Chapter 4

Chapter Title: Combining, extracting and reshaping data

ORCID: 0000-0002-7233-1461

Name: A. John Bailer

Abstract:

This chapter deals with programming of one or more data sets to construct a final data set for a particular analysis. Adding new observations to a data set using SET statements or new variables to a data set using MERGE operations are then illustrated. SQL procedure statements are used to mimic these operations. Additional preparation of a data set may require reshaping it from a wide to a long format or from a long to a wide format. This chapter concludes with an illustration of constructing training and validation data sets.

# 4.1 Adding observations by SET-ing data sets

When creating and updating a data set for analysis, you might need to add new observations. In this situation, you have the same variables for different collections of observations. This task is shown in Table 4.1. In this illustration, a data set containing seven observations is supplemented by a data set containing five observations to produce a data set containing 12 observations.

Table 4.1 Adding observations or rows—concatenating data sets

|  |  |  |  |
| --- | --- | --- | --- |
| **Observation ID** | **Variable 1** | **Variable 2** | **Variable 3** |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |
| 5 |  |  |  |
| 6 |  |  |  |
| 7 |  |  |  |

This data set is concatenated with the following data set:

|  |  |  |  |
| --- | --- | --- | --- |
| **Observation ID** | **Variable 1** | **Variable 2** | **Variable 3** |
| 8 |  |  |  |
| 9 |  |  |  |
| 10 |  |  |  |
| 11 |  |  |  |
| 12 |  |  |  |

The following data set is produced:

|  |  |  |  |
| --- | --- | --- | --- |
| **Observation ID** | **Variable 1** | **Variable 2** | **Variable 3** |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |
| 5 |  |  |  |
| 6 |  |  |  |
| 7 |  |  |  |
| 8 |  |  |  |
| 9 |  |  |  |
| 10 |  |  |  |
| 11 |  |  |  |
| 12 |  |  |  |

Suppose you (foolishly? unnecessarily?) read the nitrofen data into a separate data set for each concentration. In Program 4.1, the total brood count variable is entered into separate data sets, one for each concentration condition. A new variable is defined for each concentration, and then each concentration-specific data set is combined to create a single analysis data set using the SETcommand.

data set\_all5;

set conc\_0 conc\_80 conc\_160 conc\_235 conc\_310;

Program 4.1 Combining observations from five separate concentration-specific data sets into a single analysis data set

**data** conc\_0;

input total @@;

conc = **0**; \* define a variable containing the value of the nitrofen conc.;

datalines;

27 32 34 33 36 34 33 30 24 31

;

**run**;

**data** conc\_80;

input total @@;

conc = **80**; \* define a variable containing the value of the nitrofen conc.;

datalines;

33 33 35 33 36 26 27 31 32 29

;

**run**;

**data** conc\_160;

input total @@;

conc = **160**; \* define a variable containing the value of the nitrofen conc.;

datalines;

29 29 23 27 30 31 30 26 29 29

;

**run**;

**data** conc\_235;

input total @@;

conc = **235**; \* define a variable containing the value of the nitrofen conc.;

datalines;

23 21 7 12 27 16 13 15 21 17

;

**run**;

**data** conc\_310;

input total @@;

conc = **310**; \* define a variable containing the value of the nitrofen conc.;

datalines;

6 6 7 0 15 5 6 4 6 5

;

**data** set\_all5;

set conc\_0 conc\_80 conc\_160 conc\_235 conc\_310;

**run**;

**proc** **print** data=set\_all5;

var conc total;

**run**;

WORTH NOTING: The SET statements places the contents of one (or more) data set(s) into a new data set.

The SAS log documents that 10 observations were read into the five intermediate data sets, and that the constructed data set—set\_all5—contains 50 observations with two variables:

NOTE: There were 10 observations read from the data set WORK.CONC\_0.

NOTE: There were 10 observations read from the data set WORK.CONC\_80.

NOTE: There were 10 observations read from the data set WORK.CONC\_160.

NOTE: There were 10 observations read from the data set WORK.CONC\_235.

NOTE: There were 10 observations read from the data set WORK.CONC\_310.

NOTE: The data set WORK.SET\_ALL5 has 50 observations and 2 variables.

The printout of the concatenation of the five separate concentration-specific data sets is shown in Table 4.2.

Table 4.2 Combined data set formed from concatenating the five concentration-specific data sets

| **Obs** | **conc** | **total** |
| --- | --- | --- |
| **1** | 0 | 27 |
| **…** | … | … |
| **11** | 80 | 33 |
| **…** | … | … |
| **20** | 80 | 29 |
| **21** | 160 | 29 |
| **…** | … | … |
| **30** | 160 | 29 |
| **31** | 235 | 23 |
| **…** | … | … |
| **40** | 235 | 17 |
| **41** | 310 | 6 |
| **…** | … | … |
| **50** | 310 | 5 |

# 4.2 Adding variables by MERGE-ing data sets

In addition to adding new observations to an analysis data set, you might need to combine variables from multiple data sources into a single analysis data set. In this situation, you have different variables for the same collections of observations. This section focuses on adding variables from multiple data sources to build a single analysis data set. This task is shown in Table 4.3. The key point is that a common variable—in this case, OBSERVATION ID—is available for gluing together these data sources. Table 4.3 shows the first data set with seven observations and three variables. The second data set has seven observations and two variables that are not in the first data set. Care must be taken when merging data sets that contain variables with the same names, or that contain observations that exist in one data set, but not in the other. This is not a problem, but you need to be aware of what might result from a merge. For example, you could overwrite the values of common variables when merging. One option is to rename common variables before you merge.

Table 4.3 Adding variables and columns—merging data sets

|  |  |  |  |
| --- | --- | --- | --- |
| **Observation ID** | **Variable 1** | **Variable 2** | **Variable 3** |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |
| 4 |  |  |  |
| 5 |  |  |  |
| 6 |  |  |  |
| 7 |  |  |  |

This data set is merged with the following data set:

|  |  |  |
| --- | --- | --- |
| **Observation ID** | **Variable 4** | **Variable 5** |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |
| 6 |  |  |
| 7 |  |  |

The following data set is produced:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Observation ID** | **Variable 1** | **Variable 2** | **Variable 3** | **Variable 4** | **Variable 5** |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| 3 |  |  |  |  |  |
| 4 |  |  |  |  |  |
| 5 |  |  |  |  |  |
| 6 |  |  |  |  |  |
| 7 |  |  |  |  |  |

Suppose the subset of the SMSA data described in Chapter 2 is stored as three data sets that need to be combined to form an analysis data set. For example, one data set might contain the weather data, a second might contain demographic information for city residents, and a third might contain pollution-monitoring data. These three data sets need to be combined into a single file for additional analysis. Program 4.2 shows the SAS code to do this.

Program 4.2 Merge using DATA step programming

**data** SMSA\_subset\_weather;

length city $ **27**;

input city & JanTemp JulyTemp RelHum Rain;

datalines;

Akron, OH 27 71 59 36

Albany-Schenectady-Troy, NY 23 72 57 35

Baltimore, MD 35 77 55 43

Allentown, Bethlehem, PA-NJ 29 74 54 44

Atlanta, GA 45 79 56 47

;

**run**;

**data** SMSA\_subset\_demog;

length city $ **27**;

input city & Mortality Education PopDensity

pct\_NonWhite pct\_WC pop pop\_per\_house income;

datalines;

Akron, OH 921.87 11.4 3243 8.8 42.6 660328 3.34 29560

Albany-Schenectady-Troy, NY 997.87 11.0 4281 3.5 50.7 835880 3.14 31458

Baltimore, MD 1071.29 9.6 6441 24.4 43.7 2199531 3.44 32368

Allentown, Bethlehem, PA-NJ 962.35 9.8 4260 0.8 39.4 635481 3.21 31856

Atlanta, GA 982.29 11.1 3125 27.1 50.2 2138231 3.41 32452

;

**run**;

**data** SMSA\_subset\_pollution;

length city $ **27**;

input city & HCPot NOxPot S02Pot NOx;

datalines;

Akron, OH 21 15 59 15

Albany-Schenectady-Troy, NY 8 10 39 10

Baltimore, MD 43 38 206 38

Allentown, Bethlehem, PA-NJ 6 6 33 6

Atlanta, GA 18 8 24 8

;

**run**;

**proc** **sort** data=SMSA\_subset\_weather;

by city;

**run**;

**proc** **sort** data=SMSA\_subset\_demog;

by city;

**run**;

**proc** **sort** data=SMSA\_subset\_pollution;

by city;

**run**;

**data** all\_subset;

merge SMSA\_subset\_weather SMSA\_subset\_demog SMSA\_subset\_pollution;

by city;

**run**;

**proc** **print** data=all\_subset;

var city JanTemp mortality NOx;

**run**;

Each data set must be sorted by CITY—the common key for the later merge—before merging is done using the MERGE statement.

data all\_subset;

merge SMSA\_subset\_weather SMSA\_subset\_demog SMSA\_subset\_ pollution;

by city;

The SAS log confirms that the five observations in the three SMSA subset data sets are merged into a single data set—all\_subset—containing five observations:

NOTE: There were 5 observations read from the data set WORK.SMSA\_SUBSET\_WEATHER.

NOTE: There were 5 observations read from the data set WORK.SMSA\_SUBSET\_DEMOG.

NOTE: There were 5 observations read from the data set WORK.SMSA\_SUBSET\_POLLUTION.

NOTE: The data set WORK.ALL\_SUBSET has 5 observations and 17 variables.

A printout of the merged data set in Table 4.4 confirms the success of this process.

* + - 1. Table 4.4 Common SMSA data set after the merge

| **Obs** | **city** | **JanTemp** | **Mortality** | **NOx** |
| --- | --- | --- | --- | --- |
| **1** | Akron, OH | 27 | 921.87 | 15 |
| **2** | Albany-Schenectady-Troy, NY | 23 | 997.87 | 10 |
| **3** | Allentown, Bethlehem, PA-NJ | 29 | 962.35 | 6 |
| **4** | Atlanta, GA | 45 | 982.29 | 8 |
| **5** | Baltimore, MD | 35 | 1071.29 | 38 |

Program 4.3 explores merging when observations are not identical in two data sets that are being merged. Two subsets of the weather and demography data from the SMSA study are used. There are three cities in common in these two data sets.

Program 4.3 Merging with DATA steps—indicators that observations that are part of a particular data set (IN=) are defined

options nodate nonumber; \*\*\* comment 0;

**data** SMSA\_subset\_weather2;

length city $ **27**;

input city & JanTemp JulyTemp RelHum Rain;

datalines;

Akron, OH 27 71 59 36

Baltimore, MD 35 77 55 43

Allentown, Bethlehem, PA-NJ 29 74 54 44

Atlanta, GA 45 79 56 47

;

**run**;

**data** SMSA\_subset\_demog2;

length city $ **27**;

input city & Mortality Education PopDensity

pct\_NonWhite pct\_WC pop pop\_per\_house income;

datalines;

Akron, OH 921.87 11.4 3243 8.8 42.6 660328 3.34 29560

Albany-Schenectady-Troy, NY 997.87 11.0 4281 3.5 50.7 835880 3.14 31458

Baltimore, MD 1071.29 9.6 6441 24.4 43.7 2199531 3.44 32368

Allentown, Bethlehem, PA-NJ 962.35 9.8 4260 0.8 39.4 635481 3.21 31856

;

**run**;

**proc** **sort** data=SMSA\_subset\_weather2; \*\*\* comment 1;

by city;

**run**;

**proc** **sort** data=SMSA\_subset\_demog2;

by city;

**run**;

**data** in\_either; \* city in either weather2 or demog2 data set;

merge SMSA\_subset\_weather2 (in=in1)

SMSA\_subset\_demog2 (in=in2); \*\*\* comment 2;

by city;

weather2\_in = in1; \* save indicator of presence in data set;

demog2\_in = in2;

**run**;

**data** in\_both; \* city in both data sets;

set in\_either; \*\*\* comment 3;

if weather2\_in=**1** and demog2\_in=**1**; \*\*\* comment 4;

**run**;

**data** in\_weather; \* city in weather2 data set;

set in\_either;

if weather2\_in=**1**;

**run**;

**data** in\_demog; \* city in demog2 data set;

set in\_either;

if demog2\_in=**1**;

**run**;

ods rtf file="&dir\&subdir\ch4-table4.x3.rtf"

image\_dpi=**300**

style=sasuser.customSapphire

bodytitle;

\*\*\* comment 5;

**proc** **print** data=in\_either; \*\*\* comment 6;

title "city in EITHER weather OR demog data set OR BOTH data sets";

var city weather2\_in demog2\_in JanTemp Rain Mortality income;

**run**;

**proc** **print** data=in\_both;

title "city in BOTH weather AND demog data sets";

var city weather2\_in demog2\_in JanTemp Rain Mortality income;

**run**;

**proc** **print** data=in\_weather;

title "city in weather data set";

var city weather2\_in demog2\_in JanTemp Rain Mortality income;

**run**;

**proc** **print** data=in\_demog;

title "city in demography data set";

var city weather2\_in demog2\_in JanTemp Rain Mortality income;

**run**;

ods rtf close;

Let’s take a look at the LOG information. The first set of lines confirm the number of observations read from each data set and the number of variables in each set.

193 data SMSA\_subset\_weather2;

194 length city $ 27;

195 input city & JanTemp JulyTemp RelHum Rain;

196 datalines;

NOTE: The data set WORK.SMSA\_SUBSET\_WEATHER2 has 4 observations and 5 variables.

...

204 data SMSA\_subset\_demog2;

205 length city $ 27;

206 input city & Mortality Education PopDensity

207 pct\_NonWhite pct\_WC pop pop\_per\_house income;

208 datalines;

NOTE: The data set WORK.SMSA\_SUBSET\_DEMOG2 has 4 observations and 9 variables.

It confirms that the sorted versions of the two data sets have the same number of observations and variables:

216 proc sort data=SMSA\_subset\_weather2; \*\*\* comment 1;

217 by city;

...

NOTE: There were 4 observations read from the data set WORK.SMSA\_SUBSET\_WEATHER2.

NOTE: The data set WORK.SMSA\_SUBSET\_WEATHER2 has 4 observations and 5 variables.

...

220 proc sort data=SMSA\_subset\_demog2;

221 by city;

222 run;

NOTE: There were 4 observations read from the data set WORK.SMSA\_SUBSET\_DEMOG2.

NOTE: The data set WORK.SMSA\_SUBSET\_DEMOG2 has 4 observations and 9 variables.

The SAS log also indicates that the merge of the two subset data sets produces a data set with 13 variables from the two data sets. This data set has two variables that indicate whether an observation is in the first data set, in the second data set, or in both:

224 data in\_either; \* city in either weather2 or demog2 data set;

225 merge SMSA\_subset\_weather2 (in=in1)

226 SMSA\_subset\_demog2 (in=in2); \*\*\* comment 2;

227 by city;

228 weather2\_in = in1; \* save indicator of presence in data set;

229 demog2\_in = in2;

230 run;

NOTE: There were 4 observations read from the data set WORK.SMSA\_SUBSET\_WEATHER2.

NOTE: There were 4 observations read from the data set WORK.SMSA\_SUBSET\_DEMOG2.

NOTE: The data set WORK.IN\_EITHER has 5 observations and 15 variables.

The subsetting IF statements select observations that are in both data sets or in one or the other data set. This is confirmed in the next section of the SAS log:

232 data in\_both; \* city in both data sets;

233 set in\_either; \*\*\* comment 3;

234 if weather2\_in=1 and demog2\_in=1; \*\*\* comment 4;

235 run;

...

NOTE: There were 5 observations read from the data set WORK.IN\_EITHER.

NOTE: The data set WORK.IN\_BOTH has 3 observations and 15 variables.

...

237 data in\_weather; \* city in weather2 data set;

238 set in\_either;

239 if weather2\_in=1;

240 run;

...

NOTE: There were 5 observations read from the data set WORK.IN\_EITHER.

NOTE: The data set WORK.IN\_WEATHER has 4 observations and 15 variables.

...

242 data in\_demog; \* city in demog2 data set;

243 set in\_either;

244 if demog2\_in=1;

245 run;

NOTE: There were 5 observations read from the data set WORK.IN\_EITHER.

NOTE: The data set WORK.IN\_DEMOG has 4 observations and 15 variables.

Key features of this program are denoted by **\*\*\* comment *#***.

\*\*\* comment 0

Denotes options that suppress date and page number information that is included by default in SAS output.

\*\*\* comment 1

Denotes data sets that need to be sorted by the variable that will be used in the merge operation. In this example, the CITY variable matches observations to be merged.

\*\*\* comment 2

The parenthetical statement **(in=in1)** defines an indicator variable **in1** that identifies that a particular city is in the weather2 data set. The variable **in1** is available for use in this DATA step. However, it needs to be assigned to a different variable to be used in other steps. Here, the **in1** variable is assigned to the variable **weather2\_in**. The default behavior of a merge operation is equivalent to the observation being in either or both data sets. In equivalent SAS code,

if weather2\_in=1 OR demog2\_in=1;

\*\*\* comment 3

Denotes that if the contents of the merged data set contain cities in either the weather or demography data set, then the cities are assigned to the in\_both data set.

\*\*\* comment 4

Denotes that a subsetting IF statement is used to select observations that are in both the weather and the demography data sets.

\*\*\* comment 5

This is a common option that I use in programs. **ODS RTF** requests the RTF output destination. **BODYTITLE** includes any title in the body of the RTF file, versus just including a title if it is part of the header information. These options make copying and pasting information into another document easier. A specific file location is provided in this output.

WORTH NOTING: The default with ODS RTF is to place title in the header of a document. The BODYTITLE option in ODS RTF keeps the title as part of document.

\*\*\* comment 6

The next four PROC PRINT statements display the results of merging the data with various criteria. The first PROC PRINT is for cities in either data set. The second is for cities in both data sets. The third is for cities in the weather data set (first data set). The fourth is for cities in the demography data set (second data set).

The data sets are displayed in Table 4.5. The first data set is the result of the merge operation. The second data set is the result of selecting whether a city was in both data sets. The third data set is the result of selecting cities in the weather data set. The fourth data set is the result of selecting cities in the demography data set. Variables indicating whether the city was in the weather data set or the demography data set are displayed. Two variables from the weather data set (JANTEMP and RAIN) and two variables from the demography data set (MORTALITY and INCOME) are also displayed.

Table 4.5 PROC PRINT of the data sets created from merging two data sets with a subset of common observations

**city in EITHER weather OR demog data set OR BOTH data sets**

| **Obs** | **city** | **weather2\_in** | **demog2\_in** |
| --- | --- | --- | --- |
| **1** | Akron, OH | 1 | 1 |
| **2** | Albany-Schenectady-Troy, NY | 0 | 1 |
| **3** | Allentown, Bethlehem, PA-NJ | 1 | 1 |
| **4** | Atlanta, GA | 1 | 0 |
| **5** | Baltimore, MD | 1 | 1 |

| **Obs** | **JanTemp** | **Rain** | **Mortality** | **income** |
| --- | --- | --- | --- | --- |
| **1** | 27 | 36 | 921.87 | 29560 |
| **2** | . | . | 997.87 | 31458 |
| **3** | 29 | 44 | 962.35 | 31856 |
| **4** | 45 | 47 | . | . |
| **5** | 35 | 43 | 1071.29 | 32368 |

**city in BOTH weather AND demog data sets**

| **Obs** | **city** | **weather2\_in** | **demog2\_in** |
| --- | --- | --- | --- |
| **1** | Akron, OH | 1 | 1 |
| **2** | Allentown, Bethlehem, PA-NJ | 1 | 1 |
| **3** | Baltimore, MD | 1 | 1 |

| **Obs** | **JanTemp** | **Rain** | **Mortality** | **income** |
| --- | --- | --- | --- | --- |
| **1** | 27 | 36 | 921.87 | 29560 |
| **2** | 29 | 44 | 962.35 | 31856 |
| **3** | 35 | 43 | 1071.29 | 32368 |

**city in weather data set**

| **Obs** | **city** | **weather2\_in** | **demog2\_in** |
| --- | --- | --- | --- |
| **1** | Akron, OH | 1 | 1 |
| **2** | Allentown, Bethlehem, PA-NJ | 1 | 1 |
| **3** | Atlanta, GA | 1 | 0 |
| **4** | Baltimore, MD | 1 | 1 |

| **Obs** | **JanTemp** | **Rain** | **Mortality** | **income** |
| --- | --- | --- | --- | --- |
| **1** | 27 | 36 | 921.87 | 29560 |
| **2** | 29 | 44 | 962.35 | 31856 |
| **3** | 45 | 47 | . | . |
| **4** | 35 | 43 | 1071.29 | 32368 |

**city in demography data set**

| **Obs** | **city** | **weather2\_in** | **demog2\_in** |
| --- | --- | --- | --- |
| **1** | Akron, OH | 1 | 1 |
| **2** | Albany-Schenectady-Troy, NY | 0 | 1 |
| **3** | Allentown, Bethlehem, PA-NJ | 1 | 1 |
| **4** | Baltimore, MD | 1 | 1 |

| **Obs** | **JanTemp** | **Rain** | **Mortality** | **income** |
| --- | --- | --- | --- | --- |
| **1** | 27 | 36 | 921.87 | 29560 |
| **2** | . | . | 997.87 | 31458 |
| **3** | 29 | 44 | 962.35 | 31856 |
| **4** | 35 | 43 | 1071.29 | 32368 |

# 4.3 Working with tables in PROC SQL

As an alternative to DATA step manipulation, the SAS implementation of a structured query language in PROC SQL might be a better alternative. (This is above and beyond the obvious applications of querying a data table and constructing summary tables.) In fact, programmers who come to SAS with a broad exposure to database programming (say, via formal computer science or MIS training) might find SQL to be more natural for data set manipulation than old-time SAS programmers. This section provides an overview of this procedure with an emphasis on using PROC SQL to concatenate and merge data sets. Background references that might be interesting include documentation about SAS Advanced Certification.

The big idea is that PROC SQL looks at data sets as tables that can be manipulated, processed, and displayed using command queries. Here is the basic syntax, excerpted from SAS Help:

PROC SQL;

CREATE TABLE name;

SELECT variables FROM list-of-tables/dataset

GROUP BY /\* optional \*/

HAVING /\* optional \*/

;

Program 4.4 shows a set of simple queries of a data set.

Program 4.4 Running simple queries on a SAS data set containing the highest nitrofen concentration data

**data** conc\_310;

input total @@;

conc = **310**; \* define a variable containing the value of the nitrofen conc.;

datalines;

6 6 7 0 15 5 6 4 6 5

;

**run**;

**proc** **sql**;

select conc,total from conc\_310;

select total from conc\_310;

select \* from conc\_310

where total > **10**;

**quit**;

The first query selects two variables (or columns) from the conc\_310 SAS data set (or table). The second query selections only the total column from the conc\_310 table, and the third query selects the rows of the table where the total exceeds 10. The views produced from these queries are shown in Table 4.6. SELECT alone (with no CREATE) prints only the selected table entries of interest.

Table 4.6 Result of the simple queries with the total number of young produced in the highest nitrofen concentration

| **conc** | **total** |
| --- | --- |
| 310 | 6 |
| 310 | 6 |
| 310 | 7 |
| 310 | 0 |
| 310 | 15 |
| 310 | 5 |
| 310 | 6 |
| 310 | 4 |
| 310 | 6 |
| 310 | 5 |

| **total** |
| --- |
| 6 |
| 6 |
| 7 |
| 0 |
| 15 |
| 5 |
| 6 |
| 4 |
| 6 |
| 5 |

| **total** | **conc** |
| --- | --- |
| 15 | 310 |

WORTH NOTING: You can have multiple queries in a single invocation of PROC SQL. Enter a quit; statement to close SQL.

The real power of PROC SQL surfaces as you manipulate table relations. For example, five nitrofen concentration data sets are concatenated using UNION in Program 4.5. In PROC SQL, the SAS data set all\_sql is created as a result of the following command:

create table all\_sql as

All variables are selected from each data set as a result of the following command:

select \* from conc\_0

The **\*** is a wildcard representing the selection of all variables. UNION stacks the table into a final table. Then, all values are sorted by ID within CONC as a result of the following command:

order by conc,id;

Program 4.5 Concatenating data sets using UNION in PROC SQL

**data** conc\_0;

input total @@;

conc = **0**; \* define a variable containing the value of the nitrofen conc.;

id=\_n\_; \* define an animal ID corresponding the observation number;

datalines;

27 32 34 33 36 34 33 30 24 31

;

**run**;

**data** conc\_80;

input total @@;

conc = **80**; \* define a variable containing the value of the nitrofen conc.;

id=\_n\_; \* define an animal ID corresponding the observation number;

datalines;

33 33 35 33 36 26 27 31 32 29

**run**;

**data** conc\_160;

input total @@;

conc = **160**;\* define a variable containing the value of the nitrofen conc.;

id=\_n\_; \* define an animal ID corresponding the observation number;

datalines;

29 29 23 27 30 31 30 26 29 29

;

**run**;

**data** conc\_235;

input total @@;

conc = **235**;\* define a variable containing the value of the nitrofen conc.;

id=\_n\_; \* define an animal ID corresponding the observation number;

datalines;

23 21 7 12 27 16 13 15 21 17

;

**run**;

**data** conc\_310;

input total @@;

conc = **310**;\* define a variable containing the value of the nitrofen conc.;

id=\_n\_; \* define an animal ID corresponding the observation number;

datalines;

6 6 7 0 15 5 6 4 6 5

;

**run**;

**proc** **sql**;

create table all\_sql as

select \* from conc\_0

union

select \* from conc\_80

union

select \* from conc\_160

union

select \* from conc\_235

union

select \* from conc\_310

order by conc,id;

**quit**; \* RUN has no effect on SQL because statements are immediately executed;

\* QUIT ends SQL;

**proc** **print** data=all\_sql;

**run**;

The union of these five tables (or data sets) produces a new table named all\_sql that is then printed. The contents of this table are shown in Table 4.7.

Table 4.7 Result of combining five nitrofen concentration tables into a single table using PROC SQL

| **Obs** | **total** | **conc** | **id** |
| --- | --- | --- | --- |
| **1** | 27 | 0 | 1 |
| **2** | 32 | 0 | 2 |
| **…** | … | … | … |
| **48** | 4 | 310 | 8 |
| **49** | 6 | 310 | 9 |
| **50** | 5 | 310 | 10 |

In addition to concatenating tables, PROC SQL merges tables using join commands. In particular, an inner join in PROC SQL is equivalent to a merge in DATA step programming. Program 4.6 contains the PROC SQL code for merging the SMSA subset data sets (i.e., an analogous operation to the merge in Program 4.2).

Program 4.6 An inner join of the three SMSA subset data sets using PROC SQL

**data** SMSA\_subset\_weather;

length city $ **27**;

input city & JanTemp JulyTemp RelHum Rain;

datalines;

Akron, OH 27 71 59 36

Albany-Schenectady-Troy, NY 23 72 57 35

Baltimore, MD 35 77 55 43

Allentown, Bethlehem, PA-NJ 29 74 54 44

Atlanta, GA 45 79 56 47

;

**run**;

**data** SMSA\_subset\_demog;

length city $ **27**;

input city & Mortality Education PopDensity

pct\_NonWhite pct\_WC pop pop\_per\_house income;

datalines;

Akron, OH 921.87 11.4 3243 8.8 42.6 660328 3.34 29560

Albany-Schenectady-Troy, NY 997.87 11.0 4281 3.5 50.7 835880 3.14 31458

Baltimore, MD 1071.29 9.6 6441 24.4 43.7 2199531 3.44 32368

Allentown, Bethlehem, PA-NJ 962.35 9.8 4260 0.8 39.4 635481 3.21 31856

Atlanta, GA 982.29 11.1 3125 27.1 50.2 2138231 3.41 32452

;

**run**;

**data** SMSA\_subset\_pollution;

length city $ **27**;

input city & HCPot NOxPot S02Pot NOx;

datalines;

Akron, OH 21 15 59 15

Albany-Schenectady-Troy, NY 8 10 39 10

Baltimore, MD 43 38 206 38

Allentown, Bethlehem, PA-NJ 6 6 33 6

Atlanta, GA 18 8 24 8

;

**run**;

**proc** **sql**;

create table SMSA\_subset\_sql as

select \* from

SMSA\_subset\_weather w, SMSA\_subset\_demog d,

SMSA\_subset\_pollution p

where w.city=d.city and d.city=p.city;

quit;

**proc** **print** data=SMSA\_subset\_sql;

var city JulyTemp education S02Pot;

**run**;

The PROC SQL commands include the definition of an alias for each data set. For example:

select \* from

SMSA\_subset\_weather w, SMSA\_subset\_demog d,

SMSA\_subset\_pollution p;

This code allows **w.city** to reference the variable CITY in the SMSA\_subset\_weather data set. In addition, a join criterion is declared that matches rows of the combined data set when CITY is the same in the three data sets. For example:

where w.city=d.city and d.city=p.city;

The data set from the PROC SQL inner join is shown in Table 4.8.

* + - 1. **Table 4.8** Data set produced by the inner join of the three SMSA subset data sets

| **Obs** | **city** | **JulyTemp** | **Education** | **S02Pot** |
| --- | --- | --- | --- | --- |
| **1** | Akron, OH | 71 | 11.4 | 59 |
| **2** | Albany-Schenectady-Troy, NY | 72 | 11.0 | 39 |
| **3** | Baltimore, MD | 77 | 9.6 | 206 |
| **4** | Allentown, Bethlehem, PA-NJ | 74 | 9.8 | 33 |
| **5** | Atlanta, GA | 79 | 11.1 | 24 |

One difference between merging data sets using DATA step programming versus PROC SQL is that the input data sets need to be sorted by some key variable before merging with DATA step programming. Sorting is not required in PROC SQL.

An interesting question is, “What happens if the data doesn’t have complete matches in the data sets that are being combined?” In the next series of displays, this question is answered. In Program 4.7, two of the SMSA subset data sets that have three of the five cities are shown. The first table shows the results of an inner join containing the three cities that are in common. The second and third tables show the results of an outer join[[1]](#footnote-1) and the elements of the first data set in the join (a left join). The fourth table shows the results of an outer join and the elements of the second data set in the join (a right join). The fifth table shows the results of an outer join and the union of elements from both data sets (a full join). SAS documentation illustrates these joins with shaded Venn diagrams, which is a nice model for conceptualizing these operations. Output from these table operations is shown in Table 4.9.

Program 4.7 Left, right, or full outer joins with a subset of the SMSA observations

**data** SMSA\_subset\_weather2;

length city $ **27**;

input city & JanTemp JulyTemp RelHum Rain;

datalines;

Akron, OH 27 71 59 36

Baltimore, MD 35 77 55 43

Allentown, Bethlehem, PA-NJ 29 74 54 44

Atlanta, GA 45 79 56 47

;

**run**;

**data** SMSA\_subset\_demog2;

length city $ **27**;

input city & Mortality Education PopDensity

pct\_NonWhite pct\_WC pop pop\_per\_house income;

datalines;

Akron, OH 921.87 11.4 3243 8.8 42.6 660328 3.34 29560

Albany-Schenectady-Troy, NY 997.87 11.0 4281 3.5 50.7 835880 3.14 31458

Baltimore, MD 1071.29 9.6 6441 24.4 43.7 2199531 3.44 32368

Allentown, Bethlehem, PA-NJ 962.35 9.8 4260 0.8 39.4 635481 3.21 31856

;

**run**;

/\* inner join / conventional join \*/

**proc** **sql**;

title "inner join";

select w.city,JanTemp,JulyTemp,Education,income from

SMSA\_subset\_weather2 as w,SMSA\_subset\_demog2 as d

where w.city=d.city;

/\* LEFT outer join - with duplicate columns \*/

title "LEFT outer join with duplicate columns";

select \* from

SMSA\_subset\_weather2 as w

left join

SMSA\_subset\_demog2 as d

on w.city=d.city;

/\* LEFT outer join - eliminating duplicate columns using COALESCE \*/

title "LEFT outer join eliminating duplicate columns";

select coalesce(w.city, d.city),JanTemp,JulyTemp,Education,income from

SMSA\_subset\_weather2 as w

left join

SMSA\_subset\_demog2 as d

on w.city=d.city;

/\* RIGHT outer join \*/

title "RIGHT outer join";

select coalesce(w.city, d.city),JanTemp,JulyTemp,Education,income from

SMSA\_subset\_weather2 as w

right join

SMSA\_subset\_demog2 as d

on w.city=d.city;

/\* FULL outer join \*/

title "FULL outer join";

select coalesce(w.city, d.city),JanTemp,JulyTemp,Education,income from

SMSA\_subset\_weather2 as w

full join

SMSA\_subset\_demog2 as d

on w.city=d.city;

**quit**;

Table 4.9 Results from inner join, left outer join, right outer join, and full outer join

***inner join***

| **city** | **JanTemp** | **JulyTemp** | **Education** | **income** |
| --- | --- | --- | --- | --- |
| Akron, OH | 27 | 71 | 11.4 | 29560 |
| Baltimore, MD | 35 | 77 | 9.6 | 32368 |
| Allentown, Bethlehem, PA-NJ | 29 | 74 | 9.8 | 31856 |

***left outer join with duplicate columns***

| **city** | | **JanTemp** | | **JulyTemp** | **RelHum** | | **Rain** | | **city** | | **Mortality** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Education** | **PopDensity** | | **pct\_NonWhite** | | | **pct\_WC** | | **pop** | | **pop\_per\_house** | |
| **income** | | | | | | | | | | | |
| Akron, OH | | 27 | | 71 | 59 | | 36 | | Akron, OH | | 921.87 |
| 11.4 | 3243 | | 8.8 | | | 42.6 | | 660328 | | 3.34 | |
| 29560 | | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Allentown, Bethlehem, PA-NJ | | 29 | | 74 | 54 | | 44 | | Allentown, Bethlehem, PA-NJ | | 962.35 |
| 9.8 | 4260 | | 0.8 | | | 39.4 | | 635481 | | 3.21 | |
| 31856 | | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Atlanta, GA | | 45 | | 79 | 56 | | 47 | |  | | . |
| . | . | | . | | | . | | . | | . | |
| . | | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Baltimore, MD | | 35 | | 77 | 55 | | 43 | | Baltimore, MD | | 1071.29 |
| 9.6 | 6441 | | 24.4 | | | 43.7 | | 2199531 | | 3.44 | |
| 32368 | | | | | | | | | | | |

***left outer join eliminating duplicate columns***

|  | **JanTemp** | **JulyTemp** | **Education** | **income** |
| --- | --- | --- | --- | --- |
| Akron, OH | 27 | 71 | 11.4 | 29560 |
| Allentown, Bethlehem, PA-NJ | 29 | 74 | 9.8 | 31856 |
| Atlanta, GA | 45 | 79 | . | . |
| Baltimore, MD | 35 | 77 | 9.6 | 32368 |

***right outer join***

|  | **JanTemp** | **JulyTemp** | **Education** | **income** |
| --- | --- | --- | --- | --- |
| Akron, OH | 27 | 71 | 11.4 | 29560 |
| Albany-Schenectady-Troy, NY | . | . | 11 | 31458 |
| Allentown, Bethlehem, PA-NJ | 29 | 74 | 9.8 | 31856 |
| Baltimore, MD | 35 | 77 | 9.6 | 32368 |

***full outer join***

|  | **JanTemp** | **JulyTemp** | **Education** | **income** |
| --- | --- | --- | --- | --- |
| Akron, OH | 27 | 71 | 11.4 | 29560 |
| Albany-Schenectady-Troy, NY | . | . | 11 | 31458 |
| Allentown, Bethlehem, PA-NJ | 29 | 74 | 9.8 | 31856 |
| Atlanta, GA | 45 | 79 | . | . |
| Baltimore, MD | 35 | 77 | 9.6 | 32368 |

Table 4.10 shows a table comparing the coding for DATA step merges of the weather and demography data sets (from Programs 4.3) with the PROC SQL joins (from Program 4.7) of these data sets.

Table 4.10 Table comparing merges of data sets using the DATA step with joins of data sets using PROC SQL

| **Location of CITY** | **DATA Step** | **PROC SQL** |
| --- | --- | --- |
| Either in the weather or demography data set | data in\_either;  merge  SMSA\_subset\_weather2  (in=in1)  SMSA\_subset\_demog2  (in=in2);  by city;  weather2\_in = in1;  demog2\_in = in2;  run; | proc sql;  title "FULL OUTER JOIN";  select coalesce(w.city, d.city),   JanTemp,JulyTemp, Education,income  from  SMSA\_subset\_weather2 as w  full join  SMSA\_subset\_demog2 as d  on w.city=d.city;  quit; |
| In both the weather and demography data sets | data in\_both;  set in\_either;  if weather2\_in=1 and   demog2\_in=1;  run; | proc sql;  title "INNER JOIN";  select  w.city,JanTemp,JulyTemp, Education,income  from  SMSA\_subset\_weather2 as w,  SMSA\_subset\_demog2 as d  where  w.city=d.city;  quit; |
| In weather data set | data in\_weather;  set in\_either;  if weather2\_in=1;  run; | proc sql;  title "LEFT OUTER JOIN";  select coalesce(w.city, d.city),   JanTemp,JulyTemp,Education,income  from  SMSA\_subset\_weather2 as w  left join  SMSA\_subset\_demog2 as d  on w.city=d.city;  quit; |
| In demography data set | data in\_demog;  set in\_either;  if demog2\_in=1;  run; | proc sql;  title "RIGHT OUTER JOIN";  select coalesce(w.city, d.city),   JanTemp,JulyTemp,Education,income  from  SMSA\_subset\_weather2 as w  right join  SMSA\_subset\_demog2 as d  on w.city=d.city;  quit; |

From this table, you can see that an inner join is equivalent to a merge that requires observations to be in both data sets. A full outer join is equivalent to the default merge in which observations are included if they are in either or both of the data sets. The left or right outer join is equivalent to a merge with specified restrictions about whether an observation is included from the first or second data set in a merge.

There are times when you are interested in the complete Cartesian product of the observations in two data sets (i.e., all rows from one data set paired with all rows from another data set). Consider a toy example in which you want to enumerate the sample space resulting from tossing a coin and rolling a six-sided die. Now, this can be constructed using DATA step programming and DO-END loops (as illustrated in the code at the bottom of Program 4.8). However, the application of PROC SQL is especially nice in this case. Program 4.8 defines two data sets: one with the coin toss outcomes enumerated, and one with the six-sided die roll outcomes enumerated. The sample space of all 12 combinations of coin tosses and die rolls is produced and shown in Table 4.11.

Program 4.8 Constructing the sample space based on tossing a coin and rolling a die

**data** coin\_toss;

toss="Heads"; output;

toss="Tails"; output;

**run**;

**data** die\_roll;

face=**1**; output; face=**2**; output;

face=**3**; output; face=**4**; output;

face=**5**; output; face=**6**; output;

**run**;

**proc** **sql**;

create table roll\_n\_toss as

select \* from coin\_toss,die\_roll;

**quit**;

**proc** **print** data=roll\_n\_toss;

**run**;

**data** roll\_n\_toss2; \* using ARRAYS (see Ch. 9 for details);

array array\_toss(\*) $ toss1-toss2 ("Heads", "Tails");

array array\_face(\*) face1-face6 (**1**, **2**, **3**, **4**, **5**, **6**);

do itoss=**1** to **2**;

do iface=**1** to **6**;

toss = array\_toss(itoss);

face = array\_face(iface);

keep toss face;

output;

end;

end;

**run**;

**proc** **print** data=roll\_n\_toss2;

**run**;

The Cartesian product of all combinations of the coin toss and the die roll are shown in Table 4.11.

Table 4.11 Data set constructed from joining all rows from two tables

| **Obs** | **toss** | **face** |
| --- | --- | --- |
| **1** | Heads | 1 |
| **2** | Heads | 2 |
| **3** | Heads | 3 |
| **4** | Heads | 4 |
| **5** | Heads | 5 |
| **6** | Heads | 6 |
| **7** | Tails | 1 |
| **8** | Tails | 2 |
| **9** | Tails | 3 |
| **10** | Tails | 4 |
| **11** | Tails | 5 |
| **12** | Tails | 6 |

# 4.4 Converting wide to long formats

Different procedures in SAS might expect data to have different structures. For example, some of the multivariate procedures expect all of the variables to be defined in a single record. In other cases, such as in mixed models applied to longitudinal data, different measurements are expected to be defined in different records. It is important to be able to program the expansion of a single record to multiple records, or to be able to condense multiple records into a single record.

To illustrate the reshaping of a data set from a wide format to a long format, consider the following example. Suppose you have data, say an activities of daily living (ADL) score, on three nursing home residents measured at seven time points. The original data in its original shape is displayed in Table 4.12.

Table 4.12: Wide data version of ADL data set

| **Obs** | **name** | **sex** | **t1** | **t2** | **t3** | **t4** | **t5** | **t6** | **t7** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | Smith | M | 6 | 6 | 5 | 5 | 5 | 4 | 3 |
| **2** | Jones | F | 7 | 5 | 4 | 4 | 3 | 2 | 1 |
| **3** | Fisher | M | 5 | 5 | 5 | 3 | 2 | 2 | 1 |

This is not is a form where we could easily plot the ADL value versus time. As part of reshaping this data set, you want to add a new variable, say time, that takes values 1-7 and another variable, say ADL, that takes the appropriate value of t1-t7. Program 4.9 does this by first storing the 7 ADL values for each individual in an array and then looping over this array.

* + - 1. Program 4.9 Converting from wide to long format using DATA step programming

**data** wide\_original;

input name $ sex $ t1 t2 t3 t4 t5 t6 t7;

datalines;

Smith M 6 6 5 5 5 4 3

Jones F 7 5 4 4 3 2 1

Fisher M 5 5 5 3 2 2 1

;

**run**;

**data** wide2long;

input name $ gender $ t1 t2 t3 t4 t5 t6 t7;

array num\_array{\*} \_NUMERIC\_;

do time = **1** to dim(num\_array);

ADL = num\_array{time};

output;

end;

keep name gender time ADL;

datalines;

Smith M 6 6 5 5 5 4 3

Jones F 7 5 4 4 3 2 1

Fisher M 5 5 5 3 2 2 1

;

**run**;

**proc** **print** data=original\_shape;

**run**;

**proc** **print** data=wide2long;

**run**;

The **array** statement collects all of the numeric variables. The **dim** function returns the length of this array **(=7** in this example). The **time** variable is defined in the loop, and the **ADL** variable is defined in terms of each array element. The results of this iterative assignment are output to the data set. Only four of the variables are preserved in the data set by the **keep** statement. **Try this at home:** See what happens if you run this code without the **keep** statement.

As a final remark about arrays, an array can be defined in more than one dimension, although you might prefer to use matrices in PROC IML before resorting to higher dimensional data arrays. The reshaped “long” version of the data set is displayed in Table 4.13.

Table 4.13 Resulting “long” version of the ADL data set

| **Obs** | **name** | **gender** | **time** | **ADL** |
| --- | --- | --- | --- | --- |
| **1** | Smith | M | 1 | 6 |
| **2** | Smith | M | 2 | 6 |
| **3** | Smith | M | 3 | 5 |
| **4** | Smith | M | 4 | 5 |
| **5** | Smith | M | 5 | 5 |
| **6** | Smith | M | 6 | 4 |
| **7** | Smith | M | 7 | 3 |
| **8** | Jones | F | 1 | 7 |
| **9** | Jones | F | 2 | 5 |
| **10** | Jones | F | 3 | 4 |
| **11** | Jones | F | 4 | 4 |
| **12** | Jones | F | 5 | 3 |
| **13** | Jones | F | 6 | 2 |
| **14** | Jones | F | 7 | 1 |
| **15** | Fisher | M | 1 | 5 |
| **16** | Fisher | M | 2 | 5 |
| **17** | Fisher | M | 3 | 5 |
| **18** | Fisher | M | 4 | 3 |
| **19** | Fisher | M | 5 | 2 |
| **20** | Fisher | M | 6 | 2 |
| **21** | Fisher | M | 7 | 1 |

The SAS TRANSPOSE procedure provides an alternative for this reshaping task. Program 4.10 uses TRANSPOSE to reshape from wide to long data structure.

Program 4.10 Wide to long format reshaping using PROC TRANSPOSE

**proc** **sort** data=wide\_original;

by name;

**run**;

**proc** **transpose** data=wide\_original out=twide2long;

by name;

var t1-t7;

**run**;

**proc** **print** data=twide2long;

**run**;

**data** wide2long2;

set twide2long;

time = input(substr(\_NAME\_,**2**,**1**),**2.**);

ADL = COL1;

drop \_NAME\_ COL1;

**run**;

**proc** **print** data=wide2long2;

**run**;

While this transposition does the reshaping, we see in Table 4.14 that a little clean up work will be needed. The variable \_NAME\_ contains the names of the variables from the original data set that were transformed into rows. You need to extract the time value from the \_NAME\_ variable and rename the COL1 variable in a more natural way.

Table 4.14: Output from PROC TRANSPOSE

| **Obs** | **name** | **\_NAME\_** | **COL1** |
| --- | --- | --- | --- |
| **1** | Fisher | t1 | 5 |
| **2** | Fisher | t2 | 5 |
| **…** | … | … | … |
| **20** | Smith | t6 | 4 |
| **21** | Smith | t7 | 3 |

You can use the SUBSTR function to extract a particular part of a character string, here you can select the substring of the \_NAME\_ variable starting at the second position and select a substring of length 1. After this, the resulting string, e.g. “1”, is converted to a numeric variable using the INPUT function.

WORTH NOTING: The INPUT function is different from an INPUT statement. This function is used to convert a character value into a numeric value.

# 4.5 Converting long to wide formats

In a complementary way, you may want to move from long to wide formats. It may be that rows represent measurements of the same variable at different times as in our previous section, or it could be that different variables for the same observation are in different rows. This second case is common in some spreadsheets available from organizations such as the World Bank. This reshaping might be a step you need to take before calculating correlations between these variables.

We consider two strategies for doing this reshaping from long to wide format. In the first strategy, we will retain values of variables while we process each observation and then output the record when we finish processing an observation, see Program 4.11. In the second case, we directly apply PROC TRANSPOSE to do the work.

Program 4.11 begins by sorting the data set by name and time. This allows you to access the indicator variables, FIRST.NAME and LAST.NAME. When it is the first row for an individual, FIRST.NAME=1, you initialize variables for storing name and gender along with a counter of time and the first measurement for that time. For all other observations, you increment the time counter and save the measurement for that time. Finally, after you process the last time for an individual, you output this record to the wide data set that you are constructing. Arrays are used to store the times.

Program 4.11 Long to wide format reshaping using RETAIN and arrays

**data** long\_original;

input name $ gender $ time ADL;

datalines;

Smith M 1 6

Smith M 2 6

Smith M 3 5

Smith M 4 5

Smith M 5 5

Smith M 6 4

Smith M 7 3

Jones F 1 7

Jones F 2 5

Jones F 3 4

Jones F 4 4

Jones F 5 3

Jones F 6 2

Jones F 7 1

Fisher M 1 5

Fisher M 2 5

Fisher M 3 5

Fisher M 4 3

Fisher M 5 2

Fisher M 6 2

Fisher M 7 1

;

**run**;

**proc** **sort** data=long\_original;

by name time;

**run**;

**proc** **print** data=long\_original;

**run**;

**data** long2wide;

set long\_original;

by name;

array t(**7**) t1-t7;

array ADL\_array(**7**) ADL1-ADL7;

retain count\_time wname wgender t1-t7 ADL1-ADL7;

if first.name = **1** then do;

count\_time = **1**;

wname = name;

wgender = gender;

t(count\_time) = time;

ADL\_array(time) = ADL;

end;

else do;

count\_time = count\_time + **1**;

t(count\_time) = time;

ADL\_array(count\_time) = ADL;

end;

if last.name = **1** then output;

keep wname wgender t1-t7 ADL1-ADL7;

**run**;

**proc** **print** data=long2wide;

var wname wgender t1-t7 ADL1-ADL7;

**run**;

It is much easier to do this reshaping using PROC TRANSPOSE as we see in Program 4.12.

Program 4.12 Long to wide format reshaping using PROC TRANSPOSE

**proc** **transpose** data=long\_original out=tlong2wide prefix=t;

by name gender;

id time;

var ADL;

**run**;

**proc** **print** data=tlong2wide;

**run**;

# 4.6 Case Study: Reshaping a World Bank data set

You can apply the logic from to the two previous sections to a more complicated data set. Consider the data first introduced in Chapter 1 from the World Bank. Download the data on health, nutrition and population statistics can be downloaded from the World Bank, e.g. [http://databank.worldbank.org/data/reports.aspx?source=health-nutrition-and-population-statistics-by-wealth-quintile#](http://databank.worldbank.org/data/reports.aspx?source=health-nutrition-and-population-statistics-by-wealth-quintile) or you can download this directly for the web site for this book.

Data set was extracted from this site using the API to specify countries, variables, and years. An extract of the data downloaded from this site is given below. You see a column for country, variable (“Series Name”) and data from different years.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Series Name |  | Series Code | Country Name | Country Code | 1998 [YR1998] | 1999 [YR1999] | 2000 [YR2000] |
| Cause of death, by injury (% of total) |  | SH.DTH.INJR.ZS | Afghanistan | AFG | .. | .. | 12.9 |
| Cause of death, by injury (% of total) |  | SH.DTH.INJR.ZS | Albania | ALB | .. | .. | 7.5 |
| Cause of death, by injury (% of total) |  | SH.DTH.INJR.ZS | Algeria | DZA | .. | .. | 12.6 |
| Cause of death, by injury (% of total) |  | SH.DTH.INJR.ZS | American Samoa | ASM | .. | .. | .. |

As discussed previously, a restructuring of the data would allow for better analysis and visualization options. You see that variables are measured over time and stored in different columns; thus, you need to do some wide-to-long reshaping. In addition, you see different variables for each country are stored in different rows; thus, you need to do some long-to-wide reshaping. In Chapter 1, we presented a solution to reshaping a small prototypic data set that captured key features of this World Bank data. Here, we tackle the actual data set.

Program 4.13 reads in the data set and prints observations 100 to 115.

Program 4.13: Read the World Bank data set and print a few observations

**data** WB;

infile "&dir\&subdir\World-Bank-HNP-05apr18.csv" dsd

firstobs=**2**;

input SeriesName $ SeriesCode $ CountryName $ CountryCode $

YR1998 $ YR1999 $ YR2000

YR2001 $ YR2002 $ YR2003 $ YR2004 $ YR2005 $ YR2006 $

YR2007 $ YR2008 $ YR2009 $ YR2010 $ YR2011 $ YR2012 $

YR2013 $ YR2014 $ YR2015 $ YR2016 $ YR2017 $ ;

**run**;

**proc** **print** data=WB (firstobs=**100** obs=**115**);

var SeriesName -- YR2001;

**run**;

proc contents data=WB;

run;

WORTH NOTING: You can specify a range of variable names by either – (2 dashes) for variables that do not share a common prefix (e.g. SeriesName – YR2001) or – (single dash) for variables that share a common prefix (e.g. YR2001-YR2017).

In this program, we read the CSV data file starting with the second observation. All columns are read as character variables. Sixteen observations starting at observation 100 are then printed as we see in Table 4.15. Table 4.15 also displays the portion of the PROC CONTENTS output describing variable type and length.

Table 4.15 Print of 15 observations and select variables from the World Bank data set and a portion of PROC CONTENTS output

| **Obs** | **SeriesName** | **SeriesCode** | **CountryCode** | **YR1998** | **YR2000** |
| --- | --- | --- | --- | --- | --- |
| **100** | Cause of | SH.DTH.I | ITA | .. | 4.8 |
| **101** | Cause of | SH.DTH.I | JAM | .. | 8.0 |
| **102** | Cause of | SH.DTH.I | JPN | .. | 7.9 |
| **103** | Cause of | SH.DTH.I | JOR | .. | 11.6 |
| **104** | Cause of | SH.DTH.I | KAZ | .. | 13.1 |
| **105** | Cause of | SH.DTH.I | KEN | .. | 7.1 |
| **106** | Cause of | SH.DTH.I | KIR | .. | 7.2 |
| **107** | Cause of | SH.DTH.I | PRK | .. | 8.3 |
| **108** | Cause of | SH.DTH.I | KOR | .. | 11.8 |
| **109** | Cause of | SH.DTH.I | XKX | .. | . |
| **110** | Cause of | SH.DTH.I | KWT | .. | 16.1 |
| **111** | Cause of | SH.DTH.I | KGZ | .. | 8.2 |
| **112** | Cause of | SH.DTH.I | LAO | .. | 9.0 |
| **113** | Cause of | SH.DTH.I | LVA | .. | 10.1 |
| **114** | Cause of | SH.DTH.I | LBN | .. | 8.1 |
| **115** | Cause of | SH.DTH.I | LSO | .. | 7.3 |

| **Alphabetic List of Variables and Attributes** | | | |
| --- | --- | --- | --- |
| **#** | **Variable** | **Type** | **Len** |
| **4** | CountryCode | Char | 8 |
| **3** | CountryName | Char | 8 |
| **2** | SeriesCode | Char | 8 |
| **1** | SeriesName | Char | 8 |
| **5** | YR1998 | Char | 8 |
| **6** | YR1999 | Char | 8 |
| **…** | … | … | … |
| **24** | YR2017 | Char | 8 |

Table 4.15 highlights that we need to change the length of some of the variables and read the YR1998, YR1999, … Y2017 variables as numeric. Missing values for variables in the original data set were denoted by two periods (..).

WORTH NOTING: Reading all variables as characters initially is a good place to start. If you read character variables as numeric, then this generates a missing value. This may be what you want.

Program 4.14 addresses these issues by increasing the lengths (some experimenting before I decided at the values used here).

Program 4.14 Reading and reshaping World Bank data - second attempt

**data** WB;

infile "&dir\&subdir\World-Bank-HNP-05apr18.csv" dsd firstobs=**2**;

length SeriesName $ **80** SeriesCode $ **35** CountryName $ **40**;

input SeriesName $ SeriesCode $ CountryName $ CountryCode $

YR1998 $ YR1999 $ YR2000

YR2001 $ YR2002 $ YR2003 $ YR2004 $ YR2005 $ YR2006 $

YR2007 $ YR2008 $ YR2009 $ YR2010 $ YR2011 $ YR2012 $

YR2013 $ YR2014 $ YR2015 $ YR2016 $ YR2017 $ ;

**run**;

**proc** **print** data=WB (firstobs=**100** obs=**115**);

var SeriesName -- YR2001;

**run**;

Table 4.16 shows that the changes in Program 4.14 appear to be producing the desired results.

Table 4.16 Print of 16 observations from the World Bank data set – 2nd attempt to read

| **Obs** | **SeriesName** | **SeriesCode** | **CountryName** |
| --- | --- | --- | --- |
| **100** | Cause of death, by injury (% of total) | SH.DTH.INJR.ZS | Italy |
| **101** | Cause of death, by injury (% of total) | SH.DTH.INJR.ZS | Jamaica |
| **102** | Cause of death, by injury (% of total) | SH.DTH.INJR.ZS | Japan |
| **103** | Cause of death, by injury (% of total) | SH.DTH.INJR.ZS | Jordan |
| **104** | Cause of death, by injury (% of total) | SH.DTH.INJR.ZS | Kazakhstan |
| **105** | Cause of death, by injury (% of total) | SH.DTH.INJR.ZS | Kenya |
| **106** | Cause of death, by injury (% of total) | SH.DTH.INJR.ZS | Kiribati |
| **107** | Cause of death, by injury (% of total) | SH.DTH.INJR.ZS | Korea, Dem. People’s Rep. |
| **108** | Cause of death, by injury (% of total) | SH.DTH.INJR.ZS | Korea, Rep. |
| **109** | Cause of death, by injury (% of total) | SH.DTH.INJR.ZS | Kosovo |
| **110** | Cause of death, by injury (% of total) | SH.DTH.INJR.ZS | Kuwait |
| **111** | Cause of death, by injury (% of total) | SH.DTH.INJR.ZS | Kyrgyz Republic |
| **112** | Cause of death, by injury (% of total) | SH.DTH.INJR.ZS | Lao PDR |
| **113** | Cause of death, by injury (% of total) | SH.DTH.INJR.ZS | Latvia |
| **114** | Cause of death, by injury (% of total) | SH.DTH.INJR.ZS | Lebanon |
| **115** | Cause of death, by injury (% of total) | SH.DTH.INJR.ZS | Lesotho |

| **Obs** | **CountryCode** | **YR1998** | **YR1999** | **YR2000** | **YR2001** |
| --- | --- | --- | --- | --- | --- |
| **100** | ITA | .. | .. | 4.8 | .. |
| **101** | JAM | .. | .. | 8.0 | .. |
| **102** | JPN | .. | .. | 7.9 | .. |
| **103** | JOR | .. | .. | 11.6 | .. |
| **104** | KAZ | .. | .. | 13.1 | .. |
| **105** | KEN | .. | .. | 7.1 | .. |
| **106** | KIR | .. | .. | 7.2 | .. |
| **107** | PRK | .. | .. | 8.3 | .. |
| **108** | KOR | .. | .. | 11.8 | .. |
| **109** | XKX | .. | .. | . | .. |
| **110** | KWT | .. | .. | 16.1 | .. |
| **111** | KGZ | .. | .. | 8.2 | .. |
| **112** | LAO | .. | .. | 9.0 | .. |
| **113** | LVA | .. | .. | 10.1 | .. |
| **114** | LBN | .. | .. | 8.1 | .. |
| **115** | LSO | .. | .. | 7.3 | .. |

Now we need to start reshaping this data set. The variable / column SeriesName contains the names of variables. For future analyses, you need to move these to columns. In addition, the separate columns associated with years (YR1998-YR2017) need to be moved to a single column, say YEAR, that has values 1998, … , 2017. Program 4.15 tackles the years-as-different-columns problem.

Program 4.15 Moving YR1998-YR2017 columns to single column

\* read in World Bank data;

**data** WB;

infile "&dir\&subdir\World-Bank-HNP-05apr18.csv" dsd firstobs=**2**;

length SeriesName $ **80** SeriesCode $ **35** CountryName $ **40** ;

input SeriesName $ SeriesCode $ CountryName $ CountryCode $

YR1998 $ YR1999 $ YR2000 $

YR2001 $ YR2002 $ YR2003 $ YR2004 $ YR2005 $ YR2006 $

YR2007 $ YR2008 $ YR2009 $ YR2010 $ YR2011 $ YR2012 $

YR2013 $ YR2014 $ YR2015 $ YR2016 $ YR2017 $ ;

**run**;

/

/\* First: get the YR variables in a column with year and a column with

variable value \*/

proc sort data=WB;

by CountryName SeriesName SeriesCode;

**run**;

**proc** **transpose** data=WB out=WBlong let;

by CountryName SeriesName SeriesCode;

var YR1998-YR2017;

**run**;

**proc** **freq** data=WB;

table CountryName;

**run**;

**proc** **contents** data=WBlong; \* check the contents of the transposed data;

**run**;

**proc** **freq** data=WBlong;

table \_NAME\_ CountryName; \* check for valid entries;

**run**;

**proc** **print** data=WBlong; \* 51 rows contain missing CountryNames;

where CountryName="";

**run**;

**data** WBlong2; \* remove rows with missing CountryName;

set WBlong;

if (CountryName = "") then delete;

keep CountryName SeriesName SeriesCode Col1 \_NAME\_;

**run**;

ods rtf file="&dir\&subdir\ch4-fig4.6.rtf"

image\_dpi=**300**

style=sasuser.customSapphire;

**proc** **print** data=WBlong2 (firstobs=**10** obs=**15**);

**run**;

ods rtf close;

/\* Second: extract the year (e.g. 1960) from the character value (e.g. YR1960) and make it numeric \*/

**data** WBlong3;

set WBlong2;

year = **1.**\*substr(\_NAME\_,**3**,**4**); \* extract year & makes this variable numeric;

SeriesCode = tranwrd(SeriesCode,".","\_"); \* replace . by \_;

drop \_NAME\_;

**run**;

**proc** **print** data=WBlong3 (firstobs=**10** obs=**15**);

**run**;

The TRANSPOSE procedure is used to move the YR1998-YR2017 variable/column names into a new column with SAS default name \_NAME\_ and the values for each of the YRxxxx variables now stored as a variable named COL1. Table 4.17 shows a few lines from the resulting data set.

Table 4.17 Result of transposing year data stored in different columns to different rows of a single column

| **Obs** | **CountryName** | **SeriesName** | **SeriesCode** | **\_NAME\_** | **COL1** |
| --- | --- | --- | --- | --- | --- |
| **10** | Afghanistan | Age at first marriage, female | SP.DYN.SMAM.FE | YR2007 | .. |
| **11** | Afghanistan | Age at first marriage, female | SP.DYN.SMAM.FE | YR2008 | 15 |
| **12** | Afghanistan | Age at first marriage, female | SP.DYN.SMAM.FE | YR2009 | .. |
| **13** | Afghanistan | Age at first marriage, female | SP.DYN.SMAM.FE | YR2010 | 21.5 |
| **14** | Afghanistan | Age at first marriage, female | SP.DYN.SMAM.FE | YR2011 | 21.2 |
| **15** | Afghanistan | Age at first marriage, female | SP.DYN.SMAM.FE | YR2012 | .. |

Program 4.7 then extracts the xxxx from YRxxxx using the SUBSTR function and multiplies the result by one. Since SUBSTR(“YR2010”,3,4) extracts the character value “2010”, 1.\* SUBSTR(“YR2010”,3,4) coerces this from character to numeric, i.e. from “2010” to 2010. Values of the variable SeriesCode are then modified to replace all “.” (periods) with “\_” (underscores). The results of this is saved in the dataset WBlong3, a portion of which is displayed in Table 4.18.

Table 4.18 Transformed year and modified SeriesCode data in WBlong3

| **Obs** | **CountryName** | **SeriesName** | **SeriesCode** | **COL1** | **year** |
| --- | --- | --- | --- | --- | --- |
| **10** | Afghanistan | Age at first marriage, female | SP\_DYN\_SMAM\_FE | .. | 2007 |
| **11** | Afghanistan | Age at first marriage, female | SP\_DYN\_SMAM\_FE | 15 | 2008 |
| **12** | Afghanistan | Age at first marriage, female | SP\_DYN\_SMAM\_FE | .. | 2009 |
| **13** | Afghanistan | Age at first marriage, female | SP\_DYN\_SMAM\_FE | 21.5 | 2010 |
| **14** | Afghanistan | Age at first marriage, female | SP\_DYN\_SMAM\_FE | 21.2 | 2011 |
| **15** | Afghanistan | Age at first marriage, female | SP\_DYN\_SMAM\_FE | .. | 2012 |

Each row of the WBlong3 data set corresponds to the value of a particular variable for a country in a particular year. For example, 21.2 (observation 14 from Table 4.7) is the “Age at first marriage, female” in Afghanistan in 2011. To answer the question has age at first marriage changed over time, we need to have the age at first marriage as a column in an analysis data set. Program 4.16 completes this task.

Program 4.16 Completing the reshaping of the World Bank data

/\* Third: move the variableName column to separate distinct columns

label the variable names with the SeriesCode \*/

**data** WB;

infile "&dir\&subdir\World-Bank-HNP-05apr18.csv" dsd firstobs=**2**;

length SeriesName $ **80** SeriesCode $ **35** CountryName $ **40** ;

input SeriesName $ SeriesCode $ CountryName $ CountryCode $

YR1998 $ YR1999 $ YR2000 $

YR2001 $ YR2002 $ YR2003 $ YR2004 $ YR2005 $ YR2006 $

YR2007 $ YR2008 $ YR2009 $ YR2010 $ YR2011 $ YR2012 $

YR2013 $ YR2014 $ YR2015 $ YR2016 $ YR2017 $ ;

**run**;

/\* First: get the YR variables in a column with year and a column with

variable value \*/

**proc** **sort** data=WB;

by CountryName SeriesName SeriesCode;

**run**;

**proc** **transpose** data=WB out=WBlong let;

by CountryName SeriesName SeriesCode;

var YR1998-YR2017;

**run**;

**data** WBlong2; \* remove rows with missing CountryName;

set WBlong;

if (CountryName = "") then delete;

keep CountryName SeriesName SeriesCode COL1 \_NAME\_;

**run**;

/\* Second: extract the year (e.g. 1960) from the character value (e.g. YR1960) and make it numeric \*/

**data** WBlong3;

set WBlong2;

year = **1.**\*substr(\_NAME\_,**3**,**4**); \* extract year & makes this variable numeric;

SeriesCode = tranwrd(SeriesCode,".","\_"); \* replace . by \_;

drop \_NAME\_;

VALUE = **1.**\*COL1; \* make sure values are numeric;

**run**;

**proc** **contents** data=WBlong3;

**run**;

/\* Third: move the variableName column to separate distinct columns

label the variable names with the SeriesCode \*/

**proc** **sort** data=WBlong3;

by CountryName year;

**run**;

**proc** **transpose** data=WBlong3 out=WBfinal;

by CountryName Year;

var VALUE;

id SeriesCode;

idlabel SeriesCode;

**run**;

**proc** **contents** data=WBfinal;

**run**;

**proc** **print** data=WBfinal (firstobs=**10** obs=**15**);

**run**;

**proc** **sgplot** data=WBfinal;

title "Plot of Age at first marriage for females vs. time";

series x=year y=SP\_DYN\_SMAM\_FE / group=CountryName;

where SP\_DYN\_SMAM\_FE NE **.**;

**run**;

**proc** **sgplot** data=WBfinal;

title "Plot of Life Expectancy at birth vs. time";

series x=year y=SP\_DYN\_LE00\_FE\_IN / group=CountryName;

**run**;

This code produces a data set (see Table 4.16 for the 5 observations from this dataset) and Figures 4.1 and 4.2.

Table 4.16 Display of final World Bank Analysis data set (subset of observations and variables) and plots of variables over time

| **Obs** | **CountryName** | **year** | **\_NAME\_** | **SP\_DYN\_SMAM\_FE** |
| --- | --- | --- | --- | --- |
| **10** | Afghanistan | 2007 | VALUE | . |
| **11** | Afghanistan | 2008 | VALUE | 15.0 |
| **12** | Afghanistan | 2009 | VALUE | . |
| **13** | Afghanistan | 2010 | VALUE | 21.5 |
| **14** | Afghanistan | 2011 | VALUE | 21.2 |
| **15** | Afghanistan | 2012 | VALUE | . |

| **Obs** | **SP\_DYN\_SMAM\_MA** | **SP\_DYN\_CBRT\_IN** | **SH\_DTH\_INJR\_ZS** |
| --- | --- | --- | --- |
| **10** | . | 42.860 | . |
| **11** | 25.3 | 41.697 | . |
| **12** | . | 40.474 | . |
| **13** | . | 39.232 | 15.8 |
| **14** | . | 38.016 | . |
| **15** | . | 36.863 | . |

…

| **Obs** | **SL\_UEM\_TOTL\_MA\_ZS** | **SL\_UEM\_TOTL\_ZS** | **SP\_URB\_TOTL\_IN\_ZS** |
| --- | --- | --- | --- |
| **10** | 7.50000 | 8.30000 | 23.587 |
| **11** | 7.40000 | 8.20000 | 23.946 |
| **12** | 7.40000 | 8.20000 | 24.313 |
| **13** | 7.30000 | 8.10000 | 24.689 |
| **14** | 7.40000 | 8.20000 | 25.074 |
| **15** | 7.20000 | 8.00000 | 25.468 |

| **Obs** | **SP\_DYN\_WFRT** |
| --- | --- |
| **10** | . |
| **11** | . |
| **12** | . |

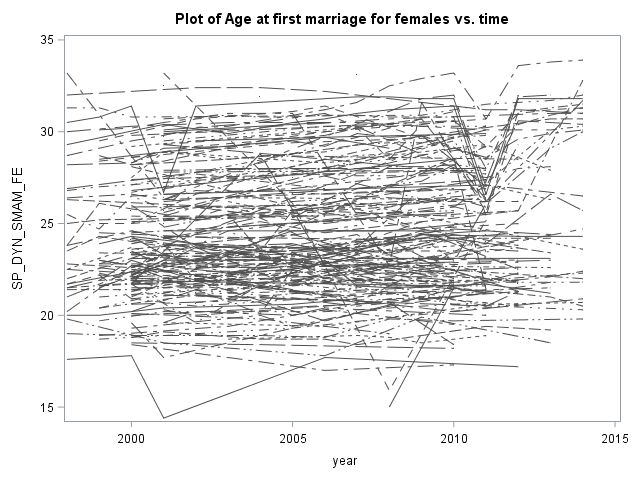


Figure 4.1: Plot of Age at first marriage for females versus time with different line segments for each country



Figure 4.2: Plot of life expectancy versus time with different line segments for each country

You now have a viable data set for exploring variable trends over time and comparing different countries. For example, what is the country with the youngest age at first marriage? What might explain countries with a decrease in life expectancy in the early 2000s?

The data set can be cleaned up more. You can provide more meaningful variable names and assign descriptive labels. Details about the character functions (SUBSTR, TRANWRD) will be provided in later chapters as will discussions of graphical displays of data.

# 4.7 Building training and validation data sets

The final section in this chapter focuses on a common task that may be part of a data analysis: partitioning a data set into a component used to fit / train models and another component used to test or validate the models built on the training data set. A model might capture the nuance of a particular data set at the cost of generalizing to other data sets. This build on a training data set and evaluation on a test data set is one strategy to help select a model that performs well outside of the data used to fit the model.

We explore this with an example investigating which of 3 models might be best for predicting highway miles/gallon based on car weight. The 3 competing models are linear models that include polynomial terms; here, linear, quadratic and cubic models are compared. One option when selecting among competing models is to see which model has the best value based on some criteria such as the mean-squared error (MSE) or coefficient of determination (R2) or some likelihood-based criterion. One potential problem with this strategy is that you may overfit features of the data and the selected model may not perform as well with new data, i.e. data not used to develop the model fit. One new-data-prediction criterion for selecting models is to compare the observed response and the model predicted response for a separate data set that was not used to develop the model. The sum of squares of the prediction error (SSPE) can be used to compare models. A new data set can be collected and then used for SSPE construction. Alternatively, a data set can be divided into a training set and a test set where models are built with the training set data and SSPE is constructed with test data.

In Program 4.17, observations are assigned to be in the test or train set using DATA step programming. An example data set from the SASHELP library, SASHELP.CARS, is used for this example. Next, a new variable (Y\_MPG\_Highway) is assigned for the response variable (MPG\_Highway) where this new variable is missing for the test data. This allows for models to be fit using only the training data but will generate predictions for all observations in the test and training data sets. The linear, quadratic and cubic models are then fit, and the output data sets containing their predictions are merged, and the squared prediction error is produced for each observation. Finally, the SSPE is calculated for the test data set along with the SSE for the training data set.

Program 4.17 Selecting a subset of a data set to train a model

\* select 75%/25% of the observations for a training/test set;

\* using RETAIN allows control of exactly how many observations

are assigned to training and test data sets;

**data** train\_test;

set SASHELP.CARS;

call streaminit(**7525**);

retain ntest ntrain **0**;

pick\_test = RAND("uniform");

if (pick\_test <= **.25**) then do;

if (ntest < **107**) then do;

ntest = ntest + **1**;

dsn= "test ";

end;

else do; \* test sample filled - put obs into training set;

ntrain = ntrain + **1**;

dsn = "train";

end;

end;

else do;

if (ntrain < **321**) then do;

ntrain = ntrain + **1**;

dsn = "train";

end;

else do; \* training sample filled - put obs into test set;

ntest = ntest + **1**;

dsn= "test ";

end;

end;

**run**;

\* define new variable to be Y for the training data but

missing for the test data - produces prediction for both

data sets - also construct the quadratic and cubic terms;

**data** reg\_train;

set train\_test;

obs\_no = \_N\_;

if dsn="train" then Y\_MPG\_Highway = MPG\_Highway;

else Y\_MPG\_Highway = **.**;

Weight2 = Weight\*\***2**;

Weight3 = Weight\*\***3**;

**run**;

\* debugging code to check variable and data set construction;

**proc** **print** data=reg\_train (obs=**4**);

**run**;

**proc** **freq** data=reg\_train;

table dsn;

**run**;

\* fit linear, quadratic and cubic models to the training data set;

**proc** **reg** data=reg\_train;

M1: model Y\_MPG\_Highway = Weight;

output out=pred\_mpg1 pred=yhat1;

M2: model Y\_MPG\_Highway = Weight Weight2;

output out=pred\_mpg2 pred=yhat2;

M3: model Y\_MPG\_Highway = Weight Weight2 Weight3;

output out=pred\_mpg3 pred=yhat3;

**quit**;

/\* combine 3 data sets with predicted values and

calculate (y-yhat)^2 for each observation \*/

**data** all\_fit;

merge pred\_mpg1 pred\_mpg2 pred\_mpg3;

by obs\_no;

lin\_pred\_SS = (MPG\_Highway - yhat1)\*\***2**;

quad\_pred\_SS = (MPG\_Highway - yhat2)\*\***2**;

cubic\_pred\_SS = (MPG\_Highway - yhat3)\*\***2**;

**run**;

**proc** **print** data=all\_fit (obs=**4**);

where dsn="test ";

**run**;

\* generate table with SS of prediction error for the test data

(and SSE for the training data set);

**proc** **means** data=all\_fit n sum;

class dsn;

var lin\_pred\_SS quad\_pred\_SS cubic\_pred\_SS;

**run**;

\* compare the 3 fits on the test data set;

**proc** **sort** data=all\_fit;

by weight;

**run**;

**proc** **sgplot** data=all\_fit;

scatter x=Weight y=MPG\_Highway / markerattrs = (COLOR = GRAY4F);

series x=Weight y=yhat1 / lineattrs=(color=lightgrey pattern=**1**);

series x=Weight y=yhat2 / lineattrs=(color=black pattern=**2**);

series x=Weight y=yhat3 / lineattrs=(color=grey pattern=**4**);

where dsn="test";

**run**;

WORTH NOTING: Most of the modeling procedures in SAS use a MODEL statement to specify the response variable and predictor variables. The form of the statement is:

Model response\_variable = predictor\_variable\_1 … predictor\_variable\_k;

(Also) Worth Noting: The OUTPUT statement with PROC REG contains the prediction of the variable Y\_MPG\_Highway for all observations including those observations not used to develop parameter estimates, i.e. the test set data.

Summaries from the model are presented in Table 4.17. The first 4 columns are values that compare the linear, quadratic and cubic models using training data only. The fifth column, SSPE, displays model predictive performance on the test data set

Table 4.17 Comparing polynomial models for predicting Highway MPG from weight for the SASHELP.CARS data set

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | R2 | Adj. R2 | MSE | SSE | SSPE(Test) |
| Linear | 62.08% | 61.96% | 13.17 | 4200.25 | 1070.40 |
| Quadratic | 66.75% | 66.54% | 11.58 | 3683.75 | 1044.29 |
| Cubic | 68.15% | 67.85% | 11.13 | 3528.10 | 1057.30 |

From this table, the cubic model has the best performance on all indices, i.e. largest R2 and Adj. R2 and smallest MSE and SSE. This is not the case when you examine the SSPE for the test data. Here, the quadratic model has the smallest SSPE among the three polynomial models. Thus, one might argue that the added curvature introduced by the cubic model may be overfitting to the training data. Figure 4.3 displays the plot of the Highway MPG vs. Weight for the 3 model fits in the test data doesn’t suggest a dramatic difference between the cubic and cubic fits.

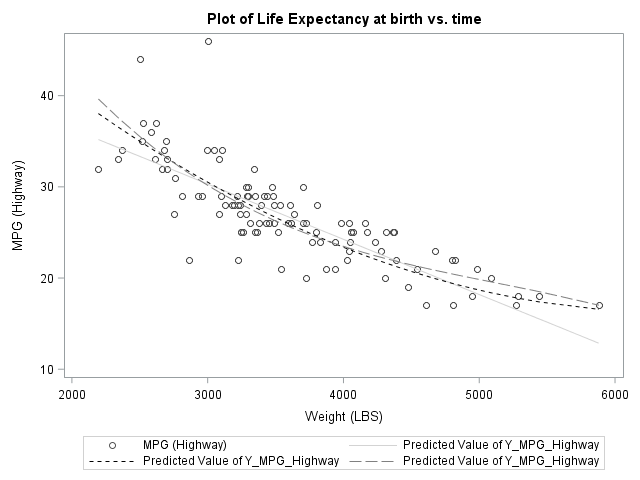


Figure 4.3: Plot of the Highway MPG vs. Weight with superimposed linear, quadratic and cubic fits for the SASHELP.CARS data test set

4.7 References

Bailer, A. John, and James T. Oris. 1994. “Assessing toxicity pollutants in aquatic systems.” *Case Studies in Biometry*. N. Lange, L. Ryan, L. Billard, D. Brillinger, L. Conquest, and J. Greenhouse, eds. New York, NY: John Wiley & Sons, Inc. [SHERRY - referenced previously – not needed here?]

4.8 Exercises

1. Use DATA step programming and PROC SQL programming to enumerate the sample space of rolling 2 fair two-sided die. Next, calculate the sum of the pips showing on the two rolled dice. Calculate the probability that the sum of the faces exceeds 8.
2. Fifty animals were exposed to one of five concentration levels of nitrofen (10 animals per group, but some observations might be missing in the data). The data was recorded separately for three broods produced by each of the 50 animals. Thus, each animal can have data in each of the three brood data sets. A particular animal is uniquely identified by the ID variable. Produce a combined data set containing observations (animals and IDs) that has data on all three broods. In addition, construct an additional variable for the total number of young produced in all three broods. Compare the different joins for this data. In particular, construct the following:

Brood1 [inner join] Brood2 [inner join] Brood3

(Brood1 [left join] Brood2) [left join] Brood3

(Brood1 [right join] Brood2) [right join] Brood3

Brood1 [full join] Brood2 [full join] Brood3

Print the results for the different combined data sets. Multiple observations are given on each line.

data B1; \* Brood=1 data;

input ID conc number of young @@;

datalines;

3 0 6 4 0 6 5 0 6 6 0 5 7 0 6 8 0 5 9 0 3 10 0 6

12 80 5 13 80 6 14 80 5 15 80 8 16 80 3 17 80 5 18 80 7

19 80 5 20 80 3

21 160 6 22 160 6 23 160 2 24 160 6 25 160 6 26 160 6

27 160 6 28 160 5 30 160 6

31 235 4 32 235 6 34 235 6 35 235 6 36 235 6 37 235 7

38 235 4 39 235 6 40 235 7

41 310 6 42 310 6 43 310 7 44 310 0 45 310 5 47 310 6

48 310 4 49 310 6 50 310 5

;

run;

data B2; \* Brood=2 data;

input ID conc number of young @@;

datalines;

1 0 14 2 0 12 3 0 11 4 0 12 6 0 14 7 0 12 8 0 13

9 0 10 10 0 11

11 80 11 13 80 11 14 80 12 15 80 13 16 80 9 17 80 9

18 80 12 19 80 13 20 80 12

21 160 12 22 160 12 23 160 8 24 160 10 25 160 11

26 160 13 27 160 12 29 160 13 30 160 12

31 235 13 32 235 10 33 235 5 34 235 0 35 235 13

36 235 0 37 235 0 38 235 2 39 235 8 40 235 0

41 310 0 42 310 0 43 310 0 45 310 10 46 310 0 47 310 0

48 310 0 49 310 0 50 310 0

;

run;

data B3; \* Brood=3 data;

input ID conc number of young @@;

datalines;

1 0 10 2 0 15 3 0 17 4 0 15 5 0 15 6 0 15 7 0 15

8 0 12 10 0 14

11 80 16 12 80 16 13 80 18 14 80 16 15 80 15 16 80 14

17 80 13 18 80 12 19 80 14 20 80 14

21 160 11 22 160 11 23 160 13 24 160 11 25 160 13 26 160 12

27 160 12 28 160 11 29 160 10 30 160 11

31 235 6 32 235 5 33 235 0 34 235 6 35 235 8 36 235 10

38 235 9 39 235 7 40 235 10

41 310 0 42 310 0 43 310 0 44 310 0 45 310 0 46 310 0

48 310 0 49 310 0 50 310 0

;

run;

1. Repeat the training-test comparison in Section 4.6 using different seeds, say 45056 and 45065. Does the quadratic fit always have the best SSPE among the three models? What do you conclude from this analysis?
2. The World Bank data set (WBfinal) in section 4.2 had horrible variable names. Modify this program to make the variable names more human friendly. Add labels to variables if this would help produce clearer output.
3. Modify the program in Section 4.6 to determine the number of observations in the SASHELP.CARS data set and then calculate the number of observations needed in a 75% training set and a 25% test set.

4.9 Self-study lab

[SHERRY – This section could be only made available online if printed text running long]

The following code examples provide a collection of more advanced PUT, SET, MERGE, and SQL operations. You are encouraged to work through each DATA step and PROC SQL query in sequence to develop more insight into DATA step manipulation and data set construction.

/\* ==================================================== \*/

/\* PUT examples ... \*/

/\* ==================================================== \*/

data;

put "Hello World!!!";

put @20 "Hello World!!!"; \* start at column 20;

put 3\*"Hello World!!!"; \* 3 copies;

run;

data;

put;

put "Hello World!!!" /;

put @20 "Hello World!!!" /; \* start at column 20;

put 3\*"Hello World!!!" /; \* 3 copies;

put;

run;

data;

input name $ @@;

put "Hello " name ", welcome to SAS Statistical Programming." /;

datalines;

Dave Hal

;

run;

data;

file " C:\Users\baileraj.IT\Desktop\put-example.TXT";

\* replace path in FILE with folder on your system;

input name $ @@;

put "Hello " name ", welcome to SAS Statistical Programming." /;

datalines;

Dave Hal

;

run;

/\* ==================================================== \*/

/\* CONCATENATING ("row binding") data sets \*/

/\* ==================================================== \*/

/\* clean example \*/

data d1;

input v1 v2 v3;

datalines;

1 2 3

4 5 6

;

run;

data d2;

input v1 v2 v3;

datalines;

11 12 13

14 15 16

17 18 19

;

run;

data d12;

set d1 d2;

run;

proc print;

run;

/\* not-so-clean example \*/

data d1a;

input v1 v2 v3;

datalines;

1 2 3

4 5 6

;

run;

data d2a;

input var1 var2 var3;

datalines;

11 12 13

14 15 16

17 18 19

;

run;

data d12a;

set d1a d2a;

run;

proc print data=d12a;

run;

/\* fixing not-so-clean example \*/

data d1b;

input v1 v2 v3;

datalines;

1 2 3

4 5 6

;

data d2b;

input var1 var2 var3;

v1=var1;

v2=var2;

v3=var3;

drop var1-var3;

datalines;

11 12 13

14 15 16

17 18 19

;

data d12b;

set d1b d2b;

run;

proc print data=d12b;

run;

/\* one last concatenation example \*/

options formdlim="-";

data d1c;

input v1 v2 v3;

datalines;

1 2 3

4 5 6

;

run;

data d2c;

input var1 var2 var3;

datalines;

11 12 13

14 15 16

17 18 19

;

run;

data d12c;

set d1c d2c (rename=(var1=v1 var2=v2 var3=v3));

run;

proc print data=d12c;

run;

/\* merging data examples \*/

options formdlim="-";

data m1;

input ID v1 v2;

datalines;

1 2 3

2 5 6

4 7 8

;

run;

data m2;

input ID var1 var2 var3;

datalines;

1 11 12 13

2 14 15 16

3 17 18 19

;

run;

data M12;

merge m1 m2;

by ID;

run;

proc print data=m12;

title "First Merge data example";

run;

/\* suppose you have common variables in the data sets that are merged? \*/

options formdlim="-";

data m1b;

input ID v1 v2;

datalines;

1 2 3

2 5 6

4 7 8

;

run;

data m2b;

input ID v1 v2 var3;

datalines;

1 11 12 13

2 14 15 16

3 17 18 19

;

data M12b;

merge m1b m2b;

by ID;

run;

proc print data=m12b;

title "Second Merge data example - common variables in 2 data sets";

run;

data M12b2;

merge m2b m1b;

by ID;

run;

proc print data=m12b2;

title "Second Merge data example - diff merge order";

run;

proc print data=m1b;

run;

proc print data=m2b;

run;

/\* what if the data sets have multiple records with an ID? \*/

data m1c;

input ID v1 v2;

datalines;

1 2 3

1 4 5

2 6 7

;

run;

data m2c;

input ID var1;

datalines;

1 11

2 21

2 22

;

run;

data m12c;

merge m1c m2c;

by ID;

proc print data=m12c;

title "many to one merging issues";

run;

data m1d;

input ID v1 v2;

datalines;

1 2 3

1 4 5

;

run;

data m2d;

input ID var1;

datalines;

1 11

1 12

1 13

;

run;

data m12d;

merge m1d m2d;

by ID;

run;

proc print data=m12d;

title "many to many merging issues";

run;

/\* ================================================================ \*/

/\* BASIC SQL stuff >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> \*/

/\* ================================================================ \*/

options formdlim="-";

data junk;

input cgroup $ x y @@;

datalines;

a 1 2 B 3 4 B 5 6 A 7 8 A 9 10

;

run;

/\* basic PROC stuff \*/

proc print data=junk;

run;

proc sort data=junk;

by cgroup;

run;

proc print data=junk;

run;

proc means data=junk;

class cgroup;

var x y;

run;

/\* notes:

1. Multiple select statements can be used to generate different views of the data

2. SQL continues until QUIT; or a DATA/PROC step

\*/

proc sql;

select \*

from junk; \* select and display all variables;

select cgroup

from junk; \* select particular variable;

select cgroup,x,y

from junk

order by cgroup; \* order the rows of the view table;

select cgroup,x,y

from junk

order by x;

select cgroup,x,y, x/(x+y)\*100 as pctsum

from junk; \* construct and name a new variable/column;

select cgroup,x,y, x/(x+y)\*100 as pctsum

label='% sum' format=4.1

from junk;

select avg(x) label='avg x',avg(y) label='avg y'

from junk; \* summary functions of SQL;

select cgroup label='Variable', count(\*) label='n',

avg(x) label='avg x',avg(y) label='avg y'

from junk

group by cgroup; \* grouping rows/summaries in SQL;

select cgroup label='Variable', count(\*) label='n',

avg(x) label='avg x',avg(y) label='avg y'

from junk

where cgroup in ('A','B')

group by cgroup ; \* subsetting rows in SQL;

select cgroup label='Variable', count(\*) label='n',

avg(x) label='avg x',avg(y) label='avg y'

from junk

where cgroup in ('A','B')

group by cgroup

having avg(x) > 5;

/\* to check a query before running it \*/

/\*

validate

select . . ..

\*/

/\* ================================================================ \*/

/\* SQL: CONCATENATE examples >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> \*/

/\* ================================================================ \*/

/\* clean example \*/

data d1;

input v1 v2 v3;

datalines;

1 2 3

4 5 6

;

run;

data d2;

input v1 v2 v3;

datalines;

11 12 13

14 15 16

17 18 19

;

run;

proc sql;

select \* from d1;

select \* from d2;

select \* from d1,d2;

select \*

from d1

outer union

select \*

from d2;

\* SET operators in SQL - outer union, union, except, intersect;

select \*

from d1

outer union corr

select \*

from d2; \* CORR overlays common columns;

select \*

from d1

union

select \*

from d2; \* even simpler;

create table work.d12 as

select \*

from d1

outer union corr

select \*

from d2; \* produce a SAS data set from an SQL query;

/\* not-so-clean example \*/

data d1a;

input v1 v2 v3;

datalines;

1 2 3

4 5 6

;

run;

data d2a;

input var1 var2 var3;

datalines;

11 12 13

14 15 16

17 18 19

;

run;

proc sql;

select \* from d1a,d2a;

select \*

from d1a

outer union

select \*

from d2a;

select \*

from d1a

union

select \*

from d2a;

select \*

from d1a

outer union corr

select var1 as v1, var2 as v2, var3 as v3

from d2a; \* renaming as part of selection;

/\* ================================================================ \*/

/\* merging data examples >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> \*/

/\* ================================================================ \*/

options formdlim="-";

data m1;

input ID v1 v2;

datalines;

1 2 3

2 5 6

4 7 8

;

run;

data m2;

input ID var1 var2 var3;

datalines;

1 11 12 13

2 14 15 16

3 17 18 19

;

run;

proc sql;

select \*

from m1,m2;

select \*

from m1,m2

where m1.id=m2.id;

select m1.ID,v1,v2,var1,var2,var3

from m1,m2

where m1.id=m2.id;

select \*

from m1 inner join m2

on m1.id=m2.id;

select \*

from m1 right join m2

on m1.id=m2.id;

select \*

from m1 left join m2

on m1.id=m2.id;

select \*

from m1 full join m2

on m1.id=m2.id;

/\* multiple records per id \*/

data m1c;

input ID v1 v2;

datalines;

1 2 3

1 4 5

2 6 7

;

run;

data m2c;

input ID var1;

datalines;

1 11

2 21

2 22

;

run;

proc sql;

select \*

from m1c,m2c;

select \*

from m1c,m2c

where m1c.id=m2c.id;

select m1c.ID,v1,v2,var1,var2,var3

from m1c,m2c

where m1c.id=m2c.id;

select \*

from m1c inner join m2c

on m1c.id=m2c.id;

select \* from m1c;

select \* from m2c;

/\* many to many merging issues \*/

data m1d;

input ID v1 v2;

datalines;

1 2 3

1 4 5

;

run;

data m2d;

input ID var1;

datalines;

1 11

1 12

1 13

;

run;

proc sql;

select \* from m1d;

select \* from m2d;

select \*

from m1d, m2d

where m1d.id=m2d.id;

quit;

1. An outer join is the contents of an inner join (the intersection of two tables) plus additional observations from the tables in the join. [↑](#footnote-ref-1)