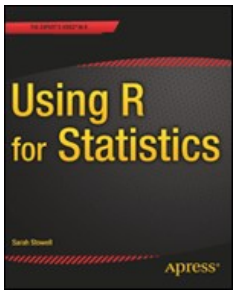

Using R for Statistics

by Sarah Stowell

Chapter 4: Combining and Restructuring Datasets

Overview

This chapter explains how to make major changes to the structure of a dataset. You may need to combine data from two or more different sources into one dataset. This may be as simple as attaching one dataset to the bottom or side of another (known as *appending* or *concatenating*), or it may involving using a common variable to match the observations (known as merging). Alternatively, you may need to combine the values from several variables into one variable (stacking) or split one variable into several variables (unstacking). Finally, you may need to reshape a dataset so that data that was once held in separate variables in now held in separate observations, or vice versa (also known as *rotating* or *transposing*).

You will learn how to:

- append rows
- append columns
- merge two datasets using a common variable
- stack a dataset
- unstack a dataset
- reshape a dataset

This chapter uses the `CIAdata1`, `CIAdata2`, `WHOdata`, `CPIdata`, `bigcats`, `endangered`, `grades1`, `resistance`, and `vitalsigns` datasets, which are available with the downloads for this book (www.apress.com/9781484201404). For more details about these datasets, see Appendix C. It also uses the `iris` dataset, which is included with R. To view more information about this dataset, enter `help(iris)`.

Appending Rows

The `rbind` function allows you to attach one dataset on to the bottom of the other, which is known as appending or concatenating the datasets. This is useful when you want to combine two datasets that contain different observations for the same variables, such as the `CIAdata1` and `CIAdata2` datasets shown in Figure 4-1a, b.

	country	lifeExp	urban	pcGDP
1	Finland	79.41	85	36700
2	Slovakia	76.03	55	23600
3	UK	80.17	80	36600
4	Ukraine	68.74	69	7300
5	Spain	81.27	77	31000

(a) CIAdata1

	country	pcGDP	lifeExp	urban
1	Italy	30900	81.86	68
2	Croatia	18400	75.99	58
3	Slovakia	23600	76.03	55

(b) CIAdata2

Figure 4-1: Data from the CIA World Factbook (see Appendix C for more details)

Before using the `rbind` function, make sure that each dataset contains the same number of variables and that all of the variable names match. You may need to remove or rename some variables first, as explained in the first two sections in Chapter 3. The variables do not need to be arranged in the same order within the datasets, as the `rbind` function automatically matches them by name.

Once the datasets are prepared, append them with the `rbind` function, as shown here for the `CIAdata1` and `CIAdata2` datasets:

```
> CIAdata<-rbind(CIAdata1, CIAdata2)
```

The new `CIAdata` dataset is shown in Figure 4-2. Notice that the new dataset contains all of the original data in the original order, including

two copies of the data for Slovakia. The `rbind` function does not identify duplicates or sort the data. You can do this with the `unique` and `order` functions as explained in Chapter 3 (under "Removing Duplicate Observations" and "Sorting a Dataset").

	country	lifeExp	urban	pcGDP
1	Finland	79.41	85	36700
2	Slovakia	76.03	55	23600
3	UK	80.17	80	36600
4	Ukraine	68.74	69	7300
5	Spain	81.27	77	31000

(a) CIAdata1

	country	pcGDP	lifeExp	urban
1	Italy	30900	81.86	68
2	Croatia	18400	75.99	58
3	Slovakia	23600	76.03	55

(b) CIAdata2

	country	lifeExp	urban	pcGDP
1	Finland	79.41	85	36700
2	Slovakia	76.03	55	23600
3	UK	80.17	80	36600
4	Ukraine	68.74	69	7300
5	Spain	81.27	77	31000
6	Italy	81.86	68	30900
7	Croatia	75.99	58	18400
8	Slovakia	76.03	55	23600

(c) CIAdata

Figure 4-2: The result of appending the `CIAdata1` and `CIAdata2` datasets to create the `CIAdata` dataset

You can append three or more datasets in a similar manner:

```
> newdataset<-rbind(dataset1, dataset2, dataset3)
```

Appending Columns

The `cbind` function pastes one dataset on to the side of another. This is useful if the data from corresponding rows of each dataset belong to the same observation, as is the case for the `CIAdata1` and `WHOdata` datasets shown in Figure 4-3a, b.

	country	lifeExp	urban	pcGDP
1	Finland	79.41	85	36700
2	Slovakia	76.03	55	23600
3	UK	80.17	80	36600
4	Ukraine	68.74	69	7300
5	Spain	81.27	77	31000

(a) CIAdata1

	alcohol	mortality
1	12.52	91
2	13.33	130
3	13.37	77
4	15.6	274
5	11.62	68

(b) WHOdata

Figure 4-3: Data from the CIA World Factbook and from the World Health Organisation (see Appendix C for more details)

You can only use the `cbind` function to combine datasets that have the same number of rows. If your datasets have a common variable or variables that can be used to match the observations, use the `merge` function to combine the datasets, as explained in the following section.

This command combines the `CIAdata1` and `WHOdata` datasets to create a new dataset called `CIAWHOdata;` the result is shown in Figure 4-4c:

```
> CIAWHOdata<-cbind(CIAdata1, WHOdata)
```

	country	lifeExp	urban	pcGDP
1	Finland	79.41	85	36700
2	Slovakia	76.03	55	23600
3	UK	80.17	80	36600
4	Ukraine	68.74	69	7300
5	Spain	81.27	77	31000

(a) CIAdata1

	alcohol	mortality
1	12.52	91
2	13.33	130
3	13.37	77
4	15.6	274
5	11.62	68

(b) WHOdata

	country	lifeExp	urban	pcGDP	alcohol	mortality
1	Finland	79.41	85	36700	12.52	91
2	Slovakia	76.03	55	23600	13.33	130
3	UK	80.17	80	36600	13.37	77
4	Ukraine	68.74	69	7300	15.6	274
5	Spain	81.27	77	31000	11.62	68

(c) CIAWHOdata

Figure 4-4: The result of appending the `CIAdata1` and `WHOdata` datasets to create the `CIAWHOdata` dataset

You can combine three or more datasets in a similar way:

```
> newdataset<-cbind(dataset1, dataset2, dataset3)
```

Merging Datasets by Common Variables

The `merge` function allows you to combine two datasets by matching the observations according to the values of common variables.

Consider the `CIAdata1` and `CPIdata` datasets shown in Figure 4-5. The datasets have a common variable called `country`, which can be used to match corresponding observations.

	country	lifeExp	urban	pcGDP
1	Finland	79.41	85	36700
2	Slovakia	76.03	55	23600
3	UK	80.17	80	36600
4	Ukraine	68.74	69	7300
5	Spain	81.27	77	31000

(a) CIAdata1

	country	CPI
1	Spain	80.24
2	UK	100.13
3	Croatia	67.54
4	Italy	94.82
5	Ukraine	51.1
6	Finland	99.69
7	Spain	80.24

(b) CPIdata1

Figure 4-5: Data from the CIA World Factbook and Numbeo (see Appendix C for more details)

The following command shows how you would combine the `CIAdata1` and `CPIdata` datasets; the result is shown in Figure 4-6:

```
> CIACPIdata<-merge(CIAdata1, CPIdata)
```

	country	lifeExp	urban	pcGDP
1	Finland	79.41	85	36700
2	Slovakia	76.03	55	23600
3	UK	80.17	80	36600
4	Ukraine	68.74	69	7300
5	Spain	81.27	77	31000

(a) CIAdata1

	country	CPI
1	Spain	80.24
2	UK	100.13
3	Croatia	67.54
4	Italy	94.82
5	Ukraine	51.1
6	Finland	99.69
7	Spain	80.24

(b) CPIdata

	country	lifeExp	urban	pcGDP	CPI
1	Finland	79.41	85	36700	99.69
2	Spain	81.27	77	31000	80.24
3	Spain	81.27	77	31000	80.24
4	UK	80.17	80	36600	100.13
5	Ukraine	68.74	69	7300	51.1

(c) CIACPIdata

Figure 4-6: The result of merging the `CIAdata1` and `CPIdata` datasets to create the `CIACPIdata` dataset

The `merge` function identifies variables with the same name and uses them to match up the observations. In this example, both datasets contain a variable named `country`, so R automatically uses this variable to match the observations. If your datasets have more than one common variable, R matches the observations by the unique combinations of all of the common variables.

If the names of the common variables are not identical in the two datasets, the simplest solution is to rename the variables, as explained in Chapter 3 (see "Renaming Variables"). Alternatively, you can use the `by.x` and `by.y` arguments to specify which variables to use for matching the observations. For example, to merge two datasets in which the common variable is named `var1` in the first dataset and `VAR1` in the second dataset, use the command:

```
newdataset<-merge(dataset1, dataset2, by.x="var1", by.y="VAR1")
```

When you combine two datasets with the `merge` function, R automatically excludes any unmatched observations that appear in only one of the datasets. In Figure 4-6, you can see that Slovakia (which appears only in `CIAdata1`) and Italy and Croatia (which appear only in `CPIdata`) are all excluded from the merged dataset.

The `all`, `all.x`, and `all.y` arguments allow you to control how R deals with any unmatched observations. To keep all unmatched observations, set the `all` argument to `T`:

```
> allCIACPIdata<-merge(CIAdata1, CPIdata, all=T)
```

The results are shown in Figure 4-7. Notice that Slovakia, Italy, and Croatia have now been included, and where corresponding data is missing, the missing data symbol `NA` has been substituted.

	country	lifeExp	urban	pcGDP
1	Finland	79.41	85	36700
2	Slovakia	76.03	55	23600
3	UK	80.17	80	36600
4	Ukraine	68.74	69	7300
5	Spain	81.27	77	31000

(a) CIAdata1

	country	CPI
1	Spain	80.24
2	UK	100.13
3	Croatia	67.54
4	Italy	94.82
5	Ukraine	51.1
6	Finland	99.69
7	Spain	80.24

(b) CIAdata

	country	lifeExp	urban	pcGDP	CPI
1	Finland	79.41	85	36700	99.69
2	Slovakia	76.03	55	23600	NA
3	Spain	81.27	77	31000	80.24
4	Spain	81.27	77	31000	80.24
5	UK	80.17	80	36600	100.13
6	Ukraine	68.74	69	7300	51.1
7	Croatia	NA	NA	NA	67.54
8	Italy	NA	NA	NA	94.82

(c) allCIACPIdata

Figure 4-7: The result of merging the `CIAdata` and `CPIdata` datasets with `all=T` to create the `allCIACPIdata` dataset

Alternatively, you can use the `all.x` and `all.y` arguments to include unmatched cases from just one of the datasets instead of both. So the command:

```
> allCIAGPIdata<-merge(CIAdata1, CPIdata, all.x=T)
```

includes the data for Slovakia but not Italy and Croatia, while the command:

```
> allCIAGPIdata<-merge(CIAdata1, CPIdata, all.y=T)
```

includes the data for Italy and Croatia but not Slovakia.

When there are multiple matches for an observation, R creates an observation in the final dataset for every possible match. This is known as the *Cartesian product*. Consider the `bigcats` and `endangered` datasets shown in Figure 4-8a and b. Both datasets have two observations with a name of 'Leopard'. When these two datasets are merged by the `Name` variable, the final dataset (shown in Figure 4-8c) contains four rows for the Leopard data.

	Name	Weight
1	Tiger	203
2	Leopard	64
3	Leopard	58
4	Lion	200

(a) bigcats

	Name	Status
1	Lion	Vulnerable
2	Leopard	Least Concern
3	Tiger	Endangered
4	Leopard	Endangered

(b) endangered

	Name	Weight	Status
1	Leopard	64	Least Concern
2	Leopard	64	Endangered
3	Leopard	58	Least Concern
4	Leopard	58	Endangered
5	Lion	200	Vulnerable
6	Tiger	203	Endangered

(c) Merged dataset

Figure 4-8: Datasets giving information about four big cat species (see Appendix C for more details)

To ensure that you get the desired result from a merge, make sure that you are using a common variable or variables that allow R to correctly identify matches. You may also need to remove duplicate observations beforehand, as described in Chapter 3 (see "Removing Duplicate Observations").

R automatically sorts the merged dataset by the same variables that were used to match the observations. If you want to prevent this, set the `sort` argument to `F`:

```
> newdataset<-merge(dataset1, dataset2, sort=F)
```

Stacking and Unstacking a Dataset

Stacking a dataset means to convert it from *unstacked* form to *stacked* form, while unstacking a dataset is the opposite procedure, that is, converting a dataset from stacked form to unstacked form. To illustrate the difference between the two forms, consider the `grades1` and `grades2` datasets shown in Figure 4-9. The `grades1` dataset is in unstacked form. It gives the test results of fifteen students, arranged in separate columns according to which class they belong. The `grades2` dataset gives the same data in stacked form. Here the data is arranged in two columns, the first giving the test result and the second identifying to which class the student belongs.

	ClassA	ClassB	ClassC
1	87	83	80
2	64	97	86
3	89	97	80
4	82	99	80
5	59	92	84

(a) grades1

	values	ind
1	87	ClassA
2	64	ClassA
3	89	ClassA
4	82	ClassA
5	59	ClassA
6	83	ClassB
7	97	ClassB
8	97	ClassB
9	99	ClassB
10	92	ClassB
11	80	ClassC
12	86	ClassC
13	80	ClassC
14	80	ClassC
15	84	ClassC

(b) grades2

Figure 4-9: The same data in stacked and unstacked form

Having your data in stacked form is useful if you want to summarize all of the data values together, or if you want to model the values against the grouping variable. Having your data in unstacked form is useful as an alternative to dividing your data into subsets when you want to analyze the values in each group separately.

Many commonly used R functions such as `aov` (for analysis of variance), `lm` (for regression and general linear models), and `glm` (for generalized linear models) require data in stacked form. Most of R's hypothesis testing functions such as `t.test` are able to handle data in either stacked or unstacked form.

You should only stack or unstack a dataset if the values are numeric, in no particular order, and if there is no correspondence between the values in each of the groups. For more complex situations, such as when there is an additional variable present that indicates which of the values in each group correspond to each other, refer to "Reshaping a dataset."

Stacking Data

To convert a dataset from unstacked to stacked form, use the `stack` function:

```
> grades2<-stack(grades1)
```

To stack only some of the columns in your dataset, use the `select` argument. For example, to stack only the `ClassA` and `ClassC` variables from the `grades1` dataset, use the command:

```
> newdataset<-stack(grades1, select=c("ClassA", "ClassC"))
```

If there are any non-numeric variables in your dataset, then R will exclude them from the stacked dataset.

The `stack` function automatically names the new variables `values` and `ind`, but you can change the names to something more informative with the `names` function:

```
> names(grades2)<-c("Result", "Class")
```

Figure 4-10 shows the final stacked dataset.

	Result	Class
1	87	ClassA
2	64	ClassA
3	89	ClassA
4	82	ClassA
5	59	ClassA
6	83	ClassB
7	97	ClassB
8	97	ClassB
9	99	ClassB
10	92	ClassB
11	80	ClassC
12	86	ClassC
13	80	ClassC
14	80	ClassC
15	84	ClassC

Figure 4-10: The `grades2` dataset, after the variable names are changed

Unstacking Data

You can unstack a dataset with the `unstack` function. If your dataset has only two variables, the first of which gives the data values and the second of which identifies which group the observation belongs to (like the `grades2` dataset), use the function:

```
> grades1<-unstack(grades2)
```

If your dataset has more than two variables or the variables are in reverse order then you must specify which variables you want to unstack:

```
> unstackeddata<-unstack(stackeddata, values~groups)
```

Here, `values` is the name of the variable containing the data values and `groups` is the name of the variable that identifies to which group the observation belongs.

For example, to unstack the `Sepal.Width` column in the `iris` dataset (included with R) to create a new dataset with a column for each iris species, use the command:

```
> sepalwidths<-unstack(iris, Sepal.Width~Species)
```

If you unstack a variable that does not have an equal number of values in each group, R cannot arrange the values to create a new data frame. In this case, R creates a *list* object instead of a data frame. However, you can still access each group of values in the new object by using the dollar notation:

```
> unstackeddata$groupA
```

Reshaping a Dataset

Reshaping a dataset is also known as rotating or transforming a dataset. It usually applies to datasets in which repeat measurements have been taken, but it is useful in other situations, too. Reshaping is the process of changing between long form (with repeat measurements in separate columns) and wide form (with repeat measurements in the same column).

To illustrate the difference between the long and wide forms, consider the `resistance` dataset shown in Figure 4-11a, which gives cubic resistance measurements for four concrete formulations taken at three, seven, and fourteen days after setting. This dataset is in wide form. Figure 4-11b shows the same dataset in long form, with all the measurements in one variable and another variable giving the day that the measurement was taken.

	Formula	Day3	Day7	Day14
1	A	10.8	18.5	41.3
2	B	20.1	29.2	37
3	C	3.7	17.5	28.9
4	D	18.1	27.2	37.6

(a) resistance

	row.names	Formula	Day	Resistance
1	A.3	A	3	10.8
2	B.3	B	3	20.1
3	C.3	C	3	3.7
4	D.3	D	3	18.1
5	A.7	A	7	18.5
6	B.7	B	7	29.2
7	C.7	C	7	17.5
8	D.7	D	7	27.2
9	A.14	A	14	41.3
10	B.14	B	14	37
11	C.14	C	14	28.9
12	D.14	D	14	37.6

(b) resistance2

Figure 4-11: The `resistance` dataset in long and wide forms (see Appendix C for more details)

Converting your dataset to long form is useful if you want to build a model that includes the time point as an explanatory variable. For example, converting the `resistance` dataset to long form allows you to model the resistance against the number of days elapsed since setting the concrete.

The `reshape` function allows you to convert a dataset from wide to long form. For example, to convert the `resistance` dataset to long form, use the command:

```
> resistance2<-reshape(resistance, direction="long", varying=list(c("Day3", "Day7", "Day14")),
                       times=c(3, 7, 14), idvar="Formula", v.names="Resistance", timevar="Day")
```

Use the `varying` argument to specify the variables that you want to combine into one column. Use the `times` argument to give the new time values or replicate numbers for each of these variables. Note that the `times` can be character values instead of numeric. Use the `idvar` argument to specify the variable that will group the records together. The `v.names` and `timevar` arguments are optional and give the names for the new variables.

The new dataset is shown in Figure 4-11b.

You can also use the `reshape` function to perform the opposite procedure, that is, converting a dataset from long to wide form.

Consider the `vitalsigns` dataset shown in Figure 4-12a, which gives measurements of systolic blood pressure, diastolic blood pressure, and pulse for four patients. All of the measurements are held in the `result` variable, while the `test` variable identifies the parameter.

	subject	test	result
1	1733	Pulse	120
2	1733	DiaBP	79
3	1733	SysBP	79
4	1734	Pulse	121
5	1734	DiaBP	86
6	1734	SysBP	72
7	1735	Pulse	130
8	1735	DiaBP	94
9	1735	SysBP	74
10	1736	Pulse	142
11	1736	DiaBP	99
12	1736	SysBP	87

(a) vitalsigns

	subject	result.Pulse	result.DiaBP	result.SysBP
1	1733	120	79	79
2	1734	121	86	72
3	1735	130	94	74
4	1736	142	99	87

(b) vitalsigns2

Figure 4-12: The `vitalsigns` dataset in long and wide forms (see Appendix C for more details)

Converting your dataset to wide form is useful if you want to compare data from two or more time points or parameters, or include them as distinct variables in the same model. For example, reshaping the `vitalsigns` dataset allows you to calculate the correlation between systolic and diastolic blood pressure.

To split the different types of measurements in to separate columns, use the command:

```
> vitalsigns2<-reshape(vitalsigns, direction="wide", v.names="result", timevar="test",
```

```
                        idvar="subject")
```

Use the `v.names` argument to specify the variable that you want to separate into different columns. Use the `timevar` argument to specify the variable that indicates which column the value belongs to (i.e., the variable giving the time point, replicate number, or category for the record). Use the `idvar` argument to specify which variable is used to group the records together.

Figure 4-12b shows the new wide form dataset. Notice that R automatically generates names for the new variables. Alternatively, you can specify the new names with the `varying` argument:

```
> vitalsigns2<-reshape(vitalsigns, direction="wide", v.names="result", timevar="test",
                       idvar="subject", varying=list(c("SysBP", "DiaBP", "Pulse")))
```

Summary

You should now be able to combine two or more datasets, either by appending them or by merging them using a common variable. You should understand the difference between stacked and unstacked forms, and be able to stack and unstack a dataset as required. You should also understand the difference between long and wide forms and be able to convert between the two.

This table summarizes the most important commands covered in this chapter.

Task	Command
Append datasets vertically	`rbind(dataset1,dataset2)`
Append datasets horizontally	`cbind(dataset1,dataset2)`
Merge datasets	`merge(dataset1,dataset2)`
Stack dataset	`stack(dataset, select=c("var1", "var2", "var3"))`
Unstack dataset	`unstack(dataset,values~groups)`
Reshape (wide to long)	`reshape(dataset, direction="long", varying=list(c("var1", "var2", "var3")), times=c(t1,t2,t3), idvar="identifier")`
Reshape (long to wide)	`reshape(dataset, direction="wide", v.names="values", timevar="groups", idvar="identifier")`

In the next chapter, we will move on to summarizing your data, beginning with continuous variables.