Week 9: PROC TABULATE (Chapter 19)

We continue exploring primarily describing data to make it easier to present and understand. PROC TABULATE is especially useful for qualitative variables or for breaking down quantitative variables for different class variables.

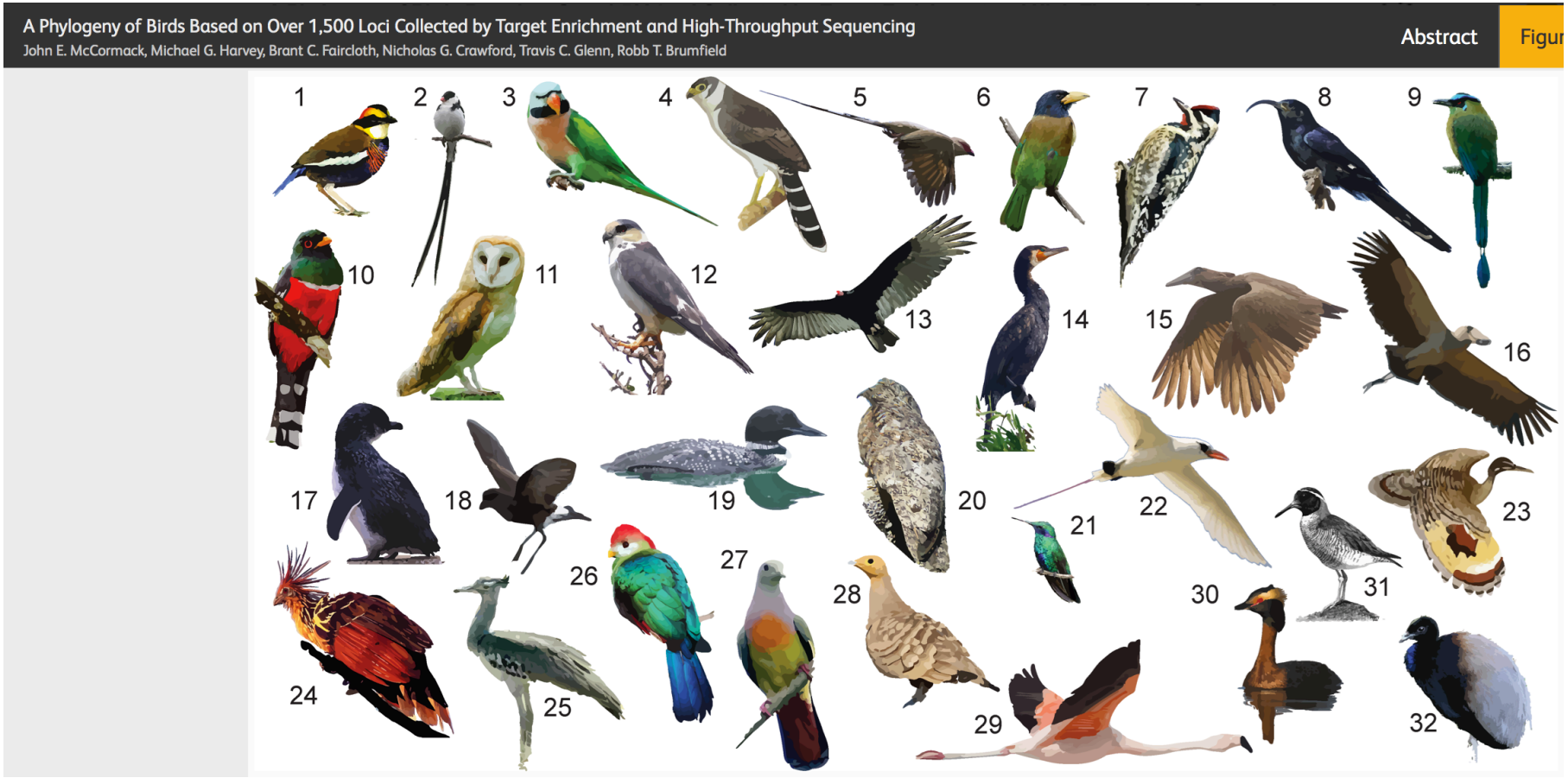
The subject is rich enough that an entire book is devoted to PROC TABULATE called *PROC TABULATE by Example*, by Lauren E. Haworth.

genome dataset: partial list

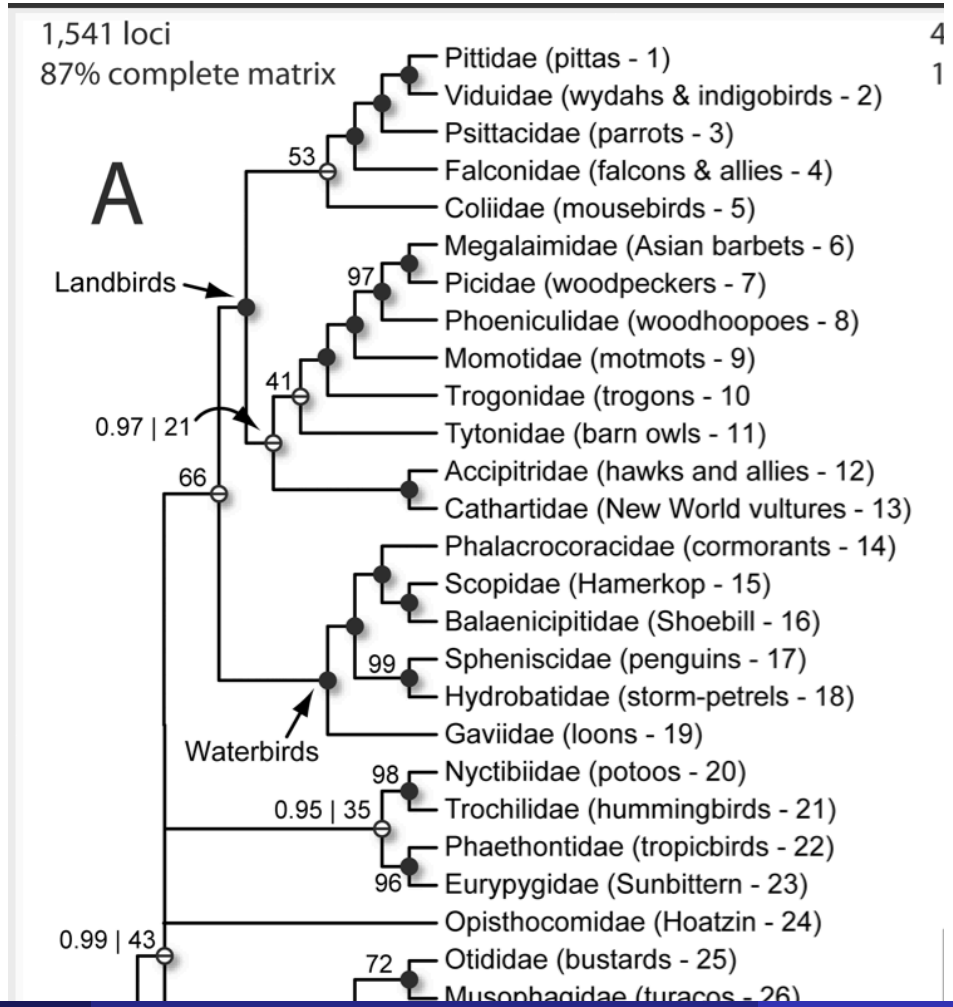
Data from McCormack et al, PLoS ONE, 2013, DOI: 10.1371/journal.pone.0054848

chromosome	size	mutation_type	species with mutation
chr8_4091	2	deletion	Rhinopomastus, Sphyrapicus
chr1_32309	3	insertion	Pitta, Rhinopomastus, Psittacula, Momotus, Podiceps
chr3_5661	2	insertion	Rhinopomastus, Sphyrapicus
chr3_5661	3	deletion	Eurypyga, Opisthocomus
chr13_707	6	deletion	Eurypyga, Treron
chr9_3551	4	deletion	Colibri, Rhinopomastus, Treron, Eurypyga
chr9_3551	7	deletion	Megalaima, Sphyrapicus
chr9_3551	3	deletion	Psittacula, Ardeotis
chr2_21162	4	deletion	Opisthocomus, Treron, Phoenicopterus, Podiceps
chr13_2902	3	insertion	Gampsonyx, Phalacrocorax
chr7_6244	5	insertion	Balaeniceps, Phalacrocorax
chr2_3317	4	deletion	Scopus, Balaeniceps
chr15_3386	4	deletion	Psittacula, Gampsonyx
chr15_3386	4	deletion	Urocolius, Scopus
chr1_32247	4	deletion	Momotus, Urocolius
chr1_32247	4	deletion	Phoenicopterus, Podiceps
chr3_5522	10	deletion	Sphyrapicus, Phaethon
chr5_10912	2	deletion	Megalaima, Sphyrapicus
chr2_23600	5	insertion	Megalaima, Sphyrapicus

Birds


A Phylogeny of Birds Based on Over 1,500 Loci Collected by Target Enrichment and High-Throughput Sequencing
John E. McCormack, Michael G. Harvey, Brant C. Faircloth, Nicholas G. Crawford, Travis C. Glenn, Robb T. Brumfield
Abstract
Figur
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32

Bird phylogeny (evolutionary tree)



genome dataset

The data lists the chromosome number for various mutations found. The mutations consist of either insertion or deletion of genetic material. The genetic material can be represented by sequences of letters, such as `GATTACA`. An insertion of two letters, for example, `GG`, might change this sequence to `GATGGTACA`. The variable `size` indicates the lengths of these insertions or deletion events.

The data lists insertions and deletions detected for 33 bird species in comparison to a reference species (chicken, I think) and indicates their location in terms of chromosome number and genomic coordinate on the reference genome. The last variable in the dataset indicates on which species the mutation was found, for mutations found on at least two species. Mutations shared by multiple species are likely to have occurred further back in the past.

genome dataset

CODE LOG RESULTS

```
filename foo url "http://math.unm.edu/~james/indels.txt";

data indel;
  infile foo firstobs=2 dlm="09"x;
  input chromosome :$20. size mutation :$20. species :$200.;
  chrom = scan(chromosome,1,"_");
  location = scan(chromosome,2,"_");
run;

proc print data=indel;
run;
```

PROC PRINT: genome dataset

Obs	chromosome	size	mutation	species	chrom	location
1	chr8_4091	2	deletion	Rhinopomastus, Sphyrapicus	chr8	4091
2	chr1_32309	3	insertion	Pitta, Rhinopomastus, Psittacula, Momotus, Podiceps, Gampsonyx, Tyto, Pterocles, Colibri, Sphyrapicus, Nyctibius, Cathartes, Phoenicopterus, Eurypyga, Megalaima, Urocolius, Gavia, Treron	chr1	32309
3	chr3_5661	2	insertion	Rhinopomastus, Sphyrapicus	chr3	5661
4	chr3_5661	3	deletion	Eurypyga, Opisthocomus	chr3	5661
5	chr13_707	6	deletion	Eurypyga, Treron	chr13	707
6	chr9_3551	4	deletion	Colibri, Rhinopomastus, Treron, Eurypyga	chr9	3551
7	chr9_3551	7	deletion	Megalaima, Sphyrapicus	chr9	3551
8	chr9_3551	3	deletion	Psittacula, Ardeotis	chr9	3551
9	chr2_21162	4	deletion	Opisthocomus, Treron, Phoenicopterus, Podiceps	chr2	21162
10	chr13_2902	3	insertion	Gampsonyx, Phalacrocorax	chr13	2902
11	chr7_6244	5	insertion	Balaeniceps, Phalacrocorax	chr7	6244
12	chr2_3317	4	deletion	Scopus, Balaeniceps	chr2	3317
13	chr15_3386	4	deletion	Psittacula, Gampsonyx	chr15	3386
14	chr15_3386	4	deletion	Urocolius, Scopus	chr15	3386
15	chr1_32247	4	deletion	Momotus, Urocolius	chr1	32247

PROC FREQ and PROC TABULATE: genome dataset

```
proc tabulate data=indel;
class chrom;
tables chrom;
run;

proc freq data=indel;
tables chrom;
run;
```

PROC FREQ and PROC TABULATE: genome dataset

chrom											
chr1	chr11	chr12	chr13	chr15	chr2	chr3	chr5	chr6	chr7	chr8	chr9
N	N	N	N	N	N	N	N	N	N	N	N
8	2	1	2	3	7	4	3	3	4	3	4

The FREQ Procedure

chrom	Frequency	Percent	Cumulative Frequency	Cumulative Percent
chr1	8	18.18	8	18.18
chr11	2	4.55	10	22.73
chr12	1	2.27	11	25.00
chr13	2	4.55	13	29.55
chr15	3	6.82	16	36.36
chr2	7	15.91	23	52.27
chr3	4	9.09	27	61.36
chr5	3	6.82	30	68.18
chr6	3	6.82	33	75.00
chr7	4	9.09	37	84.09
chr8	3	6.82	40	90.91
chr9	4	9.09	44	100.00

Note that variables listed in a TABLES (=TABLE) statement must also be listed in either a CLASS statement or a VAR statement (you can treat quantitative variables as classes).

PROC FREQ and PROC TABULATE: genome dataset

```
proc freq data=indel;
  tables chrom mutation;
run;

proc tabulate data=indel;
  class chrom mutation;
  tables chrom mutation;
run;
```

PROC FREQ and PROC TABULATE: genome dataset

chrom												mutation	
chr1	chr11	chr12	chr13	chr15	chr2	chr3	chr5	chr6	chr7	chr8	chr9	deletion	insertion
N	N	N	N	N	N	N	N	N	N	N	N	N	N
8	2	1	2	3	7	4	3	3	4	3	4	37	7

The FREQ Procedure

chrom	Frequency	Percent	Cumulative Frequency	Cumulative Percent
chr1	8	18.18	8	18.18
chr11	2	4.55	10	22.73
chr12	1	2.27	11	25.00
chr13	2	4.55	13	29.55
chr15	3	6.82	16	36.36
chr2	7	15.91	23	52.27
chr3	4	9.09	27	61.36
chr5	3	6.82	30	68.18
chr6	3	6.82	33	75.00
chr7	4	9.09	37	84.09
chr8	3	6.82	40	90.91
chr9	4	9.09	44	100.00

mutation	Frequency	Percent	Cumulative Frequency	Cumulative Percent
deletion	37	84.09	37	84.09
insertion	7	15.91	44	100.00

Cross-tabulation in PROC FREQ versus PROC TABULATE

```
proc freq data=indel;
  tables chrom*mutation;
run;

proc tabulate data=indel;
  class chrom mutation;
  tables chrom*mutation;
run;
```

PROC FREQ and PROC TABULATE: genome dataset

```
                    The FREQ Procedure

             Table of chrom by mutation

chrom     mutation

Frequency|
Percent  |
Row Pct  |
Col Pct  |deletion|insertio|  Total
         |        |n       |
---------+--------+--------+
chr3     |      3 |      1 |      4
         |   6.82 |   2.27 |   9.09
         |  75.00 |  25.00 |
         |   8.11 |  14.29 |
---------+--------+--------+
chr5     |      3 |      0 |      3
         |   6.82 |   0.00 |   6.82
         | 100.00 |   0.00 |
         |   8.11 |   0.00 |
---------+--------+--------+
chr6     |      1 |      2 |      3
         |   2.27 |   4.55 |   6.82
         |  33.33 |  66.67 |
         |   2.70 |  28.57 |
---------+--------+--------+
chr7     |      3 |      1 |      4
         |   6.82 |   2.27 |   9.09
         |  75.00 |  25.00 |
         |   8.11 |  14.29 |
---------+--------+--------+
chr8     |      3 |      0 |      3
         |   6.82 |   0.00 |   6.82
         | 100.00 |   0.00 |
         |   8.11 |   0.00 |
---------+--------+--------+
chr9     |      4 |      0 |      4
         |   9.09 |   0.00 |   9.09
         | 100.00 |   0.00 |
         |  10.81 |   0.00 |
---------+--------+--------+
Total          37        7       44
            84.09    15.91   100.00
```

PROC FREQ and PROC TABULATE: genome dataset

chrom																	
chr1		chr11		chr12		chr13		chr15		chr2		chr3					
mutation		mutation		mutation		mutation		mutation		mutation		mutation					
deletion	insertion	deletion		deletion		deletion	insertion	deletion		deletion	insertion	deletion					
N	N	N		N		N	N	N		N	N	N					
7.00	1.00	2.00		1.00		1.00	1.00	3.00		6.00	1.00	3.00					

(Continued)

The SAS System

chrom																	
chr3		chr5		chr6				chr7				chr8		chr9			
mutation		mutation		mutation				mutation				mutation		mutation			
insertion		deletion		deletion		insertion		deletion		insertion		deletion		deletion			
N		N		N		N		N		N		N		N			
1.00		3.00		1.00		2.00		3.00		1.00		3.00		4.00			

Cross-tabulation

This code did cross-tabulation for PROC FREQ but for PROC TABULATE, it nested mutation type within chromosome.

```
proc freq data=indel;
  tables chrom*mutation;
run;

proc tabulate data=indel;
  class chrom mutation;
  tables chrom*mutation;
run;
```

Cross-tabulation

To create cross-tabulated data in PROC TABULATE, use a comma instead of an asterisk.

```
proc tabulate data=indel;
  class chrom mutation;
  tables chrom,mutation;
run;
```

The SAS System

	mutation	
	deletion	insertion
	N	N
chrom		
chr1	7.00	1.00
chr11	2.00	.
chr12	1.00	.
chr13	1.00	1.00
chr15	3.00	.
chr2	6.00	1.00
chr3	3.00	1.00
chr5	3.00	.
chr6	1.00	2.00
chr7	3.00	1.00
chr8	3.00	.
chr9	4.00	.

Cross-tabulation

You can also create 3-way tables using a combination of commas and asterisks. I'm not sure how to create a 3-way table using PROC FREQ.

```
proc tabulate data=indel;
  class chrom mutation bigmut ;
  tables chrom,mutation*bigmut;
run;
```

	mutation			
	deletion		insertion	
	bigmut		bigmut	
	0	1	0	1
	N	N	N	N
chrom				
chr1	5.00	2.00	1.00	.
chr11	2.00	.	.	.
chr12	.	1.00	.	.
chr13	.	1.00	1.00	.
chr15	1.00	2.00	.	.
chr2	3.00	3.00	.	1.00
chr3	2.00	1.00	1.00	.
chr5	3.00	.	.	.
chr6	.	1.00	.	2.00
chr7	2.00	1.00	.	1.00
chr8	2.00	1.00	.	.
chr9	1.00	3.00	.	.

Dressing up the table

As usual, we should try to improve the table's appearance a bit by doing things like adding labels, getting chromosomes in the right order, and so on.

First, the order that the chromosomes is listed isn't ideal. It is in alphanumeric order, so that `char15` comes before `char2` (because 1 comes before 2). How can we fix this?

Dressing up the table: order of values

As with PROC REPORT and PROC FREQ, there are options for changing the order in which values are listed, for example using ORDER=DATA and ORDER=FREQ. In this case, if we wish to order things by chromosome, then neither option works well.

If the chromosomes were labeled `chr01, chr02, ..., chr10, ...`, then it would be in the right order. So we could try to insert a 0 in the middle for chromosomes with numbers less than 10. Another solution is to get rid of `chr` when we read in this value and convert the chromosome to a numeric number.

Is chromosome number numeric or qualitative?

Without knowing a little bit of the biology, it is hard to say whether this variable should be considered numeric or not. Usually, chromosomes are numbered so that longer chromosomes (more DNA letters) have lower numbers, but often with a few exceptions. (This arrangement applies to both humans and chickens...) So chromosome number correlates (imperfectly) with amount of DNA, and we might expect with the number of mutations.

We also see this in this data, although we don't know what the sampling scheme for the data is (it is based on about 1500 genes covering less than 0.1% of the chicken genome). The longest chromosome (chromosome 1) has the most mutations, the second longest (chromosome 2) has the second highest number etc., so it might make sense to order the chromosomes numerically.

Re-ordering the chromosome number

For this data, I would probably treat the chromosome number as quantitative to help order the values, but the issue of integers not being alphabetical when they miss the leading 0 comes up a lot in real data and messes up the order of things, so we'll try both approaches.
I had a drum instructional DVD where the author ran into this problem. He wanted the `.mp3` files to be saved in a certain order on your computer, and instead of writing the names of the tracks as `01-slow.mp3`, `01-FAST.mp3`, etc., he named them `aa 1-slow.mp3`, `ab 1-FAST.mp3` etc.

alphabetizing integers

Another place this comes up is in simulations, particularly with scripting. If you submit a large number of files to be run to a computer, it is convenient to call them things like, `SAS1.sas`, `SAS2.sas`, ..., `SAS10.sas`. When you list these files, they will not be in the right order when listed alphabetically, and this can be a pain if you want the files to be submitted (like to a cluster) in the right order....We will see things like this a little bit when we get into macros.

One way to alphabetize integers....

```
Jamess-MacBook-Pro:Groove Essentials superjames$ ls
aa 1-slow.mp3                    bc 12-slow.mp3
ab 1-FAST.mp3                    bd 12-FAST.mp3
ac 2-slow.mp3                    bf 13-slow.mp3
ad 2-FAST.mp3                    bg 13-FAST.mp3
ae 3-slow.mp3                    ca 14-slow.mp3
af 3-FAST.mp3                    cb 14-FAST.mp3
ag 4-slow.mp3                    cc 15-slow.mp3
ah 4-FAST.mp3                    cd 15-FAST.mp3
ai 5-slow.mp3                    ce 16-slow.mp3
aj 5-FAST.mp3                    cf 16-FAST.mp3
ak 6-slow.mp3                    cg 17-slow.mp3
al 6-FAST.mp3                    ch 17-FAST.mp3
am 7-slow.mp3                    da 18-slow.mp3
an 7-FAST.mp3                    db 18-FAST.mp3
ao 8-slow.mp3                    dc 19-slow.mp3
ap 8-FAST.mp3                    dd 19-FAST.mp3
aq 9-slow.mp3                    de 20-slow.mp3
ar 9-FAST.mp3                    df 20-FAST.mp3
as 10-slow.mp3                   dg 21-slow.mp3
at 10-FAST.mp3                   dh 21-FAST.mp3
ba 11-slow.mp3                   di 22-slow.mp3
bb 11-FAST.mp3                   dj 22-FAST.mp3
```

Re-ordering the chromosome number

The easiest approach is to remove characters from the strings `chr1`, `chr2`, ... so that you just have numeric data. This can be done using the COMPRESS function.

```
filename foo url "http://math.unm.edu/~james/indels.txt";

data indel;
  infile foo firstobs=1 dlm="09"x;
  input chromosome :$20. size mutation :$20. species :$200.;
  chrom = scan(chromosome,1,"_");
  chrom2 = compress(chrom,"chr");
  location = scan(chromosome,2,"_");
  if size>3 then bigmut = 1;
  else bigmut=0;
run;

proc tabulate data=indel;
  class chrom2 mutation bigmut ;
  tables chrom2,mutation*bigmut;
run;
```

Re-ordering the chromosome number

This didn't quite do we want. Why not?

	mutation			
	deletion		insertion	
	bigmut		bigmut	
	0	1	0	1
	N	N	N	N
chrom2				
1	5.00	2.00	1.00	.
11	2.00	.	.	.
12	.	1.00	.	.
13	.	1.00	1.00	.
15	1.00	2.00	.	.
2	3.00	3.00	.	1.00
3	2.00	1.00	1.00	.
5	3.00	.	.	.
6	.	1.00	.	2.00
7	2.00	1.00	.	1.00
8	2.00	1.00	.	.
9	1.00	3.00	.	.

Re-ordering the chromosome number

Now, we convert the chromosome number to be numeric and add some labels and a format.

```
filename foo url "http://math.unm.edu/~james/indels.txt";

data indel;
  infile foo firstobs=1 dlm="09"x;
  input chromosome :$20. size mutation :$20. species :$200.;
  chrom = scan(chromosome,1,"_");
  chrom2 = input(compress(chrom,"chr"),8.);
  label chrom2="Chromsome";
  location = scan(chromosome,2,"_");
  if size>3 then bigmut = 1;
  else bigmut=0;
  label bigmut="More than three letters?";
run;

proc format;
  value indel 0="No" 1="Yes";
run;

proc tabulate data=indel;
  class chrom2 mutation bigmut ;
  tables chrom2,mutation*bigmut;
  format bigmut indel.;
run;
```

Re-ordering the chromosome number

	mutation			
	deletion		insertion	
	More than three letters?		More than three letters?	
	No	Yes	No	Yes
	N	N	N	N
Chromsome				
1	5.00	2.00	1.00	.
2	3.00	3.00	.	1.00
3	2.00	1.00	1.00	.
5	3.00	.	.	.
6	.	1.00	.	2.00
7	2.00	1.00	.	1.00
8	2.00	1.00	.	.
9	1.00	3.00	.	.
11	2.00	.	.	.
12	.	1.00	.	.
13	.	1.00	1.00	.
15	1.00	2.00	.	.

Re-ordering the chromosome number

Instead of converting the chromosome number to an integer, let's try inserting a 0 in the right place. We can assume that the chromosome number is less than 100 so that our strings for chromosome number either have exactly four or exactly five digits. If they have five digits, they don't need to be modified. If they have 4 digits, we need to inset the 0.

Note that both approaches only require one line of code, although the insert-0 approach is a slightly longer line. Neither solution is better than the other, it just depends on what you want your data to look like.

Re-ordering the chromosome number

```
filename foo url "http://math.unm.edu/~james/indels.txt";

data indel;
  infile foo firstobs=1 dlm="09"x;
  input chromosome :$20. size mutation :$20. species :$200.;
  chrom = scan(chromosome,1,"_");
  if length(chrom)=4 then chrom= compress("chr0" || substr(chromosome,4,1),"");
  label chrom="Chromsome";
  location = scan(chromosome,2,"_");
  if size>3 then bigmut = 1;
  else bigmut=0;
  label bigmut="More than three letters?";
run;

proc format;
  value indel 0="No" 1="Yes";
run;

proc tabulate data=indel;
  class chrom mutation bigmut ;
  tables chrom,mutation*bigmut;
  format bigmut indel.;
run;
```

Re-ordering the chromosome number

	mutation			
	deletion		insertion	
	More than three letters?		More than three letters?	
	No	Yes	No	Yes
	N	N	N	N
Chromsome				
chr01	5.00	2.00	1.00	.
chr02	3.00	3.00	.	1.00
chr03	2.00	1.00	1.00	.
chr05	3.00	.	.	.
chr06	.	1.00	.	2.00
chr07	2.00	1.00	.	1.00
chr08	2.00	1.00	.	.
chr09	1.00	3.00	.	.
chr11	2.00	.	.	.
chr12	.	1.00	.	.
chr13	.	1.00	1.00	.
chr15	1.00	2.00	.	.

Removing the extra 0

Note that now that we've alphabetized, we can use a FORMAT to remove the leading 0s if we want.

```
filename foo url "http://math.unm.edu/~james/indels.txt";

data indel;
  infile foo firstobs=1 dlm="09"x;
  input chromosome :$20. size mutation :$20. species :$200.;
  chrom = scan(chromosome,1,"_");
  if length(chrom)=4 then chrom= compress("chr0" || substr(chromosome,4,1),"");
  label chrom="Chromsome";
  location = scan(chromosome,2,"_");
  if size>3 then bigmut = 1;
  else bigmut=0;
  label bigmut="More than three letters?";
run;

proc format;
  value indel 0="No" 1="Yes";

  value $chr chr01="chr1" chr02="chr2" chr03="chr3"
             chr05="chr5" chr06="chr6" chr07="chr7"
             chr08="chr8" chr09="chr9" chr11="chr11"
             chr12="chr12" chr13="chr13" chr15="chr15";

run;

proc tabulate data=indel;
  class chrom mutation bigmut ;
  tables chrom,mutation*bigmut;
  format bigmut indel. chrom $chr.;
```

Removing the extra 0

	mutation			
	deletion		insertion	
	More than three letters?		More than three letters?	
	No	Yes	No	Yes
	N	N	N	N
Chromsome				
chr1	5.00	2.00	1.00	.
chr2	3.00	3.00	.	1.00
chr3	2.00	1.00	1.00	.
chr5	3.00	.	.	.
chr6	.	1.00	.	2.00
chr7	2.00	1.00	.	1.00
chr8	2.00	1.00	.	.
chr9	1.00	3.00	.	.
chr11	2.00	.	.	.
chr12	.	1.00	.	.
chr13	.	1.00	1.00	.
chr15	1.00	2.00	.	.

Other things to change the table

We can also apply formats to the cell counts. In this case, since we have integers, we'd likely want a shorter format such as `3.`

```
proc tabulate data=indel format=3.;
  class chrom mutation bigmut ;
  tables chrom,mutation*bigmut;
  format bigmut indel. chrom $chr. ;
run;
```

The SAS System

	mutation			
	deletion		insertion	
	More than three letters?		More than three letters?	
	No	Yes	No	Yes
	N	N	N	N
Chromsome				
chr01	5	2	1	.
chr02	3	3	.	1
chr03	2	1	1	.
chr05	3	.	.	.
chr06	.	1	.	2
chr07	2	1	.	1
chr08	2	1	.	.
chr09	1	3	.	.
chr11	2	.	.	.
chr12	.	1	.	.

Other things to change the table

Three characters might be too narrow for the label, so we can improve it...

```
proc tabulate data=indel format=6.;
  class chrom mutation bigmut ;
  tables chrom,mutation*bigmut;
  format bigmut indel. chrom $chr. ;
run;
```

The SAS System

	mutation			
	deletion		insertion	
	More than three letters?		More than three letters?	
	No	Yes	No	Yes
	N	N	N	N
Chromsome				
chr1	5	2	1	.
chr2	3	3	.	1
chr3	2	1	1	.
chr5	3	.	.	.
chr6	.	1	.	2
chr7	2	1	.	1
chr8	2	1	.	.
chr9	1	3	.	.
chr11	2	.	.	.
chr12	.	1	.	.
chr13	.	1	1	.

Statistics for quantitative variables

You can get means, min, max, etc. for quantitative variables listed in a VAR statement.

```
proc tabulate data=indel format=6.2;
  class chrom  mutation;
  var size;
  tables chrom,size*(n mean);
  format bigmut indel. chrom $chr. ;
run;
```

The SAS System

	size	
	N	Mean
Chromsome		
chr1	8.00	3.00
chr2	7.00	3.29
chr3	4.00	4.25
chr5	3.00	2.00
chr6	3.00	4.67
chr7	4.00	3.50
chr8	3.00	3.67
chr9	4.00	5.00
chr11	2.00	3.00
chr12	1.00	4.00
chr13	2.00	4.50
chr15	3.00	3.33

Marginal totals and other statistics

You can get marginal totals and other statistics using the keyword ALL, which is different from `_ALL_`, which is normally used to analyzed all variables. Marginal subtotals are a little confusing for a 3-way table. Here I present it just for a two-way table, but you can do three-way tables also. The quantitative statistic this time is `size`, which refers to the length of the mutation (number of DNA letters inserted or deleted). The NOSEPS option makes the table more compact.

Marginal totals and other statistics

```
proc tabulate data=indel format=6.2 noseps;
  class chrom  mutation;
  var size;
  tables (chrom all),size*(n mean median min max all);
  format bigmut indel. chrom $chr. ;
run;
```

	size					
	Sum					
	N	Mean	Median	Min	Max	All
Chromsome						
chr1	8.00	3.00	3.00	2.00	4.00	24.00
chr2	7.00	3.29	4.00	2.00	5.00	23.00
chr3	4.00	4.25	2.50	2.00	10.00	17.00
chr5	3.00	2.00	2.00	2.00	2.00	6.00
chr6	3.00	4.67	4.00	4.00	6.00	14.00
chr7	4.00	3.50	3.50	2.00	5.00	14.00
chr8	3.00	3.67	3.00	2.00	6.00	11.00
chr9	4.00	5.00	5.00	3.00	7.00	20.00
chr11	2.00	3.00	3.00	3.00	3.00	6.00
chr12	1.00	4.00	4.00	4.00	4.00	4.00
chr13	2.00	4.50	4.50	3.00	6.00	9.00
chr15	3.00	3.33	4.00	2.00	4.00	10.00
All	44.00	3.59	3.00	2.00	10.00	158.00

More formatting

We can improve the table appearance by more specific formatting. We'll start with this example of a 3-dimensional table.

```
proc tabulate data=indel format=6.2 noseps;
  class chrom  mutation;
  var size;
  tables (chrom all)*mutation,size*(n mean median min max all);
  format bigmut indel. chrom $chr. ;
run;
```

		size					
		Sum					
		N	Mean	Median	Min	Max	All
Chromsome	mutation						
chr1	deletion	7.00	3.00	3.00	2.00	4.00	21.00
	insertion	1.00	3.00	3.00	3.00	3.00	3.00
chr2	deletion	6.00	3.00	3.00	2.00	4.00	18.00
	insertion	1.00	5.00	5.00	5.00	5.00	5.00
chr3	deletion	3.00	5.00	3.00	2.00	10.00	15.00
	insertion	1.00	2.00	2.00	2.00	2.00	2.00
chr5	deletion	3.00	2.00	2.00	2.00	2.00	6.00
chr6	deletion	1.00	4.00	4.00	4.00	4.00	4.00
	insertion	2.00	5.00	5.00	4.00	6.00	10.00
chr7	deletion	3.00	3.00	3.00	2.00	4.00	9.00
	insertion	1.00	5.00	5.00	5.00	5.00	5.00
chr8	deletion	3.00	3.67	3.00	2.00	6.00	11.00
chr9	deletion	4.00	5.00	5.00	3.00	7.00	20.00
chr11	deletion	2.00	3.00	3.00	3.00	3.00	6.00
chr12	deletion	1.00	4.00	4.00	4.00	4.00	4.00
chr13	deletion	1.00	6.00	6.00	6.00	6.00	6.00
	insertion	1.00	3.00	3.00	3.00	3.00	3.00
chr15	deletion	3.00	3.33	4.00	2.00	4.00	10.00
All	deletion	37.00	3.51	3.00	2.00	10.00	130.00

More formatting

Here we formatted integers as integers, but retained 2 decimals for the mean, and we removed the variable names for chromosome and mutation.

```
proc tabulate data=indel format=6.2 noseps;
  class chrom  mutation;
  var size;
  tables (chrom=" " all)*mutation=" ",
     size*(n*f=3. mean median*f=6. min*f=5. max*f=4. all*f=7.);
  format bigmut indel. chrom $chr. ;
run;
```

		size					
		Sum					
		N	Mean	Median	Min	Max	All
chr1	deletion	7	3.00	3	2	4	21
	insertion	1	3.00	3	3	3	3
chr2	deletion	6	3.00	3	2	4	18
	insertion	1	5.00	5	5	5	5
chr3	deletion	3	5.00	3	2	10	15
	insertion	1	2.00	2	2	2	2
chr5	deletion	3	2.00	2	2	2	6
chr6	deletion	1	4.00	4	4	4	4
	insertion	2	5.00	5	4	6	10
chr7	deletion	3	3.00	3	2	4	9
	insertion	1	5.00	5	5	5	5
chr8	deletion	3	3.67	3	2	6	11
chr9	deletion	4	5.00	5	3	7	20
chr11	deletion	2	3.00	3	3	3	6
chr12	deletion	1	4.00	4	4	4	4
chr13	deletion	1	6.00	6	6	6	6
	insertion	1	3.00	3	3	3	3
chr15	deletion	3	3.33	4	2	4	10
All	deletion	37	3.51	3	2	10	130

More formatting

For cases where N appears repeatedly to show the sample size, you might want to remove this. It is mostly useful if you want to contrast it with other statistics.

```
proc tabulate data=indel noseps;
  class chrom mutation bigmut;
  tables chrom,mutation;
run;

/* make format wide enough for word "insertion" */
proc tabulate data=indel format=9. noseps;
  class chrom mutation bigmut;
  tables chrom,mutation*n=" ";
run;
```

More formatting

	mutation	
	deletion	insertion
	N	N
Chromsome		
chr01	7.00	1.00
chr02	6.00	1.00
chr03	3.00	1.00
chr05	3.00	.
chr06	1.00	2.00
chr07	3.00	1.00
chr08	3.00	.
chr09	4.00	.
chr11	2.00	.
chr12	1.00	.
chr13	1.00	1.00
chr15	3.00	.

The SAS System

	mutation	
	deletion	insertion
Chromsome		
chr01	7	1
chr02	6	1
chr03	3	1
chr05	3	.
chr06	1	2
chr07	3	1
chr08	3	.
chr09	4	.
chr11	2	.
chr12	1	.
chr13	1	1
chr15	3	.

More formatting: Keylabel statement

```
/* make format wide enough for word "Number observed" */
proc tabulate data=indel format=15. noseps;
  class chrom mutation bigmut;
  tables chrom all,mutation all;
  keylabel all="Total" n="Number observed";
run;
```

	mutation		
	deletion	insertion	Total
	Number observed	Number observed	Number observed
Chromsome			
chr01	7	1	8
chr02	6	1	7
chr03	3	1	4
chr05	3	.	3
chr06	1	2	3
chr07	3	1	4
chr08	3	.	3
chr09	4	.	4
chr11	2	.	2
chr12	1	.	1
chr13	1	1	2
chr15	3	.	3
Total	37	7	44

Percentages

You can have PROC TABULATE give percentages, but it's tricky.

```
/* This gives the percentage but doesn't include a percent sign */
proc tabulate data=indel format=15. noseps;
  class chrom mutation bigmut;
  tables chrom,mutation pctn;
  keylabel all="Total" n="Number observed";
run;

/* This gives a percent sign but percent formats multiply values by 100 first
   making them incorrect */
proc tabulate data=indel format=15. noseps;
  class chrom mutation bigmut;
  tables chrom,mutation pctn*f=percent7.1;
  keylabel all="Total" n="Number observed";
run;

/* From book: to get around this, create a user-defined format for percentages
   This format allows 3 digits before the decimal and one after the decimal
   The 9s are placeholders for any value. */
proc format;
  picture pctfmt low-high='009.9%';
run;

proc tabulate data=indel format=15. noseps;
  class chrom mutation bigmut;
  tables chrom,mutation pctn*f=pctfmt7.1;
  keylabel all="Total" n="Number observed";
run;
```

Percentages: no percent format

	mutation		
	deletion	insertion	
	Number observed	Number observed	PctN
Chromsome			
chr01	7	1	18
chr02	6	1	16
chr03	3	1	9
chr05	3	.	7
chr06	1	2	7
chr07	3	1	9
chr08	3	.	7
chr09	4	.	9
chr11	2	.	5
chr12	1	.	2
chr13	1	1	5
chr15	3	.	7

The SAS System

Percentages

Here is the result of the incorrect use of percent format.

	mutation		
	deletion	insertion	
	Number observed	Number observed	PctN
Chromsome			
chr01	7	1	1818%
chr02	6	1	1591%
chr03	3	1	909%
chr05	3	.	682%
chr06	1	2	682%
chr07	3	1	909%
chr08	3	.	682%
chr09	4	.	909%
chr11	2	.	455%
chr12	1	.	227%
chr13	1	1	455%
chr15	3	.	682%

Percentages

Here is the result of the incorrect use of percent format.

	mutation		
	deletion	insertion	
	Number observed	Number observed	PctN
Chromsome			
chr01	7	1	1818%
chr02	6	1	1591%
chr03	3	1	909%
chr05	3	.	682%
chr06	1	2	682%
chr07	3	1	909%
chr08	3	.	682%
chr09	4	.	909%
chr11	2	.	455%
chr12	1	.	227%
chr13	1	1	455%
chr15	3	.	682%

Percentages

Fixing things with a user-defined format.

	mutation		
	deletion	insertion	
	Number observed	Number observed	PctN
Chromsome			
chr01	7	1	18.1%
chr02	6	1	15.9%
chr03	3	1	9.0%
chr05	3	.	6.8%
chr06	1	2	6.8%
chr07	3	1	9.0%
chr08	3	.	6.8%
chr09	4	.	9.0%
chr11	2	.	4.5%
chr12	1	.	2.2%
chr13	1	1	4.5%
chr15	3	.	6.8%

Percentages with colpctn

```
proc tabulate data=indel format=15. noseps;
  class chrom mutation bigmut;
  tables chrom all,mutation*(n colpctn*f=pctfmt7.1) all*(n colpctn*f=pctfmt7.1);
  keylabel all="Indels" n="Number observed";
run;
```

	mutation					
	deletion		insertion		Indels	
	Number observed	ColPctN	Number observed	ColPctN	Number observed	ColPctN
Chromsome						
chr01	7	18.9%	1	14.2%	8	18.1%
chr02	6	16.2%	1	14.2%	7	15.9%
chr03	3	8.1%	1	14.2%	4	9.0%
chr05	3	8.1%	.	.	3	6.8%
chr06	1	2.7%	2	28.5%	3	6.8%
chr07	3	8.1%	1	14.2%	4	9.0%
chr08	3	8.1%	.	.	3	6.8%
chr09	4	10.8%	.	.	4	9.0%
chr11	2	5.4%	.	.	2	4.5%
chr12	1	2.7%	.	.	1	2.2%
chr13	1	2.7%	1	14.2%	2	4.5%
chr15	3	8.1%	.	.	3	6.8%
Indels	37	100.0%	7	100.0%	44	100.0%

Missing values

Fixing things with a user-defined format.

```
1  data missing;
2    infile datalines dsd;
3    input (A B C) ($);
4    put _all_;
5    datalines;
6  x,y,z
7  x,y,y
8  z,z,z
9  x,x,
10 y,z,
11 x,,
12   ;
13 run;
14
15 proc print data=missing;
16 run;
17
18 proc tabulate data=missing format=4.;
19   class A B;
20   table A all, B all;
21 run;
22
23 proc tabulate data=missing format=6.;
24   class A B C;
25   table A all, B all;
26 run;
27
```

Missing values

Obs	A	B	C
1	x	y	z
2	x	y	y
3	z	z	z
4	x	x	
5	y	z	
6	x		

	B			All
	x	y	z	
	N	N	N	N
A				
x	1	2	.	3
y	.	.	1	1
z	.	.	1	1
All	1	2	2	5

	B		All
	y	z	
	N	N	N
A			
x	2	.	2
z	.	1	1
All	2	1	3

Missing values

It is ok to include a class variable that doesn't get used in a TABLES statement. However, missing values in one of the class variables cause the entire observation to be deleted, even if the variable isn't used in the TABLES statement.

Note that the observations with A equal to x when B and C are both missing isn't tabulated. There were four observations where A was x, but the total is 3.

Missing values: Missing option

```
proc tabulate data=missing format=4. missing;
  class A B;
  table A all, B all;
run;

proc tabulate data=missing format=6. missing;
  class A B C;
  table A all, B all / misstext="miss";
run;
```

	B				All
		x	y	z	
	N	N	N	N	N
A					
x	1	1	2	.	4
y	.	.	.	1	1
z	.	.	.	1	1
All	1	1	2	2	6

	B				All
		x	y	z	
	N	N	N	N	N
A					
x	1	1	2	miss	4
y	miss	miss	miss	1	1
z	miss	miss	miss	1	1
All	1	1	2	2	6

Number of species

Suppose we wanted to count how many species each mutation affected. How can we do this?

A trickier question is how to count how many times each species occurs in the data set. How could we do this one?

Number of species

First we look at the number of species. We can use string functions to get this fairly easily assuming that each species is separated by a comma. Then for each observation, the number of species equals the number of commas plus 1.

We'll also use this new data to create a 4-way table.

Number of species

```
filename foo url "http://math.unm.edu/~james/indels.txt";

data indel;
  infile foo firstobs=1 dlm="09"x;
  input chromosome :$20. size mutation :$20. species :$200.;
  chrom = input(compress(scan(chromosome,1,"_"),"chr"),8.);
  location = scan(chromosome,2,"_");
  nspecies = countc(species,",") + 1;
  if size>3 then bigmut=1;
  else bigmut=0;
  if chrom <= 5 then macro=1;
  else macro=0;
run;

proc print data=indel;
run;

proc format;
  value macro 0="Chrom 1-5"
              1="Chrom 6-15";
run;

/* A four-dimensional table */
proc tabulate;
  class macro mutation nspecies size;
  tables macro*size all,mutation*nspecies all;
  format macro macro.;
  keylabel all="Total";
run;
```

Number of species

Obs	chromosome	size	mutation	species	chrom	location	nspecies	bigmut	macro
1	chr8_4091	2	deletion	Rhinopomastus, Sphyrapicus	8	4091	2	0	0
2	chr1_32309	3	insertion	Pitta, Rhinopomastus, Psittacula, Momotus, Podiceps, Gampsonyx, Tyto, Pterocles, Colibri, Sphyrapicus, Nyctibius, Cathartes, Phoenicopterus, Eurypyga, Megalaima, Urocolius, Gavia, Treron	1	32309	18	0	1
3	chr3_5661	2	insertion	Rhinopomastus, Sphyrapicus	3	5661	2	0	1
4	chr3_5661	3	deletion	Eurypyga, Opisthocomus	3	5661	2	0	1
5	chr13_707	6	deletion	Eurypyga, Treron	13	707	2	1	0
6	chr9_3551	4	deletion	Colibri, Rhinopomastus, Treron, Eurypyga	9	3551	4	1	0
7	chr9_3551	7	deletion	Megalaima, Sphyrapicus	9	3551	2	1	0

		mutation							Total
		deletion				insertion			
		nspecies				nspecies			
		2	3	4	5	2	6	18	
		N	N	N	N	N	N	N	N
macro	size								
Chrom 1-5	2	2	.	.	1	.	.	.	3
	3	2	3	.	.	1	.	.	6
	4	3	2	1	.	1	.	.	7
	5	.	.	.	.	1	.	.	1
	6	3	.	.	.	.	1	.	4
	7	1	.	.	.	.	.	.	1
Chrom 6-15	2	7	2	.	.	1	.	.	10
	3	3	1	.	.	.	.	1	5
	4	4	.	1	.	.	.	.	5
	5	.	.	.	.	1	.	.	1
	10	1	.	.	.	.	.	.	1
Total		26	8	2	1	5	1	1	44

Number of times each species occurs

String functions again!

```
31 data indel2;
32   set indel;
33   do i = 1 to nspecies-1;
34     species2 = scan(species,i,",");
35     output;
36   end;
37   keep species2;
38 run;
39
40 proc print data=indel2;
41 run;
42
43 proc freq data=indel2;
44   table species2;
45 run;
```

Number of times each species occurs

PROC FREQ was used instead of PROC TABULATE so that the table is vertical rather than horizontal.

Obs	species2
1	Rhinopomastus
2	Pitta
3	Rhinopomastus
4	Psittacula
5	Momotus
6	Podiceps
7	Gampsonyx
8	Tyto
9	Pterocles
10	Colibri
11	Sphyrapicus
12	Nyctibius
13	Cathartes
14	Phoenicopterus
15	Eurypyga
16	Megalaima
17	Urocolius
18	Gavia
19	Rhinopomastus
20	Eurypyga
21	Eurypyga

The FREQ Procedure

species2	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Cathartes	1	1.27	1	1.27
Colibri	1	1.27	2	2.53
Eurypyga	1	1.27	3	3.80
Gampsonyx	2	2.53	5	6.33
Gavia	1	1.27	6	7.59
Megalaima	3	3.80	9	11.39
Momotus	1	1.27	10	12.66
Motmotus	2	2.53	12	15.19
Nyctibius	1	1.27	13	16.46
Phoenicopterus	2	2.53	15	18.99
Podiceps	2	2.53	17	21.52
Psittacula	4	5.06	21	26.58
Pterocles	2	2.53	23	29.11
Rhinopomastus	3	3.80	26	32.91
Sphyrapicus	2	2.53	28	35.44
Tauraco	1	1.27	29	36.71
Treron	3	3.80	32	40.51
Tyto	1	1.27	33	41.77
Urocolius	2	2.53	35	44.30
Balaeniceps	3	3.80	38	48.10
Cathartes	1	1.27	39	49.37
Colibri	3	3.80	42	53.16
Eurypyga	2	2.53	44	55.70
Gampsonyx	1	1.27	45	56.96
Megalaima	8	10.13	53	67.09

ARRAYS (Chapter 13)

Arrays have multiple uses. One use is to convert data sets from wide to narrow, for example when you have repeated measures data.

We'll first cover arrays, and then go over ways to convert data sets from wide to narrow and vice versa, using either arrays within data steps or using PROC TRANSPOSE.

ARRAYS

From the book: "Cody's rule of SAS programming goes something like this: if you are writing a SAS program, and it is becoming very tedious, stop. There is a good chance that there is a SAS tool, perhaps arrays or macros, that will make your task less tedious."

I think I would add that it is also important for your code to be understandable to you, so that if writing fancier code saves a few lines of code and a little bit of tedium, but means that you won't understand your own code one year later, it might be worth having more tedious but more understandable code.

At the same time, if your job calls for a lot of SAS programming, then you (should) want to improve your skills as a SAS programmer, and this might involve figuring out more than one way to do things. Doing something a more difficult way might not be useful for one project but could turn out useful for a project in the future.

How much SAS should you know?

There isn't a good answer to this—it will depend on your job and access to SAS quite a bit.

I had an internship in the pharmaceutical industry about 10 years ago. In one department (at one site), there were about 50 PhD statisticians (over 200 PhD statisticians in the company as a whole). For the department with 50 PhD statisticians, there were about 30 SAS programmers who were not statisticians, but who were there to provide programming support to the statisticians.

In an environment like this, it is possible for the statistician to concentrate on statistical issues — modeling, analysis, etc. — instead of all of the detailed programming. Still, the more SAS you know, the better you can communicate what you need. In other environments, however, you might be expected the local SAS expert...

Back to ARRAYS

Usually programming languages use arrays to store data.

For SAS, the array is a collection of variables in a certain order, and you can refer to the variable by indexing the list of variables instead of referring to variables by their name. This can be especially useful if you want to perform the same operation on multiple columns.

The main reason for this is to save time—it is less tedious to loop over your variables in an array rather than refer to all of them individually.

ARRAYS: temperature data

As an example, we'll use some temperature data online on average US temperatures and some other information regarding precipitation. The data has four columns for average temperature for January, April, July, and October, using the Farenheit scale.

First, I'll discuss some difficulties reading in the data. The data was obtained from this website
`http://www.infoplease.com/ipa/A0762183.html`

Screenshot of data and first attempt to read in

```
[jamdeg@vulcan SAS]$ cat city.sas
options nodate;

filename foo url "http://math.unm.edu/~james/citytemp.txt";

data city;
  infile foo dlm="09"x;
  input city :$50. jan apr jul oct rain days snow years :10.;
run;

proc print data=city;
run;
```

```
Albany, N.Y.    22.2    46.6    71.1    49.3    38.60   136     64.4    57
Albuquerque, N.M.       35.7    55.6    78.5    57.3    9.47    60      11.0    64
Anchorage, Alaska       15.8    36.3    58.4    34.1    16.08   115     70.8    39 / 60
Asheville, N.C.         35.8    54.1    73.0    55.2    47.07   126     15.3    39
Atlanta, Ga.            42.7    61.6    80.0    62.8    50.20   115     2.1     69 / 65
Atlantic City, N.J.     32.1    50.6    75.3    55.1    40.59   113     16.2    60 / 54
Austin, Texas   50.2    68.3    84.2    70.6    33.65   85      0.9     62 / 58
Baltimore, Md.          32.3    53.2    76.5    55.4    41.94   115     21.5    53
Baton Rouge, La.        50.1    66.6    81.7    68.1    63.08   110     0.2     52 / 46
```

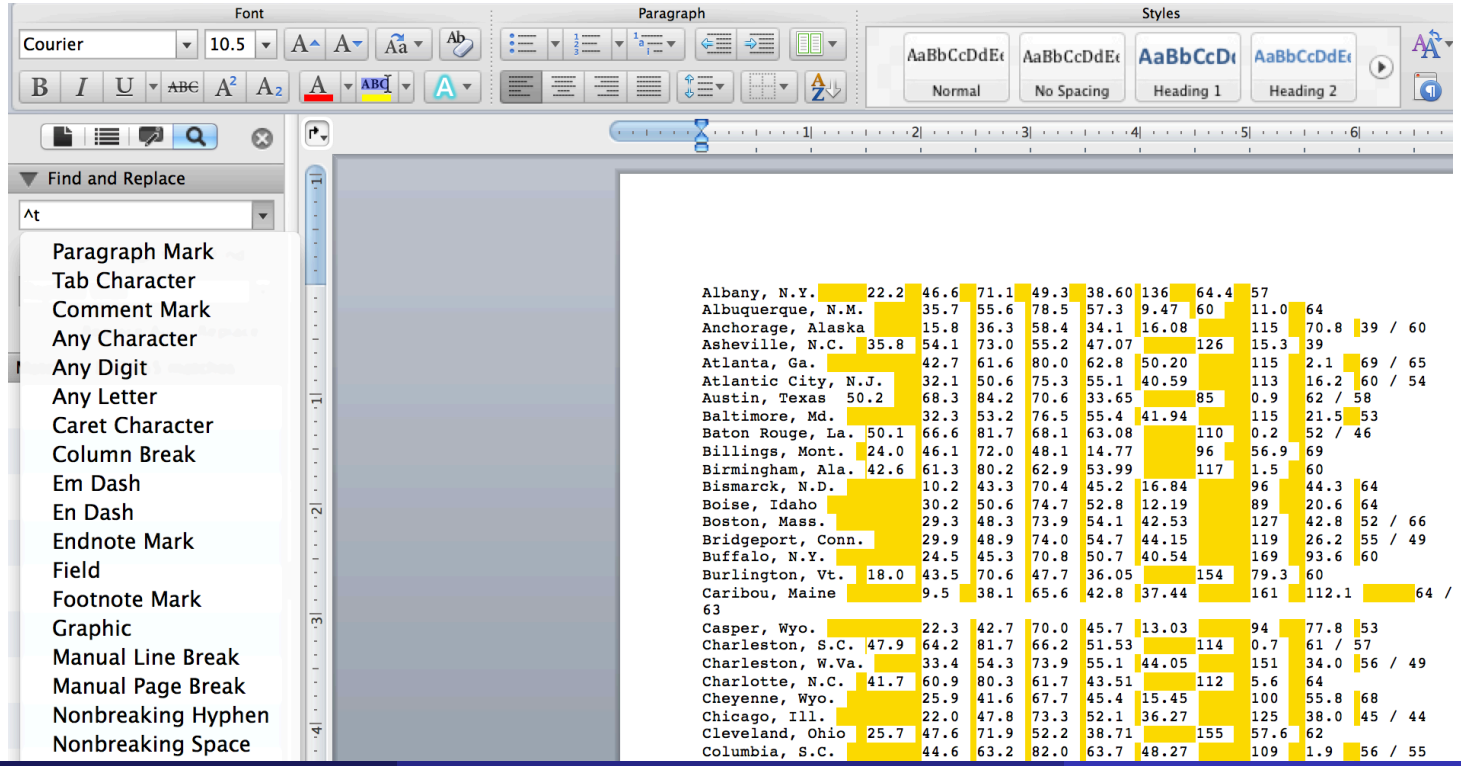
ARRAYS: temperature data

Trouble with tabs....Notice that the line for Austin is incorrect in PROC PRINT

Obs	city	jan	apr	jul	oct	rain	days	snow	years
1	Albany, N.Y.	22.2	46.6	71.1	49.30	38.60	136.0	64.4	57
2	Albuquerque, N.M.	35.7	55.6	78.5	57.30	9.47	60.0	11.0	64
3	Anchorage, Alaska	15.8	36.3	58.4	34.10	16.08	115.0	70.8	.
4	Asheville, N.C.	35.8	54.1	73.0	55.20	47.07	126.0	15.3	39
5	Atlanta, Ga.	42.7	61.6	80.0	62.80	50.20	115.0	2.1	.
6	Atlantic City, N.J.	32.1	50.6	75.3	55.10	40.59	113.0	16.2	.
7	Austin, Texas 50.2	68.3	84.2	70.6	33.65	85.00	0.9	.	.
8	Baton Rouge, La.	50.1	66.6	81.7	68.10	63.08	110.0	0.2	.
9	Billings, Mont.	24.0	46.1	72.0	48.10	14.77	96.0	56.9	69
10	Birmingham, Ala.	42.6	61.3	80.2	62.90	53.99	117.0	1.5	60

ARRAYS: temperature data

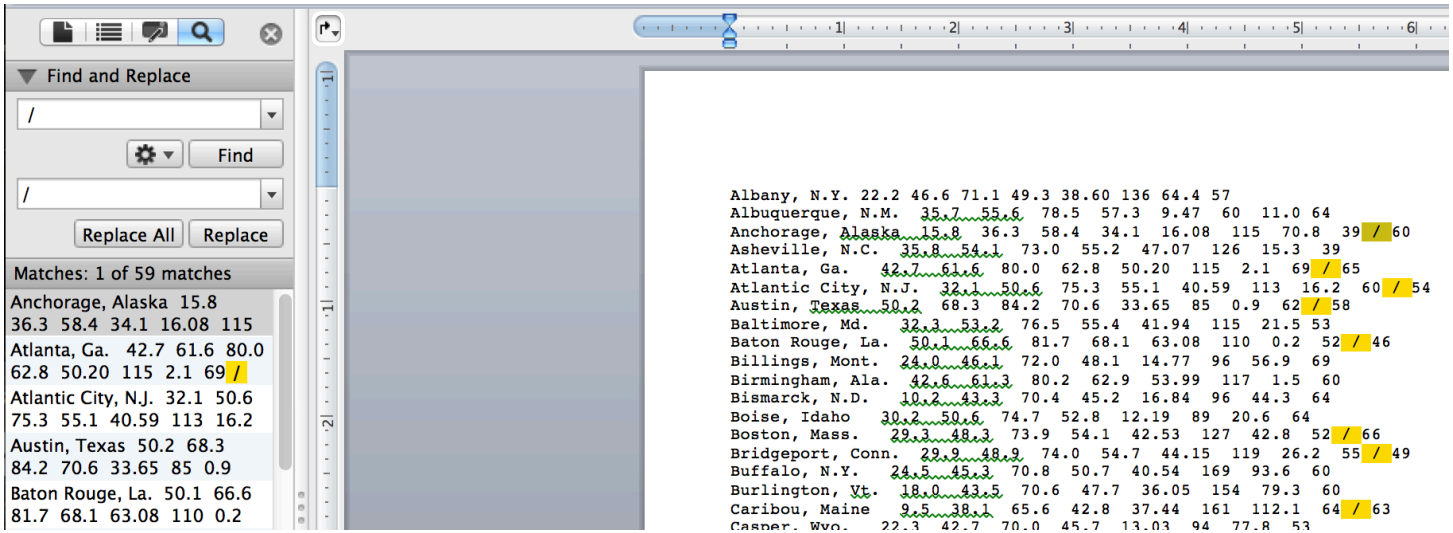
Find and replace tabs with two spaces (two prevent ending up with exactly one space separating two variables). We'll read this in using `dlmstr` and using two spaces as the delimiter.



ARRAYS: temperature data

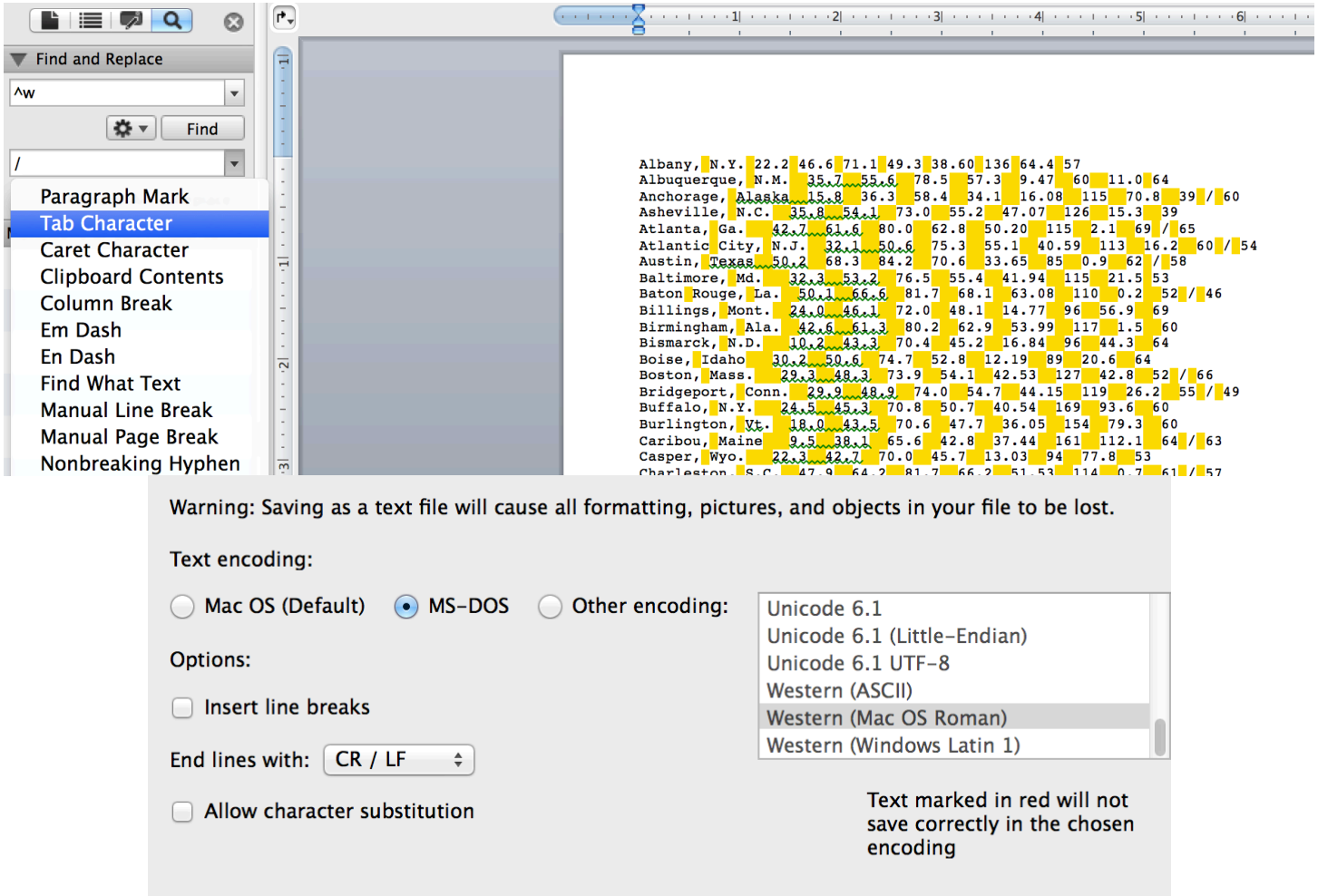
Also remove spaces around slash in case that causes problems

Replace " / " with "/"



ARRAYS: temperature data

After some trial and error, I saved the file this way (not the default).



ARRAYS: temperature data

Success at last!

```
filename foo url "http://math.unm.edu/~james/city2.txt";

data city;
  infile foo dlmstr="  "; /* two spaces */
  input city :$50. jan apr jul oct rain days snow years :$10.;
run;

proc print data=city;
run;
```

The SAS System

Obs	city	jan	apr	jul	oct	rain	days	snow	years
1	Albany, N.Y.	22.2	46.6	71.1	49.3	38.60	136	64.4	57
2	Albuquerque, N.M.	35.7	55.6	78.5	57.3	9.47	60	11.0	64
3	Anchorage, Alaska	15.8	36.3	58.4	34.1	16.08	115	70.8	39/60
4	Asheville, N.C.	35.8	54.1	73.0	55.2	47.07	126	15.3	39
5	Atlanta, Ga.	42.7	61.6	80.0	62.8	50.20	115	2.1	69/65
6	Atlantic City, N.J.	32.1	50.6	75.3	55.1	40.59	113	16.2	60/54
7	Austin, Texas	50.2	68.3	84.2	70.6	33.65	85	0.9	62/58
8	Baltimore, Md.	32.3	53.2	76.5	55.4	41.94	115	21.5	53
9	Baton Rouge, La.	50.1	66.6	81.7	68.1	63.08	110	0.2	52/46
10	Billings, Mont.	24.0	46.1	72.0	48.1	14.77	96	56.9	69

ARRAYS: temperature data

Suppose we wanted to convert the temperatures to Celsius. This could be done by typing

```
jan = (jan-32)*5/9
apr = (apr-32)*5/9
aug = (aug-32)*5/9
oct = (oct-32)*5/9
```

in the data step. This is only slightly tedious. It would be more tedious if we had 12 months and one column per month. Or suppose we have a questionnaire with 100 questions on a Likert scale (1=strongly disagree, 5=strongly agree, 99=missing) and we want to recode missing values as periods?

ARRAYS: temperature data

A way to automate applying the same code to many variables is to use arrays.

```
filename foo url "http://math.unm.edu/~james/city2.txt";

data city;
  infile foo dlmstr="  "; /* two spaces */
  input city :$50. jan apr jul oct rain days snow years :$10.;
run;

data cityC;
  set city;
  array temps{4} jan apr jul oct;
  do i=1 to 4;
    temps{i} = (temps{i}-32)*5/9;
  end;
  drop i;
run;

proc print data=cityC; run;
```

ARRAYS: temperature data

Now everything is Celsius

Obs	city	jan	apr
1	Albany, N.Y.	-5.4444	8.1111
2	Albuquerque, N.M.	2.0556	13.1111
3	Anchorage, Alaska	-9.0000	2.3889
4	Asheville, N.C.	2.1111	12.2778
5	Atlanta, Ga.	5.9444	16.4444
6	Atlantic City, N.J.	0.0556	10.3333
7	Austin, Texas	10.1111	20.1667
8	Baltimore, Md.	0.1667	11.7778
9	Baton Rouge, La.	10.0556	19.2222
10	Billings, Mont.	-4.4444	7.8333

ARRAYS: temperature data

instead of writing out all variables, you can use some abbreviations, such as

```
array temps{4} jan -- oct; /* for all variables starting
         with jan going up to oct in the order they appear */
array x{*} _numeric_; /* for all numeric variables */
array variables{*} $_character_ /* all character variables */
array Q{100} $ Q1-Q20 /* single hyphen indicates that
               variables  Q1, Q2, ...., Q20 are used even if
               other variables exist in between */
```

You can also use other characters instead of braces for arrays, such as square brackets or parentheses, but it is good to be consistent.

ARRAYS

A common application of arrays is to convert missing value codes. Data prepared for SPSS (often used in Psychology, for example), often uses 99 or 999 as a missing value code. To convert this for a long list of variables in a questionnaire, you can use

```
data new;
  set dataSPSS;
  array myvars{*}  _all_;
  do i = 1 to dim(myvars);  /* length of array */
    if myvars{i} = 999 then myvars{i} = .;
  end;
  drop i; /* no need to keep index variable */
run;
```

An alternative is to use

```
if myvars{i} = 999 then call missing(myvars{i});
```

ARRAYS

If there are multiple missing value codes, you can use IN as a special character function:

```
data new;
  set dataSPSS;
  array myvars{*}  _all_;
  do i = 1 to dim(myvars);  /* length of array */
    if myvars{i} in (NA,?,999) then call missing myvars{i};
  end;
  drop i; /* no need to keep index variable */
run;
```

ARRAYS

Another common use of arrays to clean up your data is to convert all character data to lower case across all variables. For the crime data, we had only one variable (`city`) that needed to be standardized in terms of capitalization, but in general, you might have many variables that need to be standardized.
Here is code for that

```
data lower;
  set old_data;
  array all_chars{*} _character_;
  do i = 1 to dim(all_chars);
    all_chars{i} = lowcase(all_chars{i});
   end;
  drop i;
run;
```

ARRAYS: creating new variables

You can also specify an array of variables that are not based on old data, and are assigned values during the data step. If we wanted both Celsius and Fahrenheit temperatures, for example, we can do the following.

ARRAYS: creating new variables

```
filename foo url "http://math.unm.edu/~james/city2.txt";

data city;
  infile foo dlmstr="  "; /* two spaces */
  input city :$50. jan apr jul oct rain days snow years :$10.;
run;

data cityC;
  set city;
  array temps{4} jan -- oct;
  array C{4};
  do i=1 to 4;
    C{i} = (temps{i}-32)*5/9;
  end;

  drop i;
run;

proc print data=cityC;
  var city jan--oct C1-C4;
run;
```

ARRAYS: creating new variables

Obs	city	jan	apr	jul	oct	C1	C2	C3	C4
1	Albany, N.Y.	22.2	46.6	71.1	49.3	-5.4444	8.1111	21.7222	9.6111
2	Albuquerque, N.M.	35.7	55.6	78.5	57.3	2.0556	13.1111	25.8333	14.0556
3	Anchorage, Alaska	15.8	36.3	58.4	34.1	-9.0000	2.3889	14.6667	1.1667
4	Asheville, N.C.	35.8	54.1	73.0	55.2	2.1111	12.2778	22.7778	12.8889
5	Atlanta, Ga.	42.7	61.6	80.0	62.8	5.9444	16.4444	26.6667	17.1111
6	Atlantic City, N.J.	32.1	50.6	75.3	55.1	0.0556	10.3333	24.0556	12.8333
7	Austin, Texas	50.2	68.3	84.2	70.6	10.1111	20.1667	29.0000	21.4444
8	Baltimore, Md.	32.3	53.2	76.5	55.4	0.1667	11.7778	24.7222	13.0000
9	Baton Rouge, La.	50.1	66.6	81.7	68.1	10.0556	19.2222	27.6111	20.0556
10	Billings, Mont.	24.0	46.1	72.0	48.1	-4.4444	7.8333	22.2222	8.9444
11	Birmingham, Ala.	42.6	61.3	80.2	62.9	5.8889	16.2778	26.7778	17.1667

ARRAYS: creating new variables

Note that this created new variables `C1, C2, ...` without specifying the names. This could also be a way to shorten annoyingly long variable names, especially if they don't mean much to you (you are analyzing someone else's data...) The purpose is just to save you some typing (and typos) in later code.

```
data cleanup;
  set messy;
  array annoying{100} LongVariableName1-LongVariableName100;
  array v{100};
  do i=1 to 100;
     v{i} = annoying{i};
  end;
  drop LongVariableName1-LongVariableName100;
run;
```

Array bounds

You can also change bounds of arrays so that instead of having the indexing start at 1, it starts at some other number. For example if your data has variables `rain10, rain11, rain12, rain13` for rainfall in 2010, 2011, ..., 2013, you can use

```
array rain{10:13} rain10-rain13;
```

This can help prevent typos in your code. Indexing can also vary for different languages. For example, R indexes starting at 1, but C indexes starting at 0. This can cause a lot of off-by-one errors when switching between programming languages.

Temporary arrays

You can also create a temporary array that has no variable names using the keyword _TEMPORARY_. This essentially acts as a constant that can be used for comparison to any observation. The following code (from the book) stores an answer key in a temporary array to grade student answers.

Temporary arrays

```
data score;
  array ans{10} $ 1; /* the 1 is not needed but indicates that
                        each value is 1 byte */
  array key{10} $ 1 _temporary_
  ('A','B','C','D','E','E','D','C','B','A');
  input ID (Ans1-Ans10)($1.);
  RawScore = 0;
  do Ques = 1 to 10;
    RawScore + (key{Ques} eq ans{Ques});
  end;
  Percent = 100*RawScore/10;
  keep ID RawScore Percent;
datalines;
123 ABCDEDDDCA
126 ABCDEEDCBA
129 DBCBCEDDEB
;
run;
```

Two-dimensional arrays

You can do temporary two-dimensional arrays using the syntax

```
array A{3,4} _temporary_;
```

This can be useful for having a look-up table that is available for every observation. Here you need data to populate the array. As an example, you could have a table that gave distances between cities (or prices for airline trips). For a customer traveling between cities, the look up table would indicate the distance for the trip.

Converting a data set from one observation per subject to one observation per visit

It is common in medical data to have one record per clinic or hospital visit so that the same patient has multiple records, or to have one record per patient, with multiple variables (for example, repeated measures, that need to be converted into one record per patient per time point.
Typical repeated measures data might look like this, where we have a patient with age and blood pressure reading at 4 time points.

```
patient  age bp1 bp2 bp3 bp4
0001 67 130 138 140 136
0002 61  150 145 144 142
0003 72  121 135 122 140
0004 51  118 115 126 120
```

Restructuring data: wide to narrow

Suppose we want the data to look like this

```
0001 67 130
0001 67 138
0001 67 140
0001 67 136
0002 61 150
0002 61 145
0002 61 144
0002 61 142
0003 72 121
...
```

ARRAYS: restructuring data, wide to narrow

```
data bp;
  infile "bp.txt" firstobs=2;
  input id $ age bp1 bp2 bp3 bp4;
run;

data bp2;
  set bp;
  array bparray{4} bp1-bp4;
  do i=1 to 4;
    bp = bparray{i};
    output;
  end;
  keep id age bp;
run;
```

Obs	id	age	bp
1	0001	67	130
2	0001	67	138
3	0001	67	140
4	0001	67	136
5	0002	61	150
6	0002	61	145
7	0002	61	144
8	0002	61	142
9	0003	72	121
10	0003	72	135
11	0003	72	122
12	0003	72	140
13	0004	51	118
14	0004	51	115
15	0004	51	126
16	0004	51	120

ARRAYS: restructuring data, wide to narrow

```
title "No arrays, no do loops";
data bp3;
  set bp;
  bp=bp1; output;
  bp=bp2; output;
  bp=bp3; output;
  bp=bp4; output;
  keep id age bp;
run;

proc print data=bp3; run;
```

No arrays, no do loops

Obs	id	age	bp
1	0001	67	130
2	0001	67	138
3	0001	67	140
4	0001	67	136
5	0002	61	150
6	0002	61	145
7	0002	61	144
8	0002	61	142
9	0003	72	121
10	0003	72	135
11	0003	72	122
12	0003	72	140
13	0004	51	118
14	0004	51	115
15	0004	51	126
16	0004	51	120

ARRAYS: restructuring data

For a small example like this, there isn't much difference between using an array or not. But the length of the code will not change if there are 12 or 100 blood pressure readings, while it would be tedious to do this without arrays and loops for so many readings.

Now, we'll look at going in the other direction, narrow to wide. So we'll assume we're starting with the data in the narrow format we just saw.

ARRAYS: restructuring data, narrow to wide

```
title "No arrays, no do loops";
data bp3;
  set bp;
  bp=bp1; output;
  bp=bp2; output;
  bp=bp3; output;
  bp=bp4; output;
  keep id age bp;
run;

title "SET statement within DO LOOP";
data bp4;
  array newbp{4};
  do i=1 to 4;
    set bp3;
    newbp{i} = bp;
  end;
  drop i bp;
run;

proc print data=bp4; run;
```

ARRAYS: restructuring data, narrow to wide

SET statement within DO LOOP

Obs	newbp1	newbp2	newbp3	newbp4	id	age
1	130	138	140	136	0001	67
2	150	145	144	142	0002	61
3	121	135	122	140	0003	72
4	118	115	126	120	0004	51

ARRAYS: restructuring data, narrow to wide

The book has an another solution, which also uses arrays but doesn't use a DO loop.

```
proc sort data=bp3 out=manyper; by id; run;

/* Book's approach for narrow to wide */

title "Books approach without DO LOOP";
data oneper;
  set manyper;
  by id;

  /* create counter to keep track of clinic visit */
  if first.id then time=1;
  else time+1;

  array BParray{4};
  retain BParray1-BParray4;

  /* check for missing values */
  if first.Subj then call missing(of BParray1-BParray4);

  BParray{time} = bp;
  if last.id then output;

run;
```

ARRAYS: restructuring data, narrow to wide

The book has an another solution, which also uses arrays but doesn't use a DO loop.

Books approach without DO LOOP

Obs	id	age	bp	time	BParray1	BParray2	BParray3	BParray4
1	0001	67	136	4	130	138	140	136
2	0002	61	142	4	150	145	144	142
3	0003	72	140	4	121	135	122	140
4	0004	51	120	4	118	115	126	120

ARRAYS: restructuring data, narrow to wide

I used the approach in the book, but adapted for this data and modified a bit. The book's solution assumes that the counter (called `time` in my code) is a variable in the narrow data set. Here I created it on the fly while also restructuring the data, and this works too.

The book's solution has the advantage that the order of the variables comes out in the order you might like. This can be fixed using LENGTH statements in the data step that has the DO LOOP.

The book's solution I think is slightly harder than mine, but works just fine. It is a good idea to change the code and see what happens. For example, what happens if the RETAIN is commented out?

Changing the code to understand it better

Here the RETAIN statement was commented out, making the variables reset to missing the next time you go through the data step, so only the last observation from each ID is output as non-missing.

Commenting out the RETAIN statement from the book's code

Books approach without DO LOOP

Obs	id	age	bp	time	BParray1	BParray2	BParray3	BParray4
1	0001	67	136	4	.	.	.	136
2	0002	61	142	4	.	.	.	142
3	0003	72	140	4	.	.	.	140
4	0004	51	120	4	.	.	.	120

ARRAYS: restructuring data, narrow to wide

```
title "SET statement within DO LOOP";
data bp4;
  length id $8 age 8.;
  array newbp{4};
  do i=1 to 4;
    set bp3;
    newbp{i} = bp;
  end;
  drop i bp;
run;

proc print data=bp4; run;
```

SET statement within DO LOOP

Obs	id	age	newbp1	newbp2	newbp3	newbp4
1	0001	67	130	138	140	136
2	0002	61	150	145	144	142
3	0003	72	121	135	122	140
4	0004	51	118	115	126	120

Changing the order of variables

Actually, it is not uncommon to want to change the order of variables in a dataset. One way to do this is with a LENGTH statement.

Changing the order of variables

```
title "SET statement within DO LOOP";
data bp4;
  length id $8 age 8.;
  array newbp{4};
  do i=1 to 4;
    set bp3;
    newbp{i} = bp;
  end;
  drop i bp;
run;

proc print data=bp4; run;

title "Rearranging variables with LENGTH STATEMENT";
data bp5;
  length newbp4 8. newbp3 8. newbp2 8. newbp1 8. age 8. id $8;
  set bp4;
run;
proc print data=bp5; run;


proc sort data=bp3 out=manyper; by id; run;
```

Changing the order of variables

SET statement within DO LOOP

Obs	id	age	newbp1	newbp2	newbp3	newbp4
1	0001	67	130	138	140	136
2	0002	61	150	145	144	142
3	0003	72	121	135	122	140
4	0004	51	118	115	126	120

Rearranging variables with LENGTH STATEMENT

Obs	newbp4	newbp3	newbp2	newbp1	age	id
1	136	140	138	130	67	0001
2	142	144	145	150	61	0002
3	140	122	135	121	72	0003
4	120	126	115	118	51	0004

Restructuring the data with PROC TRANSPOSE

Another way of changing data from multiple observations per patient to one observation per patient and vice versa is through PROC TRANSPOSE.

PROC TRANSPOSE

```
/* transposing the original data */
/* I assume data is sorted by id */
title "Using PROC TRANSPOSE";
proc transpose data=bp out=bp_Transpose1;
  by id;
  var bp1-bp4;
run;

proc print data=bp_Transpose1;
run;
```

Using PROC TRANSPOSE

Obs	id	_NAME_	COL1
1	0001	bp1	130
2	0001	bp2	138
3	0001	bp3	140
4	0001	bp4	136
5	0002	bp1	150
6	0002	bp2	145
7	0002	bp3	144
8	0002	bp4	142
9	0003	bp1	121
10	0003	bp2	135
11	0003	bp3	122
12	0003	bp4	140
13	0004	bp1	118
14	0004	bp2	115
15	0004	bp3	126
16	0004	bp4	120

PROC TRANSPOSE

The idea is that for each ID variable, the columns become rows and the rows become columns. For ID 0001, the data was a row which was 1x4, so now for ID 0001, the data is 4x1.

Note that the age variable was lost in the process, so we'll have to do something to get it back.

The variable names have now become a column of data. Usually, you wouldn't want to keep this (although sometimes you might want to), and instead you would want `COL1` to be called blood pressure.

PROC TRANSPOSE: wide to narrow

```
/* transposing the original data */
/* I assume data is sorted by id */
title "Using PROC TRANSPOSE";
proc transpose data=bp
                out=bp_Transpose1(rename=(col1=BP) drop=_name_);
  by id age;
  var bp1-bp4;
run;

proc print data=bp_Transpose1;
run;
```

Using PROC TRANSPOSE

Obs	id	age	BP
1	0001	67	130
2	0001	67	138
3	0001	67	140
4	0001	67	136
5	0002	61	150
6	0002	61	145
7	0002	61	144
8	0002	61	142
9	0003	72	121
10	0003	72	135
11	0003	72	122
12	0003	72	140
13	0004	51	118
14	0004	51	115
15	0004	51	126
16	0004	51	120

PROC TRANSPOSE

If you have missing data due to missing observations at some time points, then this will show up as periods in the wide data, but you might want these rows to be deleted. For example,

```
0001 67 130
0001 67 138
0001 67 140
0001 67   .
0002 61 150
0002 61 145
0002 61 144
0002 61 142
```

To have the row deleted, so that the number of rows is equal to the number of times the patient received a measurement, use

```
out=bo_Transpose1(rename(col1=BP) drop=_name_
       where BP is not null);
```

PROC TRANSPOSE: narrow to wide

```
proc transpose data=bp_Transpose1
                out=bp_Transpose2;
  by id;
  var BP;
run;

proc print data=bp_Transpose2; run;
```

Using PROC TRANSPOSE

Obs	id	_NAME_	COL1	COL2	COL3	COL4
1	0001	BP	130	138	140	136
2	0002	BP	150	145	144	142
3	0003	BP	121	135	122	140
4	0004	BP	118	115	126	120

PROC TRANSPOSE: a whole dataset

To transpose an entire dataset, you can omit the BY statement:

```
filename foo url "http://math.unm.edu/~james/citytemp.txt";

data city;
  infile foo dlm="09"x firstobs=1 obs=5;
  input city :$50. jan apr jul oct rain days snow years :10.;
run;

proc transpose data=city out=city2;
  var jan apr jul oct rain days;
run;

proc print data=city2; run;
```

The SAS System

Obs	_NAME_	COL1	COL2	COL3	COL4	COL5
1	jan	22.2	35.70	15.80	35.80	42.7
2	apr	46.6	55.60	36.30	54.10	61.6
3	jul	71.1	78.50	58.40	73.00	80.0
4	oct	49.3	57.30	34.10	55.20	62.8
5	rain	38.6	9.47	16.08	47.07	50.2
6	days	136.0	60.00	115.00	126.00	115.0