Dr. Ali Ajouz

Data analyst with a strong math background and experience in machine learning, computational number theory , and statistics. Passionate about explaining data science to non-technical business audiences.

SAS CHEAT SHEET

Version: 1.0.2

- *The following example codes often works. Refer to SAS docs for all arguments.*

April 17, 2017

Linkedin: https://www.linkedin.com/in/ajouz/ Email: aliajouz@gmail.com

From Raw Data to Final Analysis

1. You begin with *raw data*, that is, a collection of data that has not yet been processed by SAS.
2. You use a set of statements known as a *DATA step* to get your data into a *SAS data set*.
3. Then you can further process your data with additional DATA step programming or with *SAS procedures*.

Reading raw data from instream data lines

```
data names;
   infile datalines delimiter=',';
   length first last $25.;
   input first $ last $;
   datalines;
      Ali, Ajouz
      Karam, Ghawi
;
run;
```

Obs	first	last
1	Ali	Ajouz
2	Karam	Ghawi

DATA STEP

INFILE: identifies an external file to read

INPUT: specifies the variables in the new data

LENGTH: the number of bytes for storing variables

DATALINES: indicates that data lines follow

RUN: execute the DATA step.

Some useful infile options are: DELIMITER, MISSOVER, DLM, FIRSTOBS

Reading raw data from external file

```
proc import
datafile="C:\file.csv"
        out=outdata
        dbms=csv
        replace;
        getnames=yes;
```

Using: PROC IMPORT

DATAFILE: the location of file you want to read.

DBMS : specifies the type of data to import.

```
data names;
   infile "file-path" delimiter=',';
   length first last $25.;
   input first $ last $;
run;
```

Using: DATA STEP

using INFILE statement to identify an external file to read with an

INPUT statement

Reading raw data (3)

INFORMATS
Informats tell SAS how to read the data.

VS

FORMATS
formats tell SAS how to display the data.

USING FORMAT AND INFORMAT

```
data names;
infile datalines delimiter=',';
informat first $15. last $15. birthday ddmmyy10.;
format birthday date9.;
input first $ last $ birthday;
datalines;
Ali,Ajouz, 04/11/1980
;
```

first	last	birthday
Ali	Ajouz	04NOV1980

POPULAR INFORMATS

$w.	Reads in character data of length *w*.
w.d	Reads in numeric data of length *w* with *d* decimal points
MMDDYYw.	Reads in date data in the form of 04-11-80

Reading raw data (4)

Beob.	FullName	Addresse
1	Ali Ajouz	Danziger str.7
2	Karam Ghawi	Berliner str.32

We can use multiple INPUT statements to read group of records as single observation.

We can use double trailing @@ to read a record into a group of observations

Single observations from multiple records

```
data multilines;
        infile datalines;
        length FullName Addresse $36;
        input FullName $36.;
        input Addresse $36.;
        datalines;
        Ali Ajouz
        Danziger str.7
        Karam Ghawi
        Berliner str.32
        ;
```

Multiple observations from single record

```
data multilines;
infile datalines delimiter=",";
length FullName Addresse $36;
input FullName$  Addresse $ @@;
datalines;
Ali Ajouz, Danziger str.7, Karam Ghawi,Berliner
str.32
;
run;
```

Selecting columns

Selecting columns by name

```
data selected;
      set sashelp.cars;
      keep model type;
```

Deleting columns by name

```
data selected;
      set sashelp.cars;
      drop model type;
```

Change column label

```
data selected;
      set sashelp.cars;
      label model="car model"
            type="car type";
```

Selecting columns by position

```
%let lib= sashelp;
%let mydata= class;
%let to_keep=(1:3);  /* range */
proc sql noprint;
select name into :vars separated by ' '
from dictionary.columns
where  libname="%upcase(&lib.)" and
memname="%upcase(&mydata)"
and varnum in &to_keep;
quit;
data want;
      set &lib..&mydata (keep=&vars.);
```

OR

```
%let to_keep=(1,3,5);  /*list */
```

Selecting Rows

Ignore the first N rows

```
data want;
      set sashelp.buy (firstobs=5);
```

Limit numbers of rows to be read

```
data want;
      set sashelp.buy (obs=5);
```

Select rows by using IF statement

```
data want;
      set sashelp.buy;
      if amount >=-1000;
```

Selecting rows by index

```
data want;
      do Slice=2,1,7,4;
            set sashelp.buy point=Slice;
            output;
        end;
         stop;
run;
```

Random sample

```
proc surveyselect data=sashelp.cars
  method=srs n=10 seed=12345
out=SampleSRS;
run;
```

Data attributes + columns list

```
proc datasets ;
   contents
data=SASHELP.HEART
order=collate;
quit;
```

Frequency tables

```
proc freq data=sashelp.class;
   tables SEX/ plots=freqplot;
```

Data summary statistics

```
proc means data=sashelp.iris
N mean std min max q1 median
q3;
```

Peek at the data set contents

Data header (first 5 rows)

```
proc print data=sashelp.class (obs=5);
```

Number of rows

```
data _NULL_;
   if 0 then set sashelp.class nobs=x;
   call symputx('nobs', x);
   stop;
run;
%put #rows = &nobs;
```

Sorting a data set

```
data Score ;
   infile datalines dsd;
   input Name$ Score;
   datalines;
      Ali,66
      Karam,82
      Roaa,57
      Hala,57
;
proc sort data=Score out=sorted;
   by  descending score;
run;
proc print data=sorted;
   var Name Score;
   title 'Listed by descending Score';
run;
```

Listed by descending Score

Obs	Name	Score
1	Karam	82
2	Ali	66
3	Roaa	57
4	Hala	57

The *Sort procedure* order SAS data set observations by the values of one or more character or numeric variables .

Arithmetic Operators

+	addition
-	subtraction
*	multiplication
**	exponentiation
/	division

Logical operators

&	AND
|	OR
^	NOT

Comparison operators

=	EQ	Equal to
^=	NE	Not Equal to
>	GT	greater than
<	LT	less than
>=	GE	greater than or equal to
<=	LE	less than or equal to
	IN	is one of

Some SAS Operators

A SAS ***operator*** is a symbol that represents a comparison, arithmetic calculation, or logical operation; a SAS function; or grouping parentheses.

```
data names;
    infile datalines;
    input Name$;
    datalines;
Ali
Karam
Roaa
Hala
;
data titles;
    set names;
    if name="Ali" then
       do;
            title="Papa";
       end;
    else if name="Karam" then
       do;
            title="Mama";
       end;
    else
       title="Kid";
```

If statement

Obs	Name	title
1	Ali	Papa
2	Karam	Mama
3	Roaa	Kid
4	Hala	Kid

SAS evaluates the expression in an *IF-THEN/ELSE* statement to produce a result that is either nonzero, zero, or missing. A nonzero and nonmissing result causes the expression to be true; a result of zero or missing causes the expression to be false.

```
data names;
    infile datalines;
    input Name$;
    datalines;
Ali
Karam
Roaa
Hala
;
data titles;
    set names;
    select (name);
        when ("Ali") title="Papa";
        when ("Karam")
title="Mama";
        otherwise title="Kid";
    end;
run;
```

Select statement

Obs	Name	title
1	Ali	Papa
2	Karam	Mama
3	Roaa	Kid
4	Hala	Kid

Select VS If:

When you have a long series of mutually exclusive conditions, using a *SELECT group* is more efficient than using a series of *IF-THEN* statements because CPU time is reduced.

DO Statement

```
data df;
   do x=1 to 3;
      y=x**2;
      output;
   end;
run;
```

```
data df;
   do x=1,2,3;
      y=x**2;
      output;
   end;
run;
```

```
data df;
   x=1;
   do while (x<4);
      y=x**2;
      output;
      x=x + 1;
   end;
run;
```

```
data df;
   x=1;
   do until (x>3);
      y=x**2;
      output;
      x=x + 1;
      end;
run;
```

Obs	x	y
1	1	1
2	2	4
3	3	9

The basic iterative *DO statement* in SAS has the syntax *DO value = start TO stop*. An *END statement* marks the end of the loop, as shown in the following examples.

Transposing data with the data step

Step 1: Sample data

```
data raw;
infile datalines
delimiter=",";
input ID $ Name$ Try
result;
datalines;
1,Ali,1,160
1,Ali,2,140
2,Karam,1,141
2,Karam,3,161
;
```

Step 2: Sorting by ID

```
proc sort
  data=raw;
       by id;
run;
```

Step 3: Transposing the data

```
data rawflat;
 set raw;
by id;
keep id Name Try1-Try3;
retain Try1-Try3;
array myvars {*} Try1-Try3;

if first.id then
       do i = 1 to
dim(myvars);
                myvars{i} = .;
       end;

myvars{Try} = result;
if last.id;
run;
```

ID	Name	Try	result
1	Ali	1	160
1	Ali	2	140
2	Karam	1	141
2	Karam	3	161

ID	Name	Try1	Try2	Try3
1	Ali	160	140	.
2	Karam	141	.	161

Labels and Formats

```
data Salary;
    infile datalines dsd;
    input Name$ Salary;
    format Salary dollar12.;
    Label Name="First Name"
          Salary="Annual Salary";
    datalines;
      Ali,45000
      Karam,50000
;
run;
proc print data=Salary label;
run;
```

Name	Salary
Ali	45000
Karam	50000

First Name	Annual Salary
Ali	$45,000
Karam	$50,000

We can use *LABEL Statement* to assigns descriptive labels to variables. Also, *FORMAT Statement* to associates formats with variables.

```
data Score ;
   infile datalines dsd;
   input Name$ Score;
   datalines;
      Ali,66
      Karam,82
      Roaa,57
      Hala,57
;
proc rank data=Score
out=order descending ties=low;
   var Score ;
   ranks ScoreRank;
run;

proc print data=order;
   title "Rankings of Participants'
Scores";
run;
```

Ranking

Rankings of Participants' Scores

Obs	Name	Score	ScoreRank
1	Ali	66	2
2	Karam	82	1
3	Roaa	57	3
4	Hala	57	3

The *RANK procedure* computes ranks for one or more numeric variables across the observations of a SAS data set and writes the ranks to a new SAS data set.

Working with numeric values

function-name (var1, var2, ..., varn)

Some useful functions

SUM	Sum of nonmissing arguments
MEAN	Average of arguments
MIN	Smallest value
MAX	Largest value
CEIL	Smallest integer greater than or equal to argument
FLOOR	Greatest integer less than or equal to argument

Apply functions

to all variables x1 to xn

function-name(of x1-xn)

to all variables begin with same string

function-name(of x:)

to special name list

function-name(of _ALL_)
function-name(of _Character_)
function-name(of _Numeric_)

Summarizing data

Summarizing data by groups

```
proc sort data=sashelp.class
out=Class;
      by sex;

data Class (keep= sex count);
      set Class;
      by sex;
      if first.sex then
            do;
                  count=0;
            end;
      count+1;
      if last.sex;
```

Accumulating variable using (+)

```
data total;
      set sashelp.buy;
      TotalAmount + Amount;
```

In the DATA step, SAS identifies the beginning and end of each BY group by creating two temporary variables for each BY variable: FIRST.*variable* and LAST.*variable*.

Beob.	Sex	count
1	F	9
2	M	10

A *missing value* is a valid value in SAS. A missing character value is displayed as a blank, and a missing numeric value is displayed as a period. *STDIZE procedure* with the *REPONLY option* can be used to replace only the missing values. The *METHOD= option* enables you to choose different location measures such as the mean, median, and midrange. Only numeric input variables should be used in PROC STDIZE.

Working with missing values

Step 1: Setup

```
/* data set*/
%let mydata
=sashelp.baseball;
/* vars to impute */
%let inputs = salary;
/* missing Indicators*/
%let MissingInd =
MIsalary;
```

Step 2: Missing indicators

```
data MissingInd
(drop=i);
  set &mydata;
  array mi{*}
&MissingInd;
  array x{*} &inputs;
  do i=1 to dim(mi);
       mi{i}=(x{i}=.);
  end;
run;
```

Step 3: Impute missing values

```
proc stdize data=MissingInd
        reponly
        method=median
        out=imputed;
         var &inputs;
run;
```

Last observation carried forward (LOCF)

Step 1: Sample data

```
data raw;
infile datalines
delimiter=",";
input ID $  LastTry;
datalines;
1,160
1,.
1,140
2,141
2,.
;
```

Step 2: Sorting by ID

```
proc sort
  data=raw;
       by id;
run;
```

Step 3: Compute LOCF

```
data rawflat;
set raw;
by id;
retain LOCF;

* set to missing to avoid being carried
to the next ID;

if first.id then
        do;
              LOCF=.;
        end;

* update from last non missing value;
if LastTry not EQ . then
LOCF=LastTry;
run;
```

LOCF

One method of handling missing data is simply to impute, or fill in, values based on existing data.

ID	LastTry	LOCF
1	160	160
1	.	160
1	140	140
2	141	141
2	.	141

Working with character values

Some useful functions

Function	Details	Example
SUBSTR(char, position ,n)	Extracts a substring from an argument	phone = "(01573) 3114800" ; area = substr(phone, 2, 5) ;
SCAN(char, n, 'delimiters')	Returns the nth word from a string.	Name = "Dr. Ali Ajouz" ; title = scan(Name, 1, ".") ;
INDEX(source,excerpt)	Searches a character for a string	INDEX('Ali Ajouz','Ajo');
COMPRESS	Remove characters from a string	phone="(01573)- 3114800"; phone_c= compress(phone,'(-) ');

Working with character values (2)

Some useful functions

Function	Details	Example
TRANWRD	Replaces or removes given word within a string	old ="I eat Apple"; new = TRANWRD(old,"Apple","Pizza");
CATX(sep, str1,str2, .)	Removes leading and trailing blanks, inserts delimiters, and returns a concatenated character string.	First="Ali"; Last = "Ajouz"; UserName=catx(".", First, Last);
STRIP(char)	Removes leading and trailing blanks	STRIP(" aliajouz@gmail.com ");

Also UPCASE for uppercase and LOWCASE for lowercase

SAS dates and times

SAS DATE VALUE: is a value that represents the number of days between January 1, 1960, and a specified date.

December 31, 1959	-1
January 1, 1960	0
January 2, 1960	1

Raw date to SAS date

```
data Birthday;
infile datalines;
input date ddmmyy10.;
datalines;
04/11/1980
;
```

Converting a text date to SAS date USING INFORMATS

INPUT	INFORMAT
04/11/1980	mmddyy10.
04/11/80	mmddyy8.
04nov1980	date9.
nov 04, 1980	worddate12.

Refer to SAS docs for all available INFORMATS.

Working with dates, and times

Some useful functions

DATEPART(datetime)	Extracts the date from a SAS datetime value.
TIMEPART(datetime)	Extracts a time value from a SAS datetime value.
DAY(date)	Returns the day of the month from a SAS date value.
MONTH(date)	Returns the month from a SAS date value.
YEAR(date)	Returns the year from a SAS date value.
DATE / TODAY	Returns the current date

Example

```
data Me (drop=DateTime);
   format Date ddmmyy10. Time time.;

DateTime='04Nov80:12:15'dt;
   Date=datepart(DateTime);
   Time=timepart(DateTime);
   Day=day(Date);
   Month=month(Date);
   Year=year(Date);
run;
```

Obs	Date	Time	Day	Month	Year
1	04/11/1980	12:15:00	4	11	1980

Concatenating data sets rows

We can use ***SET statement*** to combine data sets ***vertically***.

```
data result;
      set data1 data2;
run;
```

data1

Obs	A	B
1	A0	B0
2	A1	B1
3	A2	B2

+

data2

Obs	A	B
1	A3	B3
2	A4	B4
3	A5	B5

=

result

Obs	A	B
1	A0	B0
2	A1	B1
3	A2	B2
4	A3	B3
5	A4	B4
6	A5	B5

Concatenating data sets columns

We can use ***Merge statement*** to combine data sets ***horizontally (One-to-One)*** .

```
data result;
      merge data1 data2;
run;
```

data1

Obs	Name	Gender
1	Ali	M
2	Karam	F
3	Roaa	F

+

data2

Obs	Salary
1	50000
2	40000
3	.

=

result

Obs	Name	Gender	Salary
1	Ali	M	50000
2	Karam	F	40000
3	Roaa	F	.

We can use *IN option* in the *MERGE statement* with *IF statement* to control the output.

Merging data sets on a common column

data1

ID	Name
1	Ali
2	Karam

data2

ID	Name
1	Ali
3	Roaa

+

```
data result;
merge data1(in=InData1)
data2(in=InData2);
  by ID;
/* code here */
```

if InData1=1 and InData2=0;

result

ID	Name
2	Karam

if InData1=0 and InData2=1;

result

ID	Name
3	Roaa

if InData1=1 and InData2=1;

result

ID	Name
1	Ali

to be continued ..

I welcome comments,
suggestions and corrections.
aliajouz@gmail.com