Data Manipulation

Statistics 135

Autumn 2005

Reading Files on Lab Machines

When you want to read in data files, for example, on the lab computers, you need to specify the full path name. The earlier examples, which were run under a unix setup, were able to reuse relative path position to specify the file.

For example, for an example I will be looking at later, needed the following code

```
PROC IMPORT DATAFILE = 'F:\Autumn 2005\SAS\Shingles.txt'
  DBMS = dlm OUT = shingles REPLACE;
```

This was being run from a USB memory key, which happened to be set to drive `F:` on the PC I was using. It may be different on different machines. I've seen it some times come up as `E:` or `F:` on my laptop.

Data Manipulation

As we have seen before, it is easy to create new variables using standard functions. For example, for the `shingles` data set all ready read in

Roofing Shingle Sales 48
19:46 Sunday, November 20, 2005

Obs	sales	promotion	accounts	brands	potential
1	79.3	5.5	31	10	8
2	200.1	2.5	55	8	6
3	163.2	8	67	12	9
4	200.1	3	50	7	16

we can create a new variable `value` by

```
DATA shingles2;
  set shingles;
  value = sales * 51.25; /* assuming $51.25 per 1000 squares */
```

It is also possible to do conditional assignments inside a data set with the IF-THEN statement. For example

```
DATA shingles2;    /* part of a more complex data statement */
  set shingles;
  IF brands < 10 THEN compete = "low";
```

This will create a new variable compete which will take the value low if brands < 10 and . (missing) if brands >= 10.

A better approach probably would use an IF-THEN/ELSE statement. For example

```
DATA shingles2;    /* part of a more complex data statement */
  set shingles;
  IF brands < 10 THEN compete = "low";
    ELSE compete = "hi";
```

In this case, compete will have a value for every observation.

Note that more than one command can be run within each part of an IF-THEN statement. In that case, the structure needs to change slightly to

```
IF condition THEN DO;
  action1;
  action2;
END;
ELSE DO;
  action1;
  action2;
END;
```

Of course you can have a different number of steps in each block.

In addition, you can extend the IF-THEN/ELSE statement can be extended to more conditions with the use of ELSE IF.

For example

```
IF potential > 15 then potentcat = "High";
ELSE IF potential < 6 then potentcat = "Low";
ELSE potentcat = "Moderate";
```

This creates a string variable with 3 different values depending, one for each different range of potential considered.

You can include as many `ELSE IF` statements as needed, though this may not be the best way to do it. If there are many options, `SELECT` maybe preferable. One example is

```
SELECT;
  WHEN (0 LE potential LT 5) grade = 0;
  WHEN (5 LE potential LT 10) grade = 1;
  WHEN (10 LE potential LT 15) grade = 2;
  WHEN (15 LE potential LT 20) grade = 3;
  WHEN (potential GE 20) grade = 4;
END;
```

The conditional in the previous `SELECT` statement is a section way of defining conditionals. Conditionals can be defined using the operators

Symbolic	Mnemonic	Meaning
=	EQ	equals
^=, ~= or ¬=	NE	not equal
>	GT	greater than
<	LT	less than
>=	GE	greater than or equal
<=	LE	less than or equal

In addition condition can be strung together with `AND` or `OR` with

Symbolic	Mnemonic	Meaning
&	AND	all comparisons must be true
| or !	OR	only one comparison must be true

Any combination of Symbolic and Mnemonic may be used. However I suggest that you have some level of consistency in your own code.

You need to be careful when dealing with conditions like

```
IF potential <= 5 THEN grade = 0;
```

This would have any observation where `potential` is missing having `grade = 0`, which would probably be undesirable. Instead the following might be better

```
IF 0 <= potential <= 5 THEN grade = 0;
```

There is one other type of comparison available, the `IN` operator. `IN` compares the value of a variable to a list of possible values. For example

```
IF grade in (0, 1) THEN pass = "F";
  ELSE pass = "T";
```

There are a few automatic variables available in **SAS**. Two that are useful are

- `_N_`: This indicates the number of times **SAS**has looped through a `DATA` step. Note that this isn't necessarily the same as the observation number as the dataset might have been subsetted (coming next).

  One example of how it can be used is the following

```
IF _N_ < 27 THEN training = "TRUE";
  ELSE training = "FALSE";
```

- `_ERROR_`: This variable takes the value 1 if there is a data error for that observation and 0 if there isn't. Things that can cause errors include invalid data (assigning characters to a numeric variable), division by 0, or illegal arguments to functions (like `sqrt(-1)`).

When working with data, sometimes you will want to remove variables from a dataset. This can be done in a `DATA` as follows

```
DATA dropdata;
  SET shingle2;
  DROP grade pass;
```

This would remove the variables `grade` and `pass` from the dataset and write the rest to `dropdata`.

You can also state which variables you would like to keep. This could be done as follows by

```
DATA keepdata;
  SET shingle2;
  KEEP sales compete potential;
```

In this case only 3 variables will be written to the new dataset.

Subsetting Your Data

As we have seen before, it is often desirable to only look at a subset of your data. This can be easily accomplished in a `DATA` step with an `IF` statement. For example, suppose we only wanted the first 5 observations from the dataset. One way of doing this is

```
DATA shingles3;
  SET shingles2;
  IF _N_ < 6;  /* ok since _N_ is a integer starting at 1 */
```

There is an equivalent approach that will also work

```
DATA shingles 4;
  SET shingles2;
  IF _N_ > 5 THEN DELETE;
```

When deciding which version to use, generally you will want to pick the form that as the easier to write out.

There are other ways to subset your data. We will see these later on as they work better in some PROCs instead of creating new datasets.

We can see that the two approaches lead to the same answer with the code

```
PROC PRINT DATA = shingles3;
  TITLE 'shingles3 version';

PROC PRINT DATA = shingles4;
  TITLE 'shingles4 version';
```

shingles3 version 00:10 Monday, November 21, 2005 10

Obs	sales	promotion	accounts	brands	potential
1	79.3	5.5	31	10	8
2	200.1	2.5	55	8	6
3	163.2	8	67	12	9
4	200.1	3	50	7	16
5	146	3	38	8	15

shingles4 version 00:10 Monday, November 21, 2005 11

Obs	sales	promotion	accounts	brands	potential
1	79.3	5.5	31	10	8
2	200.1	2.5	55	8	6
3	163.2	8	67	12	9
4	200.1	3	50	7	16
5	146	3	38	8	15

SORTING

So procedures in **SAS** required the data to be sorted. For example, suppose we wanted to sort the data by `sales`. We could do this by

```
PROC SORT DATA = shingles2 OUT = shingles5;
  BY sales;
```

This will create a new dataset `shingles5` where the observations will be ordered in increasing order of `sales`

```
                    Increasing in Sales    00:10 Monday, November 21, 2005  27

Obs    sales       promotion        accounts          brands       potential

  1    30.9              8             30               12               8
  2    47.7            6.1             38               13              10
  3    48              7.5             46               14               3
  4    64.7            5.8             24               10               8
  5    73.4            6.7             53               13               5
...
```

In this output, only the variables sales, promotion, accounts, brands, and potential as I restricted which variables will be printed out with the VAR statement

```
PROC PRINT DATA = shingles5;
  TITLE 'Increasing in Sales';
  VAR sales promotion accounts brands potential;
```

The VAR statement indicates which variables are to be printed.

In you want to sort in decreasing order, add the DESCENDING option to the BY statement. For example

```
PROC SORT DATA = shingles2;
  BY DESCENDING sales;
```

This will overwrite the dataset shingles2 by listing the observations by decreasing sales. It will overwrite the dataset since a OUT option was not set.

Decreasing in Sales 00:10 Monday, November 21, 2005 28

Obs	sales	promotion	accounts	brands	potential
1	339.4	6.5	73	5	16
2	331.2	5.6	71	4	9
3	295.8	5.4	70	6	8
4	291.9	9	56	5	10
5	291.5	5.3	70	7	10

...

Using SAS Procedures

The common structure for **SAS**PROC statements is the following

```
PROC whatever <options>;
  command1;
  command2;
  ...
  commandn;
RUN;
```

Now there can be great flexibility in how these are combined. All that is required is PROC and the procedure name.

Thus

```
PROC PRINT;
```

is a valid command. This will print the current dataset. Usually there will be more to it.

As an example of the possibilities available, lets look at `PROC UNIVARIATE`

```
PROC UNIVARIATE < options > ;
BY variables ;
CLASS variable-1 <(v-options)> < variable-2 <(v-options)> >
       < / KEYLEVEL= value1 | ( value1 value2 ) >;
FREQ variable ;
HISTOGRAM < variables > < / options > ;
ID variables ;
INSET keyword-list < / options > ;
OUTPUT < OUT=SAS-data-set >
         < keyword1=names...keywordk=names >
         < percentile-options >;
PROBPLOT < variables > < / options > ;
QQPLOT < variables > < / options > ;
VAR variables ;
WEIGHT variable ;
```

So we can see, for many PROCs there are a wide range of options available. One that is always available is DATA = . This says which dataset is to be used in the current PROC. If it is not given, **SAS** will use the most recent dataset.

Another options that is often available, but not always, is the OUT = option. This indicates the name of the dataset that should be written out as the result of the PROC. For example, we saw it with PROC SORT. However it doesn't exist for PROC REG, one approach to regression in **SAS**.

There are a number of common commands that are available with all (almost all?) PROCs. These include

- `BY`: This command tells **SAS** to perform a separate analysis for each level of a desired variable (or variables). For example the code

```
PROC UNIVARIATE DATA = shingles2;
  BY potentcat;
  VAR promotion brands;
```

  will give separate analyses of `promotion` and `brands` for each level of potentcat.

  However for this to work, the dataset must be sorted by the `BY` variables.

  In fact, the only `PROC` to require the `BY` command is `PROC SORT`.

- `TITLE` and `FOOTNOTE`: We've seen `TITLE` before. `FOOTNOTE` is the opposite of `TITLE`; it prints at the end of each page of output.

  These can be put anywhere in your code, but generally you want to put them `PROC` where you change them. If you don't change them, they will stay active.

  It is possible to have up to 10 `TITLE` and `FOOTNOTE` active at a time. This is done by adding numbers to the keywords, e.g.

```
TITLE2 'Here''s a second title';
FOOTNOTE3 "Here's another footnote";
```

  Note that these two examples show the two ways of including a ' into text.

  If you wish to get rid other either of name, a null statement will eliminate them

```
TITLE;
FOOTNOTE;
```

- LABEL: This allows for more descriptive labels to be given to variable names. For example

```
LABEL sales = 'Shingle Sales'
LABEL potentcat = 'Sales Potential Category';
```

  If a label is used in a DATA step, it becomes part of the data set. However if it is used in a PROC, it is only active in that PROC.