**Chapter 6**

# Data Frames

The weights, prices, and types data structures are all deeply tied together, if you think about it. If you add a new weight sample, you need to remember to add a new price and type, or risk everything falling out of sync. To avoid trouble, it would be nice if we could tie all these variables together in a single data structure.

Fortunately, R has a structure for just this purpose: the data frame. You can think of a data frame as something akin to a database table or an Excel spreadsheet. It has a specific number of columns, each of which is expected to contain values of a particular type. It also has an indeterminate number of rows - sets of related values for each column.

## Data Frames 6.1

Our vectors with treasure chest data are perfect candidates for conversion to a data frame. And it's easy to do. Call the data.frame function, and pass weights, prices, and types as the arguments. Assign the result to the treasure variable:

 > treasure <- data.frame(weights, prices, types)

 Now, try printing treasure to see its contents:

> print(treasure)

weights prices types

1 300 9000 gold

2 200 5000 silver

3 100 12000 gems

4 250 7500 gold

5 150 18000 gems

There's your new data frame, neatly organized into rows, with column names (derived from the variable names) across the top.

##  Data Frame Access 6.2

Just like matrices, it's easy to access individual portions of a data frame.

You can get individual columns by providing their index number in double-brackets. Try getting the second column (prices) of treasure:

> treasure[[2]]

[1] 9000 5000 12000 7500 18000

 You could instead provide a column name as a string in double-brackets. (This is often more readable.) Retrieve the "weights" column:

> treasure[["weights"]]

[1] 300 200 100 250 150

 Typing all those brackets can get tedious, so there's also a shorthand notation: the data frame name, a dollar sign, and the column name (without quotes). Try using it to get the "prices" column:

> treasure$prices

[1] 9000 5000 12000 7500 18000

 Now try getting the "types" column:

> treasure$types

[1] gold silver gems gold gems

Levels: gems gold silver

##  Loading Data Frames 6.3

Typing in all your data by hand only works up to a point, obviously, which is why R was given the capability to easily load data in from external files.

We've created a couple data files for you to experiment with:

> list.files()

[1] "targets.csv" "infantry.txt"

Our "targets.csv" file is in the CSV (Comma Separated Values) format exported by many popular spreadsheet programs. Here's what its content looks like:

"Port","Population","Worth"

"Cartagena",35000,10000

"Porto Bello",49000,15000

"Havana",140000,50000

"Panama City",105000,35000

You can load a CSV file's content into a data frame by passing the file name to the read.csv function. Try it with the "targets.csv" file:

> read.csv("targets.csv")

Port Population Worth

1 Cartagena 35000 10000

2 Porto Bello 49000 15000

3 Havana 140000 50000

4 Panama City 105000 35000

 The "infantry.txt" file has a similar format, but its fields are separated by tab characters rather than commas. Its content looks like this:

Port Infantry

Porto Bello 700

Cartagena 500

Panama City 1500

Havana 2000

For files that use separator strings other than commas, you can use the read.table function. The sep argument defines the separator character, and you can specify a tab character with "\t".

Call read.table on "infantry.txt", using tab separators:

> read.table("infantry.txt", sep="\t")

V1 V2

1 Port Infantry

2 Porto Bello 700

3 Cartagena 500

4 Panama City 1500

5 Havana 2000

 Notice the "V1" and "V2" column headers? The first line is not automatically treated as column headers with read.table. This behavior is controlled by the header argument. Call read.table again, setting header to TRUE:

> read.table("infantry.txt", sep="\t", header=T)

Port Infantry

1 Porto Bello 700

2 Cartagena 500

3 Panama City 1500

4 Havana 2000

##  Merging Data Frames 6.4

We want to loot the city with the most treasure and the fewest guards. Right now, though, we have to look at both files and match up the rows. It would be nice if all the data for a port were in one place...

R's merge function can accomplish precisely that. It joins two data frames together, using the contents of one or more columns. First, we're going to store those file contents in two data frames for you, targets and infantry.

The merge function takes arguments with an x frame (targets) and a y frame (infantry). By default, it joins the frames on columns with the same name (the two Port columns). See if you can merge the two frames:

> targets <- read.csv("targets.csv")

> infantry <- read.table("infantry.txt", sep="\t", header=TRUE)

> merge(x = targets, y = infantry)

Port Population Worth Infantry

1 Cartagena 35000 10000 500

2 Havana 140000 50000 2000

3 Panama City 105000 35000 1500

4 Porto Bello 49000 15000 700

## Chapter 6 Completed

Thirty paces south from the gate of the fort, and dig… we've unearthed another badge!

When your data grows beyond a certain size, you need powerful tools to organize it. With data frames, R gives you exactly that. We've shown you how to create and access data frames. We've also shown you how to load frames in from files, and how to cobble multiple frames together into a new data set.

Time to take what you've learned so far, and apply it. In the next chapter, we'll be working with some real-world data!