Merging datasets, Chapter 10 in book

We often want to merge datasets into one new dataset in order to combine variables.

We can also take one dataset and split it into new datasets, for example by conditioning on a variable.

Splitting datasets

```
data salvageTitle cleanTitle Other;
  infile "cars.txt" firstobs=2;
  input year price miles title $;
  if title="salvage" then output salvageTitle;
  else if title="clean" then output cleanTitle;
  /* else delete; */
run;

proc print data=salvageTitle;
run;
```

Splitting datasets

Note that the new dataset names must be in the `data` statement at the beginning. In this case, the dataset named `Other` had no observations output to it, and is created by SAS but is empty. (It is listed in the WORK directory.) The statement to delete observations that don't meet the criteria (either `salvage` or `clean`) doesn't seem to make a difference in this case.

Combining datasets: concatenation

If two data sets have the same variables, they can be concatenated to create a larger dataset. For example, from geonet.org, if you request one year of earthquakes, to make the file sizes not too big, it splits it up into two separate `.csv` files with exactly the same variables.

If my earthquake files are say, `2010a.csv` and `2010b.csv`, where the first goes from January 2010 to June 2010, and the second from July to December 2010, I can read them in separately and combine them into one SAS dataset without having to combine my `.csv` files outside of SAS. To do this, I can do the following:

Combining datasets: concatenation

```
data quake2010a;
  infile "2010a.csv" dsd firstobs=2;
  input id eventtype <etc.>;
run;

data quake2010b;
  infile "2010b.csv" dsd firstobs=2;
  input id eventtype <etc.>;
run;

data quake2010;
  set quake2010a quake2010b;
run;
```

Combining datasets: concatenation

It is also easy to concatenate datasets in linux using `cat`

```
$ cat 2010a.csv 2010b.csv > 2010.csv
```

Here you would want to be careful that the `2010b.csv` had the header line removed.

Combining datasets: concatenating when variables differ

If you try to concatenate two datasets that have different variables, the resulting concatenated datasets have all variables contained in both data sets.

As an example, suppose you have the following datasets:

```
id name age salary
3387 Wang 29 32000
2445 Jones 45 33000
2432 Gonzalez 32 34000
1179 DeGiorgio 31 45000
1108 Castillo 26 18000
```

```
id name startDate insurance
2445 Jones 01/01/2010 BCBS
2432 Gonzalez 01/15/2009 BCBS
1179 Degiorgio 04/22/2006 UNM
1108 Castillo 10/05/2013 none
3387 Wang 08/08/2011 Pres
```

Concatenating data with differing variables

Note that the `name` variable has different lengths. This produces a warning in the `log` file that the `name` variable
might get truncated; however, SAS does not treat them as distinct variables.

```
data salary;
  infile "salary.txt";
  input id $ name :$20. age salary;
run;

data insurance;
  infile "insurance.txt";
  input id $ name :$30. startdate mmddyy10. insurance $;
  format startdate date9.;
run;

data sal_ins;
  set salary insurance;
run;
```

Concatenating data with differing variables

Note that `startdate` is missing for the first 5 observations and has a period. Insurance is
also missing because it has a space, the missing string for character variables.

Obs	id	name	age	salary	startdate	insurance
1	3387	Wang	29	32000	.	
2	2445	Jones	45	33000	.	
3	2432	Gonzalez	32	34000	.	
4	1179	DeGiorgio	31	45000	.	
5	1108	Castillo	26	18000	.	
6	2445	Jones	.	.	01JAN2010	BCBS
7	2432	Gonzalez	.	.	15JAN2009	BCBS
8	1179	Degiorgio	.	.	22APR2006	UNM
9	1108	Castillo	.	.	05OCT2013	none
10	3387	Wang	.	.	08AUG2011	Pres

Merging datasets

What we might want with this data is to have one line per employee with all of the variables: id, name, age, salary, startdate, and insurance. Note that the names are not in the same order in the two datasets. Also note that `DeGiorgio` is spelled with a capital `G` in the first but not second dataset.

To merge the datasets this way, we need to sort the datasets by the variable we want to use for merging, and use a BY statement to merge using that variable.

Merging datasets

```
data salary;
  infile "salary.txt";
  input id $ name :$20. age salary;
run;

data insurance;
  infile "insurance.txt";
  input id $ name :$30. startdate mmddyy10. insurance $;
  format startdate date9.;
run;

proc sort data=salary;    by id; run;
proc sort data=insurance; by id; run;

data sal_ins;
  set salary insurance;
  by id;
run;
```

Merging datasets

```
data salary;
  infile "salary.txt";
  input id $ name :$20. age salary;
run;

data insurance;
  infile "insurance.txt";
  input id $ name :$30. startdate mmddyy10. insurance $;
  format startdate date9.;
run;

proc sort data=salary;    by id; run;
proc sort data=insurance; by id; run;

data sal_ins;
  set salary insurance;
  by id;
run;
```

Merging datasets

Obs	id	name	age	salary	startdate	insurance
1	1108	Castillo	26	18000	.	
2	1108	Castillo	.	.	05OCT2013	none
3	1179	DeGiorgio	31	45000	.	
4	1179	Degiorgio	.	.	22APR2006	UNM
5	2432	Gonzalez	32	34000	.	
6	2432	Gonzalez	.	.	15JAN2009	BCBS
7	2445	Jones	45	33000	.	
8	2445	Jones	.	.	01JAN2010	BCBS
9	3387	Wang	29	32000	.	
10	3387	Wang	.	.	08AUG2011	Pres

Merging datasets

```
data salary;
  infile "salary.txt";
  input id $ name :$20. age salary;
run;

data insurance;
  infile "insurance.txt";
  input id $ name :$30. startdate mmddyy10. insurance $;
  format startdate date9.;
run;

proc sort data=salary;    by id; run;
proc sort data=insurance; by id; run;

data sal_ins;
  set salary insurance;
  by id;
  age = max(age,lag1(age));
  salary = max(salary,lag1(salary));
  startdate = max(startdate,lag1(startdate));
  startdate2 = max(salary,lag1(salary));
  format startdate2 date9.;
  lagins = lag1(insurance);
  if length(lagins) > length(insurance) then insurance = lagins;
  /* if last.id; */
  /* drop lagins; */
run;
```

Merging datasets

Obs	id	name	age	salary	startdate	insurance	startdate2	lagins
1	1108	Castillo	26	18000	.		13APR2009	
2	1108	Castillo	26	18000	05OCT2013	none	13APR2009	
3	1179	DeGiorgio	31	45000	05OCT2013	none	16MAR2083	none
4	1179	Degiorgio	31	45000	22APR2006	UNM	16MAR2083	
5	2432	Gonzalez	32	34000	22APR2006	UNM	16MAR2083	UNM
6	2432	Gonzalez	32	34000	15JAN2009	BCBS	01FEB2053	
7	2445	Jones	45	33000	15JAN2009	BCBS	01FEB2053	BCBS
8	2445	Jones	45	33000	01JAN2010	BCBS	08MAY2050	
9	3387	Wang	29	32000	01JAN2010	BCBS	08MAY2050	BCBS
10	3387	Wang	29	32000	08AUG2011	Pres	12AUG2047	

Merging datasets: the hard way

```
data salary;
  infile "salary.txt";
  input id $ name :$20. age salary;
run;

data insurance;
  infile "insurance.txt";
  input id $ name :$30. startdate mmddyy10. insurance $;
  format startdate date9.;
run;

proc sort data=salary;    by id; run;
proc sort data=insurance; by id; run;

data sal_ins;
  set salary insurance;
  by id;
  age = max(age,lag1(age));
  salary = max(salary,lag1(salary));
  startdate = max(startdate,lag1(startdate));
  startdate2 = max(salary,lag1(salary)); /* funny typo */
  format startdate2 date9.;
  lagins = lag1(insurance);
  if length(lagins) > length(insurance) then insurance = lagins;
  if last.id;
  drop lagins;
run;

proc print data=sal_ins; run;
```

Merging datasets: the hard way

Obs	id	name	age	salary	startdate	insurance	startdate2
1	1108	Castillo	26	18000	05OCT2013	none	13APR2009
2	1179	Degiorgio	31	45000	22APR2006	UNM	16MAR2083
3	2432	Gonzalez	32	34000	15JAN2009	BCBS	01FEB2053
4	2445	Jones	45	33000	01JAN2010	BCBS	08MAY2050
5	3387	Wang	29	32000	08AUG2011	Pres	12AUG2047

Notes on typos/bugs

1. Some bugs in your SAS code will generate error messages or warnings in the log. If your output isn't what you expect, or you get NO output check the log first.
2. Common problems might be that your filename or path is incorrect, in which case your SAS dataset might have 0 observations. Also common is that the variables are read in incorrectly
3. Some bugs in your code do not generate errors in the log, and is good to know which type of error you have to start off with (for example, my weird dates)
4. If you are spending more than 50% of your time debugging instead of analyzing data, then this is frustrating, but normal

Notes on typos/bugs

1. To minimize how much time you spend debugging, run your code VERY often. Try not to write more than one procedure at a time without running your code to make sure it is doing what you expect. You can go to your libraries to view spreadsheets of SAS datasets you create instead of printing everything out, or just print a few observations to make sure things look ok
2. When doing datastep programming, you might want to run your code after every few lines
3. When using `first.` and `last.` to subset your data, you might want to see what happens without these lines first to make sure everything is ok
4. You can use `put` statements to output what is going on in more detail to the log to help debug.

Merging datasets: an easier (less hard) way

Recall that the `first.` and `last.` variables can be created by you instead of relying on SAS to do it for you, but it is easier to let SAS do it. The same is true for merging. Merging datasets is a strength for SAS, and you don't have to do everything from scratch.

To merge two data sets, use the MERGE statement instead of SET. The MERGE statement assumes that your datasets are sorted by the variables used for the MERGE. For the salary/insurance example, the data is sorted by the `id` variable.

Merging datasets: the MERGE statement

```
data salary;
  infile "salary.txt";
  input id $ name :$20. age salary;
run;

data insurance;
  infile "insurance.txt";
  input id $ name :$30. startdate mmddyy10. insurance $;
  format startdate date9.;
run;

proc sort data=salary;    by id; run;
proc sort data=insurance; by id; run;

data sal_ins;
  merge salary insurance; /* this line is different */
  by id;
run;

proc print data=sal_ins; run;
```

Merging datasets: the MERGE statement

The output looks the same as before except that `startdate2` hasn't been added. Note that the second instance of `Degiorgio` is used. This is sensitive to the order in which the merge occurs. If you type `merge insurance salary;`, then `DeGiorgio` will be used instead.

Obs	id	name	age	salary	startdate	insurance
1	1108	Castillo	26	18000	05OCT2013	none
2	1179	Degiorgio	31	45000	22APR2006	UNM
3	2432	Gonzalez	32	34000	15JAN2009	BCBS
4	2445	Jones	45	33000	01JAN2010	BCBS
5	3387	Wang	29	32000	08AUG2011	Pres

Merging datasets by name instead of ID

What happens if we merge by `name` instead of `id`?

```
data salary;
  infile "salary.txt";
  input id $ name :$20. age salary;
run;

data insurance;
  infile "insurance.txt";
  input id $ name :$30. startdate mmddyy10. insurance $;
  format startdate date9.;
run;

/* Now sort by name */
proc sort data=salary;    by name; run;
proc sort data=insurance; by name; run;

data sal_ins;
  merge salary insurance;
  by name;
run;

proc print data=sal_ins; run;
```

Merging datasets by name instead of ID

This causes a problem. But sometimes we don't have unique identifiers. For a name like `DeGiorgio`, several variants might be common:

```
DeGiorgio
Degiorgio
De Giorgio
de Giorgio
```

What can be done?

Solution. String functions to the rescue!

Merging datasets by name instead of ID

For the `DeGiorgio` example, the problem in the original data can be solved if we ignore case. We might also have to ignore spaces for the third and fourth variants of the name.

To ignore case, the easiest thing to do is to convert the name to all uppercase or all lowercase using the functions UPCASE or LOWCASE. Here will try LOWCASE.

Merging datasets by name instead of ID

```
data salary;
  infile "salary.txt";
  input id $ name :$20. age salary;
  lowName = lowcase(name);
run;

data insurance;
  infile "insurance.txt";
  input id $ name :$30. startdate mmddyy10. insurance $;
  format startdate date9.;
  lowName = lowcase(name);
run;

/* Now sort by name */
proc sort data=salary;    by lowName; run;
proc sort data=insurance; by lowName; run;

data sal_ins;
  merge salary insurance;
  /* would the next line work? try it! */
  /* by upcase(name); */
  by lowName;
run;

proc print data=sal_ins; run;
```

Merging datasets by name instead of ID

Here we sorted by `lowName` rather than `name`.

Obs	id	name	age	salary	lowName	startdate	insurance
1	1108	Castillo	26	18000	castillo	05OCT2013	none
2	1179	Degiorgio	31	45000	degiorgio	22APR2006	UNM
3	2432	Gonzalez	32	34000	gonzalez	15JAN2009	BCBS
4	2445	Jones	45	33000	jones	01JAN2010	BCBS
5	3387	Wang	29	32000	wang	08AUG2011	Pres

Merging datasets by name instead of ID

```
data salary;
  infile "salary.txt";
  input id $ name :$20. age salary;
  lowName = lowcase(name);
run;

data insurance;
  infile "insurance.txt";
  input id $ name :$30. startdate mmddyy10. insurance $;
  format startdate date9.;
  lowName = lowcase(name);
run;

/* Now sort by name */
proc sort data=salary;    by lowName; run;
proc sort data=insurance; by lowName; run;

data sal_ins;
  merge salary insurance;
  by lowName;
  /* drop lowName; */
  keep name age insurance;
run;

proc print data=sal_ins; run;
```

Merging datasets by name instead of ID: keeping a subset of variables

Note that you could drop the variable `lowName` in the same datastep as you used for merging, so that one of the formatted names (with capitalization) appears, even though it was merged based on the lower case version of the string. This is done by putting

```
drop lowName
```

at the end of the datastep. Often when merging, you only want a subset of the variables available. You can also use a KEEP statement (instead of DROP) to list the variables you want to KEEP. You can list multiple variables to DROP or KEEP, depending on whichever is more convenient.

Merging datasets by name instead of ID

Obs	name	age	insurance
1	Castillo	26	none
2	Degiorgio	31	UNM
3	Gonzalez	32	BCBS
4	Jones	45	BCBS
5	Wang	29	Pres

Merging datasets by name instead of ID

What if we also had names such as `De Giorgio` or `de Giorgio`?

To get rid of spaces, we can use the COMPRESS function. Here, we can do something like this `name2 = compress(lowcase(name)," ")` where I've used a single space between the quotes.

Merging datasets by name instead of ID

```
data salary;
  infile "salary.txt" dsd;
  input id $ name :$20. age salary;
  lowName = compress(lowcase(name)," ");
run;

data insurance;
  infile "insurance.txt" dsd;
  /* subtle change in date informat due to comma delimter */
  input id $ name :$30. startdate :mmddyy10. insurance $;
  format startdate date9.;
  lowName = compress(lowcase(name)," ");
  /* debugging code I did for myself to be able to scroll through log file
     more easily */
  put "++++++++++++++++++++" insurance "++++++++++++++++++++++++++++++";
  put _all_;
run;

/* Now sort by name */
proc sort data=salary;    by lowName; run;
proc sort data=insurance; by lowName; run;

data sal_ins;
  merge salary insurance;
  by lowName;
  /* drop lowName; */
  /* keep id name age insurance lowName; */
run;

proc print data=sal_ins; run;
```

Merging datasets by name instead of ID

The merge was done based on lower case names with no spaces.

Obs	id	name	age	salary	lowName	startdate	insurance
1	1108	Castillo	26	18000	castillo	05OCT2013	none
2	1179	De giorgio	31	45000	degiorgio	22APR2006	UNM
3	2432	Gonzalez	32	34000	gonzalez	15JAN2009	BCBS
4	2445	Jones	45	33000	jones	01JAN2010	BCBS
5	3387	Wang	29	32000	wang	08AUG2011	Pres

The COMPRESS function

The COMPRESS function is incredibly useful. In addition to removing spaces, you can remove other unwanted characters. For example, for the weird time format in the New Zealand data, you could type something like

```
date2 = compress(date1,"Z");
```

to get rid of the Z at the end. If you had strings representing dollar amounts, such as US$1000 and GBP$500, you could remove all (upper case) alphabetic characters using

```
money2 = compress(money,"ABCDEFGHIJKLMNOPQRSTUVWXYZ$");
```

Removing characters to make formatting compatible

Suppose you want to merge two datasets by phone number but the phone numbers are stored differently in the two datasets. The first dataset has data like this

```
phone city carrier
505-333-0904 Albuquerque Verizon
505-325-0999 Albuquerque T-Mobile
```

The second like this

```
phone name
5053330904 Tarzan
5053240999 Jane
```

A further complication is that the second dataset might store phone numbers as numeric, whereas the first is character. A way of dealing with this is to remove hyphens from the first dataset and convert to numeric.

```
phone2 = input(compress(phone,"-"),10.);
```

The COMPRESS function

The COMPRESS function removes any of the characters matching the list. You can also have a modifier as a third argument to the function. For example

```
newvar = compress(oldvar, ,"d")
```

Removes all digits from the string. This can be handy when you have essentially numeric data, but someone has put in notes like `unknown` for missing values instead of a standard value.

There are many modifiers available for the COMPRESS function:

The COMPRESS function (search online for complete modifiers)

```
a adds alphabetic characters to the list of characters.
c adds control characters to the list of characters.
d adds digits to the list of characters.
f adds the underscore character and English letters to the list of c
g adds graphic characters to the list of characters.
h adds a horizontal tab to the list of characters.
i ignores the case of the characters to be kept or removed.
k keeps the characters in the list instead of removing them.
l adds lowercase letters to the list of characters.
n adds digits, the underscore character, and English letters to the
o processes the second and third arguments once rather than every ti
     modifier in the DATA step (excluding WHERE clauses), or in the
     faster when you call it in a loop where the second and third ar
p adds punctuation marks to the list of characters.
s adds space characters (blank, horizontal tab, vertical tab, carria
     characters.
```

Merging by two variables

Sometimes a variable you'd like to merge by is not really unique, or you are not sure if it is unique. This is sort of the opposite of the `DeGiorgio` problem. In that case, the same person could have multiple spellings of their name. It is also possible of course, for multiple people to have the exact same name, which can cause errors when merging.

In my case, I am James Degnan IV (the fourth), with my father and grandfather (now deceased) having the exact same name, including middle name, and none of using suffixes like III, IV, etc. or jr. (junior). As I result I have sometimes gotten my father's mail (who also lives in Albuquerque), although I never received my grandfather's mail (who lived in Pennsylvania).

Merging by two variables

Sorting by two variables, such as `name` and `address` can help with these problems by making the combination of variables more likely to be unique (note that `address` might not be unique with multiple people at the same address). Adding more variables or being more exact about the merging variable makes it less likely that you will mix up two people, but runs the risk that you will think the same person is two separate people, such as a person who has changed their address.

It is easy to see how these problems can lead to getting on the wrong mailing list or getting duplicate mailings sent to the same address.

Moral: Mergining can't always be done perfectly.

Merging with the IN= option

If you want your merged dataset to only have observations that are common to both incoming datasets, you can use the (IN=) option when you merge. This is similar to the `first.` and `last.` variables in that it creates a temporary variable that is not saved with your dataset, but you can use it to subset your data based on the observations that are in one or the other (or both) datasets.

Merging with the IN= option

```
data math;
  infile "math.txt" firstobs=2;
  input id $ year gpa;
run;

data physics;
  infile "physics.txt" firstobs=2;
  input id $ year gpa;
run;

proc sort data=math; by id; run;
proc sort data=physics; by id; run;

data doubleMajors;
  merge math(in=mathMajor)
        physics(in=physicsMajor);
  by id;
  put _all_;
  if mathMajor and physicsMajor;
run;

proc print data=doubleMajors;
run;
```

Merging with the IN= option: double majors

The log file. Note that `first.` and `last.` were also created.

```
13
14          data doubleMajors;
15            merge math(IN=mathMajor)
16                  physics(IN=physicsMajor);
17            by id;
18            put _all_;
19            if mathMajor and physicsMajor;
20          run;

mathMajor=0 physicsMajor=1 id=1050601 year=2 gpa=3.9 FIRST.id=1 LAST.id=1 _ERROR_=0 _N_=1
mathMajor=1 physicsMajor=1 id=1095553 year=4 gpa=3.1 FIRST.id=1 LAST.id=1 _ERROR_=0 _N_=2
mathMajor=1 physicsMajor=0 id=1432534 year=4 gpa=3.6 FIRST.id=1 LAST.id=1 _ERROR_=0 _N_=3
mathMajor=1 physicsMajor=0 id=1543332 year=3 gpa=3.2 FIRST.id=1 LAST.id=1 _ERROR_=0 _N_=4
mathMajor=1 physicsMajor=1 id=1654993 year=4 gpa=2.8 FIRST.id=1 LAST.id=1 _ERROR_=0 _N_=5
mathMajor=0 physicsMajor=1 id=2001008 year=1 gpa=3.4 FIRST.id=1 LAST.id=1 _ERROR_=0 _N_=6
mathMajor=1 physicsMajor=1 id=2009833 year=2 gpa=1.7 FIRST.id=1 LAST.id=1 _ERROR_=0 _N_=7
mathMajor=1 physicsMajor=1 id=2019203 year=2 gpa=3.9 FIRST.id=1 LAST.id=1 _ERROR_=0 _N_=8
mathMajor=0 physicsMajor=1 id=2098188 year=2 gpa=2.5 FIRST.id=1 LAST.id=1 _ERROR_=0 _N_=9
```

Merging with the IN= option: double majors

The output has only those students who were in both the `math` and `physics` datasets.

The SAS System

Obs	id	year	gpa
1	1095553	4	3.1
2	1654993	4	2.8
3	2009833	2	1.7
4	2019203	2	3.9
5	3040032	3	3.6

Merging with the IN= option: only math majors

Suppose you want all math majors and an indicator of whether or not they are majoring in physics as well? But you don't want a list of physics majors not double majoring in math?

```
data math;
  infile "math.txt" firstobs=2;
  input id $ year gpa;
run;

data physics;
  infile "physics.txt" firstobs=2;
  input id $ year gpa;
run;

proc sort data=math; by id; run;
proc sort data=physics; by id; run;

data doubleMajors;
  merge math(in=mathMajor)
        physics(in=physicsMajor);
  by id;
  physMaj = physicsMajor;

  /* Note: if mathMajor=1; also works */
  if mathMajor;
run;
```

Merging with the IN= option: only math majors

The SAS System

Obs	id	year	gpa	phys Maj
1	1095553	4	3.1	1
2	1432534	4	3.6	0
3	1543332	3	3.2	0
4	1654993	4	2.8	1
5	2009833	2	1.7	1
6	2019203	2	3.9	1
7	3040032	3	3.6	1

Merging and renaming variables

Sometimes you want to merge SAS datasets by a certain variable that has slightly different names. For example, ID and StudentID. Possible solutions are: (1) either create a new variable such as `ID=studentID` in a previous dataset, (2) change the code for how the variable was read in initially (might not be desirable if it will screw up other procedures you've already written), and (3), use the RENAME option a previous dataset. This is done by something like this

```
data ToBeMergedLater;
  set originalData(rename=(StudentID=ID));
run;
```

One-to-one Merging

It is also possible to merge without using a BY statement if you have two datasets that are the same length. This could be useful if, for example, you have data for a matched pairs t-test but the two datasets are in different files, and you want to combine them. This could happen, for example, if the data were generated at different time points for the same subject.

This type of merging only works if observations are ordered in exactly the same way in the two files. If there is a subject ID or any other variable in common that uniquely identifies each observation, you can merge by that variable instead of assuming that the data are in the right order. If you don't have an ID in each dataset, you can still merge them.

One-to-one merging

```
[jamdeg@mizar SAS]$ cat ttest1.txt
4.563
3.541
2.009
1.8
1.055
[jamdeg@mizar SAS]$ cat ttest2.txt
3.665
2.987
1.045
2.045
1.005
```

One-to-one merging

```
00042 data one;
00043   infile "ttest1.txt";
00044   input x;
00045 run;
00046
00047 data two;
00048   infile "ttest2.txt";
00049   input y;
00050 run;
00051
00052 data one_two;
00053   merge one two;
00054   diff = x-y;
00055   put _all_;
00056 run;
00057
00058 proc print data=one_two;
00059 run;
00060
```

One-to-one merging

Obs	x	y	diff
1	4.563	3.665	0.898
2	3.541	2.987	0.554
3	2.009	1.045	0.964
4	1.800	2.045	-0.245
5	1.055	1.005	0.050

One-to-one merging

The TTEST Procedure

Variable: diff

N	Mean	Std Dev	Std Err	Minimum	Maximum
5	0.4442	0.5288	0.2365	-0.2450	0.9640

Mean	95% CL Mean		Std Dev	95% CL Std Dev	
0.4442	-0.2124	1.1008	0.5288	0.3168	1.5196

DF	t Value	Pr > |t|
4	1.88	0.1335

One-to-one merging

One-to-one merging is a bit like adding a new column to a dataset, similar to `cbind()` in R, which allows you to attach columns to a dataframe assuming that the dimensions match.

One-to-many merging

You might have one file with one observation per subject and another file with many observations per subject. This situation is fine for merging.

```
[jamdeg@mizar SAS]$ cat math2.txt
ID  course
1432534 math314
1432534 math316
1543332 math311
1543332 math316
1543332 math317
1654993 math264
2009833 math264
2019203 math264
2019203 stat145
3040032 math321
3040032 math361
```

One-to-many merging

```
data math;
  infile "math.txt" firstobs=2;
  input id $ year gpa;
run;

data courses;
  infile "math2.txt" firstobs=2;
  input id $ course $;
run;

proc sort data=math; by id; run;
proc sort data=courses; by id; run;

data merged;
  merge math courses;
  by id;
run;
```

One-to-many merging

The SAS System

Obs	id	year	gpa	course
1	1095553	4	3.1	
2	1432534	4	3.6	math314
3	1432534	4	3.6	math316
4	1543332	3	3.2	math311
5	1543332	3	3.2	math316
6	1543332	3	3.2	math317
7	1654993	4	2.8	math264
8	2009833	2	1.7	math264
9	2019203	2	3.9	math264
10	2019203	2	3.9	stat145
11	3040032	3	3.6	math321
12	3040032	3	3.6	math361

Many-to-many merging

If you have multiple observations per BY variable in each data set, SAS won't generate an error, but might give unpredictable results. It is usually best to make at least on dataset have one observation per BY variable before merging with a BY statement.

Many-to-many merging

```
data courses1;
  infile "math2.txt" firstobs=2;
  input id $ mathCourse $;
run;

data courses2;
  infile "physics2.txt" firstobs=2;
  input id $ physicsCourse $;
run;

proc sort data=courses1; by id; run;
proc sort data=courses2; by id; run;

data coursesBoth;
  merge courses1 courses2;
  by id;
run;

proc print data=coursesBoth;
run;
```

Many-to-many merging

```
[jamdeg@mizar SAS]$ nl math2.txt
     1  ID  course
     2  1432534 math314
     3  1432534 math316
     4  1543332 math311
     5  1543332 math316
     6  1543332 math317
     7  1654993 math264
     8  2009833 math264
     9  2019203 math264
    10  2019203 stat145
    11  3040032 math321
    12  3040032 math361
```

```
[jamdeg@mizar SAS]$ nl physics2.tx
     1  ID  course
     2  1432534 phys262
     3  1543332 phys290
     4  1654993 phys262
     5  1654993 phys262L
     6  2009833 phys162
     7  2019203 phys161
     8  2019203 phys161L
     9  3040032 phys302
    10  3040032 phys302L
    11  3040032 phys303
```

The SAS System

Obs	id	math Course	physics Course
1	1432534	math314	phys262
2	1432534	math316	phys262
3	1543332	math311	phys290
4	1543332	math316	phys290
5	1543332	math317	phys290
6	1654993	math264	phys262
7	1654993	math264	phys262L
8	2009833	math264	phys162
9	2019203	math264	phys161
10	2019203	stat145	phys161L
11	3040032	math321	phys302
12	3040032	math361	phys302L
13	3040032	math361	phys303

Many-to-many merging

In this case the output might or might not be reasonable, depending on what you want. If a student is taking, say 2 math courses and 1 physics course, it just lists the physics course twice. If the student is taking 2 math courses and 3 physics courses (might be a hard semester...), then the second math course gets listed twice.

Fuzzy merging

It is also possible to do fuzzy merging, but this is a fairly complicated issue.

The idea with fuzzy merging is that you consider two observations from different data sets to match if they are close enough, within some tolerance.

An example where this might be useful is for earthquake data, where you have a timestamp on seismic events from different field station. The NZ data is recorded to the nearest 100th of a second. But different field stations that are say, 100km apart will have slightly different times at which the earthquakes are recorded. For example, an earthquake in between the stations might be 10km from one station and 90km from the other. Or the clocks (or computers) at the different stations might be calibrated a little differently.

It still might be reasonable to infer that two observations with reasonably close time stamps are considered the same event.

Fuzzy merging

I don't have an empirical example of fuzzy merging.

The NZ data is already largely cleaned, although revisions for earthquake data are often necessary. We don't have the "raw" data. As statisticians, we often pretend that we are analyzing raw data, but this is often not the case. For the NZ data, the location of the earthquake might have already been inferred from comparing seismic data from different stations (one website listed 34 stations with different designs and depths), and there is therefore some measurement error in the data. Probably some sort of fuzzy merging has already taken place before we got the data in the relatively clean version we got from the web.

Nevertheless, to illustrate fuzzy merging, we'll some example data.

Fuzzy merge example

File 1:

```
ID day hour magnitude
1001 2010-05-31 23:16:23.130 3.48
1002 2010-05-31 22:45:29.809 2.65
1003 2010-05-31 22:44:56.291 2.11
1004 2010-05-31 22:38:28.000 2.44
```

File 2:

```
ID day hour magnitude
2001 2010-05-31 23:16:54.224 2.15
2002 2010-05-31 22:55:29.912 1.35
2003 2010-05-31 22:44:16.012 2.02
2004 2010-05-31 22:36:28.000 2.77
```

Fuzzy merge example

Fuzzy merges aren't built into SAS, so instead we'll concatenate the data, then sort by time. This will create interleaved data which should roughly alternative between the two datasets. Seismic events that recorded at both stations should be next to each other in the sorted data.

Fuzzy merging example

```
/* 12 characters are necessary to preserve seconds to
   thousandths place */
data quake1;
  infile "quaketime1.txt" firstobs=2;
  input id $ day :yymmdd10. hour :time12.3 magnitude;
  format hour time12.3;
run;

data quake2;
   infile "quaketime2.txt" firstobs=2;
   input id $ day :yymmdd10. hour :time12.3 magnitude;
   format hour time12.3;
run;

proc sort data=quake1; by hour; run;
proc sort data=quake2; by hour; run;

data fuzz1;
  set quake1 quake2;
  by hour;
run;

proc print data=fuzz1;
run;
```

Fuzzy merging example

The SAS System

Obs	id	day	hour	magnitude
1	2004	18413	22:36:28.000	2.77
2	1004	18413	22:38:28.000	2.44
3	2003	18413	22:44:16.012	2.02
4	1003	18413	22:44:56.291	2.11
5	1002	18413	22:45:29.809	2.65
6	2002	18413	22:55:29.912	1.35
7	1001	18413	23:16:23.130	3.48
8	2001	18413	23:16:54.224	2.15

Fuzzy merging example

```
data fuzz1;
  set quake1(in=in1) quake2(in=in2);
  by hour;
  /* retain keeps the previous value
     of the variable instead of
     initializing to missing */
  retain time1 time2 id1 id2;

  /* if record comes from data set 1
     update time1 and id1, but make
     time2 and id2 keep the value
     from the previous observation */
  if in1 then do;
    time1=hour;
    id1=id;
  end;
  if in2 then do;
    time2=hour;
    id2=id;
  end;
run;
```

Fuzzy merging example

Obs	id	day	hour	magnitude	time1	time2	id1	id2
1	2004	18413	22:36:28.000	2.77	.	81388.00		2004
2	1004	18413	22:38:28.000	2.44	81508.00	81388.00	1004	2004
3	2003	18413	22:44:16.012	2.02	81508.00	81856.01	1004	2003
4	1003	18413	22:44:56.291	2.11	81896.29	81856.01	1003	2003
5	1002	18413	22:45:29.809	2.65	81929.81	81856.01	1002	2003
6	2002	18413	22:55:29.912	1.35	81929.81	82529.91	1002	2002
7	1001	18413	23:16:23.130	3.48	83783.13	82529.91	1001	2002
8	2001	18413	23:16:54.224	2.15	83783.13	83814.22	1001	2001

Fuzzy merging example

```
data fuzz1;
  set quake1(in=in1) quake2(in=in2);
  by hour;
  /* retain keeps the previous value
     of the variable instead of
     initializing to missing */
  retain time1 time2 id1 id2;

  /* if record comes from data set 1
     update time1 and id1, but make
     time2 and id2 keep the value
     from the previous observation */
  if in1 then do;
    time1=hour;
    id1=id;
  end;
  if in2 then do;
    time2=hour;
    id2=id;
  end;
  /* compute the difference in times
     if greater than 60s, this is a new event */
  diff = abs(time1-time2);
  if diff > 60.0 or diff = . then newid = compress(id1||id2," ");
  else newid = "repeat";
  drop day time1 time2;
run;

proc print data=fuzz1;
```

Fuzzy merging example

To create an ID for each event, I used the concatenation operator ||, which concatenates two strings. I also wanted to remove spaces from the concatenated string.

Fuzzy merging example

```
data fuzz1;
  set quake1(in=in1) quake2(in=in2);
  by hour;
  /* retain keeps the previous value
     of the variable instead of
     initializing to missing */
  retain time1 time2 id1 id2;

  /* if record comes from data set 1
     update time1 and id1, but make
     time2 and id2 keep the value
     from the previous observation */
  if in1 then do;
    time1=hour;
    id1=id;
  end;
  if in2 then do;
    time2=hour;
    id2=id;
  end;
  file = 2 - in1;
  /* compute the difference in times
     if greater than 60s, this is a new event */
  diff = abs(time1-time2);
  if file ne lag1(file) and
    (diff > 60.0 or diff = .) then newid = compress(id1||id2," ");
  else newid = "repeat";
  drop day time1 time2;
run;

proc print data=fuzz1;
  where newid ne "repeat";
run;
```

Fuzzy merging example

Finally, to create one record per event, I subset those events that don't have the value "repeat" for `newid`.

```
proc print data=fuzz1;
  where newid ne "repeat";
run;
```

Note that special handling might be required if you have three or more observations within a 60 second window, or say three earthquakes that are 45 seconds apart each. In this case you could make an arbitrary decision, such as if quake 1 and quake 2 are "the same" and quake 2 and quake 3 are "the same", then you arbitrarily call the first two quakes the same, and treat quake 3 as a separate event.

Other criteria could also be used to decide whether two quakes were the same event, such as their magnitudes being reasonably similar.

Fuzzy merging example

Obs	id	hour	magnitude	id1	id2	file	diff	newid
1	2004	22:36:28.000	2.77		2004	2	.	2004
2	1004	22:38:28.000	2.44	1004	2004	1	120.00	10042004
3	2003	22:44:16.012	2.02	1004	2003	2	348.01	10042003
6	2002	22:55:29.912	1.35	1002	2002	2	600.10	10022002
7	1001	23:16:23.130	3.48	1001	2002	1	1253.22	10012002

Phonetic merging

Phonetic merging is also possible. In this case, you might want to merge one record against a long list of possible records and find the match. To do a phonetic merge you can follow certain rules (which we won't entirely enumerate) to convert a string into a phonetic approximation to the string in both files, then merge on the phonetic approximations. This can be done "on the fly" in real-time applications such as phone voice-recognition systems where you say your name and phone number and the computer tries to pull up the record of who you are. Here it can match a phonetic version of the name you gave against a phonetic version of your name its records. Example of rules for converting a word to a phonetic equivalent are

1. Keep the first letter
2. Remove all vowels (A, E, I, O, U, Y) after the first letter.
3. Remove the letters W and H, after the first letter.
4. Convert the letters B, F, P, and V to 1.
5. etc.

Probabilistic merging

Sometimes this is called Probabilistic Record Linkage. The idea is that for each pair of values for a variable (such as name), a similarity score is computed and decisions to match or not are based on the similarity score.

Typically, you might want to clean the data first, by doing things like removing blanks in names, so that De Giorgio becomes DeGiorgio, removing capitalization, and standardizing address information (Street, ST., and ST should all be the same). If one address says AVE and another says ST, you might penalize this matching only partially.

Probabilistic merging

Instead of phonetic matching, you can also compute distances between words so that mispelled words have a small distance to the correct spelling. You might want to have a small distance between say, DEGNAN and DEGMAN, or DEGNAN and DENGAN, even though they might not match phonetically.

There are different metrics for distances between words, but a common one is the Levenshtein distance, which determines the number of edits needed to modify one string to produce another string.

Similarity scores for every combination of two records can be computed using PROC SQL in SAS, but we won't explore this further.

More on string functions: concatenation

Earlier I used the concatenation operator || to combine two strings. This was used to create a new variable that contained information about two other variables.
There are many other uses of the concatenation operator. Some examples include

1. Combining month, day, and year to produce a date string
2. Combining First Name and Last Name to produce a single string with both names
3. Combining area codes and local phone numbers to produce phone numbers with area codes (note that you might need to get an area code by merging with a file that has area codes with zip codes and another file with addresses)
4. Making an identifier unique by combining it with other information, such as subjectID with visitNumber

Other string functions

We've encountered a few string functions so far. Here is a list of some of the more common string functions:

1. UPCASE/LOW CASE
2. LENGTH
3. COMPRESS (remove individual characters such as spaces, specified letters, digits, or punctuation)
4. COMPL (similar to COMPRESS, but converts multiple spaces into one space instead of removing all spaces)
5. The concatenation operator ||
6. Concatenation functions, CAT, CATS, CATX, CATT (different ways of dealing with blank space)
7. FIND (finds a string within a string and returns the position where the string occurs)
8. FINDW (finds a word within a string)
9. ANYDIGIT, ANYNUM, ANYPUNCT, ANYSPACE (determines starting location of first type of character)

Other string functions

1. NOTDIGIT, NOTNUM, etc. (determines starting location of first non-digit, etc.)
2. VERIFY (determines the first nonvalid character from user-defined list)
3. SUBSTR (extract a substring of a string)
4. SCAN (extract the nth word of a string)
5. SPEDIS (spelling distance between words)
6. TRANSLATE (substitute one character for another)
7. TRANWRD (substitute one word for another—e.g. "ST" for "STREET")
8. TRIM (remove trailing blanks from a word)
9. REVERSE (reverses a word character by character–useful for determining the last characters in a string)

Concatenation

Instead of using the concatenation operator, you can use the CAT function and it's variants. Here are some examples:

```
data _null_;
length a b c $ 8;
a = 'some'; b = ' text'; n = 123.456; c = 'together';
cat = cat(a, b, n, c);
catt = catt(a, b, n, c);
cats = cats(a, b, n, c);
catx = catx(',', a, b, n, c);
put +1 cop= / +1 cat= / catt= / cats= /catx=; run;

Output:(from log)

 cop=sometext123.456together
 cat=some     text   123.456together
catt=some text123.456together
cats=sometext123.456together
catx=some,text,123.456,together
```

Other string functions

Often string functions get combined, and we'll illustrate some examples. Suppose you have one variable for area code and another variable for the local phone number. You might have data like this

```
areaCode local
505 255-6560
619 310-3481
```

You wish to produce a list of phone numbers that look like (505) 255-6560. How can you do this? Try the following

```
data phone;
  infile "phoneNumbers.txt" firstobs=2;
  input areaCode $ local $;
  newNumber = compress("(" || areaCode || ")"," ") || " " || local;
run;
```

Compress with concatenation operator

Note that the concatenation operator behaves similarly to the `paste` function in R. Here is the output from the previous code

Obs	area Code	local	newNumber
1	505	255-6560	(505) 255-6560
2	619	310-3481	(619) 310-3481

Phone number merging

Suppose you want to merge two files by phone number, but they are in different formats:
File 1:

```
(505) 255-6560
(619) 310-3481
```

File 2

```
505-255-6560
619-310-3481
```

Phone number merging

How do you deal with the different formats?
Easiest thing to do might be to remove non-numeric characters, including spaces, from the strings from both data sets, and then compare the 10-digit strings.

```
phone <- compress(phone,"()- ");
```

This should work for both types of phone numbers.

The FIND function

The FIND function allows you to search for a string within a longer string. Suppose you have a list of music files where some are `.mp3` and some are `.m4a` (a newer file type). You want to create two datasets, one with a list of `.mp3` files, and one with a list of `.m4a` files. Your data might look like this

```
Among The Living/01 Among The Living.mp3
Among The Living/02 Caught In A Mosh.mp3
Among The Living/03 I Am The Law.mp3
Among The Living/04 Efilnikufesin (N.F.L.).mp3
Among The Living/05 A Skeleton In The Closet.mp3
Among The Living/06 Indians.mp3
Among The Living/07 One World.mp3
Among The Living/08 A.D.I._Horror Of It All.mp3
Among The Living/09 Imitation Of Life.mp3
Persistence Of Time/01 Time.m4a
Persistence Of Time/02 Blood.m4a
Persistence Of Time/03 Keep It In The Family.m4a
Persistence Of Time/04 In My World.m4a
```

The FIND function

To read in data like this, I might assume that every file name is some maximum length, say 1000 characters. In this case, the name of the album is a directory, and the name of the file follows a slash, so I might read in the data using the forward slash as a delimiter.

```
data files;
  infile "songs.txt" dlm="/";
  input album :$1000. song :$1000.;
run;
data mp3 mp4;
  set files;
  if find(song,".mp3") then output mp3;
  else if find(song,".m4a") then output m4a;
run;
```

How would you determine the number of albums (instead of songs) that were saved using `.m4a`?

Using FINDW instead

The FINDW function finds a word in the string, and you can choose your own delimiter. Note that if I used periods as a delimiter when I first read in the data to separate `mp3` and `m4a` as a separate word, this wouldn't have worked, because some of the song titles have periods in them. However, I can specify a delimiter within the FINDW function, so I could have used

```
if findw(song,"mp3",".") then output mp3
```

instead.

The SCAN function

In the previous example, I might have known that there were only two file formats for my data, `.mp3` and `.m4a`. Suppose I don't know how many file formats, and I want to make a list of all file formats used and their frequencies. How could I do this with just a list of files names?

If I don't know the names of all the file extensions, I could try to look for the last few characters in the file extension. I don't want to assume that all file extensions have 3 characters, since occassionally you get things like `.jpeg` instead of `.jpg`. Instead, I'll think of the file extension as the last "word" in the character string. I'm not aware of a way of extracting the last word directly, so we'll reverse the string, then get the first word, then reverse the first word again.

The SCAN function

The following code could go at the end of the original data step. The COMPRESS function here wasn't necessary, but the output does funny things with the spacing without it.

```
data files;
  infile "songs.txt" dlm="/";
  input album :$1000. song :$1000.;
  extension = reverse(song);
  extension = scan(extension,1,".");
  extension = compress(reverse(extension)," ");
run;

proc freq data=files;
  tables extension;
run;
```

More phone number questions

The middle three numbers, e.g., 255 in 505-255-6560, should determine whether a phone number is associated with a cell phone or not. Suppose you have phone numbers in this format. How can get the middle three numbers?

An easy way is to think of the three blocks of numbers as words delimited by hyphens, so you can use

```
middle = findw(phone,2,"-");
```

More phone number questions

Again suppose I want the middle three numbers, but the phone number looks like (505) 255-6560. Now what to do? You could first replace the hyphen with a space (or the space with a hyphen), and then get the second word. For example

```
middle = findw(translate(phone,"-"," "),2,"-");
```

If you know that the string starts at certain position, say the 5th character and ends at the 7th, you could use the SUBSTR function instead:

```
middle = substr(phone,5,7);
```

Phylogenetic trees

Phylogenetic trees are often represented in Newick format, which looks like this
`((A:0.01342,B:0.040330):0.502001,C:0.551032);`
A second tree might be quite similar but with different numeric values:
`((A:0.02145,B:0.05231):0.41205,C:0.46139);`
Whereas a third tree might show a different relationship:
`((A:0.02145,C:0.05231):0.41205,B:0.46139);`
To compare if two trees have the same relationships but ignoring the branch length information (the quantitative values), we want to get rid of numeric values, to get the strings `((A,B),C)`, `((A,B),C)`, and `((A,C),B)`. This is most easily done by

`tree = compress(tree,":.;","d");` which removes digits from the tree strings as well as the colon and period symbols.

The VERIFY function

The VERIFY function is useful for checking that values are legitimate. For example when using a Likert scale (1=Strongly Agree, 2=Agree, ..., 5=Strongly Disagree), the only legitimate values are 1, 2, 3, 4, 5, and probably missing. Answers like 0, 6, or A, are invalid, and should either be coded as invalid or set to missing. Another example might be multiple choice tests where the possible responses are A, B, C, or D, or a yes/no question that only allows two answers.

Another example might be DNA sequences. A DNA sequence is a string of characters usually from `A, C, G, T`; however, there are also special missing value codes such as `Y` when it is known that the character should be `C` or `T`, but it is not certain which, and `?`, when it is completely unknown what the character should be. Scanning the sequence to make sure only allowable letters are used could be done with the VERIFY function.

String functions and the Wikipedia data

If you are having trouble reading in the Wikipedia data, there are some things you can do:
(1) If you are having trouble with reading in missing values, you can convert them by hand in the original data to periods for the variables you need (a lot of the missing values are for variables we aren't using). When I tried, SAS correctly converted `N/A` to `.`
(2) Be sure to use the `comma9.` informat to read in values of numeric values that have commas. Quite a few of the variables need to have this because they have counts over 1000.
(3) Also make sure you are letting your strings be long enough to not cut off names of cities or states (e.g., New Mexico is more than 8 characters).
(4) Make sure you are using Tab delimited for the delimiter (see notes from week 1 or 2, or look online or in the book for dealing with this).
(5) If you were really desperate with horrible data (but you shouldn't need to for Wikipdia), you could read in the data one line at a time as a single character variable, then parse the line using string functions. For example,

Desperate use of string functions: crime data

I don't recommend the following approach for the Wikipedia data, but this sort of approach can be useful if you have highly unstructured data that isn't organized consistently into columns by delimiters, or has lots of comments and other stuff in it. The approach is based on a linux text file, and might require a different newline character for Windows.

```
data crime;
  /* hex code for carriage return */
  infile "crimeW.txt" dlm="0D"x;
  input line :$1000.;

  /* convert tabs to semicolons and remove commas */
  line = translate(line,";","09"x);
  line = compress(line,",");

  /* assume city is the second word and population is the third word */
  city = scan(line,2,";");
  population = input(scan(line,3,";"),10.);
run;

proc print data=crime;
  var city population;

run;
```

Desperate use of string functions

The SAS System

Obs	city	population
1	Albuquerque	553684
2	Anaheim	344526
3	Anchorage	291143
4	Arlington	379295
5	Atlanta	437041
6	Aurora	336952
7	Austin	832901
8	Bakersfield	355696
9	Baltimore	625474
10	Boston	630648
11	Buffalo	262434
12	Charlotte-Mecklenburg	808504
13	Chicago	2708382