
Interleaving	First of all, be sure that each dataset is sorted or indexed in ascending order based on the BY variable(s). The same as concatenation, but observations in each BY group and dataset are read sequentially.
Match	Combines observations from two or more datasets into a single observation in a new dataset according to the values of a common variable. Be aware that the BY variable(s) must be previously sorted, and variables with the same name are overwritten by the latest data set with same variable name. Finally, just like when renaming variables, you can also drop or keep variables within the merge statement if they are not used in a if statement inside the data step.

One-to-One

data DATASET_NAME;

set <library-name>.DATASET_01;

set <library-name>.DATASET_02;

...;

run;

Concatenate

data DATASET_NAME;

set <library-name>.DATASET_01 <library-name>.DATASET_02 ...;

run;

Append

```
proc append base=<library-name>.DATASET_01;  
data=<library-name>.DATASET_02 FORCE (optional for when datasets have unmatching variable definitions);  
run;
```

Interleaving

```
data DATASET_NAME;  
set <library-name>.DATASET_01 <library-name>.DATASET_02 ...;  
by <variable-01> <variable-02> ...;  
run;
```

Match

```
data DATASET_NAME;  
merge <library-name>.DATASET_01 <library-name>.DATASET_02 ...;  
by <descending> <variable-01> <variable-02> ...;  
run;  
<descending> is optional.
```

Renaming variables

```
data DATASET_NAME;  
merge DATASET_01 (rename=<old-name>=<new-name>)  
DATASET_02 (rename=<old-name>=<new-name>)  
...;  
by <variable-01> <variable-02> ...;  
run;
```

Excluding unmatched observations

```
data DATASET_NAME;  
merge  
DATASET_01 (in=<temp-variable-name-01>)  
DATASET_02 (in=<temp-variable-name-02>)  
...;  
by <variable-01> <variable-02> ...;  
if <temp-variable-name-01> EQ 1 &  
<temp-variable-name-02> EQ 1; (INNER JOIN)  
if <temp-variable-name-01> EQ 1; (LEFT JOIN)  
if <temp-variable-name-02> EQ 1; (RIGHT JOIN)  
run;
```

Macros

Definition of macro variables

```
%let <variable-name> = <variable/object>;  
Note: Referred to with &<variable-name> (optionally, a “.” might be added last to indicate the end of the macro variable’s name).
```

Basic macro and macro parameters

```
%macro <macro-name>(<param-01>, <param-02>, ...);  
result = &<param-01> + &<param-02>; (e.g., variable creation)  
%mend <macro-name>;
```

Macro invocation

```
%<macro-name>
```

Note: macro invocation pastes the whole text/operations within the macro to this location, which is why “;” might not be required after the invocation.

Other

```
%put <variable-name>; (write text/macrovariable information to the SAS log)
```

SAS Procedures

TRANSPOSE

```
PROC TRANSPOSE data=<old-dataset>  
prefix=<prefix-id-name>(optional) out=<new-dataset>;  
by <variable-list>; (optional: if you have any grouping variables that you want to keep as variables. Dataset must be sorted before transposing)  
id <variable-list>; (optional: names the variable whose formatted values will become the new variable names. Otherwise, they will be named COL1, COL2, ...)  
var <variable-list>; (required: variables whose values you want to transpose)  
run;
```

SORT (required when using a ‘by’ statement)

```
PROC SORT data=<original-dataset> out=<new-dataset> NODUPKEY (optional) DUPOUT  
=<dataset-with-removed-observations> (optional);  
by descending (optional) <variable-01> ...;  
run;
```

PRINT

```
PROC PRINT data=<dataset>  
obs=‘<observations-label>’ (optional) noobs (optional) sumlabel = ‘<text>’ (optional) grandtotal_label = ‘<text>’ (optional);  
title ‘<text>’; (optional)  
var <variable-01> <variable-02> ...; (optional)  
ID <variable(s)-to-emphasize>; (optional)  
where <variable> = <condition>; (optional)  
<summary-function> (e.g., sum) <variable>; (optional)  
format <variable> <format>; (optional)  
by <group-variable(s)>; (optional - useful for when a summary function like ‘sum’ is used)  
run;
```

SUMMARY

```
PROC MEANS data=<dataset> MAXDEC =  
<number-digits> MISSING (optional: to treat missing values as valid summary groups for ‘class’)  
<summary-statistic(s)> (optional);  
title ‘<text>’; (optional)  
by <group-variable(s)>; (optional)  
class <group-variable(s)>; (optional: similar to ‘by’ but output is more compact and it does not require data to be sorted)  
var <variable(s)-to-assess>; (optional)  
run;  
Summary statistics: MAX, MIN, MEAN, MEDIAN, MODE, N (number of non-missing observations), NMISS, RANGE, STDDEV and SUM.
```

FREQUENCY TABLE

```
PROC FREQ data=<dataset>;  
title ‘<text>’; (optional)  
TABLES <variable-01> <variable-01> * <variable-02> / <option(s)>;  
run;  
Options: LIST (for list format instead of grid), MISSPRINT (includes missing values in frequencies), MISSING (includes missing values in frequencies and percentages), NOCOL (suppresses printing of column percentages in cross-tabulations), NOPERCENT (suppresses printing of percentages), NOROW (suppresses printing of row percentages in cross-tabulations) and OUT =<output-dataset>.
```

UNIVARIATE ANALYSIS

```
PROC UNIVARIATE data=<dataset>;  
title '<text>'; (optional)  
var <variable(s)>;  
by <group-variable(s)>; (optional)  
run;
```

CONTENT OF A SAS LIBRARY/DATASET

```
PROC CONTENT data=<libname>._ALL_ NODS  
(optional: to suppress the description of each SAS  
dataset within the report);
```

```
run;  
Note: you may specify a specific dataset within a  
library instead of assessing all the datasets within the  
library.
```

EXPORT TO .CSV AND .XLSX

```
PROC EXPORT data=<dataset>  
(where=(<variable(s)> = '<condition>'))  
DBMS = FILE.EXTENSION (e.g., csv, xlsx, etc.)  
OUTFILE = <output-location>  
REPLACE; (optional: overwrite if file already exists)  
DELIMITER = '<delimiter>'; (optional)  
SHEET = '<name-worksheet>'; (optional)  
run;
```

SQL Procedures

SELECT - General Form

```
PROC SQL  
CREATE TABLE <dataset-name> AS  
SELECT <variable(s)> (split by commas) (mandatory)  
FROM <original-dataset> (mandatory)  
WHERE <condition(s)>  
GROUP BY <variable(s)> (either: use an aggregate  
function in the SELECT clause or use HAVING clause to  
instruct how to group the data)  
HAVING <condition(s)>  
ORDER BY <variable(s)>;  
QUIT;
```

INNER JOIN (joins matching rows)

PROC SQL

```
SELECT x.<variable(s)>, y.<variable(s)>  
FROM <dataset-01> AS x, <dataset-02> AS y  
WHERE <dataset-01>.<variable> = <dataset-  
02>.<variable>
```

QUIT;

Note: Data does not have to be sorted prior to this operation.

OUTER JOIN (joins non-matching rows)

LEFT JOIN (data from dataset-02 into dataset-01)

PROC SQL

```
SELECT x.<variable(s)>, y.<variable(s)>  
  
FROM <dataset-01> AS x LEFT JOIN  
<dataset-02> AS y  
ON <dataset-01>.<variable> = <dataset-  
02>.<variable>
```

QUIT;

RIGHT JOIN (data from dataset-01 into dataset-02)

PROC SQL

```
SELECT x.<variable(s)>, y.<variable(s)>  
  
FROM <dataset-01> AS x RIGHT JOIN  
<dataset-02> AS y  
ON <dataset-01>.<variable> = <dataset-  
02>.<variable>
```

QUIT;

FULL OUTER JOIN (all data used)

PROC SQL

```
SELECT x.<variable(s)>, y.<variable(s)>  
  
FROM <dataset-01> AS x FULL JOIN  
<dataset-02> AS y  
ON <dataset-01>.<variable> = <dataset-  
02>.<variable>
```

QUIT;

Note: Data does not have to be sorted prior to these operations.

Reports Generation

General Form

```
ODS <format-01> file=<output-destination-01>  
<option(s)>;
```

...

```
ODS <format-n> file=<output-destination-n> <op-  
tion(s)>;
```

```
<SAS-program(s)>
```

```
ODS <format-01> close;
```

...

```
ODS <format-n> close;
```

Note: Optionally, you may use the command **ODS _ALL_ CLOSE;** to close simultaneously the previous ODS commands.

PDF - Example

```
ODS pdf file='/folders/xxx/<filename>.pdf';
```

```
<SAS-program(s)>
```

```
ODS pdf close;
```

Excel - Example

```
ODS excel file='/folders/xxx/<filename>.xlsx' opti-  
ons(sheet_interval = 'bygroup' -> specifies that a new  
sheet is created after each 'by' group  
sheet_label = 'Gender = ' -> specifies the prefix of a  
worksheet label
```

```
embedded_titles = 'yes' -> specifies that the title  
created by the TITLE statement is embedded in the  
Excel worksheet
```

```
embed_titles_once = 'yes'); -> specifies that the title  
created by the TITLE statement is embedded once, at  
the top of each sheet
```

```
TITLE 'Summary by Gender';
```

```
ODS NOPROCTITLE; (suppresses the writing  
of the title of the procedure that produces the results -  
e.g., PROC MEANS)
```

```
<SAS-program(s)>
```

```
ODS excel close;
```

Numeric Functions

Function	Description	Form
ABS	Removes the negative sign from the value.	abs (<value>)
CEIL	Returns the largest integer value.	ceil (<value>)
CONSTANT	Returns values of commonly used mathematical constants such as pi and e.	constant (<constant>) <constant> = 'Pi'
EXP	Raises e to the value provided in its argument (e ^x).	exp (<value>)
FLOOR	Returns the smallest integer value.	floor (<value>)
INT	Returns the integer part of a numeric value.	int (<value>)
LARGEST	Returns the n-th largest value in a list of values.	largest (<n-th>, <list-values>)
LOG	Returns the natural logarithm of its argument. To return a base 10 logarithm, use log10 instead.	log (<value>)
MAX	Returns the largest value of its arguments.	max (<values>)
MEAN	Returns the mean of its arguments.	mean (<value>)
MIN	Returns the smallest value of its arguments.	min (<value>)
ROUND	Round numbers to the nearest integer or to other place values.	round (<value>, <round-off-unit>) For two decimal places, <round-off-unit>=.01 or 2
SMALLEST	Returns the n-th smallest value in a list of variables.	smallest (<n-th>, <list-values>)
SQRT	Returns the square root of its arguments.	sqrt (<value>)
SUM	Calculates the sum of observations.	sum (<values>)

Character Functions

Function	Description	Form
CATX	Concatenates character strings, removes leading and trailing blanks, and inserts separators.	catx ('<separator>', <string-01>, ..., <string-n>)
FIND	Searches for a specific substring of characters within a character string.	find (<variable>, <expression>, <modifier>, <start-position>)
INDEX	Search a character expression for a string of characters and returns the position of the string's first character for the first occurrence of the string.	index (<variable>, <expression>) <expression> must be a character string enclosed in quotation marks (')
LOWCASE	Converts all letters in a value to lower-case.	lowcase (<variable>)
PROPCASE	Converts all letters in a value to proper case.	propcase (<variable>)
SCAN	Returns a specified word from a string.	scan (<variable>, <word-number>, '<delimiters>')
SUBSTR	Extracts a substring or replaces character values.	substr (<variable>, <start-position>, <number-characters-to-extract>)
TRANWRD	Replaces or removes all occurrences of a pattern of characters within a character string.	tranwrd (<variable>, <expression>, <replacement>)
TRIM	Trims trailing blanks from character values.	trim (<variable>)
UPPERCASE	Converts all letters in a value to upper-case.	uppercase (<variable>)

Date & Time Functions

Function	Description	Form
MDY	Create a SAS date value from numeric values that represent the month, day and year.	mdy (<month>, <day>, <year>)
YEAR	Extracts the year value from a SAS date value.	year (<date>)
QTR	Extracts the quarter value from a SAS date value.	qtr (<date>)
MONTH	Extracts the month value from a SAS date value.	month (<date>)
DAY	Extracts the day value from a SAS date value.	day (<date>)
WEEKDAY	Extracts the day of the week from a SAS date value (e.g., 1 is Sunday).	weekday (<date>)
DATE TODAY	Returns the current date from the system clock as a SAS date value.	date () today ()
INTCK	Returns the number of time intervals (day, weekday, week, tenday, semimonth, month, qtr, semiyear or year) that occur in a given time span.	intck (<interval>, <start-date>, <end-date>)
DATDIF	Calculates the difference in days between two SAS dates.	datdif (<start-date>, <end-date>, <basis>) [<basis>=30/360, ACT/ACT, ACT/360 or ACT/365]
YRDIF	Calculates the difference in years between two SAS dates.	yrdif (<start-date>, <end-date>, <basis>) [<basis>=30/360, ACT/ACT, ACT/360 or ACT/365]