# Note on homework for SAS date formats

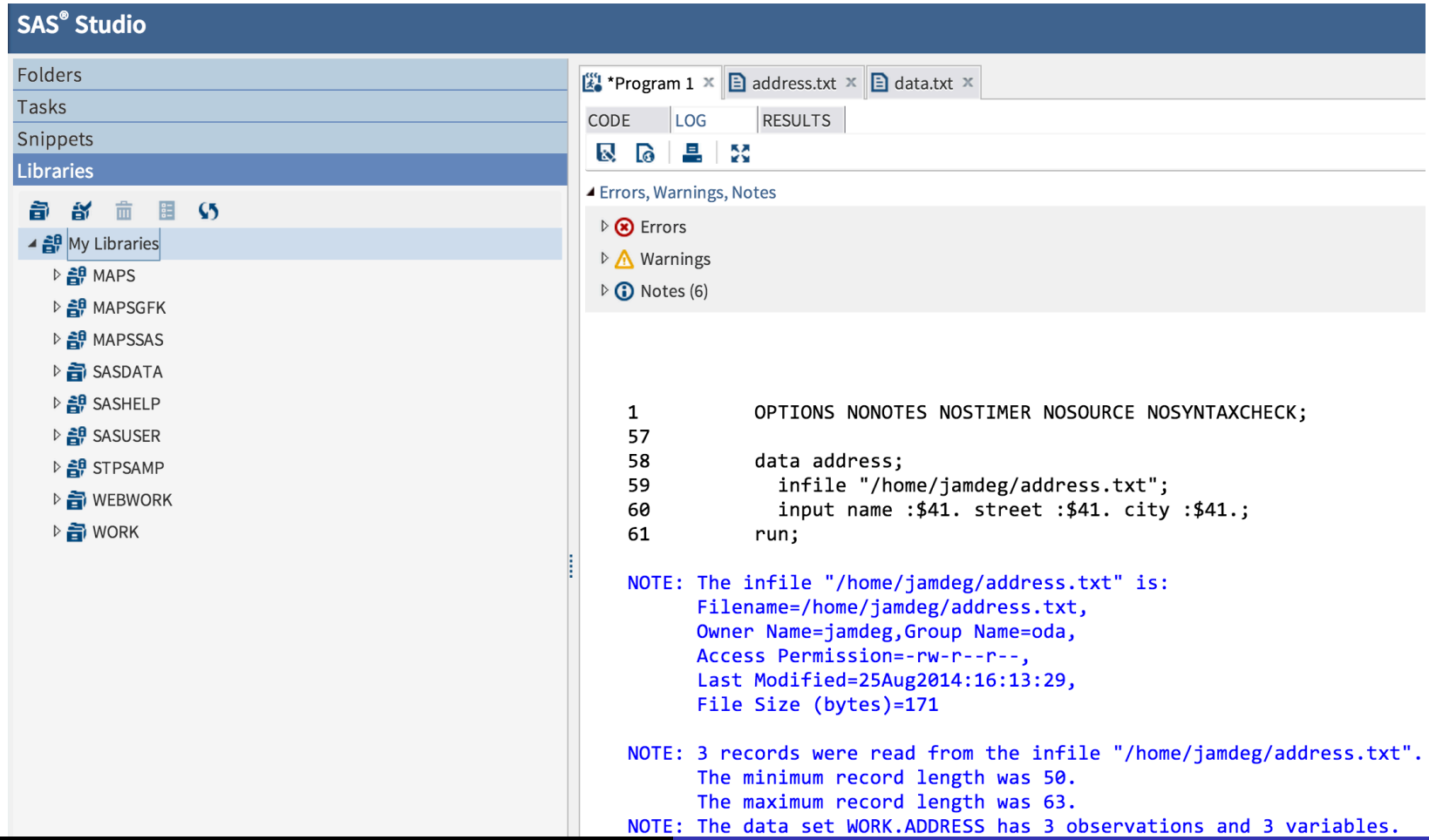
I'm getting error messages using the format `MMDDYY10D.` even though this is listed on websites for SAS date formats. Instead, `MMDDYY10` and similar (without the `D` seems to work for both hyphens and slashes.

Also note that a date format such as `MMDDYY`**`w`**`.` means that the **w** is replaced by a number indicating the width of the string (e.g., 8 or 10).

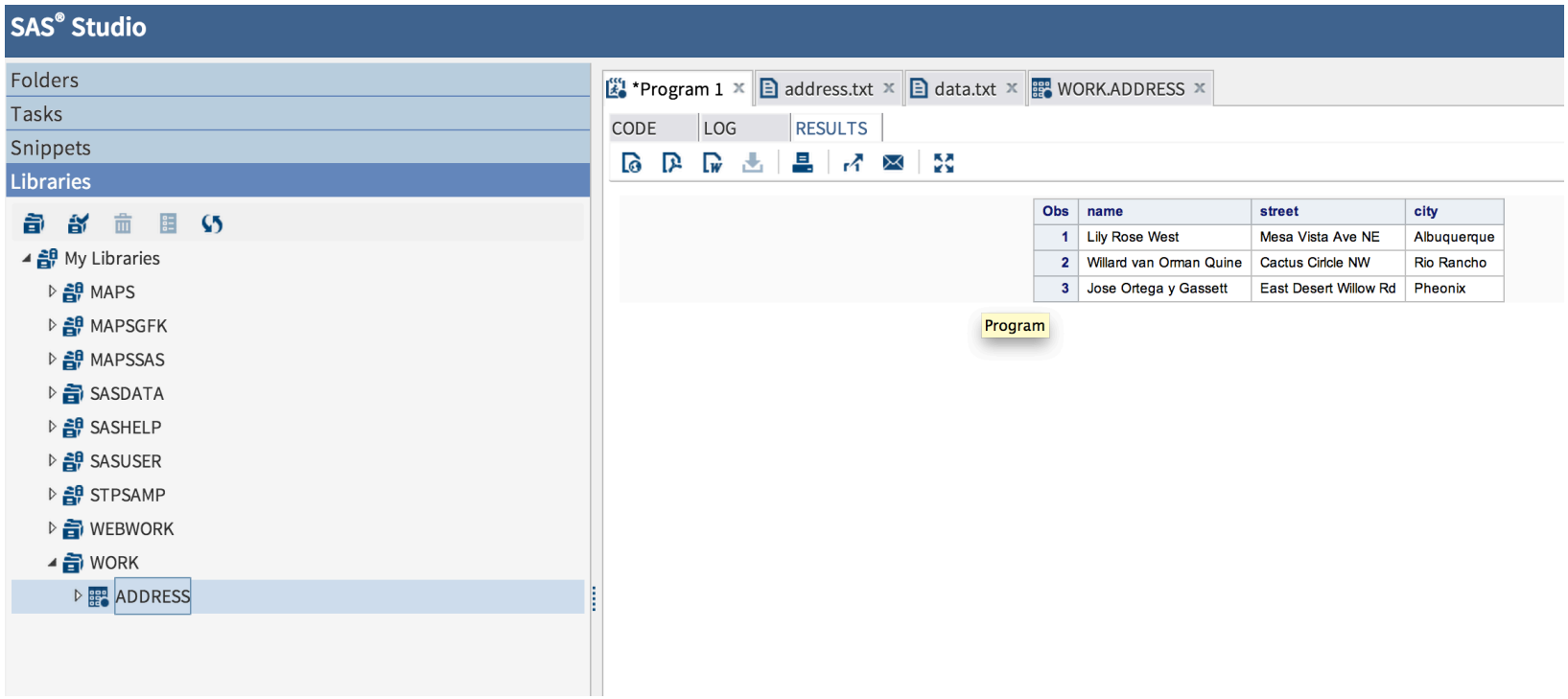
# SAS data sets (Chapter 4 of Cody book)

SAS creates data sets internally once they are read in from a Data Step. The data sets can be stored in different locations and accessed later on.
The default is to store them in WORK, so if you create a data set using
`data adress;` the logfile will say that it created a SAS dataset called WORK.ADDRESS.
You can nagivate to the newly created SAS dataset. In SAS Studio, go to the Libraries Tab on the left (Usually appears toward the bottom until you click on it). Then WORK.ADDRESS should appear.

# SAS data sets

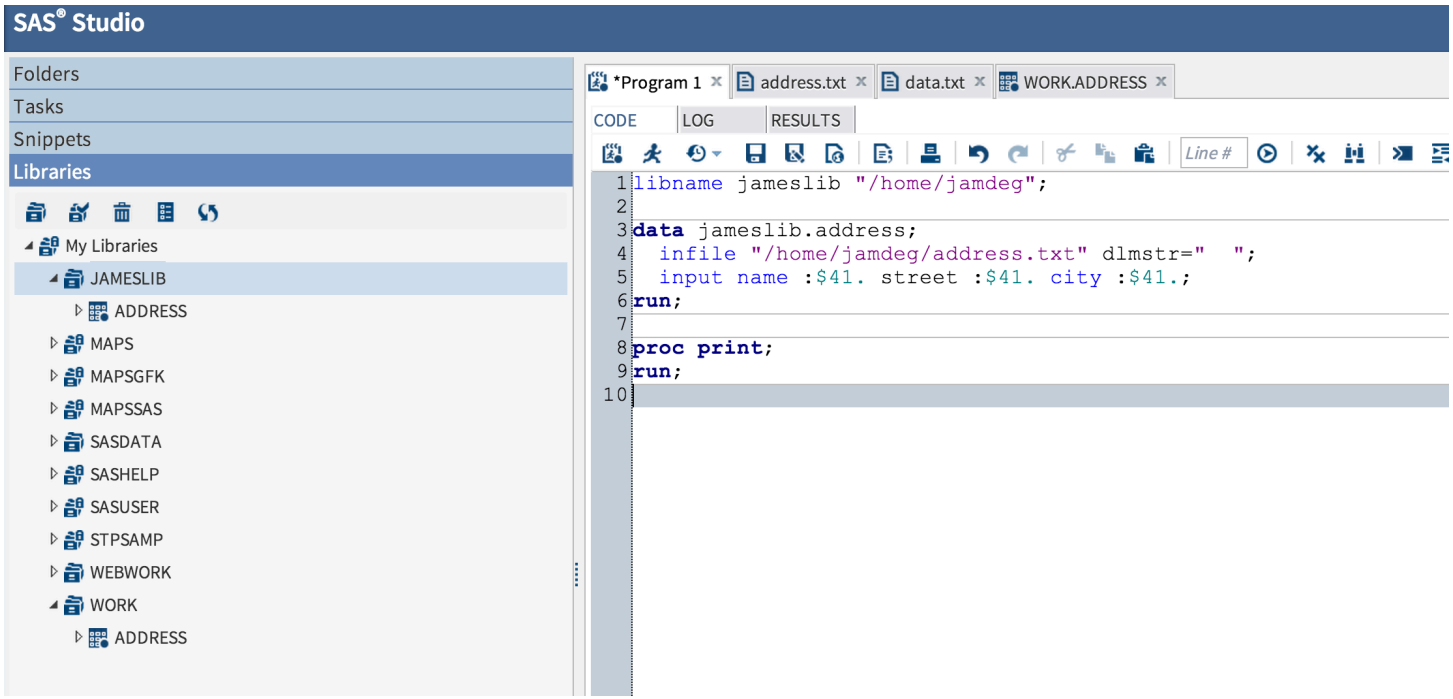


# SAS data sets



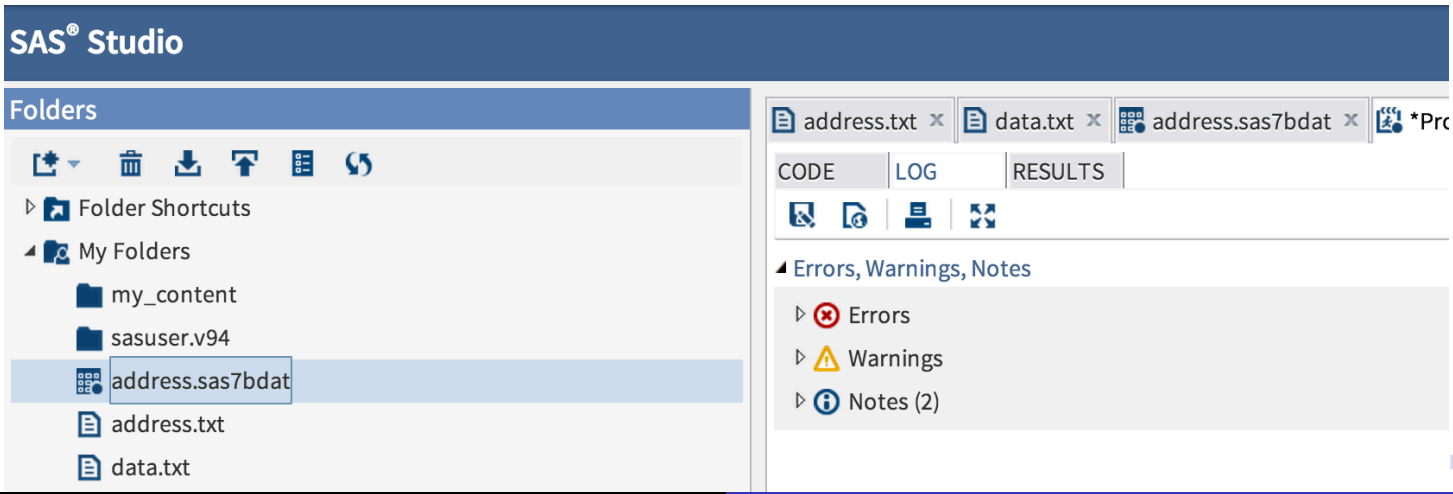
# Making datasets permanent

You can also make SAS datasets permanent. This is done using the `libname` statement. E.g.



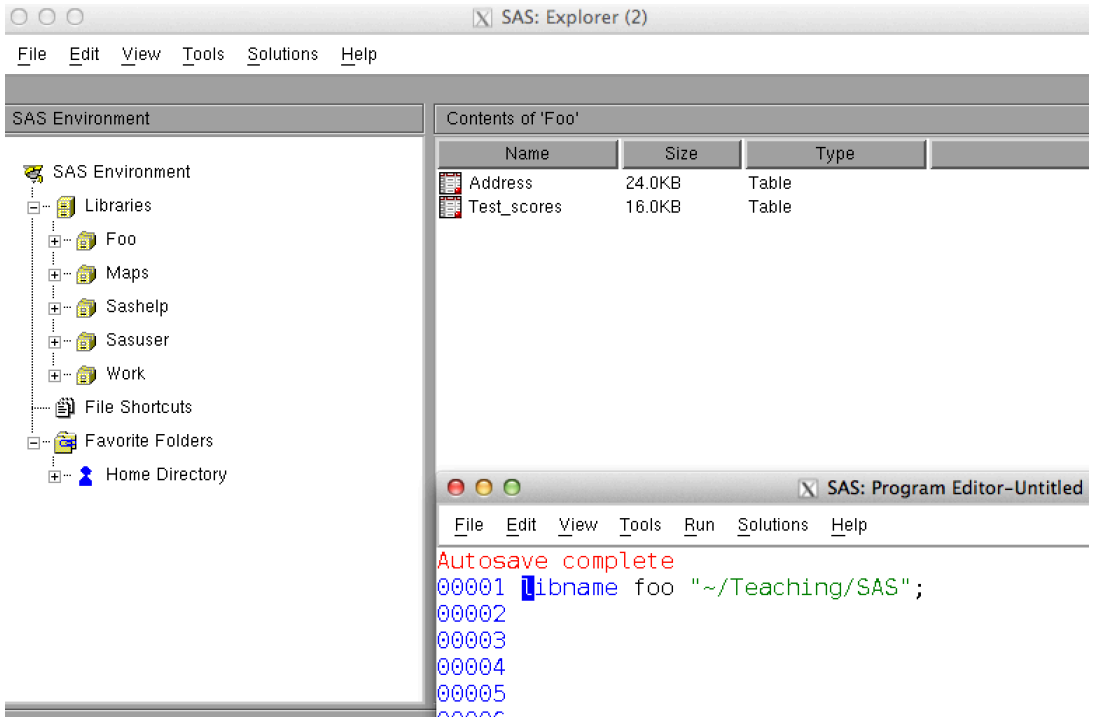
# Permanent SAS datasets

The new dataset should be available to be accessed directly from other SAS programs without reading in original data. This can save a lot of time for large datasets.
If the SAS dataset is called `mydata`, the SAS dataset will be called `mydata.sas7bdat`, where the 7 refers to the datastructures used in version 7 (and which hasn't changed up to version 9).



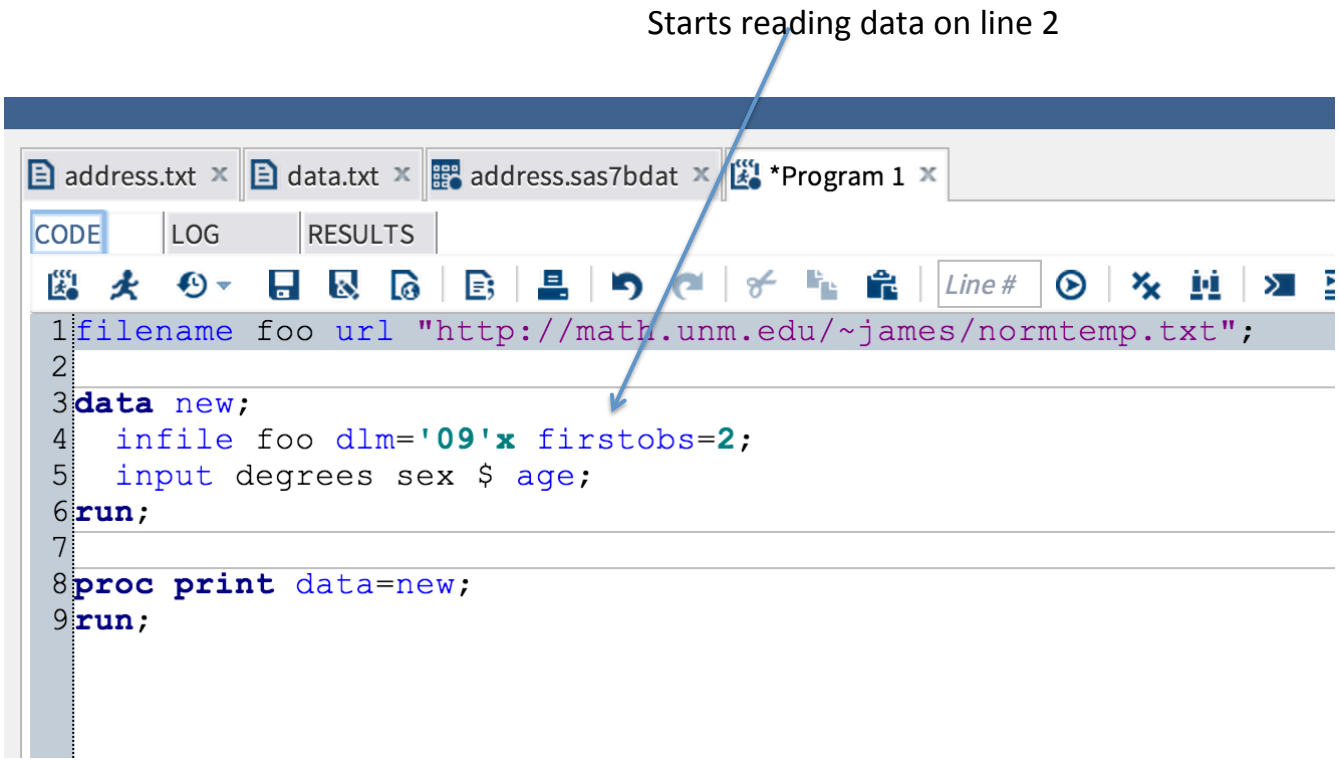
# Permanent SAS datasets

Something confusing about this is to see the data set under "My libraries", you have to run the libname statement again to create a reference to the directory where the dataset is. However, the dataset exists in the directory where it was saved as a binary file, whether or not SAS is opened.

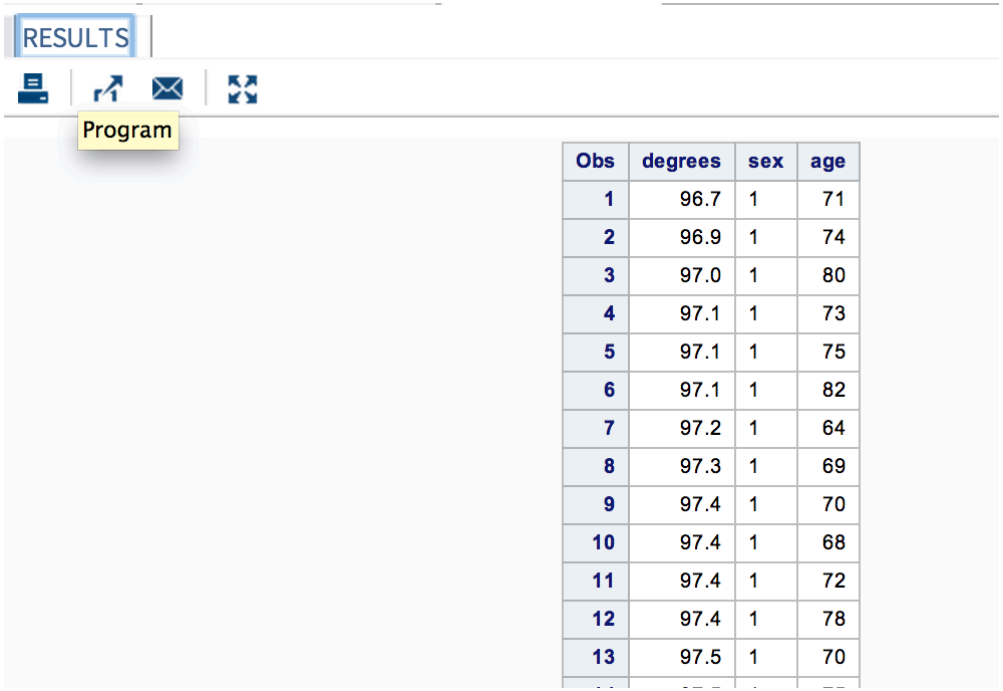


# Using a fileref

In addition to naming libraries, you can name individual files external to SAS. This is especially useful for downloading data directly from the web.



# Temperature data




| Obs | degrees | sex | age |
|---|---|---|---|
| 1 | 96.7 | 1 | 71 |
| 2 | 96.9 | 1 | 74 |
| 3 | 97.0 | 1 | 80 |
| 4 | 97.1 | 1 | 73 |
| 5 | 97.1 | 1 | 75 |
| 6 | 97.1 | 1 | 82 |
| 7 | 97.2 | 1 | 64 |
| 8 | 97.3 | 1 | 69 |
| 9 | 97.4 | 1 | 70 |
| 10 | 97.4 | 1 | 68 |
| 11 | 97.4 | 1 | 72 |
| 12 | 97.4 | 1 | 78 |
| 13 | 97.5 | 1 | 70 |

# Examining contents of datasets from other sessions

To make sure that your SAS dataset has what you think it has, it can be useful to examine the contents of the SAS dataset using PROC CONTENTS. This is especially helpful for datasets that were created in other SAS programs, and can be used to list contents of multiple datasets simultaneously.

# PROC CONTENTS example

SAS: Output-Untitled

File Edit View Tools Solutions Help

```
                          The SAS System          10:59 Tuesday, A

                        The CONTENTS Procedure

                     Engine/Host Dependent Information

Number of Data Set Repairs  0
Filename                    /nfs/user/j/jamdeg/Teaching/SAS/address.sa
Release Created             9.0202M3
Host Created                Linux
Inode Number                69559268
Access Permission           rw-------
Owner Name                  jamdeg
File Size (bytes)           24576


                  Alphabetic List of Variables and Attributes

                    #    Variable    Type    Len

                    2    address     Char     41
                    3    city        Char     41
                    1    name        Char     41
```

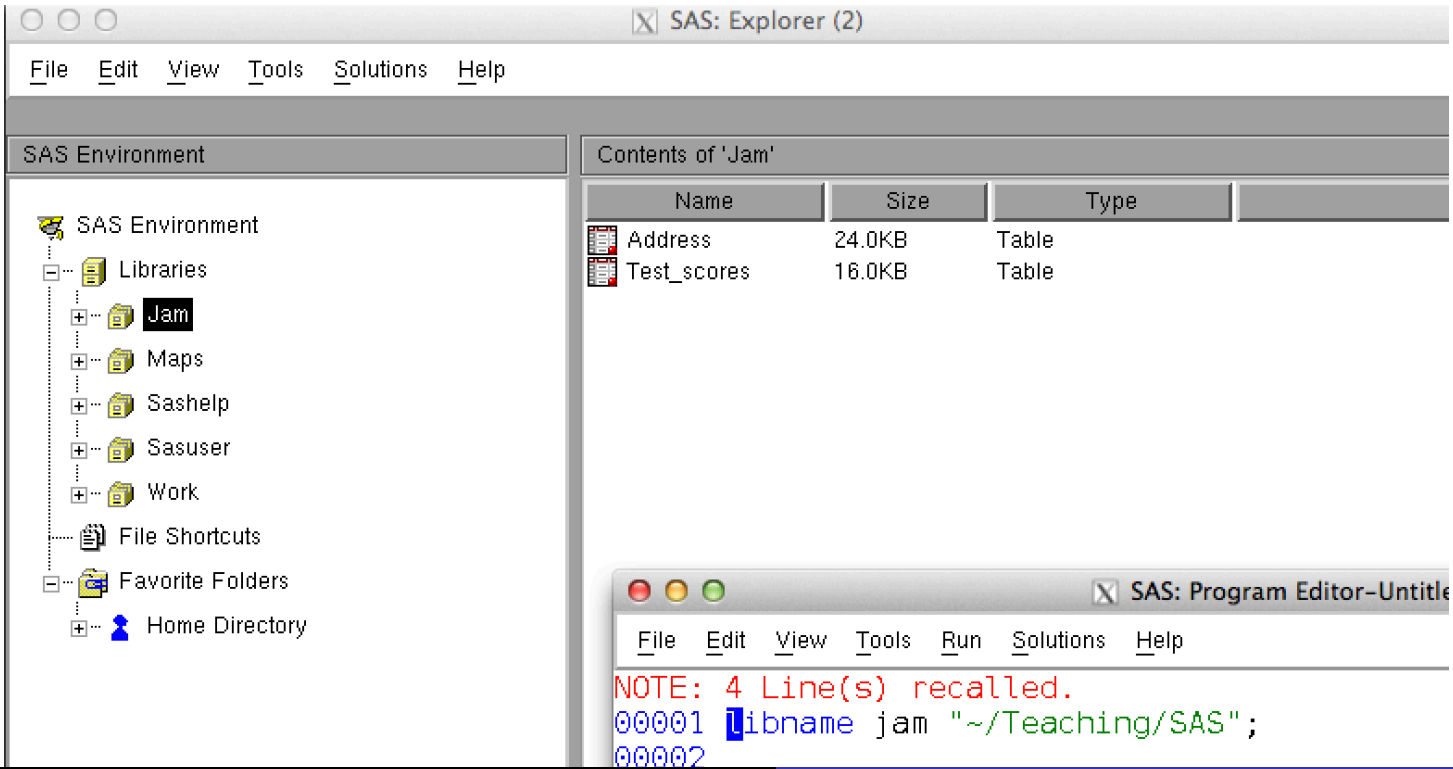
SAS: Program Editor-Untitled

File Edit View Tools Run Solutions Help

```
NOTE: 4 Line(s) recalled.
00001 libname jam "~/Teaching/SAS";
00002
00003 proc contents data=jam.Address;
00004 run;
00005
00006
```

# PROC CONTENTS on all datasets in a directory

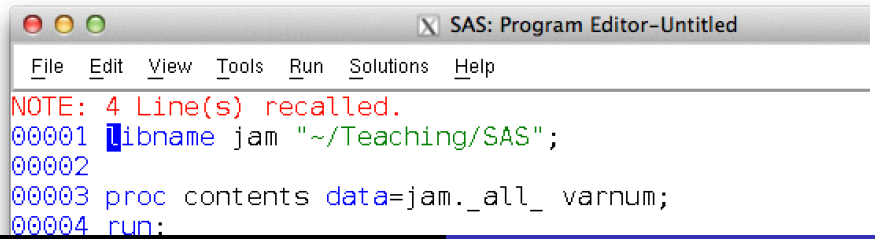
Here I listed the contents of all datasets in the directory with the `jam` libref, which is my directory for this class. The output just concatenates PROC CONTENTS runs from each SAS dataset. Note that in this program, I have run a full SAS program without using any datastep, which is unusual.



# PROC CONTENTS

PROC CONTENTS alphabetizes the names of the variables. You can have it list the variables in the order that they occur by using the option `varnum`

```
                                The SAS System          10:59

                            The CONTENTS Procedure

                        Engine/Host Dependent Information

Number of Data Set Repairs  0
Filename                    /nfs/user/j/jamdeg/Teaching/SAS/
Release Created             9.0202M3
Host Created                Linux
Inode Number                69559268
Access Permission           rw-------
Owner Name                  jamdeg
File Size (bytes)           24576


                          Variables in Creation Order

                      #    Variable    Type    Len

                      1    name        Char     41
                      2    address     Char     41
                      3    city        Char     41
```



# Tips on Linux SAS

If your connection with SAS OnDemand isn't very good, you might prefer using linux SAS. Instead of using SAS graphically, you can use it batch mode, just like in the good old days (like how I did for the Data Analysis class in the late 90s).

In some ways, this can be very fast, and is a good alternative if remote access is a problem. Instead of opening a SAS session, you have your SAS code in a separate file, say `mycode.sas`. Then at the linux prompt, you type

```
sas mycode.sas
```

# Tips on Linux SAS

If all goes well, this should produce new files: `mysas.log` and `mysas.lst`. The `mysas.log` file should be very similar to what you would see in the log window in a graphical SAS sassion from running the code. The `.lst` file lists the output assuming there are no errors.

Some things that I like about this approach are that the log and output files are written from scratch, so you don't get confused by old output. You can also very quickly identify if you have any errors in the log file using a little bit of linux:

```
cat mysas.lst | grep ERROR
```

The vertical line is called a "pipe" and means that the output from the first part is the input for the second part. This will return the lines from the `log` file that have errors if there are any. If there are no errors, it will just be silent and give you a new prompt. This can be faster than scrolling through a long logfile to find the errors.

# SAS batch processing in Linux

Another nice way to use batch processing is to apply the same SAS code to different files. Suppose you have data in 10 different states. Suppose the files are called NM.txt, AZ.txt, TX.txt, etc. You way to apply exactly the same SAS code to these 10 files. This can all be done within SAS in one session, but it could also be done separately using batch processing. To do this, try this code in Linux. Assume that your SAS code inputs a file called temp.txt.

```
cp -f NM.txt temp.txt
sas temp.txt
cp temp.lst NM.lst
cp -f AZ.txt temp.txt
sas temp.txt
cp temp.lst AZ.lst
cp -f TX.txt temp.txt
sas temp.txt
cp temp.lst TX.lst
```

# SAS batch processing in Linux

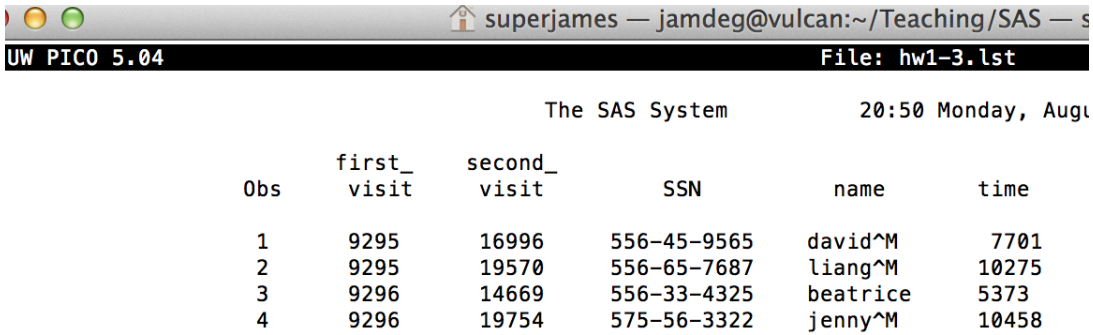
Now your SAS output is in the files ending with `.lst`, and you've run three SAS "jobs" without having to launch SAS.

This can be espcially useful in supercomputing environments, where you can send different jobs to different machines and have them run in parallel. If you are doing difficult statistics on each file or each file is very big, then instead of taking 10 hours to run all 10 data sets, you might get all of them running in parallel and get them all done in 1 hour because you've run them on multiple machines. Of course, B71 has 10 machines with SAS, so you could also log into all 10 and run parallel SAS sessions that way assuming no one else is using them...

With genetic data, I've had 10,000 jobs (one for each gene), each running MCMC and taking several minutes per job, that I wanted to run in parallel.

# Tips on Linux SAS: cleaning up carriage returns

When SAS creates output files, it puts in Window's-style carriage returns, whereas Linux/Unix/OS X use newline carriage returns only. This usually shows up in output as Ctrl-M.



# Tips on Linux SAS: cleaning up carriage returns

Carriage returns can kind of screw up the output a little bit. To clean up the carriage returns, you can type the following

```
sed -i $'s/\r//' hw1-3.lst
```

This is very obscure syntax, and comes from the UNIX/LINUX utility `sed` which does some text processing. It can do things like replace all characters or character strings in a file in an automated way, without opening the file. The usual syntax is something like

```
sed 's/oldstring/newstring/g' file
```

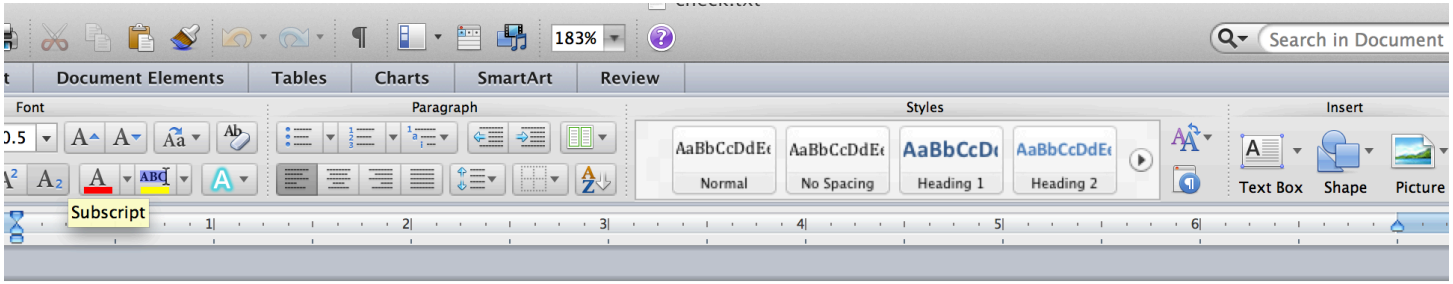
Here `\r` stands for carriage return, and because we are replacing it with nothing, we get two forward slashes in a row. There are probably other ways to do this, but this is the easiest way I know of.

# Cleaning up carriage returns

Note: why do I have integers for `first_visit` and `second_visit`?

```
[jamdeg@vulcan SAS]$ sas hw1-3.sas

Starting SAS ....

The http://Fastinfo.unm.edu  entry for SAS is:

    http://unm.custhelp.com/cgi-bin/unm.cfg/php/enduser/std_adp.php?p_faqid=6341

[jamdeg@vulcan SAS]$ cat hw1-3.lst
                                        The SAS System          17:13 Monday, August 25, 2014   1

                        first_    second_
                  Obs    visit     visit         SSN          name       time

       7701         1      9295      16996     556-45-9565    david
      10275         2      9295      19570     556-65-7687    liang
                    3      9296      14669     556-33-4325    beatrice     5373
      10458         4      9296      19754     575-56-3322    jenny
[jamdeg@vulcan SAS]$ sed -i $'s/\r//' hw1-3.lst
[jamdeg@vulcan SAS]$ cat hw1-3.lst
                                        The SAS System          17:13 Monday, August 25, 2014   1

                        first_    second_
                  Obs    visit     visit         SSN          name       time

                    1      9295      16996     556-45-9565    david       7701
                    2      9295      19570     556-65-7687    liang      10275
                    3      9296      14669     556-33-4325    beatrice     5373
                    4      9296      19754     575-56-3322    jenny      10458
[jamdeg@vulcan SAS]$
```

# How it looks in MS-Word: fine




The SAS System 20:50 Monday, August 25, 2014 1

| Obs | first_ visit | second_ visit | SSN | name | time |
|---|---|---|---|---|---|
| 1 | 9295 | 16996 | 556-45-9565 | david | 7701 |
| 2 | 9295 | 19570 | 556-65-7687 | liang | 10275 |
| 3 | 9296 | 14669 | 556-33-4325 | beatrice | 5373 |
| 4 | 9296 | 19754 | 575-56-3322 | jenny | 10458 |

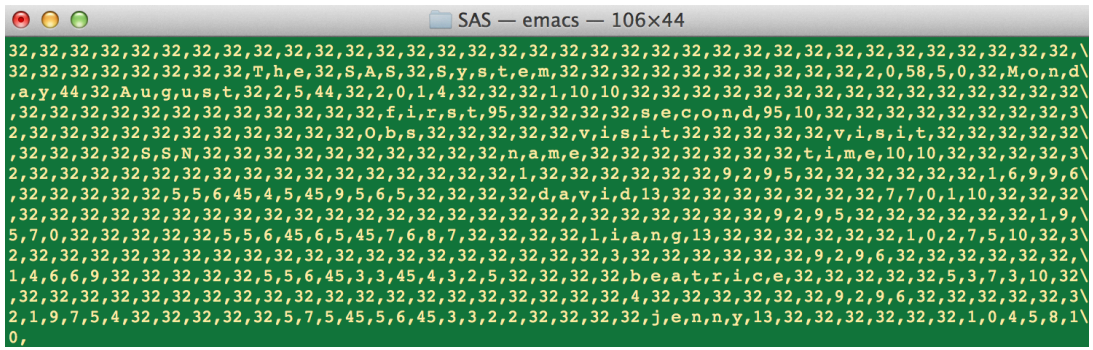
# How it looks in Mac Pages: similar to linux

The SAS System 20:50 Monday, August 25, 2014 1

| Obs | first_ visit | second_ visit | SSN | name | time |
|---|---|---|---|---|---|
| 1 | 9295 | 16996 | 556-45-9565 | david | 7701 |
| 2 | 9295 | 19570 | 556-65-7687 | liang | 10275 |
| 3 | 9296 | 14669 | 556-33-4325 | beatrice | 5373 |
| 4 | 9296 | 19754 | 575-56-3322 | jenny | 10458 |

# ASCII characters of the output

10 indicates Line Feed \n while 13 represents carriage return, \r. WORD doesn't start a new line until the Line Feed, while UNIX/LINUX do a new line at the carriage return. Look up the entry for Newline in Wikipedia for some of the history of the difference and lists of operating systems with different ways to do Newlines characters. These issues can also cause incompatibilities when programs do or do not expect a file to end with a newline character. It usually safest to include a newline at the end of a file.



# ASCII table

ASCII tables are easy to search for online in case something funky is happening and you need to know what the invisible characters really are. Notice where the TAB is in hex code (recall `'09'x` is the delimiter name in SAS). Extended ASCII goes up to number 255 (in decimal). This where there are 8 bytes to code one character of data. 8 bytes $= 2^8 = 256$ bits, so numbers between 0 and 255 can stored using 8 bytes.

| Dec | Hx | Oct | Char | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 000 | NUL (null) | 32 | 20 | 040 |   | Space | 64 | 40 | 100 | @ | @ | 96 | 60 | 140 | ` | ` |
| 1 | 1 | 001 | SOH (start of heading) | 33 | 21 | 041 | ! | ! | 65 | 41 | 101 | A | A | 97 | 61 | 141 | a | a |
| 2 | 2 | 002 | STX (start of text) | 34 | 22 | 042 | " | " | 66 | 42 | 102 | B | B | 98 | 62 | 142 | b | b |
| 3 | 3 | 003 | ETX (end of text) | 35 | 23 | 043 | # | # | 67 | 43 | 103 | C | C | 99 | 63 | 143 | c | c |
| 4 | 4 | 004 | EOT (end of transmission) | 36 | 24 | 044 | $ | $ | 68 | 44 | 104 | D | D | 100 | 64 | 144 | d | d |
| 5 | 5 | 005 | ENQ (enquiry) | 37 | 25 | 045 | % | % | 69 | 45 | 105 | E | E | 101 | 65 | 145 | e | e |
| 6 | 6 | 006 | ACK (acknowledge) | 38 | 26 | 046 | & | & | 70 | 46 | 106 | F | F | 102 | 66 | 146 | f | f |
| 7 | 7 | 007 | BEL (bell) | 39 | 27 | 047 | ' | ' | 71 | 47 | 107 | G | G | 103 | 67 | 147 | g | g |
| 8 | 8 | 010 | BS (backspace) | 40 | 28 | 050 | ( | ( | 72 | 48 | 110 | H | H | 104 | 68 | 150 | h | h |
| 9 | 9 | 011 | TAB (horizontal tab) | 41 | 29 | 051 | ) | ) | 73 | 49 | 111 | I | I | 105 | 69 | 151 | i | i |
| 10 | A | 012 | LF (NL line feed, new line) | 42 | 2A | 052 | * | * | 74 | 4A | 112 | J | J | 106 | 6A | 152 | j | j |
| 11 | B | 013 | VT (vertical tab) | 43 | 2B | 053 | + | + | 75 | 4B | 113 | K | K | 107 | 6B | 153 | k | k |
| 12 | C | 014 | FF (NP form feed, new page) | 44 | 2C | 054 | , | , | 76 | 4C | 114 | L | L | 108 | 6C | 154 | l | l |
| 13 | D | 015 | CR (carriage return) | 45 | 2D | 055 | - | - | 77 | 4D | 115 | M | M | 109 | 6D | 155 | m | m |
| 14 | E | 016 | SO (shift out) | 46 | 2E | 056 | . | . | 78 | 4E | 116 | N | N | 110 | 6E | 156 | n | n |
| 15 | F | 017 | SI (shift in) | 47 | 2F | 057 | / | / | 79 | 4F | 117 | O | O | 111 | 6F | 157 | o | o |
| 16 | 10 | 020 | DLE (data link escape) | 48 | 30 | 060 | 0 | 0 | 80 | 50 | 120 | P | P | 112 | 70 | 160 | p | p |
| 17 | 11 | 021 | DC1 (device control 1) | 49 | 31 | 061 | 1 | 1 | 81 | 51 | 121 | Q | Q | 113 | 71 | 161 | q | q |
| 18 | 12 | 022 | DC2 (device control 2) | 50 | 32 | 062 | 2 | 2 | 82 | 52 | 122 | R | R | 114 | 72 | 162 | r | r |
| 19 | 13 | 023 | DC3 (device control 3) | 51 | 33 | 063 | 3 | 3 | 83 | 53 | 123 | S | S | 115 | 73 | 163 | s | s |
| 20 | 14 | 024 | DC4 (device control 4) | 52 | 34 | 064 | 4 | 4 | 84 | 54 | 124 | T | T | 116 | 74 | 164 | t | t |
| 21 | 15 | 025 | NAK (negative acknowledge) | 53 | 35 | 065 | 5 | 5 | 85 | 55 | 125 | U | U | 117 | 75 | 165 | u | u |
| 22 | 16 | 026 | SYN (synchronous idle) | 54 | 36 | 066 | 6 | 6 | 86 | 56 | 126 | V | V | 118 | 76 | 166 | v | v |
| 23 | 17 | 027 | ETB (end of trans. block) | 55 | 37 | 067 | 7 | 7 | 87 | 57 | 127 | W | W | 119 | 77 | 167 | w | w |
| 24 | 18 | 030 | CAN (cancel) | 56 | 38 | 070 | 8 | 8 | 88 | 58 | 130 | X | X | 120 | 78 | 170 | x | x |
| 25 | 19 | 031 | EM (end of medium) | 57 | 39 | 071 | 9 | 9 | 89 | 59 | 131 | Y | Y | 121 | 79 | 171 | y | y |
| 26 | 1A | 032 | SUB (substitute) | 58 | 3A | 072 | : | : | 90 | 5A | 132 | Z | Z | 122 | 7A | 172 | z | z |
| 27 | 1B | 033 | ESC (escape) | 59 | 3B | 073 | ; | ; | 91 | 5B | 133 | [ | [ | 123 | 7B | 173 | { | { |
| 28 | 1C | 034 | FS (file separator) | 60 | 3C | 074 | < | < | 92 | 5C | 134 | \ | \ | 124 | 7C | 174 | | | \| |

# SAS in linux

I've put a link on my webpage above the course notes to a separate tutorial I've found online regarding SAS in a UNIX (similar to LINUX) environemnt. It has a list of common linux commands and examples of `sed` (which I've used in different projects) and also `awk` (which I've never used, but probably should...).

# Example SAS in program

```
libname mozart "~/Teaching/SAS";
data mozart.test_scores;
  length ID $ 3 Name $ 15;
  input ID $ Name $ Score1-Score3 ;
  datalines;
1 Sha 90 95 98
2 Yuancheng 78 77 75
3 Fiona 88 91 90
;
run;

data ave_scores;
  set mozart.test_scores;
  ave_score = mean(of score1-score3);
run;

proc print data=ave_scores;
  var name ave_score score1-score3 ;
run;
```

# Example from previous slide

There is a lot going on in the previous slide.

1. Score1-Score3 is specified using a range, and SAS understands that Score2 exists as well — it interpolates the numbers
2. Score1-Score3 is capitalized in the datastep, but not later, yet this capitalization is retianed in the printed output
3. I created a new dataset called `ave_scores`, which has a subset of the variables of the first datset
4. `mean(of score1-score3)` computes the mean of `Score1, Score2, Score3` for each row. Note that `mean(score1-score3)` would compute the mean of the difference between those two scores
5. In `proc print`, I am not printing the `ID` variable. Also, I am changing the order in which variables are printed

# Running program in batch mode

```
[jamdeg@polaris SAS]$ sas program4-1.sas

Starting SAS ....

The http://Fastinfo.unm.edu  entry for SAS is:

    http://unm.custhelp.com/cgi-bin/unm.cfg/php/enduser/std_adp.php?p_faqid=6341

[jamdeg@polaris SAS]$ cat program4-1.lst
                                                The SAS System

                                                ave_
                                  Obs    Name   score     Score1    Score2    Score3

                                   1     Sha    94.3333     90        95        98
                                   2     Yuancheng 76.6667  78        77        75
                                   3     Fiona  89.6667     88        91        90
[jamdeg@polaris SAS]$ cat program4-1.log | grep ERROR
[jamdeg@polaris SAS]$
```

# data _null_

You can also use a datastep which doesn't create a new dataset, but processes observations from another dataset and records information about observations from the other dataset. This can be done using a combination of `data _null_` and `set`

```
data _null_;
set learn.test_scores;
if score1 ge 95 or score2 ge 95 or score3 ge 95 then
put ID= Score1= Score2= Score3=;
run;
```

# More on put

The output from the put statement goes to the log file by default, but can be placed elsewhere, by putting

```
file "myfile".txt;
```

or

```
file print;
```

above the put statement. This either sends the output to an external file or to the output window.

# data _null_

data _null_ is especially useful for creating custom reports, such as automatically generating tables, and making data look nicely formatted instead of what is convenient for reading in for data analysis.

# Example with used Toyota cars on Craigslist

```
year price miles title
1995 1200 150000  clean
2004 4500 184000  salvage
1995 3200 . clean
1998 1850 152000  salvage
1998 3400 136000  clean
2004 8500 85500 clean
2007 12400 89000 clean
2002 5450 137000 clean
2007 18500 64000 clean
1996 15000 134000 clean
2008 13999 143934 clean
1997 2500 . salvage
2007 8500 129000 clean
2003 . . salvage
1986 4500 190291 clean
1983 4300 . rebuilt
1976 4500 131000 clean
```

# Printing the data to a file look nicer using data _null_

```
data cars;
  infile "cars.txt" firstobs=2 obs=10;
  input year price miles title $;
run;

data _null_;
  set cars;
  file "carsFormatted.txt";
  if title = "clean"
    then put year price miles;
run;
```

# Printing the data to a file look nicer using data _null_

```
1995 1200 150000
1995 3200 .
1998 3400 136000
2004 8500 85500
2007 12400 89000
2002 5450 137000
2007 18500 64000
```

# Printing the data to a file look nicer using data _null_

```
head -10 cars2.sas | tail -2
  if title = "clean"
    then put year price dollar8. miles;
[jamdeg@mizar SAS]$ cat carsFormatted.txt
1995    $1,200150000
1995    $3,200.
1998    $3,400136000
2004    $8,50085500
2007   $12,40089000
2002    $5,450137000
2007   $18,50064000
```

# Printing the data to a file look nicer using data _null_

```
[jamdeg@mizar SAS]$ !head
head -10 cars2.sas | tail -2
  if title = "clean"
    then put year " " price dollar8. " " miles;
[jamdeg@mizar SAS]$ cat carsFormatted.txt
1995     $1,200 150000
1995     $3,200 .
1998     $3,400 136000
2004     $8,500 85500
2007    $12,400 89000
2002     $5,450 137000
2007    $18,500 64000
```

# Printing the data to a file look nicer using data _null_

```
[jamdeg@mizar SAS]$ !h
head -10 cars2.sas | tail -2
  if title = "clean"
    then put year +1  price dollar8. +4 miles;
[jamdeg@mizar SAS]$ !c
cat carsFormatted.txt
1995     $1,200     150000
1995     $3,200     .
1998     $3,400     136000
2004     $8,500     85500
2007    $12,400     89000
2002     $5,450     137000
2007    $18,500     64000
```

# Printing the data to a file look nicer using data _null_

```
[jamdeg@mizar SAS]$ !h
head -10 cars2.sas | tail -2
  format price dollar8. miles comma8.;
  if title = "clean"
[jamdeg@mizar SAS]$ !c
cat carsFormatted.txt
1995 $1,200 150,000
1995 $3,200 .
1998 $3,400 136,000
2004 $8,500 85,500
2007 $12,400 89,000
2002 $5,450 137,000
2007 $18,500 64,000
```

# Printing the data to a file look nicer using data _null_

```
[jamdeg@mizar SAS]$ head -11 cars2.sas | tail -3
  format price dollar8.2 miles comma13.1;
  if title = "clean"
    then put @1 year @10 price @21 miles;
[jamdeg@mizar SAS]$ !c
cat carsFormatted.txt
1995      $1200.00    150,000.0
1995      $3200.00    .
1998      $3400.00    136,000.0
2004      $8500.00    85,500.0
2007      12400.00    89,000.0
2002      $5450.00    137,000.0
2007      18500.00    64,000.0
```

# Printing the data to a file look nicer using data _null_

```
data cars;
  infile "cars.txt" firstobs=2 obs=10;
  input year price miles title $;
run;

data _null_;
  set cars;
  file "carsFormatted.txt";
  format price dollar8.2 miles comma13.1;
  if _n_ = 1 then do;
    put;
    put "------Clean Title Only------";
    put "YEAR" @10 "PRICE" @25 "MILES";
    put;
  end;

  if salvage = clean
    then put @1 year @10 price @21 miles;
run;
```

# Printing the data to a file look nicer using data _null_

```
data cars;
  infile "cars.txt" firstobs=2 obs=10;
  input year price miles title $;
run;

data _null_;
  set cars;
  file "carsFormatted.txt";
  format price dollar8.2 miles comma13.1;
  if _n_ = 1 then do;
    put;
    put "------Clean Title Only------";
    put "YEAR" @10 "PRICE" @25 "MILES";
    put;
  end;

  if title = "clean" then
    put @1 year @10 price @21 miles;
run;
```

# Printing the data to a file look nicer using data _null_

-------Clean Title Only-------

| YEAR | PRICE | MILES |
|---|---|---|
| 1995 | $1200.00 | 150,000.0 |
| 1995 | $3200.00 | . |
| 1998 | $3400.00 | 136,000.0 |
| 2004 | $8500.00 | 85,500.0 |
| 2007 | 12400.00 | 89,000.0 |
| 2002 | $5450.00 | 137,000.0 |
| 2007 | 18500.00 | 64,000.0 |

# Printing the data to make a LaTeX table

```
data cars;
  infile "cars.txt" firstobs=2 obs=10;
  input year price miles title $;
run;

data _null_;
  set cars;
  file "carsFormatted.txt";
  format price dollar8.2 miles comma13.1;
  if _n_ = 1 then do;
    put;
    put "------Clean Title Only------";
    put "YEAR" @6 "&" @10 "PRICE" @22 "&" @25 "MILES" @35 "\\";
    put "\hline\\";
  end;

  if title = "clean"
    then put @1 year @6 "&" @9 "\" @10 price @19 "&" @22 miles @32 "\\";
run;
```

# Printing the data to make a LaTeX table

```
------Clean Title Only------
YEAR &   PRICE        &  MILES      \\
\hline\\
1995 &  \$1200.00 &  150,000.0 \\
1995 &  \$3200.00 &  .         \\
1998 &  \$3400.00 &  136,000.0 \\
2004 &  \$8500.00 &  85,500.0  \\
2007 &  \12400.00 &  89,000.0  \\
2002 &  \$5450.00 &  137,000.0 \\
2007 &  \18500.00 &  64,000.0  \\
```

# More on logic statements in SAS

The chapter dealing with logic in the book is in Chapter 7, so we're skipping ahead a little bit, but it was natural to use conditional processing with `data _null_`, which was introduced at the end of Chapter 4.

We'll go ahead and look more at conditional processing now since it is so useful.

# IF statements

We already introduced the IF statement with the `data _null_` example. The general syntax for an IF statement is `if`

CONDITION `then` ACTION `;` or `if` CONDITION `then do;`

ACTION1 ;
ACTION2 ;
...
`end;`

# Logical comparisons

The condition for IF statement is typically a logical comparison, such as whether one value is greater than another. Here are some common comparisons and their syntax (use either symbol or code)

| Comparison | symbol | two-letter code |
|---|---|---|
| equal to | = | eq |
| not equal to | ^= or ~= | ne |
| less than | < | lt |
| greater than | > | gt |
| less than or equal to | >= | le |
| greater than or equal to | >= | ge |
| and | & | and |
| or | \| | or |

# Example of using IF statements to create new variables

You might wish to use IF statements to create new variables that will be more useful to you. For the car example, the variable `title` had 4 observed values: `clean`, `salvage`, `rebuilt`, and missing. You might want a variable that just indicates whether or not the title is clean for example. Here you can modify the data step.

```
data cars;
  infile "cars.txt" firstobs=2 obs=10;
  input year price miles title $;
  if title = clean then cleanTitle=1;
  if title ne clean then cleanTitle=0;
run;
```

# Instead of having two IF statements, you can also use the IF-ELSE construction

```
data cars;
  infile "cars.txt" firstobs=2 obs=10;
  input year price miles title $;
  if title = "clean" then cleanTitle=1;
    else cleanTitle=0;
run;

proc print data=cars;
run;
```

# Constructing variables from ranges

Often different categories or ranges are collapsed, sometimes converting continuous variables into categorical or ordinal variables. For example, we might consider car mileage to be either low, medium, high, or very high, depending on the range. It is natural to use IF-ELSE constructions. Here is an example of doing this

```
data cars;
  infile "cars.txt" firstobs=2 obs=10;
  input year price miles title $;
  if title = "clean" then cleanTitle=1;
  if title ne "clean" then cleanTitle=0;
  if 0 < miles < 70000 then mileage="low";
    else if 70000 <= miles < 100000 then mileage="medium";
    else if 100000 <= miles < 150000 then mileage="high";
    else mileage="very high";
run;

proc print data=cars;
run;
```

# Variables from ranges

| Obs | year | price | miles | title | clean Title | mileage |
|---|---|---|---|---|---|---|
| 1 | 1995 | 1200 | 150000 | clean | 1 | ver |
| 2 | 2004 | 4500 | 184000 | salvage | 0 | ver |
| 3 | 1995 | 3200 | . | clean | 1 | ver |
| 4 | 1998 | 1850 | 152000 | salvage | 0 | ver |
| 5 | 1998 | 3400 | 136000 | clean | 1 | hig |
| 6 | 2004 | 8500 | 85500 | clean | 1 | med |
| 7 | 2007 | 12400 | 89000 | clean | 1 | med |
| 8 | 2002 | 5450 | 137000 | clean | 1 | hig |
| 9 | 2007 | 18500 | 64000 | clean | 1 | low |

# Variables from ranges

```
data cars;
  infile "cars.txt" firstobs=2 obs=10;
  input year price miles title $;
  if title = "clean" then cleanTitle=1;
  if title ne "clean" then cleanTitle=0;
  format mileage $10.;
  if 0 < miles < 70000 then mileage="low";
    else if 70000 <= miles < 100000 then mileage="medium";
    else if 100000 <= miles < 150000 then mileage="high";
    else mileage="very high";
run;

proc print data=cars;
run;
```

# Variables from ranges

| Obs | year | price | miles | title | clean Title | mileage |
|---|---|---|---|---|---|---|
| 1 | 1995 | 1200 | 150000 | clean | 1 | very high |
| 2 | 2004 | 4500 | 184000 | salvage | 0 | very high |
| 3 | 1995 | 3200 | . | clean | 1 | very high |
| 4 | 1998 | 1850 | 152000 | salvage | 0 | very high |
| 5 | 1998 | 3400 | 136000 | clean | 1 | high |
| 6 | 2004 | 8500 | 85500 | clean | 1 | medium |
| 7 | 2007 | 12400 | 89000 | clean | 1 | medium |
| 8 | 2002 | 5450 | 137000 | clean | 1 | high |
| 9 | 2007 | 18500 | 64000 | clean | 1 | low |

# Care with missing values

Missing values are set to the smallest negative number, which can cause problems when using inequalities. Note that one of the values set to "very high" is actually missing. It is best to specify ranges instead of relying on `else`.

```
data cars;
  infile "cars.txt" firstobs=2 obs=10;
  input year price miles title $;
  if title = "clean" then cleanTitle=1;
  if title ne "clean" then cleanTitle=0;
  format mileage $10.;
  if miles < 70000 then mileage="low";
    else if 70000 <= miles < 100000 then mileage="medium";
    else if 100000 <= miles < 150000 then mileage="high";
    else if miles >= 150000 then mileage="very high";
run;
```

# Variables from ranges: missing values

| Obs | year | price | miles | title | clean Title | mileage |
|---|---|---|---|---|---|---|
| 1 | 1995 | 1200 | 150000 | clean | 1 | very high |
| 2 | 2004 | 4500 | 184000 | salvage | 0 | very high |
| 3 | 1995 | 3200 | . | clean | 1 | low |
| 4 | 1998 | 1850 | 152000 | salvage | 0 | very high |
| 5 | 1998 | 3400 | 136000 | clean | 1 | high |
| 6 | 2004 | 8500 | 85500 | clean | 1 | medium |
| 7 | 2007 | 12400 | 89000 | clean | 1 | medium |
| 8 | 2002 | 5450 | 137000 | clean | 1 | high |
| 9 | 2007 | 18500 | 64000 | clean | 1 | low |

# Variables from ranges: missing values

```
data cars;
  infile "cars.txt" firstobs=2 obs=10;
  input year price miles title $;
  if title = "clean" then cleanTitle=1;
  if title ne "clean" then cleanTitle=0;
  format mileage $10.;
  if 0 <= miles < 70000 then mileage="low";
    else if 70000 <= miles < 100000 then mileage="medium";
    else if 100000 <= miles < 150000 then mileage="high";
    else if miles >= 150000 then mileage="very high";
run;

proc print data=cars;
run;
```

# Variables from ranges

| Obs | year | price | miles | title | clean Title | mileage |
|---|---|---|---|---|---|---|
| 1 | 1995 | 1200 | 150000 | clean | 1 | very high |
| 2 | 2004 | 4500 | 184000 | salvage | 0 | very high |
| 3 | 1995 | 3200 | . | clean | 1 | |
| 4 | 1998 | 1850 | 152000 | salvage | 0 | very high |
| 5 | 1998 | 3400 | 136000 | clean | 1 | high |
| 6 | 2004 | 8500 | 85500 | clean | 1 | medium |
| 7 | 2007 | 12400 | 89000 | clean | 1 | medium |
| 8 | 2002 | 5450 | 137000 | clean | 1 | high |
| 9 | 2007 | 18500 | 64000 | clean | 1 | low |

# Variables from ranges: missing values

```
data cars;
  infile "cars.txt" firstobs=2 obs=10;
  input year price miles title $;
  if title = "clean" then cleanTitle=1;
  if title ne "clean" then cleanTitle=0;
  format mileage $10.;
  if 0 <= miles < 70000 then mileage="low";
    else if 70000 <= miles < 100000 then mileage="medium";
    else if 100000 <= miles < 150000 then mileage="high";
    else if miles >= 150000 then mileage="very high";
    else if miles = . then mileage="missing";
run;

proc print data=cars;
run;
```

# Variables from ranges: missing values

| Obs | year | price | miles | title | clean Title | mileage |
|---|---|---|---|---|---|---|
| 1 | 1995 | 1200 | 150000 | clean | 1 | very high |
| 2 | 2004 | 4500 | 184000 | salvage | 0 | very high |
| 3 | 1995 | 3200 | . | clean | 1 | missing |
| 4 | 1998 | 1850 | 152000 | salvage | 0 | very high |
| 5 | 1998 | 3400 | 136000 | clean | 1 | high |
| 6 | 2004 | 8500 | 85500 | clean | 1 | medium |
| 7 | 2007 | 12400 | 89000 | clean | 1 | medium |
| 8 | 2002 | 5450 | 137000 | clean | 1 | high |
| 9 | 2007 | 18500 | 64000 | clean | 1 | low |

# Nested IF statements

IF statments can be nested inside one another, just like with any other programming language. An example:

```
data cars;
  infile "cars.txt" firstobs=2 obs=10;
  input year price miles title $;
run;

data _null_;
  set cars;
  file "carsFormatted.txt";
  format price dollar8.2 miles comma13.1;
  if title = "clean" then
    if 0 <= miles < 100000 then put year price miles "low miles";
    else if miles >= 100000 then put year price miles "high miles";
run;
```

# Nested IF statements

```
[jamdeg@mizar SAS]$ cat carsFormatted.txt
1995 $1200.00 150,000.0 high miles
1998 $3400.00 136,000.0 high miles
2004 $8500.00 85,500.0 low miles
2007 12400.00 89,000.0 low miles
2002 $5450.00 137,000.0 high miles
2007 18500.00 64,000.0 low miles
```