**STA 2183W**

**Chapter 3 – Data Management in SAS & R**

We spent the first two chapters on reviewing the fundamentals of descriptive and inferential statistics. Aside from the coding aspect, much that we covered was material you probably covered in introductory statistics. This chapter will now delve deeper into various data management and manipulation techniques that you can use in order to clean your data up and perform the statistical tests we will use for the rest of the course. While there are eight sections in this chapter, half (3.1-3.4) are SAS-only section and the other half (3.5-3.8) cover the same topics, but in R.

**Part A: Data Management Techniques in SAS**

The first half of this chapter will be dedicated to implement various data management techniques using SAS.

**3.1 Creating and Modifying Variables in SAS**

In this section, we will explore one of the most important features you can do with datasets with SAS: create and modify variables within a dataset.

|  |  |
| --- | --- |
| Consider a shortened version of the dataset ***Auto***, which we will call it ***auto\_short***. We will introduce some concepts through this mini file.  Note that often, variables will be presented in ways that require us to *modify* them in order to provide a more appropriate analysis.  For example, say we wanted to create ***cost*** so that it gives us the price in thousands of dollars. Furthermore, we want ***mpgpd***, which will stand for miles per gallon per thousand dollars.  In each case, just type  ***[variable name] =[expression for the value];*** as seen on the right. | **DATA** auto\_short;  INPUT make $ price mpg rep78 foreign;  cost = ROUND( price / **1000** );  mpgptd = mpg / price;  DATALINES;  AMC 4099 22 3 0  AMC 4749 17 3 0  AMC 3799 22 3 0  Audi 9690 17 5 1  Audi 6295 23 3 1  BMW 9735 25 4 1  Buick 4816 20 3 0  Buick 7827 15 4 0  Buick 5788 18 3 0  Buick 4453 26 3 0  Buick 5189 20 3 0  Buick 10372 16 3 0  Buick 4082 19 3 0  Cad. 11385 14 3 0  Cad. 14500 14 2 0  Cad. 15906 21 3 0  Chev. 3299 29 3 0  Chev. 5705 16 4 0  Chev. 4504 22 3 0  Chev. 5104 22 2 0  Chev. 3667 24 2 0  Chev. 3955 19 3 0  Datsun 6229 23 4 1  Datsun 4589 35 5 1  Datsun 5079 24 4 1  Datsun 8129 21 4 1  ;  **RUN**; |

If we wish to format or change our data, we can do so either by replacing the dataset or creating a new one using the SET command.

Using SAS to Create/Modify Variables within a Dataset:

**DATA** **NEW\_DATASET**;

SET **REFERENCED\_OLD\_DATASET**;

**[INSERT COMMANDS]**

**RUN**;

[If we wish to replace a dataset with what we are using, we will need to include the REPLACE command.]

For example, if we wish to *create* a new variable within the dataset, then we may need to use specific formulas based on a given *condition*. To do so, we will rely upon the **IF - THEN statement**.

General Form: IF *condition* THEN *action* ;

* If the condition is *true*, then SAS does the action. (The action is usually an assignment statement.)

For example, the variable ***rep78*** is coded 1 through 5, standing for poor, fair, average, good and excellent.

We would like to change ***rep78*** so that it has only three values, 1 through 3, standing for below average, average, and above average.

* We will do this by creating a new variable called ***repair*** and recoding the values of ***rep78*** into it.

We will also create a new variable called ***highmpg*** that is a dummy coding of ***mpg***.

Vehicles with better than 20 mpg will be coded **1** and those with 20 or less will be coded **0**.

**Example 3.1 –** Using SAS Code to Create New Variables:

**DATA** auto2;

SET auto\_short;

repair = **.**;

IF (rep78=**1**) or (rep78=**2**) THEN repair = **1**;

IF (rep78=**3**) THEN repair = **2**;

IF (rep78=**4**) or (rep78=**5**) THEN repair = **3**;

highmpg = **.**;

IF (mpg <= **20**) THEN highmpg = **0**;

IF (mpg > **20**) THEN highmpg = **1**;

**RUN**;

Note that when we create new variables, we usually do so because other variables are not useful in our data set. We can use the KEEP or DROP commands create a new dataset.

**Example 3.2 –** Using SAS Code to KEEP or DROP Variables:

**DATA** auto3;

SET auto2;

KEEP make mpg price;

**RUN**;

**PROC** **PRINT** data=auto3;

**RUN**;

**DATA** auto4;

SET auto2;

DROP rep78 hdroom trunk weight length turn displ gratio foreign;

**RUN**;

**PROC** **PRINT** data=auto4;

**RUN**;

**3.2 Labelling, Formatting, and Sorting Variables in SAS**

When coding our datasets, we rarely use the full variable name or the name of the responses. Instead, we often use *variable and response codes*, which are shorter and easier to code. However, when we wish to print graphics or run analyses, we often would prefer the full description – thus, it is useful to label and format the values. We can use the command LABEL to accurately describe what the variable code actually stands for.

**Example 3.3 –** Using SAS Code to Provide Labels to Variable Names:

**DATA** auto2;

SET auto\_short;

LABEL rep78 ="1978 Repair Record"

mpg ="Miles Per Gallon"

foreign="Where Car Was Made";

**RUN**;

**PROC** **CONTENTS** DATA=auto2;

**RUN**;

**PROC** **MEANS** DATA=auto2;

**RUN**;

Furthermore, we can *format* the variable responses by using a VALUE statement, and then attach the FORMAT statement to the variable. This procedure is useful for us to *order* qualitative variables using a *numerical* *output*, and formatting the responses so that when you print them, the output will read the appropriate qualitative responses.

**Example 3.4 –** Using SAS Code to Format Variable Names:  
**PROC** **FORMAT**;

VALUE for **0**="domestic" **1**="foreign" ;

VALUE $mak "AMC" ="American Motors"

"Buick" ="Buick (GM)"

"Cad." ="Cadillac (GM)"

"Chev." ="Chevrolet (GM)"

"Datsun" ="Datsun (Nissan)";

**RUN**;

**PROC** **FREQ** DATA=auto2;

FORMAT foreign for. make $mak.;

TABLES foreign make;

**RUN;**

Our next technique we’ll cover is to ***sort*** our data by specific categories. Such ordering can help the statistician see the split in the data and detect patterns.

**Example 3.5 –** Using SAS Code to Sort Observations:

**PROC** **SORT** DATA=auto2 OUT=autosort;

BY foreign;

**RUN**;

**PROC** **PRINT** DATA=autosort;

**RUN**;

If we wish to change the *order* of the sorting procedure, we can do so by using the BY DESCENDING procedure.

**Example 3.6 –** Using SAS Code to Sort Observations in Descending Order:

**PROC** **SORT** DATA=auto2 OUT=autosort2;

BY DESCENDING foreign;

**RUN**;

**PROC** **PRINT** DATA=autosort2;

**RUN**;

Finally, note that we can sort by a number of variables, not just a single one.

**Example –** Using SAS Code to Sort Observations by One Variable, and Then Another:

**PROC** **SORT** DATA=auto2 OUT=autosort3;

BY foreign rep78;

**RUN**;

**PROC** **PRINT** DATA=autosort3;

**RUN**;

**3.3 Missing Data, Irregular Data, and Outlier Removal in SAS**

One problem that can occur with data sets is dealing with *missing data*. We can drop observations with missing data by using the following command:

General Form: IF **[Variable] ^= . ;**

**Example 3.7 –** Using SAS Code to Get Rid of Missing Data:

**DATA** Miss;

INPUT Values @@;

DATALINES;

5 8 8 6 3 2 . 4 6

;

**RUN**;

**DATA** No\_Miss;

SET Miss;

IF Values ^=**.** ;

**RUN**;

Another issue that can arise is attempting to get rid of suspected outliers or incorrect data. Such issues occur when there are clear irregularities in the data, and we wish to *clean* the data for appropriate analysis.

Say for example, we wish to go back to the ***auto2*** dataset, and conclude that values for the variable ***rep78*** should be at most **3**. Then, we need to delete observations that are greater than 3. We can do so with the two following options:

**Example 3.8 –** Using SAS Code to Get Rid of Irregular Data:

**DATA** auto5;

SET auto2;

IF rep78 > **3** THEN DELETE;

**RUN**;

**PROC** **PRINT** data=auto5;

VAR rep78;

**run**;

**Example 3.9 –** Using SAS Code to Get Rid of Irregular Data:

**DATA** auto6;

SET auto2;

IF (rep78 <= **3**);

**RUN**;

**PROC** **PRINT** data=auto6;

VAR rep78;

**run**;

We can even get rid of both missing data and irregular entries simultaneously. We will incorporate the **AND** operator in the IF statement.

**Example 3.10 –** Using SAS Code to Get Rid of both Missing and Irregular Data:

**DATA** auto7;

SET auto2;

IF (rep78 <= **3**) AND (rep78 ^= **.**);

**RUN**;

Potentially, we will need to find *which* observations meet a specific requirement. We will use the WHERE command to achieve this task.

**Example 3.11 –** Using SAS Code to Get Find Specific Observations:

**PROC** **PRINT** DATA=auto;

WHERE (rep78 > **3**);

VAR make rep78;

**RUN**;

Note that we can use this command along with PROC UNIVARIATE to find outliers. Recall that for the ***price*** variable, and. Then, and   
. We can use those boundaries to find the potential outliers below.

**Example 3.12 –** Using SAS Code to Get Find Outliers:

**PROC** **PRINT** data=auto2;

WHERE (price > **2.5**\***8129**-**1.5**\***4453**) OR (price < **2.5**\***4453**-**1.5**\***8129**);

**RUN**;

**3.4 Concatenating, Merging, and Transposing Data in SAS**

We’ll complete our data manipulation techniques in SAS with a lesson on concatenating, merging, and transposing our data.

Concatenating Data Files

The first technique is ***concatenating data*** *files*, which simply means to stack them one of top of the other to make one large data set. This technique allows us to take multiple data sets – including data from various surveyors or from *longitudinal studies* (studies over time) – and analyze them together, namely when they share similar variables.

For example, say have two data sets, “dads” and “moms” that we wish to put together as a data set “parents”.

|  |  |
| --- | --- |
| dads  famid name inc  2 Art 22000  1 Bill 30000  3 Paul 25000 | Moms  famid name inc  1 Bess 15000  3 Pat 50000  2 Amy 18000 |

If we wish to combine the following two data sets by stacking them one on top of the other, we can use the following SAS Code:

**Example 3.13** – Concatenating the Two Data Sets

**DATA** dads;

INPUT famid name $ inc;

DATALINES;

2 Art 22000

1 Bill 30000

3 Paul 25000

;

**RUN**;

**DATA** moms;

INPUT famid name $ inc;

DATALINES;

1 Bess 15000

3 Pat 50000

2 Amy 18000

;

**RUN**;

* We will use the SET command to combine the data sets.

**DATA** parents;

SET dads moms;

**RUN**;

**PROC** **PRINT** DATA=parents;

**RUN**

Now, this concept works well, but there is one slight hiccup – we may wish to differentiate the moms from the dads. What we will need to do is create a new variable that indicates from which data set these observations originate.

**Example 3.14** – Concatenating the Two Data Sets with Indicator Variable

**DATA** dads2;

SET dads;

momdad = "dad";

**RUN**;

**DATA** moms2;

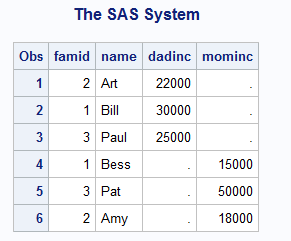
SET moms;

momdad = "mom";

**RUN**;

**DATA** parents2;

SET dads2 moms2;

**RUN**;

**PROC** **PRINT** DATA=parents2;

**RUN**;

Now suppose that income is called **dadinc** and in the dads file and called **mominc** in the moms file, as shown below. Then, there are variables that do no match. When the variables are combined in SAS, PROC PRINT would provide the following read-out to our right.

Thus, we need to derive a way to merge the to variables together to rid ourselves of this divide.

We will first consider data ***dads3*** and ***moms3*** that are identical to ***dads*** and *moms*, but for now, the variable *inc* has been replaced with *dadinc* and *mominc*, respectively

One route is to create a variable called ***inc*** that gives the income, and set our values by using IF statements. Then, upon merging, we will DROP the old variables.

**Example 3.15** – Concatenating the Two Data Sets with a Different Variable Label

**DATA** parents3a;

SET dads3(IN=dad) moms3(IN=mom);

IF dad=**1** THEN

DO;

momdad="dad";

inc=dadinc;

END;

IF mom=**1** THEN

DO;

momdad="mom";

inc=mominc;

END;

DROP dadinc mominc;

**RUN**;

We can also create a variable called ***inc*** that gives the income by using the RENAME option on the SET statement.

**Example 3.16** – Concatenating the Two Data Sets with a Different Variable Label

**DATA** parents3b;

SET dads3(RENAME=(dadinc=inc)) moms3(RENAME=(mominc=inc));

**RUN**;

Match Merging Data Files in SAS

Previously, we combined observations to create one data set, but this procedure does not provide us yet with the ability to combine specified variables.

|  |  |
| --- | --- |
| **DATA** dads;  INPUT famid name $ inc;  DATALINES;  2 Art 22000  1 Bill 30000  3 Paul 25000  ;  **RUN**; | **DATA** faminc;  INPUT famid inc96 inc97 inc98;  DATALINES;  3 75000 76000 77000  1 40000 40500 41000  2 45000 45400 45800  ;  **RUN**; |

We will begin with a **one-to-one** merge, where there is a one-to-one correspondence between the label of interest. Consider the datasets ***dads*** and ***faminc*** (family income).

Here, there is only one value (famid) that is used per family.

We will first create new datasets, modified by using ***SORT*** to order each set of data, then use the ***MERGE*** command to combine the two data sets, linked by the variable ***famid***.

**Example 3.17**– One-to-One Merging of Two Datasets

**PROC** **SORT** DATA=dads OUT=dads2;

BY famid;

**RUN**;

**PROC** **SORT** DATA=faminc OUT=faminc2;

BY famid;

**RUN**;

**DATA** dadfam;

MERGE dads2 faminc2;

BY famid;

**RUN**;

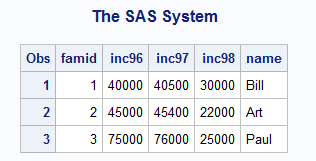
**PROC** **PRINT** DATA=dadfam;

**RUN**;

|  |
| --- |
| **DATA** dadinc;  INPUT famid name $ inc98;  DATALINES;  2 Art 22000  1 Bill 30000  3 Paul 25000  ;  **RUN**; |

Now, consider a case where two data sets, along with sharing the same labelling code, also share another variable. Such a situation could occur in longitudinal studies or other datasets where there is tracking of previous information. When we merge two datasets in one-to-one merging, SAS will overwrite of the two variables and provide a dataset with only one copy.

**Example 3.18** – Merging of Two Datasets

**PROC** **SORT** DATA=dadinc;

BY famid;

**RUN**;

**PROC** **SORT** DATA=faminc;

BY famid;

**RUN**;

**DATA** merge\_sameinc;

MERGE faminc dadinc;

BY famid;

**RUN**;

**PROC** **PRINT** DATA=merge\_sameinc;

**RUN**;

The **MERGE** procedure also works when there isn’t a one-to-one correspondence. When the label of one data set has all unique outputs, while another data set may have more than one observation with the same output, this type of merging is known as **one-to-many**merging.

We’ll again consider the***dads***file, and also incorporate a new file, ***kids***. In this file, there are numerous observations with the same family ID, and so we will merge the two data sets, sorted corresponding to the ***famid*** variable.

**Example 3.19** – One-to-Many Merging of Two Datasets

|  |
| --- |
| **DATA** kids;  INPUT famid kidname $ birth age wt sex $ ;  DATALINES;  2 Beth 1 9 60 f  1 Bob 2 6 40 m  3 Barb 3 3 20 f  2 Andy 1 8 80 m  1 Al 2 6 50 m  2 Ann 3 2 20 f  3 Pete 1 6 60 m  1 Pam 2 4 40 f  3 Phil 3 2 20 m  ;  **RUN**; |

**PROC** **SORT** DATA=kids OUT=kids2;

BY famid;

**DATA** dadkid;

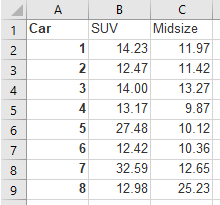
MERGE dads2 kids2;

BY famid;

**RUN**;

**PROC** **PRINT** DATA=dadkid;

**RUN**;

Transposing Data to Split Variables

Sometimes, data files will consists of just lists observations, split by populations. As in the image to our right, each “variable” is a distinct population. We could have received a file such as the one to our right for statistical analysis, or we could have created such a data set by producing a *one-to-one* *merging* of two or more datasets.

To be able to provide sufficient analysis on these populations, we will incorporate the **PROC TRANSPOSE** command, which will allow us combine our populations (and differentiating with a created categorical variables) into a set of the same observation.

The idea of ***transposing*** data has its history that often blends mathematics, statistics, and computer science. What a *transpose* operation does is take a data frame (or matrix) and ***switches the rows and columns***. With a few manipulations and nifty strategy, we can use this command to combine our populations.

|  |
| --- |
| **DATA** Cardamage;  INPUT Car SUV Midsize;  DATALINES;  1 14.23 11.97  2 12.47 11.42  3 14.00 13.27  4 13.17 9.87  5 27.48 10.12  6 12.42 10.36  7 32.59 12.65  8 12.98 25.23  ;  **RUN**; |

**Example 3.20** – Transposing Data to Combine Populations

Consider the dataset ***cardamage*** on our right.

* We will first drop the index value ***car***.

**DATA** cardamage2;

SET cardamage;

DROP Car;

**RUN**;

* Now, we will create a temporary file (tmp) that transposes our data. It will also create a column for the variable names (\_NAME\_), which leads to the *rename* command.   
  We will rename the population variable as *Car\_Type*.
* We will then sort the transposed data by the population indicator, *Car\_Type*.

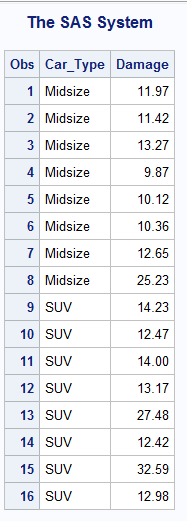
**PROC** **TRANSPOSE** data=cardamage2 out= tmp (rename = (\_NAME\_ = Car\_Type));

**RUN**;

**PROC** **SORT** data = tmp;

BY Car\_Type;

**RUN**;

* With the sorting completed, we can create a result file that transposes the data back in its proper order.
* We will clean our data, dropping the "Name" variable and renaming the observation variable “Damage”.

**PROC** **TRANSPOSE** DATA =tmp OUT=result;

BY Car\_Type;

**RUN**;

**DATA** car\_damage;

SET Result;

DROP \_NAME\_;

RENAME COL1=Damage;

**RUN**;

* If desired, we can also manipulate the data and re-insert the index term ***Car*** back into the data set using the variable creation performed in Part 1.

**Part B: Data Management Techniques in R**

The second half of this chapter will switch over to various data management techniques using R.

**3.5 Creating and Modifying Variables in R**

Obtaining new variables from others can begin as a simple procedure, but adding it into the data set does require some tweaking to get it all to work. For example, say we wish to take the ***population*** variable (which is in thousands) and instead switch it to *hundreds of thousands*. So we need to divide by 100.

**Example** **3.21** – Using R Code to Create a New Variable:  
**carseats <- read.csv("carseats.csv")**

**attach(carseats)**

**PopHT <- Population/100**

Just like that, we have a new variable. The kicker is that we need to *attach* that variable to our dataset. We can do so two different ways. The first is to explicitly define the variable. The other method is through the **cbind** command. The latter option is useful when we have more than one variable to attach (or if we are merging data sets, especially over time).

**Example** **3.22** – Using R Code to Attach a New Variable to a Data Set:

**car2 <- carseats**

**car2$PopHT <- PopHT**

or

**car3 <- cbind(carseats, PopHT)**

To transform variables for various conditions, R has a couple routes. The easiest stem when you have various ranges for the values.

**Example 3.23 –** Using R Code to Create New Variables:

**car4 <-carseats**

**Ageold <-carseats$Age**

**Ageold[carseats$Age <40] <- 0**

**Ageold[carseats$Age >=40] <- 1**

**Ageold**

**car4 <-cbind(car4, Ageold)**

Another route is to incorporate the IF/THEN statements along a ***loop***. Note that instead of using ***THEN***, we will use ***ELSE***. In this situation, we will incorporate a **for loop**, which tells us that, for the values provided, we will perform this operation. (Recall that to get the number of observations in a variable, we can use the ***length*** command.)  
  
**Example 3.24 –** Using R Code to Create New Variables:

**Ageold2 <-carseats$Age**

**for (i in 1:length(Ageold2)){**

**if (Ageold2[i] >=40) {**

**Ageold2[i]=1**

**}**

**else {Ageold2[i]=0 }**

**}**

**car5<-cbind(carseats, Ageold2)**

**3.6 Labelling, Formatting, and Sorting Variables in R**

When coding our datasets, we rarely use the full variable name or the name of the responses. Instead, we often use *variable and response codes*, which are shorter and easier to code. However, when we wish to print graphics or run analyses, we often would prefer the full description – thus, it is useful to label and format the values.

To apply labels responses, we will use the **expss** package. If we wish to apply labels to a list of variables within a dataset, we can use the **apply\_labels** command. It is typically in the form:

**DATASET = apply\_labels( DATASET,   
 VAR1= “LABEL1”,  
 VAR2= “LABEL2”, …)**

In the *carseats* data, we previously noted that the indicator variables *Urban* and *US* are not the most descript. We will remedy such.

**Example 3.25 –** Using R Code to Create Variable Labels:

**carseats=apply\_labels(carseats,**

**Urban= "Indicator of Living In Urban Area",**

**US = "Indicator of Living In US")**

Alternatively, we can use the **var\_lab­** command, as we can use for the *Advertising* variable.

**Example 3.26 –** Using R Code to Create Variable Labels:

**var\_lab(carseats$Advertising)="Local Advertising Budget"**

To format our variable response labels, we will use the **ordered** command. It is typically in the form:

**DATASET <- ordered(DATASET,**

**levels=c(LEVEL1, LEVEL2, …),**

**labels=c("LABEL1", "LABEL2",…))**

We’ll apply this procedure to the *Urban* variable, where we wish to denote “Yes” as “Urban” and “No” as “Not Urban”

**Example 3.27 –** Using R Code to Create Variable Response Formatting:

**carseats$Urban <- ordered(carseats$Urban,**

**levels=c("Yes", "No"),**

**labels=c("Urban", "Not Urban"))**

We can also ***sort*** our data by a variable using the **order** command. Below, we order through ***Income***.

**Example 3.28 –** Using R Code to Order Data by a Variable:

**carsort <- carseats[order(carseats$Income), ]**

**3.7 Missing Data, Irregular Data, and Outlier Removal in R**

If we wish to deal with ***missing data***, R is pretty rad. We can use **na.omit** to remove our missing data.

**Example** **3.29** – Using R Code to Remove Missing Data:  
**auto <- read.csv("automobile.csv")  
auto$rep78  
cleanedauto <-na.omit(auto)**

**cleanedauto$rep78**

Another issue that can arise is getting rid of suspected outliers or incorrect data. Such issues occur when there are clear irregularities in the data, and we wish to *clean* the data for appropriate analysis.

**Example 3.30–** Using R Code to Get Rid of Irregular Data:

**outauto <-cleanedauto**

**outauto$price[outauto$price>11000]<-NA**

**c.o.auto <-na.omit(outauto)**

Removing potential outliers in R is quite simple. We will use the operator **%in%** to check to see if a value on the left matches with any of the elements on the right.

**Example 3.31–** Using R Code to Get Remove Outliers:

**oauto2 <-cleanedauto**

**boxplot(oauto2$price)$out**

**oauto2$price[oauto2$price %in% boxplot(oauto2$price)$out]<-NA**

**oauto2<-na.omit(oauto2)**

**Note:**

Just because we have *potential outliers* does not mean we should remove all of them. Make sure that values you remove are appropriate for the removal and does not change the population for which you wish to sample.

**3.8 Concatenating, Merging, and Melting Data in R**

We’ll conclude with the R techniques for concatenating, merging, and melting data.

Concatenating Data with R

The first technique is ***concatenating data*** *files*, which simply means to stack them one of top of the other to make one large data set. This technique allows us to take multiple data sets – including data from various surveyors or from *longitudinal studies* (studies over time) – and analyze them together, namely when they share similar variables.

We will first create our two data sets ***dads*** and ***moms***, and we will aim to put together as a data set “parents”.

**Example** **3.32**– Concatenating Two Data Sets

**famid <- c(2,1,3)**

**name <- c("Art", "Bill", "Paul")**

**inc <- c(22000, 30000, 25000)**

**dads <- data.frame(famid, name, inc)**

**famid2 <- c(1,3,2)**

**name2 <- c("Bess", "Pat", "Amy")**

**inc2 <- c(15000, 50000, 18000)**

**moms <- data.frame(famid=famid2, name=name2, inc=inc2)**

[Note: We would have to define the variables differently at first, and then rename within the data frame in order to concatenate. ]

* We can then combine the two sets with the command **rbind**.

**parents <- rbind(dads, moms)**

Now, this concept works well, but there is one slight hiccup – we may wish to differentiate the moms from the dads. What we will need to do is create a new variable that indicates from which data set these observations originate.

If we wish to have an indicator variable to determine which population for which this derives, we will use some neat tricks to bind the datasets together. The below command utilizes three different procedures at once.

1. First, we create a variable ***momdad***.
2. Second, we set ***momdad*** to equal a *replication* of responses (Dad or Mom), and replicate it to the *length* of a single column within the data frame.
3. Third, we add that variable on with the command **cbind**.
4. Fourth, we use these cbind-ed variables and again use the **rbind** procedure.

**Example** **3.33**– Concatenating Two Data Sets with an Indicator Variable

**parents2 <- rbind(cbind(dads, momdad=replicate(length(dads[1,]),"Dad")),**

**cbind(moms, momdad=replicate(length(moms[1,]),"Mom")))**

When trying to concatenate two data sets that do not share the exact same variable list, R becomes more noncompliant and cranky.

Concatenating the Two Data Sets with a Different Variable List

> dads <-data.frame(famid, name, inc)

> moms2 <-data.frame(famid=famid2, name=name2, inc2)

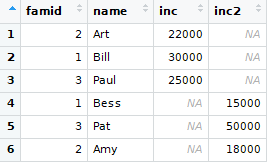
> parents3 <-rbind(dads, moms2)

Error in match.names(clabs, names(xi)) :

names do not match previous names

But, like much of what makes R rad, there is a package for that. The **plyr** packageallows us to do more with editing data sets. Using the **rbind.fill** command, R will concatenate and simply insert ***NA*** in the variable for data sets that do not originally have that variable.

**Example 3.34** – Concatenating the Two Data Sets with a Different Variable List

**moms2 <-data.frame(famid=famid2, name=name2, inc2)**

**dads <- data.frame(famid, name, inc)**

**install.packages("plyr")**

**library(plyr)**

**parents3 <- rbind.fill(dads,moms2)**

Now, when these two variables are actually the same variable (but with different labels), we would ideally like to put them together. We can do so by *changing the column names* of a data set before binding them.   
We can do so by using the **colnames** command, specifying the column we wish to change.

**Example 3.35** – Concatenating the Two Data Sets with a Different Variable Label

**moms3 = moms2**

**colnames(moms3)[3] <- "inc"**

**parents4 <- rbind.fill(dads,moms3)**

Match Merging Data Files in R

We will now work with **merging** in R, which is much easier than with SAS ^\_^. Previously, we combined observations to create one data set, but this procedure does not provide us yet with the ability to combine specified variables.

|  |
| --- |
| **dads** |
|  |
| **faminc** |
|  |

Now, consider a case where two data sets, along with sharing the same labelling code, also share another variable. Such a situation could occur in longitudinal studies or other datasets where there is tracking of previous information. When we merge two datasets in one-to-one merging, R will overwrite of the two variables and provide a dataset with only one copy.

We will begin with a **one-to-one** merge, where there is a one-to-one correspondence between the label of interest. Consider the datasets ***dads*** and ***faminc*** (family income).

Here, there is only one value (famid) that is used per family.   
We can merge the two datasets together with the **merge** command.

**Example 3.36** – One-to-One Merging of Two Datasets

**famid3 <- c(3,1,2)**

**faminc96 <- c(75000,40000,45000)**

**faminc97 <-c(76000,40500,41000)**

**faminc98 <- c(77000,41000,45800)**

**faminc <- data.frame(famid=famid3, faminc96, faminc97, faminc98)**

|  |
| --- |
| **kids** |
|  |

**faminc.dad <- merge(dads, faminc)**

The **merge** procedure also works when there isn’t a one-to-one correspondence. When the label of one data set has all unique outputs, while another data set may have more than one observation with the same output, this type of merging is known as **one-to-many**merging.

We’ll again consider the***dads***file, and also incorporate a new file, ***kids***. In this file, there are numerous observations with the same family ID, and so we will merge the two data sets, sorted corresponding to the ***famid*** variable.

**Example 3.37** – One-to-Many Merging of Two Datasets

**famidkid <- c(2,1,3,2,1,2,3,1,3)**

**kidname <- c("Beth", "Bob", "Barb", "Andy", "Al", "Ann", "Pete", "Pam", "Phil")**

**kids <- data.frame(famid=famidkid, kidname)**

**dad.kids <-merge(kids, dads)**

Combining Populations into Usable Variables in R

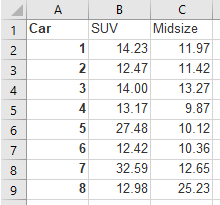
If we have two or more populations that we wish to merge together for a single data frame, we can still use the **merge** command, but we must get a little sneaky with it. We will include two extra comments:   
(1) **by= “row.names”** and (2) **all=TRUE**

**Example 3.38** – Merging Populations with Unequal Observations

**Pop1 <- data.frame(Pop1=c(4,5,62, 6))**

**Pop2 <- data.frame(Pop2=c(1,-2,55,51,23))**

**twopops <-merge(Pop1, Pop2, by= "row.names",all=TRUE)**

Sometimes, data files will consists of just lists observations, split by populations. As in the image to our right, each “variable” is a distinct population. We could have received a file such as the one to our right for statistical analysis, or we could have created such a data set by producing a *one-to-one* *merging* of two or more datasets.

When we had multiple populations in lists, SAS made us use PROC TRANSPOSE to get our results.

For R, we will utilize the package **reshape2** (a faster, upgraded version of **reshape** package).

We will again consider the dataset ***cardamage***, and we will upload it using the **read.table** command, which you may find as an easier way to transfer code from SAS or other programs to R if needed.

**Example 3.39** – Using **read.table** to Read Data, and Deleting the First Column

**cardamage <- read.table(text=" Car SUV Midsize**

**1 14.23 11.97**

**2 12.47 11.42**

**3 14.00 13.27**

**4 13.17 9.87**

**5 27.48 10.12**

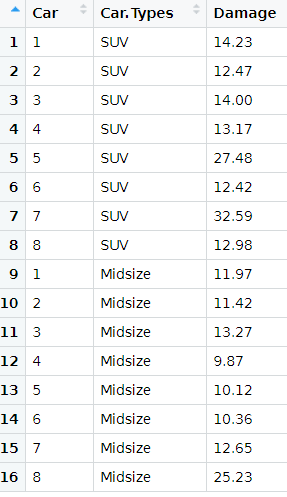
**6 12.42 10.36**

**7 32.59 12.65**

**8 12.98 25.23", header=TRUE)**

[If you wish to get rid of the index variable, you can do so using: **cardamage2 <- cardamage[,-1]** ]

Now, we will upload the **reshape2** package and use the **melt** command to combine the two populations into a dataframe we can work with. The option **variable.name** allows us the name the variable for populations, and the option **value.name** allows us to name the variable that we “melt” these observations into. We use the **id** option for all variables we wish to keep associated to a given population (this concept helps when we have *dependent* observations).



**Example 3.40** – “Melting” Data to Combine Populations

**install.packages("reshape2")**

**library(reshape2)**

**car.damage <-melt(cardamage2, value.name = "Damage", variable.name = "Car.Types", id="Car", na.rm=TRUE )**

In doing so, we obtain a dataset that looks like the image to our right.

In certain variables, we will have uneven observations, which will lead a dataset to have NA values. The option **na.rm** rids us of the NA values. If we wish to still keep the NA values, we can do so by setting the value to **FALSE**.