introduction to sas

Juan Shishido, D-Lab, UC Berkeley

March 10, 2016

introduction

what is sas

SAS is an integrated system of software solutions.

It enables:

- data management
- report generation
- plotting
- statistical and mathematical analyses
- and more

products

- Base SAS
- SAS/STAT
- SAS/ETS
- SAS Text Miner
- SAS Energy Forecasting
- and much, much more

Products & Solutions A-Z

base sas

Includes:

- a programming language
- a data management facility
- data analysis and reporting utilities

Base SAS is at the core of the SAS System

programming language

programming language

> *The SAS language contains statements, expressions, functions and CALL routines, options, formats, and informats.*

There are two main components:

- data steps
- procedure steps

SAS programs—files ending in the `.sas` file extension—typically include several DATA and PROC steps

Example of a DATA step

```
data example;
    infile 'path/to/file';
    input x1 x2 x3;
run;
```

Syntax

One of the most important rules is that **SAS statements must end with a semicolon**

SAS statements can span multiple lines

Multiple SAS statements can appear on the same line, so long as each is separated by a semicolon

A `run;` statement, which creates a "step boundary," marking the end of a step, isn't required between steps in a program, but is recommended

programming language

SAS Names

Are used for data sets, variables, and other items

In general, these names must:

- contain only letters, numbers, or underscores (_)
- begin with a letter or underscore
- have a length betwen one and 32 characters
    - maximum length varies by name type (e.g., variable names versus library references)
- not contain blanks

Names are *not* case sensitive

data management

data representation

In SAS, data is organized into rows and columns in what is called a SAS data set

x1	x2	x3
25	m	berkeley
26	f	san francisco
23	f	oakland
24	m	marin

Each row is sometimes called an “observation” and each column a “variable”

data step

DATA steps begin with the `data` statement and are typically used to create, modify, or replace SAS data sets

Data can either be read inline or from external sources, such as `.txt`, `.csv`, or `.sas7bdat` files

SAS data sets can either be temporary or permanent

Temporary data sets are stored in the `WORK` library and are deleted at the end of each SAS session

Permanent data sets are saved to disk

data step

SAS data sets are temporary, by default

In the code above, `example` is a temporary SAS data set

To read or write a *permanent* SAS data set, use dot notation such as `libref.dataset`

The `libref` is a name associated with a SAS library or directory location

It is possible to use `work.dataset` to be explicit about temporary data sets

data step

To set up a `libref` use the `libname` keyword

```
libname mylib 'path/to/dir';
```

In this example, `mylib` is a variable representing the `path/to/dir` location

Note that `libref` names can only be 8 character long and should appear before any references are made to it in your program

data step

```
data mylib.example;
    ...
run;
```

In the code above, the data set `example` will be saved to the location associated with `mylib`

data step

There are several ways to read data into a SAS data set

- `datalines`: for inline data
- `infile`: for data from an external file
- `set`: for a SAS data set

It's important to note that both the `datalines` and `infile` approaches require the use of an `input` statement, which

> *Describes the arrangement of values in the input data record and assigns input values to the corresponding SAS variables.*

data analysis

proc step

SAS procedures are built-in programs that use SAS data set values to produce specific output

These are called using PROC Steps, which begin with the `proc` keyword

There are three main types of SAS procedures:

- report writing
- statistics
- utilities

proc step

[Report writing] procedures display useful information, such as data listings (detail reports), summary reports, calendars, letters, labels, multipanel reports, and graphical reports.

proc step

[Statistics] procedures compute elementary statistical measures that include descriptive statistics based on moments, quantiles, confidence intervals, frequency counts, crosstabulations, correlations, and distribution tests. They also rank and standardize data.

proc step

[Utility] procedures perform basic utility operations. They create, edit, sort, and transpose data sets, create and restore transport data sets, create user-defined formats, and provide basic file maintenance such as to copy, append, and compare data sets.

proc step

One of the most basic procedures is PROC PRINT

```
proc print data=example;
run;
```

This prints the SAS data set `example`

proc step

PROC Steps often have several optional arguments

With PROC PRINT, for example, we can specify the number of observations (rows) as well as the variables (columns) we want printed

```
proc print data=example (obs=10);
    var x1 x2;
run;
```

running sas code

ways to run sas programs

There are several ways to execute or run SAS programs

> *They differ in the speed with which they run, the amount of computer resources that are required, and the amount of interaction that you have with the program (that is, the kinds of changes you can make while the program is running).*

The results and output—that is, the data sets and values—are the same regardless of the way the program is executed (although the *appearance* might be different)

Windowing Environment

The SAS windowing environment is a stand-alone desktop application

It includes, among other things, an editor for writing code and an output window

Entire programs or individual code blocks can be submitted

Log information and output is typically printed to their corresponding windows instead of being saved to external files

ways to run sas programs

SAS Studio

> *SAS Studio is a developmental web application for SAS that you access through your web browser.*

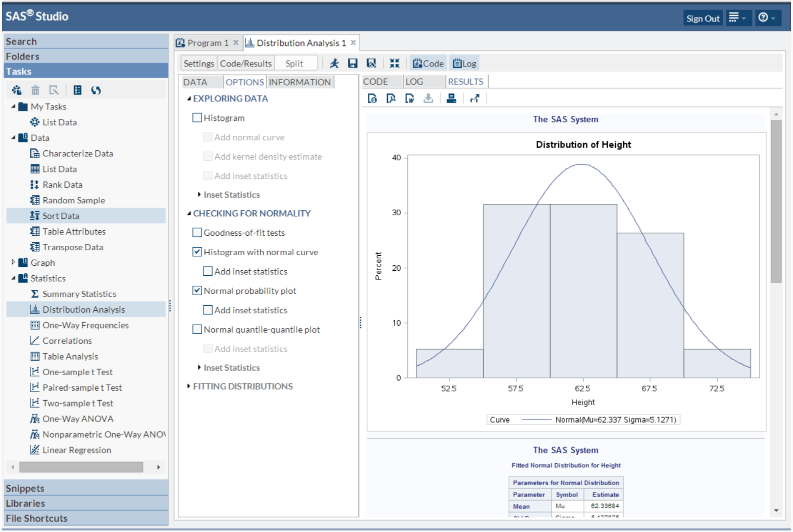
This is a web browser-based environment

> *With SAS Studio, you can access your data files, libraries, and existing programs and write new programs.*

In subsequent workshops, we'll use SAS Studio

ways to run sas programs

SAS Studio



ways to run sas programs

Noninteractive mode

With this approach, entire SAS programs are submitted

This is the only way to interact with SAS if all you have access to is a command line interface

To run a SAS program from the command line

```
$ sas filename.sas
```

The log information is saved to `filename.log` and the output, if any, to `filename.lst`

getting help

An important programming skill is knowing *how* to find help

SAS provides extensive documentation (link) with information on syntax, options, and examples

UCLA's Institute for Digital Research and Education (link) is another great resource

getting help

Often times, I rely on Google searches like: “sas average by group”

I sometimes search for specific information about a PROC

Your searches might return papers from SUGI (SAS Users Group International) or NESUG (Northeast SAS Users Group)—these are often quite good

coding

`auto`

To this point, we've described, at a high level, the two primary components of the SAS language

For the remainder of the workshop, we'll show and describe SAS code in order to get familiar with the details and work through common problems

We'll start by creating a SAS data set using inline data

Then, we'll print some summary statistics

(The code for this can be found in `code/firstprogram.sas`)

auto

```
data auto;
  input make $ price mpg rep78 foreign;
datalines;
AMC    4099 22 3 0
    ...
Datsun 8129 21 4 1
;
run;
```

In this example, we're creating a SAS data set that we're naming `auto`

We use `datalines` to let SAS know the data will provided inline

auto

```
data auto;
  input make $ price mpg rep78 foreign;
datalines;
AMC    4099 22 3 0
    ...
Datsun 8129 21 4 1
;
run;
```

Notice that semicolons (;) are *not* used at the end of each data line, only at the end of the block

auto

```
data auto;
  input make $ price mpg rep78 foreign;
datalines;
AMC    4099 22 3 0
    ...
Datsun 8129 21 4 1
;
run;
```

The the `input` statement is used to specify the variable names—in this case, there are five columns, so we list five variable names

auto

```
data auto;
  input make $ price mpg rep78 foreign;
datalines;
AMC    4099 22 3 0
    ...
Datsun 8129 21 4 1
;
run;
```

You may have noticed a $ after the `make` variable name

This lets SAS know that `make` is a character variable

Let's say we're interested in calculating the average mpg for foreign and domestic cars in our data set

We can do this using the means procedure (`proc means`)

```
proc means data=auto;
    class foreign;
    var mpg;
run;
```

average mpg

```
proc means data=auto;
    class foreign;
    var mpg;
run;
```

Here, we specify the input data (`data=auto`), the variable we want the means for (`var mpg`), and the “by” group (`class foreign`)

a note on output

If we wanted to run this code in batch mode, we would do the following

```
$ sas firstprogram.sas
```

If things go well, you won't see any output when you submit this program

So, where does the output go?

Whenever programs (or individual SAS code blocks) are run, SAS always produces a log file (with file extension `.log`)

a note on output

This gives information about the steps that were executed, how long they took, and messages related to any particular errors

In addition, if there are things that are printed (a lot of `PROC`s produce this type of output), a listing file will be created (with file extension `.lst`)

comments

SAS provides two ways to add comments

```
*message;
```

```
/*message*/
```

SAS ignores comments during processing

loading

Typically, `datalines` won't be used in practice

Instead, we load data from external sources

These can be comma-separated value files, for example, or SAS data sets

When loading data, we must use either `infile` or `set` statements, depending on the data source

loading

In `data/` there is a small CSV file named `mtcars.csv`

To load this data into SAS, use the following (a `.sas` file can also be found in `code/`)

```
data cars;
    infile '../data/mtcars.csv' dlm=',' dsd firstobs=2;
    input model : $19. mpg cyl disp hp
          drat wt qsec vs am gear carb;
run;
```

This `infile` statement includes several options we have not seen before

loading

Perhaps the most important is the `dlm` option, which specifies the delimiter that separates the variables in the file

If data contains missing values, as `mtcars.csv` does, the `dsd` options allows SAS to recognize two consecutive delimiters as such

`dsd` also allows the data to include the delimiter within quoted strings

Finally, the `firstobs` option allows us to specify the line at which SAS should start reading data from

Because `mtcars.csv` includes a “header” with variable names, we start at line 2

loading

We have previously seen that $ lets SAS know the associated column should be read in as character

By default, SAS only reads the first 8 characters, but we can specify a length

The drawback is that we have to know the maximum length, which is 19 in this case

The colon modifier (`:`) is also important here as it tells SAS to read the record until there it encounters the delimiter

The `.` in `$19.` is also necessary

loading

```
proc import datafile='../data/mtcars.csv'
    out=cars_imported
    replace;
    getnames=yes;
run;
```

Alternatively, we can use SAS's import procedure to load the data

With `proc import`, we specify the input data using `datafile`

Because SAS recognizes that `.csv` files are comma-separated, we don't have to be explicit about the delimiter

`proc import` requires that we provide an output data set using `out`

loading

```
proc import datafile='../data/mtcars.csv'
    out=cars_imported
    replace;
    getnames=yes;
run;
```

The `replace` option is used to overwrite an existing SAS data set

Use the `getnames` option to specify whether variable names should be generated from the first record in the input file

Because the first record for our input file has a missing value in the first position, SAS uses the default `VAR1` name for that column

This is what the `cars` data set looks like (from the .lst' file)

The SAS System

Obs	model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
1	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
2	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	.	0	1	4	4
3	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
4	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
5	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
6	Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
7	Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
8	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
9	Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
10	Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
11	Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
12	Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
13	Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
14	Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
15	Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
16	Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
17	Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
18	Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
19	Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
20	Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
21	Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
22	Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
23	AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
24	Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
25	Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
26	Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
27	Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
28	Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
29	Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
30	Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
31	Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
32	Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

contents

`proc contents` describes a SAS data set

```
proc contents data=cars varnum;
run;
```

This outputs information at both the data set and variable levels

For example, it returns the number of observations, the number of variables, the variable names and formats, and much more

The `varnum` option lists the variables in “creation order”

contents

proc contents output

```
                                   The CONTENTS Procedure

Data Set Name        WORK.CARS                                          Observations          32
Member Type          DATA                                               Variables             15
Engine               V9                                                 Indexes               0
Created              02/17/2016 21:41:54                                Observation Length    144
Last Modified        02/17/2016 21:41:54                                Deleted Observations  0
Protection                                                              Compressed            NO
Data Set Type                                                           Sorted                NO
Label
Data Representation  SOLARIS_X86_64, LINUX_X86_64, ALPHA_TRU64, LINUX_IA64
Encoding             latin1  Western (ISO)


                             Engine/Host Dependent Information

             Data Set Page Size          65536
             Number of Data Set Pages    1
             First Data Page             1
             Max Obs per Page            454
             Obs in First Data Page      32
             Number of Data Set Repairs  0
             Filename                    /tmp/SAS_workD57000001497_ln002.brc/cars.sas7bdat
             Release Created             9.0401M2
             Host Created                Linux
             Inode Number                66264
             Access Permission           rw-r--r--
             Owner Name                  shishido
             File Size (bytes)           131072


                              Variables in Creation Order

                          #    Variable        Type    Len

                          1    model           Char     19
                          2    mpg             Num       8
                          3    cyl             Num       8
                          4    disp            Num       8
                          5    hp              Num       8
                          6    drat            Num       8
                          7    wt              Num       8
                          8    qsec            Num       8
                          9    vs              Num       8
                         10    am              Num       8
                         11    gear            Num       8
                         12    carb            Num       8
                         13    liters          Num       8
                         14    hp_per_liter    Num       8
                         15    make            Char     19
```

creating variables

Next, let's say we're interested in converting our displacement variable (`disp`) from cubic inches to liters, using the conversion rate found here

In SAS, we can simply add the following to our existing DATA step:

```
liters = disp / 61.024;
```

We might also be interested the vehicles' power density values

creating variables

```
data cars;
    infile '../data/mtcars.csv' dlm=',' dsd firstobs=2;
    input model : $19. mpg cyl disp hp
          drat wt qsec vs am gear carb;
    liters = disp / 61.024;
    hp_per_liter = hp / liters;
run;
```

Because we're curious about which cars are the most power dense, we should sort our existing SAS data set by `hp_per_liter`, in descending order

sorting

```
proc sort data=cars out=power_density
        (keep=model hp liters hp_per_liter);
    by descending hp_per_liter;
run;
```

The `out` option tells SAS that the procedure should create a new data set called `power_density`

To this, we add a `keep` option, only keeping the `model`, `hp`, `liters`, and `hp_per_liter` variables; without this option, all variables would be written to `power_density`

sorting

```
proc sort data=cars out=power_density
        (keep=model hp liters hp_per_liter);
    by descending hp_per_liter;
run;
```

The `by` statement tells the procedure what variable(s) we'd like to sort by

By default, sorting occurs in ascending order

To sort in descending order, add the `descending` keyword *before* the variable name

sorting

The Ferrari and Lotus models are the most power dense

The SAS System

Obs	model	hp	liters	hp_per_ liter
1	Ferrari Dino	175	2.37611	73.6497
2	Lotus Europa	113	1.55840	72.5101
3	Maserati Bora	335	4.93249	67.9171
4	Toyota Corolla	65	1.16512	55.7885
5	Volvo 142E	109	1.98283	54.9720
6	Datsun 710	93	1.76980	52.5484
7	Fiat 128	66	1.28966	51.1764
8	Fiat X1-9	66	1.29457	50.9821
9	Toyota Corona	97	1.96808	49.2867
10	Porsche 914-2	91	1.97136	46.1611
11	Ford Pantera L	264	5.75184	45.8984
12	Merc 280	123	2.74646	44.7849
13	Merc 280C	123	2.74646	44.7849
14	Camaro Z28	245	5.73545	42.7168
15	Mazda RX4	110	2.62192	41.9540
16	Mazda RX4 Wag	110	2.62192	41.9540
17	Honda Civic	52	1.24050	41.9187
18	Duster 360	245	5.89932	41.5302
19	Merc 230	95	2.30729	41.1739
20	Merc 450SE	180	4.51953	39.8271
21	Merc 450SL	180	4.51953	39.8271
22	Merc 450SLC	180	4.51953	39.8271
23	Chrysler Imperial	230	7.21028	31.8989
24	AMC Javelin	150	4.98165	30.1105
25	Hornet Sportabout	175	5.89932	29.6644
26	Dodge Challenger	150	5.21106	28.7849
27	Lincoln Continental	215	7.53802	28.5221
28	Valiant	105	3.68707	28.4779
29	Pontiac Firebird	175	6.55480	26.6980
30	Cadillac Fleetwood	205	7.73466	26.5041
31	Hornet 4 Drive	110	4.22784	26.0180
32	Merc 240D	62	2.40397	25.7906

string manipulation

It might be useful to extract the make from the `model` variable

To do this, we can use the `scan()` function to our existing DATA step

```
make = scan(model, 1, ' ');
```

`scan()` takes a string as its first argument—in this case, `model`—a position, and a delimiter

The string is split on the delimiter—a single space

It then returns the first word

In some cases, it might be useful to summarize our data by taking column sums

We might be interested in knowing the total horsepower for the vehicles in our data set, for example

There are several ways to do this

One way is to use `proc print`

```
proc print data=cars;
    var hp;
    sum hp;
run;
```

This prints all of the `hp` observations in `cars`, but adds a total at the bottom of the output

In some cases, this isn't desirable

We might, instead, want to only output the total

sums

`proc summary` is one of the most powerful procedures to summarize numeric variables and place aggregated results into a new SAS data set.

```
proc summary data=cars;
    var hp;
    output out=cars_summ (drop=_TYPE_)
        sum=hp_total mean=hp_mean;
run;
```

`proc summary` requires that we specify an output data set using `output out`

This procedure is flexible, allowing us to calculate sums, means, and much more

sums

```
proc summary data=cars;
    var hp;
    output out=cars_summ (drop=_TYPE_)
        sum=hp_total mean=hp_mean;
run;
```

The `drop` option next to the new data set name, `cars_summ`, tells SAS to not include the listed variables

For the specified operations—that is, `sum` and `mean`—we list corresponding variable names *`proc sql` can sort, summarize, subset, join (merge), and concatenate datasets, create new variables, and print the results or create a new table or view all in one step.*

```
proc sql;
    create table cars_sql as
    select count(*) as _FREQ_,
           sum(hp) as hp_total,
           mean(hp) as hp_mean
    from cars;
quit;
```

Note that `proc sql` steps end with `quit;` rather than `run;`

Using SQL syntax, we can recreate the results from the `proc summary` step

sums

Obs	_FREQ_	hp_total	hp_mean
1	32	4694	146.688

loops

Loops—referred to as `do` loops in SAS—enable iteration

> *[They execute] statements between the `do` and `end` statements repetitively, based on the value of an index variable.*

loops

```
data squares;
    do x = 2 to 10 by 2;
        x_squared = x ** 2;
        output;
    end;
run;
```

This DATA step loops over the values 2 and 10 and outputs five records—the values 2, 4, . . . , 10 as well as their squares

The `output` statement "writes the current observation to data set"

loops

```
data squares;
    do x = 2 to 10 by 2;
        x_squared = x ** 2;
        output;
    end;
run;
```

The default increment in a `do` loop is 1

However, we can use the `by` keyword to increment by any positive or negative number

loops

The resulting data set looks like this

Obs	x	x_squared
1	2	4
2	4	16
3	6	36
4	8	64
5	10	100

if-then/else

SAS supports control flow with the `if` and `else` statements

```
if expression then statement;
<else statement>;
```

If the condition(s) in the `if` clause are met, the statement is executed

Otherwise, the next line is evaluated

The optional `else` statement, which must immediately follow the IF-THEN statement, is executed if the `then` clause before it isn't

if-then/else

Using our `cars` data set, let's conditionally output observations

```
data toyota mazda;
    set cars;
    if make = 'Toyota' then output toyota;
    else if make = 'Mazda' then output mazda;
run;
```

Here, we use `set` because `cars` is an existing SAS data set

We've already seen how to create permanent SAS data sets

Sometimes, we want to export data sets in other formats

```
proc export data=power_density
    outfile='../data/power_density.csv'
    replace;
run;
```

This creates a `.csv` file in `data/` based on our `power_density` data set

Let's say we wanted to get a count of the number of vehicles in each cylinder class

That is, the number of 4-, 6-, and 8-cylinder cars in our data set

For this, we could use the frequency procedure

```
proc freq data=cars;
    table cyl;
run;
```

analysis

This produces the following output

```
                The FREQ Procedure

                                   Cumulative    Cumulative
cyl   Frequency     Percent         Frequency      Percent
-------------------------------------------------------
  4          11       34.38               11        34.38
  6           7       21.88               18        56.25
  8          14       43.75               32       100.00
```

analysis

There are a relatively similar number of 4- and 8-cylinder cars in our data set

A question we might want to answer is whether the horsepower differs between these groups

For this, we can use SAS's t-test procedure

analysis

We first create a data set with only the 4- and 8-cylinder vehicles and then sort the data

For the two-sample t-test

```
proc ttest data=cars_4_8;
    class cyl;
    var hp;
run;
```

analysis

The output for the t-test procedure

p-value < 0.0001

The TTEST Procedure

Variable: hp

cyl	N	Mean	Std Dev	Std Err	Minimum	Maximum
4	11	82.6364	20.9345	6.3120	52.0000	113.0
8	14	209.2	50.9769	13.6241	150.0	335.0
Diff (1-2)		-126.6	40.7350	16.4126		

cyl	Method	Mean	95% CL Mean		Std Dev	95% CL Std Dev	
4		82.6364	68.5724	96.7004	20.9345	14.6273	36.7387
8		209.2	179.8	238.6	50.9769	36.9559	82.1259
Diff (1-2)	Pooled	-126.6	-160.5	-92.6258	40.7350	31.6598	57.1415
Diff (1-2)	Satterthwaite	-126.6	-158.1	-95.0440			

Method	Variances	DF	t Value	Pr > |t|
Pooled	Equal	23	-7.71	<.0001
Satterthwaite	Unequal	18.096	-8.43	<.0001

Equality of Variances

Method	Num DF	Den DF	F Value	Pr > F
Folded F	13	10	5.93	0.0079

references

links

- An Introduction to the SAS System
- Reading Data into SAS
- SAS DATA Steps
- Base SAS 9.2 Procedures Guide
- `infile`
- `sum` function
- `proc sql`
- IF-THEN/ELSE
- Loops in SAS
- `do` loops
- Conditionally Writing Observations
- `proc import`