How can I form all possible pairs in my dataset?

Authored by

stats writer

June 30, 2024

To form all possible pairs in a given dataset, one can utilize a combination of mathematical and computational techniques. This involves identifying all unique data points in the dataset and then using a combination formula to determine the number of possible pairs. The data points can then be paired using a systematic approach, such as sorting or shuffling, to ensure all pairs are formed without repetition. Additionally, utilizing programming languages or software tools can aid in automating the pairing process and handling large datasets efficiently. This approach allows for a comprehensive exploration of the dataset and can provide valuable insights for further analysis and decision-making.

How can I form all possible pairs in my dataset? | SPSS FAQ

Data analysis sometimes requires a dataset that contains all possible pairs of cases in your current dataset. This is particularly common when analyzing dyadic data. For example, if you have a group of children, you may want to form a dataset that pairs each child with each other child in the dataset so that you can study interaction at the dyad level. A more complex case is one in which you want to form all possible pairs within some set of groups in your data. For example, the children may be from different classrooms and you may wish to form pairs of students only within classroom.

Below we show two examples of creating a dataset of matched pairs. The first example creates all possible pairs in the dataset. The second example extends the first by forming all possible pairs within groups defined by a variable in our dataset. The examples show how to match pairs, as well as how to "clean up" the results, for example, by removing duplicate pairs (e.g. case 1 matched with case 2, and case 2 matched with case 1).

Example 1: All possible pairs in the dataset

We will start with a simple example dataset, there are 8 cases, and three variables. The variable case is the case id, the other two variables (v1 and v2) contain information about each case.

```
data list list /case v1 v2.
begin data
1001 12 0
1002 23 1
```

1003 22 0
1004 5 0
1005 12 0
1006 23 1
1007 22 0
1008 5 0
end data.

Although our dataset already contains a unique identifier (case) we
need to create a new variable id because in
order for the syntax below to work, the id numbers must be numeric and
continuous, starting with one and going to n, where n is the
number of cases. If
this is true in your data you can skip this step. Otherwise you can create a new variable
id as shown below (if your
current id variable is named id rename it to something like id_old before you
start). The $casenum indicates that the value of id for each case should

be equal to its row number.

```
compute id = $casenum.
exe.
save outfile ='d:datastart.sav'.
```

The code below creates a new variable, one, equal to 1 for each
case. This variable is then used with as the break variable in the
aggregate command. The result is that we have a variable ncase that
is equal to the number of cases (8).

```
compute one = 1.
exe.
aggregate
/outfile = * mode = addvariables
/break = one
/ncase = NU(one).
```

The first line of syntax below begins a loop. A loop tells SPSS to repeat a
series of actions until some criterion is met. In this

**case, we want to
repeat the action 8 times (i.e. once for each case in the dataset),
we use the value of ncase to tell SPSS the number of times we want to
repeat the process. Each time we repeat the loop, the macro variable cnt is set
to a different value (1-8). We use compute nid = cnt to create a new variable
nid and save the value of that variable, and the variable id for each case into a dataset
using xsave. The command xsave adds new cases to a new dataset (all.sav)
each time the loop runs, so that the new dataset gets 8 new cases each time our loop runs.**

```
loop cnt =1 to ncase.
compute nid = cnt.
xsave outfile = 'd:dataall.sav' /keep = id nid.
end loop.
exe.
```

A listing of the resulting dataset is shown below using the list command

(some of the output is omitted to save space). The dataset has two variables,
id and nid, and contains every possible combination of id
numbers (i.e. id and nid). The output also shows the number of cases in the dataset, in this case, 64, which is expected since 8×8=64. (Note that we list the data many times in this page,
this is to show the changes in the data, but it is not necessary to list the
data for the syntax to work.)

```
get file='d:dataall.sav'.
list .

id nid

1.00 1.00
1.00 2.00
1.00 3.00
1.00 4.00
1.00 5.00
1.00 6.00
1.00 7.00
```

1.00 8.00
2.00 1.00
2.00 2.00
2.00 3.00
<output omitted>
8.00 6.00
8.00 7.00
8.00 8.00

Number of cases read: 64 Number of cases listed: 64

The dataset above contains all possible combinations of id numbers, but it does not contain any additional variables. In order to get the variables associated with each case into this new dataset, we will have to merge the datasets. This will actually require two merges, once to match to id, and once to match to nid so that the variables from each member of the pair are in that row. In order to run the first merge, we have to sort both our starting dataset (start.sav), and the new

dataset containing all possible pairs (all.sav), by the variable id.
Once we have sorted both datasets, we can use the match files command to
match the two datasets. This is a one to many merge, since our original dataset
had only one row for each id, and our new dataset has 8 rows for each
id. In order to perform the one-to-many merge properly, we need to be sure
that the dataset listed on the file line (file='d:dataall.sav') is the dataset that has multiple rows for
each id and that the dataset listed on the table line (/table='d:datastart.sav')
is the dataset where each id occurs only once.

```
get file='d:datastart.sav'.
sort cases by id.
save outfile ='d:datastart.sav'.

get file = 'd:dataall.sav' .
sort cases by id.
save outfile='d:dataall.sav'.
```

```
match files file='d:dataall.sav'
/table='d:datastart.sav'
/by id.
exe.
```

Now we have the data for the first member of the pair (i.e. those identified by the variable id), but we also need the data for the second member of the pair (i.e. those identified by nid). To do this, we repeat the merging process, with a few small changes.

First we sort the dataset all.sav by nid. Then we open our starting dataset (start.sav) and rename the variables we want to merge. We change id to nid so that we can match it with nid in the other dataset. At the same time we rename all other variables that we want in the dataset, in this example we change the name by adding an n to it so that v1 becomes nv1. This is necessary so that we can distinguish the values from the first member of the pair (identified by id) from those for

**the second
member of the pair (identified by nid). After this we sort by nid, and
save the file, then we use the match files command to merge the two datasets.**

```
sort cases by nid.
save outfile='d:dataall.sav'.

get file='d:datastart.sav'.
rename variables (id case v1 v2 = nid ncase nv1 nv2).
sort cases by nid.
save outfile ='d:datastart_n.sav'.

match files file='d:dataall.sav'
/table='d:datastart_n.sav'
/by nid.
exe.
save outfile='d:dataall.sav'.
```

**Below we list the cases in the resulting dataset. The dataset contains three new variables
ncase, nv1 and nv2, these are equal to the values of case, v1,**

and v2 for the case identified in nid.

list.

id nid case v1 v2 ncase nv1 nv2

1.00 1.00 1001.00 12.00 .00 1001.00 12.00 .00
2.00 1.00 1002.00 23.00 1.00 1001.00 12.00 .00
3.00 1.00 1003.00 22.00 .00 1001.00 12.00 .00
4.00 1.00 1004.00 5.00 .00 1001.00 12.00 .00
5.00 1.00 1005.00 12.00 .00 1001.00 12.00 .00
6.00 1.00 1006.00 23.00 1.00 1001.00 12.00 .00
7.00 1.00 1007.00 22.00 .00 1001.00 12.00 .00
8.00 1.00 1008.00 5.00 .00 1001.00 12.00 .00
1.00 2.00 1001.00 12.00 .00 1002.00 23.00 1.00
2.00 2.00 1002.00 23.00 1.00 1002.00 23.00 1.00
<output omitted>
6.00 8.00 1006.00 23.00 1.00 1008.00 5.00 .00
7.00 8.00 1007.00 22.00 .00 1008.00 5.00 .00
8.00 8.00 1008.00 5.00 .00 1008.00 5.00 .00

Number of cases read: 64 Number of cases listed: 64

The dataset we have now contains all possible pairs in the dataset, this includes

id=1 matched with nid=1 (itself), and both id=5 with nid=3 and id=3 with nid=5. For some
purposes this may be desirable, for others, one or both of these conditions may be
problematic. The first line of code below keeps only those cases where id and
nid are
not equal, that is, it drops individuals matched with themselves. Finding unique pairs takes a bit more work. First we
need to create an id for each unique pair (pairid), we want each pairid to
exist twice in our dataset, for example, once for id=2 and nid=8, and once for
id=8 and nid=2. To combine them we multiply the smaller of the two by 100, and
then add the larger. Now we have a unique id, for example, pairid=208
(=2*100+8). This is done in the second and third lines of syntax below. (Note that if you have more than 99 cases, you will need to
increase the multiplier from 100 to 1000, or even higher.)

```
select if id~=nid.

if (id<nid) pairid=id*100+nid.
if (id>nid) pairid=nid*100+id.
exe.
```

Next we sort cases by pairid and create a variable flag that is equal to
0 for each case. Then we change flag to 1 if the pairid for the case is
equal to the pairid in the previous case. Now for each value of
pairid, one case has flag=0 and the other has flag=1. The
select command keeps only those cases where flag=1, dropping the
redundant cases.

```
sort cases by pairid.
compute flag = 0.
if pairid = lag(pairid) flag = 1.
exe.
select if flag=1.
exe.
```

Below we list the dataset after removal of the cases matched with themselves and duplicate pairs.

list

id nid case v1 v2 ncase nv1 nv2 pairid flag

1.00 2.00 1001.00 12.00 .00 1002.00 23.00 1.00 102.00 1.00
1.00 3.00 1001.00 12.00 .00 1003.00 22.00 .00 103.00 1.00
1.00 4.00 1001.00 12.00 .00 1004.00 5.00 .00 104.00 1.00
1.00 5.00 1001.00 12.00 .00 1005.00 12.00 .00 105.00 1.00
1.00 6.00 1001.00 12.00 .00 1006.00 23.00 1.00 106.00 1.00
1.00 7.00 1001.00 12.00 .00 1007.00 22.00 .00 107.00 1.00
1.00 8.00 1001.00 12.00 .00 1008.00 5.00 .00 108.00 1.00
2.00 3.00 1002.00 23.00 1.00 1003.00 22.00 .00 203.00 1.00
<output omitted>
6.00 7.00 1006.00 23.00 1.00 1007.00 22.00 .00 607.00 1.00
6.00 8.00 1006.00 23.00 1.00 1008.00 5.00 .00 608.00 1.00
7.00 8.00 1007.00 22.00 .00 1008.00 5.00 .00 708.00 1.00

Number of cases read: 28 Number of cases listed: 28

Example 2: All possible pairs within groups.

In this example we match pairs only within group. As mentioned above, one possible application of this would be matching students only with other students from the same classroom. In this example the starting dataset contains an additional variable, group which indicates which group each of the cases is in. The unique identifier for each case is the variable case.

```
data list list /group case v1 v2.
begin data
1 1 12 0
1 2 23 1
1 3 22 0
5 4 5 0
5 5 7 0
5 6 13 1
3 7 1 0
3 8 56 1
end data.

save outfile='d:datastart.sav'.
```

If the id variable in the dataset you are working with is named id, you should rename it before you begin, ours is named case so this is not a problem. The syntax below requires an id variable that takes on the values 1 to n within each group (where n is the number of cases in that group). The code syntax creates a new variable, id, that meets this requirement. First, the cases are sorted by group, then a new variable called id that is equal to 1 is created. The next line of syntax (starting with if) checks to see if this case is in the same group as the case before it, if it is, then id is changed to the value of id in the case before it plus 1, otherwise id=1. Finally, we save the file and use the list command to show the data.

```
sort cases by group.
compute id = 1.
if group = lag(group) id = lag(id) + 1.
```

exe.
save outfile='d:datastart.sav'.
list.

group case v1 v2 id

1.00 1.00 12.00 .00 1.00
1.00 2.00 23.00 1.00 2.00
1.00 3.00 22.00 .00 3.00
3.00 7.00 1.00 .00 1.00
3.00 8.00 56.00 1.00 2.00
5.00 4.00 5.00 .00 1.00
5.00 5.00 7.00 .00 2.00
5.00 6.00 13.00 1.00 3.00

Number of cases read: 8 Number of cases listed: 8

The syntax below creates a dataset of all possible pairs. First it creates a new variable, one, equal to one for each
case. The variables group and one are used as the break variables in the
aggregate command. The result is that we have a variable ncase that
is equal to the number of cases in each group. Below

the aggregate command is a
loop. A loop tells SPSS to repeat a series of actions until some criterion is
met. In this case, we want to repeat the action for each case in the subject's
group, and we use the value of ncase to tell SPSS the number
of times we want to repeat the process. Each time we repeat the loop, the
variable cnt is set to a different value. We use compute nid = cnt
to create a new variable, nid, and save the value of that variable, along
with the
variable id into a dataset using xsave. The command xsave
adds new cases to the dataset (all_by_group.sav) each time the loop
runs.

```
compute one = 1.
exe.
aggregate
/outfile = * mode = addvariables
```

```
/break = group one
/ncase = NU(one).

loop cnt =1 to ncase.
compute nid = cnt.
xsave outfile = 'd:dataall_by_group.sav' /keep = id nid group.
end loop.
exe.

get file = 'd:dataall_by_group.sav' .
list

id nid group

1.00 1.00 1.00
1.00 2.00 1.00
1.00 3.00 1.00
2.00 1.00 1.00
2.00 2.00 1.00
2.00 3.00 1.00
3.00 1.00 1.00
3.00 2.00 1.00
3.00 3.00 1.00
1.00 1.00 3.00
```

1.00 2.00 3.00
<output omitted>
3.00 1.00 5.00
3.00 2.00 5.00
3.00 3.00 5.00

Number of cases read: 22 Number of cases listed: 22

Now we match files in order to add the other variables in our dataset to the
matched dataset, just as we did before, except that cases are matched by both
group and id (or group and nid).

```
get file='d:datastart.sav'.
sort cases by group id.
save outfile ='d:datastart.sav'.

get file = 'd:dataall_by_group.sav' .
sort cases by group id.
save outfile='d:dataall_by_group.sav'.

match files file='d:dataall_by_group.sav'
/table='d:datastart.sav'
/by group id.
```

```
exe.
sort cases by group nid.
save outfile='d:dataall_by_group.sav'.

get file='d:datastart.sav'.
rename variables (id v1 v2 = nid nv1 nv2).
sort cases by group nid.
save outfile ='d:datastart_n.sav'.

match files file='d:dataall_by_group.sav'
/table='d:datastart_n.sav'
/by group nid.
exe.
save outfile='d:dataall_by_group.sav'.
```

As before our dataset now includes individuals matched with themselves and possibly redundant pairs (e.g. 1 with 2, and 2 with 1). Depending on your research design, you may or may not want to include these cases in your dataset. Below we show how to remove these types of cases from the dataset. As before, dropping cases of an individual matched with themselves is relatively easy,

**the command is shown in
the first line of syntax below. Dropping redundant pairs is a little trickier
this time, because group must be taken into account. The group
variable needs to be included because all groups have an instance of id=1
with nid=2 and id=2 with nid=1, and we want to make sure we drop only one of
these in each group. In the above example, we multiplied the smaller of id
or nid by 100, and added the other, to create a unique id for each pair.
This time we do that, but we also multiply the group number by 10000 and add that
to the combined ids, this results in a unique identifier for each pair. (Note that
you may need to modify the multipliers by increasing them by powers of 10, e.g.
change 100 to 1000 and 100000 to 1000000, depending on how many cases there are in
each group.)**

select if id~=nid.

if (id<nid) pairid=group*10000+id*100+nid.
if (id>nid) pairid=group*10000+nid*100+id.
exe.

Now we identify duplicate cases by first sorting the dataset by pairid,
then creating a variable called flag that is equal to one for all cases.
Then we change flag to 1 if the pairid for the case is equal to the
pairid in the previous case. Now for each value of pairid, one case has flag=0
and the other has flag=1. The select command keeps only those cases where flag=1,
dropping the redundant cases. The final line of syntax below lists resulting the
dataset.

sort cases by pairid.
compute flag = 0.
if pairid = lag(pairid) flag = 1.
select if flag=1.
exe.
list.

id nid group case v1 v2 nv1 nv2 pairid flag

2.00 1.00 1.00 2.00 23.00 1.00 12.00 .00 1102.00 .00
3.00 1.00 1.00 3.00 22.00 .00 12.00 .00 1103.00 .00
3.00 2.00 1.00 3.00 22.00 .00 23.00 1.00 1203.00 .00
2.00 1.00 3.00 8.00 56.00 1.00 1.00 .00 3102.00 .00
2.00 1.00 5.00 5.00 7.00 .00 5.00 .00 5102.00 .00
3.00 1.00 5.00 6.00 13.00 1.00 5.00 .00 5103.00 .00
3.00 2.00 5.00 6.00 13.00 1.00 7.00 .00 5203.00 .00

Number of cases read: 7 Number of cases listed: 7