R14

Data frames

Covered in R14

- Creating data frames
- Indexing
- Adding a row or column to a data frame
- Names *versus* objects

1 Creating data frames

In previous chapters we described one-dimensional objects: *vectors* (which contain numeric, character or logical data) and *factors* (which contain categorical data – either ordinal or nominal). We also looked at matrices which were two-dimensional objects. Recall that all of these objects had to contain data of the same type. So a vector could not, for example, contain numbers *and* characters.

In this chapter we look at data frames which are two-dimensional objects like matrices. However, they can contain different types of data. Essentially a data frame is a collection of column vectors/factors of the same length. The columns are the vectors/factors which contain data for a single variable (*eg* policyholder's names, ages and smoker status). The rows represent a single observation (*eg* a single policyholder).

$$\begin{pmatrix} \text{Alfie} & 34 & \text{TRUE} \\ \text{Belinda} & 28 & \text{FALSE} \\ \text{Charlie} & 31 & \text{FALSE} \\ \text{Delilah} & 38 & \text{TRUE} \end{pmatrix}$$

So we can see that whilst each column (*ie* vector/factor) contains data of the same type, the different columns (*ie* vectors/factors) can be different data types. So in the example above we have character data for the first column, numeric data for the second column and logical data for the third column.

You can find out whether an object is a data frame by using is.data.frame(<object>). The dimensions are called rows and columns and the numbers of each can be found by using the nrow(<data frame object>) and ncol(<data frame object>) commands respectively. Alternatively, you could use the dimensions command dim(<object>) to get both the number of rows and columns.

In this section we look at how to create a data frame from scratch. This will be rather long-winded and so you will often use scripts to prevent laborious retyping. In reality we will usually import data from a spreadsheet or other source to create a data frame. We cover this in the next chapter.

To create a data frame from scratch we use the data.frame( ) command. This will create a data frame out of a list of vectors (or other data frames):

data.frame(<vector1>, <vector2>, ….)

As already mentioned, vectors don't have to contain the same data type but they do need to be the same length. If not, then R will recycle/extend the shorter ones to make them the same length as the longest vector.

We could either create the vectors separately and then put them together in a data frame or we can create them inside this function. We'll look at both to show you the pros and cons of each approach.

Let's create a data frame for the example mentioned above:

Alfie	34	TRUE
Belinda	28	FALSE
Charlie	31	FALSE
Delilah	38	TRUE

To do this and store it in object A, we would type:

A <- data.frame(c("Alfie", "Belinda", "Charlie", "Delilah"), c(34, 28, 31, 38), c(TRUE, FALSE, FALSE, TRUE))

You might prefer to enter this function in two parts then just put part of the function on the next line.

```
> A <- data.frame(c("Alfie", "Belinda", "Charlie", "Delilah"),
+ c(34, 28, 31, 38), c(TRUE, FALSE, FALSE, TRUE))
> A
  c..Alfie....Belinda....Charlie....Delilah.. c.34..28..31..38.
1                                      Alfie                34
2                                    Belinda                28
3                                    Charlie                31
4                                    Delilah                38
  c.TRUE..FALSE..FALSE..TRUE.
1                        TRUE
2                       FALSE
3                       FALSE
4                        TRUE
> |
```

However when R displays this data frame it is not pretty. The reason is that a data frame automatically looks for column names from the vectors. This is great if we create a data frame out of existing vectors (as we shall see a bit later) but if, as we did above, we create it from scratch it causes rathy messy names.

We can name the vectors either within the data.frame( ) command itself or separately using the colnames( ) command or the names( ) command (which assumes you're talking about columns when applying it to a dataframe).

To define the names inside the data.frame( ) command is similar to naming elements in a vector. We put the name equal to the vector as follows:

data.frame(<name1>=<vector1>, <name2>=<vector2>, ….)

Let's create this data frame again but this time name the columns – name, age and smoker:

A <- data.frame(name = c("Alfie", "Belinda", "Charlie", "Delilah"), age = c(34, 28, 31, 38), smoker = c(TRUE, FALSE, FALSE, TRUE))

```
> A <- data.frame(name=c("Alfie", "Belinda", "Charlie", "Delilah"),
+ age=c(34, 28, 31, 38), smoker=c(TRUE, FALSE, FALSE, TRUE))
> A
     name age smoker
1   Alfie  34   TRUE
2 Belinda  28  FALSE
3 Charlie  31  FALSE
4 Delilah  38   TRUE
>
```

This is much more beautiful.

Had we wished to do this via the colnames or names functions we would have typed one of the following:

colnames(A) <- c("name", "age", " smoker")

names(A) <- c("name", "age", " smoker")

Note that the numbers 1, 2, 3, 4 down the lefthand side of the data frame are *not* index values referring to the first, second, third and fourth rows but the *names* of the rows and are, therefore, characters "1", "2", "3", "4". We can rename them using the rownames( ) function. However, there is little point in this case as the names of the individuals are included in the data frame itself.

Properties

Let's look at the properties of the data frame A:

```
> is.data.frame(A)
[1] TRUE
> class(A)
[1] "data.frame"
> nrow(A)
[1] 4
> ncol(A)
[1] 3
> dim(A)
[1] 4 3
>
```

And let's look at its names:

```
> names(A)
[1] "name"   "age"    "smoker"
> colnames(A)
[1] "name"   "age"    "smoker"
> rownames(A)
[1] "1" "2" "3" "4"
> dimnames(A)
[[1]]
[1] "1" "2" "3" "4"

[[2]]
[1] "name"   "age"    "smoker"

>
```

Recall that another way of obtaining (nearly) all this information in one go is to use the str(<object>) command which displays the structure of an R object:

```
> str(A)
'data.frame':   4 obs. of  3 variables:
 $ name  : Factor w/ 4 levels "Alfie","Belinda",..: 1 2 3 4
 $ age   : num  34 28 31 38
 $ smoker: logi  TRUE FALSE FALSE TRUE
>
```

So the data frame is the default object for data input which, like our example, assumes that the columns are the variables we are observing and the rows are the observations. Hence it says that we have 4 observations of 3 variables.

The names of the observations are given at the start of each line, followed by their data type, followed by their contents (or some of the contents if there are too many to display).

We can see that age is numeric data and smoker status is logical, however names are not characters but factors. The reason for this is that data frames assume that observations are factors unless we tell it otherwise. This is understandable as most data we observe is categorical (eg policy type, car type, postcode, *etc*). We can turn this feature off by setting the stringsAsFactors option in the data frame to FALSE (instead of the default TRUE).

```
A <- data.frame(name = c("Alfie", "Belinda", "Charlie", "Delilah"), age = c(34, 28, 31, 38), smoker = c(TRUE, FALSE, FALSE, TRUE), stringsAsFactors = FALSE)
```

When we do this we can see that the structure command now lists names as character data:

```
> A <- data.frame(name = c("Alfie", "Belinda", "Charlie", "Delilah"),
+ age = c(34, 28, 31, 38), smoker = c(TRUE, FALSE, FALSE, TRUE),
+ stringsAsFactors = FALSE)
> A
     name age smoker
1   Alfie  34   TRUE
2 Belinda  28  FALSE
3 Charlie  31  FALSE
4 Delilah  38   TRUE
> str(A)
'data.frame':   4 obs. of  3 variables:
 $ name  : chr  "Alfie" "Belinda" "Charlie" "Delilah"
 $ age   : num  34 28 31 38
 $ smoker: logi  TRUE FALSE FALSE TRUE
>
```

Creating data frames from other objects

Just like we could create a matrix from vectors, we can also create a data frame from vectors. Let's demonstrate this by creating vectors containing the heights and weights of the four individuals we used in our previous data frame.

```
> height <- c(186, 154, 166, 170)
> weight <- c(92, 72,80 , 78)
> B <- data.frame(height, weight)
> B
  height weight
1    186     92
2    154     72
3    166     80
4    170     78
> str(B)
'data.frame':   4 obs. of  2 variables:
 $ height: num  186 154 166 170
 $ weight: num  92 72 80 78
>
```

We can see that, by default, the names of the vectors will be the column names.

Similarly we can create a new data frame from other data frames. Let's now combine our two data frames A and B together:

```
> A
     name age smoker
1   Alfie  34   TRUE
2 Belinda  28  FALSE
3 Charlie  31  FALSE
4 Delilah  38   TRUE
> B
  height weight
1    186     92
2    154     72
3    166     80
4    170     78
> C <- data.frame(A,B)
> C
     name age smoker height weight
1   Alfie  34   TRUE    186     92
2 Belinda  28  FALSE    154     72
3 Charlie  31  FALSE    166     80
4 Delilah  38   TRUE    170     78
>
```

Since the arguments of the data frame function are the (column) vectors, it combines the two data frame objects side by side (*ie* as new columns).

Creating data frames using cbind and rbind

Just as we did in the matrix chapter we can use column bind (cbind) or row bind (rbind) functions to combine vectors together. However, by default this combines vectors to form a matrix (with coercion, if necessary). But if we cbind or rbind a data frame with another object (such as a vector) then it creates a data frame instead of a matrix. We'll look at this later in the chapter.

Coercing

Since each column of a data frame is a vector then each column must contain data of the same type. So if there is a mix of different data types in a column then R will try to coerce them all to the same type.

For example suppose we create a data frame, D, with two columns, c1 and c2, as follows:

```
> D <- data.frame(c1=c(5, TRUE, -1, FALSE), c2=c("A", 3, -2, TRUE))
> D
   c1   c2
1   5    A
2   1    3
3  -1   -2
4   0 TRUE
> str(D)
'data.frame':   4 obs. of  2 variables:
 $ c1: num  5 1 -1 0
 $ c2: Factor w/ 4 levels "-2","3","A","TRUE": 3 2 1 4
>
```

We can see for the first column it has converted the logical data into numeric data (TRUE becomes 1 and FALSE becomes 0) so that it is now a vector of numeric data . We can see that the second column has been converted to factors. Had we specified the stringsAsFactors = FALSE then it would have coerced into a character vector. Try it out to see for yourself.

Recycling

Since a data frame is a collection of column vectors/factors of the same length then if we try to create a data frame from vectors of differing lengths then it will recycle shorter vectors. The following data frame is made from 3 vectors (v1, v2 and v3) of lengths 1, 2 and 4:

```
> E <- data.frame(v1=c(1), v2=c(2,3), v3=c(4, 5, 6, 7))
> E
  v1 v2 v3
1  1  2  4
2  1  3  5
3  1  2  6
4  1  3  7
>
```

Since the length of the longest vector is 4, the shorter vectors have been recycled (*ie* extended by repetition) until they are also of length 4.

However, suppose that the longest vector is not a multiple of one or more of the other vectors (*eg* lengths 1, 2 and 5). When we did vector arithmetic it displayed an error message but still performed the operation. However, for data frames it displays an error message and stops:

```
> F <- data.frame(v1=c(1), v2=c(2,3), v3=c(4, 5, 6, 7, 8))
Error in data.frame(v1 = c(1), v2 = c(2, 3), v3 = c(4, 5, 6, 7, 8)) :
  arguments imply differing number of rows: 1, 2, 5
>
```

2 Indexing (or subsetting) data frames

Recall that indexing is the process of selecting only some of the elements of an object. We give the position of the element in square brackets after the name of the object.

Since data frames have two dimensions we will need to give the row and column of the element we want. Let's apply this to our first data frame object A:

$$\begin{pmatrix} \text{Alfie} & 34 & \text{TRUE} \\ \text{Belinda} & 28 & \text{FALSE} \\ \text{Charlie} & 31 & \text{FALSE} \\ \text{Delilah} & 38 & \text{TRUE} \end{pmatrix}$$

We can choose the element in the first row and second column by writing A[1,2]:

```
> A
     name age smoker
1   Alfie  34   TRUE
2 Belinda  28  FALSE
3 Charlie  31  FALSE
4 Delilah  38   TRUE
> A[1,2]
[1] 34
>
```

To display all the elements of, say, the first row, we omit the figure for the column A[1,]:

```
> A
     name age smoker
1   Alfie  34   TRUE
2 Belinda  28  FALSE
3 Charlie  31  FALSE
4 Delilah  38   TRUE
> A[1,]
   name age smoker
1 Alfie  34   TRUE
```

Similarly, to display all the elements of the second column, we omit the figure for the row A[,2]:

```
> A
     name age smoker
1   Alfie  34   TRUE
2 Belinda  28  FALSE
3 Charlie  31  FALSE
4 Delilah  38   TRUE
> A[,2]
[1] 34 28 31 38
>
```

You notice that even though we have selected the column it displays it horizontally as R does for vectors.

To display more than one row or more than one column we simply enter the multiple values using the c( ) command. For example, to display the elements in the 1st and 2nd rows of the 3rd column:

```
> A
     name age smoker
1   Alfie  34   TRUE
2 Belinda  28  FALSE
3 Charlie  31  FALSE
4 Delilah  38   TRUE
> A[c(1,2),3]
[1]  TRUE FALSE
>
```

Or the 2nd and 3rd rows of the 1st and 3rd columns:

```
> A
     name age smoker
1   Alfie  34   TRUE
2 Belinda  28  FALSE
3 Charlie  31  FALSE
4 Delilah  38   TRUE
> A[c(2,3),c(1,3)]
     name smoker
2 Belinda  FALSE
3 Charlie  FALSE
>
```

We could also select multiple rows or columns using a:b the consecutive integer function:

```
> A
     name age smoker
1   Alfie  34   TRUE
2 Belinda  28  FALSE
3 Charlie  31  FALSE
4 Delilah  38   TRUE
> A[1:2,1:3]
     name age smoker
1   Alfie  34   TRUE
2 Belinda  28  FALSE
>
```

We can use the dimension commands nrow( ) or ncol( ) to specify all the rows or columns until the end:

```
> A
     name age smoker
1   Alfie  34   TRUE
2 Belinda  28  FALSE
3 Charlie  31  FALSE
4 Delilah  38   TRUE
> A[1:nrow(A),2]
[1] 34 28 31 38
> A[3,2:ncol(A)]
  age smoker
3  31  FALSE
>
```


To specify all rows/columns elements *except* the specified ones we use a negative infront of their positions:

```
> A
     name age smoker
1   Alfie  34   TRUE
2 Belinda  28  FALSE
3 Charlie  31  FALSE
4 Delilah  38   TRUE
> A[-2,3]
[1]  TRUE FALSE  TRUE
> A[1:2,-3]
     name age
1   Alfie  34
2 Belinda  28
> A[-c(1,3),2]
[1] 28 38
> A[-c(1,3),-2]
     name smoker
2 Belinda  FALSE
4 Delilah   TRUE
>
```

We can select elements using the results of logical tests. However, since our data frame consists of different types of data we will look at how we can do logical tests just on one variable (*ie* column) in a later chapter.

We can also use the names of rows or columns to select elements. In our data frame we had variable/column names of name, age and smoker:

```
> A
     name age smoker
1   Alfie  34   TRUE
2 Belinda  28  FALSE
3 Charlie  31  FALSE
4 Delilah  38   TRUE
> A[1,"age"]
[1] 34
> A[,"smoker"]
[1]  TRUE FALSE FALSE  TRUE
> A[3,c("age", "smoker")]
  age smoker
3  31  FALSE
>
```

Subsetting using the $ notation

An alternative way of specifying a column is to use the $ notation.

```
<data frame>$<column name>
```

For example, our data frame object is called A and the first of its column names is “name”. So we can specify that column immediately using A$name. Similarly for the other columns:

```
> A
     name age smoker
1   Alfie  34   TRUE
2 Belinda  28  FALSE
3 Charlie  31  FALSE
4 Delilah  38   TRUE
> A$name
[1] "Alfie"   "Belinda" "Charlie" "Delilah"
> A$age
[1] 34 28 31 38
> A$smoker
[1]  TRUE FALSE FALSE  TRUE
>
```

This is a very useful way of obtaining a vector object from the data frame to use in calculations (*eg* mean or correlation). We’ll make use of this in a later chapter.

Note that just like with function arguments, you can abbreviate the column name to the fewest letters that uniquely define it. Since our columns all begin with different letters we could just use the first letters:

```
> A
     name age smoker
1   Alfie  34   TRUE
2 Belinda  28  FALSE
3 Charlie  31  FALSE
4 Delilah  38   TRUE
> A$n
[1] "Alfie"   "Belinda" "Charlie" "Delilah"
> A$a
[1] 34 28 31 38
> A$s
[1]  TRUE FALSE FALSE  TRUE
>
```


3 Adding new columns or rows to a data frame

Adding a new column

There are three ways of adding a new column to an existing data frame. The first is by combining the objects using the data.frame command. We did this earlier in the chapter. In the example below we combine our original data frame with a vector of heights to add this variable:

```
> A
     name age smoker
1   Alfie  34   TRUE
2 Belinda  28  FALSE
3 Charlie  31  FALSE
4 Delilah  38   TRUE
> height
[1] 186 154 166 170
> G <- data.frame(A,height)
> G
     name age smoker height
1   Alfie  34   TRUE    186
2 Belinda  28  FALSE    154
3 Charlie  31  FALSE    166
4 Delilah  38   TRUE    170
>
```

The second way is to use the cbind function. So we could have obtained the same result as follows:

```
> A
     name age smoker
1   Alfie  34   TRUE
2 Belinda  28  FALSE
3 Charlie  31  FALSE
4 Delilah  38   TRUE
> height
[1] 186 154 166 170
> H <- cbind(A, height)
> H
     name age smoker height
1   Alfie  34   TRUE    186
2 Belinda  28  FALSE    154
3 Charlie  31  FALSE    166
4 Delilah  38   TRUE    170
>
```

The third method is to use the dollar notation to define a new column. For example let's add the weight vector as another column to the data frame H above:

```
> H
     name age smoker height
1   Alfie  34   TRUE    186
2 Belinda  28  FALSE    154
3 Charlie  31  FALSE    166
4 Delilah  38   TRUE    170
> weight
[1] 92 72 80 78
> H$weight <- weight
> H
     name age smoker height weight
1   Alfie  34   TRUE    186     92
2 Belinda  28  FALSE    154     72
3 Charlie  31  FALSE    166     80
4 Delilah  38   TRUE    170     78
>
```

Adding a new row

To add a new row (*ie* a new observation for each of the variables) is more tricky. We can't use a vector as it is likely that the different variables (eg name, age, smooker status, *etc*) are different data types as they are in our example.

The only way to do it is to use the row bind function on the original data frame and a new data frame containing the observations for that individual.

Suppose we wish to add Eddie to the data frame who is aged 24, is not a smoker, is 172cm tall with a weight of 82kg. We'll put these results in a data frame called I and then add this row to data frame H above:

```
> I <- data.frame("Eddie", 24, FALSE, 172, 82)
> J <- rbind(H,I)
Error in match.names(clabs, names(xi)) :
  names do not match previous names
>
```

There is a problem as the names of our new observation data frame do not match those of the original. So we have to define the same names first, and then we can combine them:

```
> I <- data.frame(name="Eddie", age=24, smoker=FALSE, height=172, weight=82)
> J <- rbind(H,I)
> J
     name age smoker height weight
1   Alfie  34   TRUE    186     92
2 Belinda  28  FALSE    154     72
3 Charlie  31  FALSE    166     80
4 Delilah  38   TRUE    170     78
5   Eddie  24  FALSE    172     82
>
```

4 The important difference between column names and objects

Let's have a look at the objects we have created during this chapter:

```
> objects()
 [1] "A"      "B"      "C"      "D"      "E"      "G"      "H"      "height"
 [9] "I"      "J"      "weight"
>
```

We can see that we have the objects "height" and "weight" that were vectors we used to create more columns in some of our data frames such as C:

```
> height
[1] 186 154 166 170
> weight
[1] 92 72 80 78
> C
     name age smoker height weight
1   Alfie  34   TRUE    186     92
2 Belinda  28  FALSE    154     72
3 Charlie  31  FALSE    166     80
4 Delilah  38   TRUE    170     78
>
```

Let's look at what happens to the data frames if we change the weights vector.

```
> weight <- c(92,72,80,70)
> weight
[1] 92 72 80 70
> C
     name age smoker height weight
1   Alfie  34   TRUE    186     92
2 Belinda  28  FALSE    154     72
3 Charlie  31  FALSE    166     80
4 Delilah  38   TRUE    170     78
>
```

You can see that the last result in the vector weight has been changed but not the values of the weight column in the data frame.

The data frame doesn't update the values of the weight vector after it has been created. To do that you would have use the weight subset of the data frame as follows:

```
> weight <- c(92,72,80,70)
> weight
[1] 92 72 80 70
> C
     name age smoker height weight
1   Alfie  34   TRUE    186     92
2 Belinda  28  FALSE    154     72
3 Charlie  31  FALSE    166     80
4 Delilah  38   TRUE    170     78
> C$weight <- c(92,72,80,70)
> C
     name age smoker height weight
1   Alfie  34   TRUE    186     92
2 Belinda  28  FALSE    154     72
3 Charlie  31  FALSE    166     80
4 Delilah  38   TRUE    170     70
>
```

This may seem like a silly point at the moment – but it will be very important in the next chapter when we look at attaching data frames.

5 Summary

Key terms

Data frame	A two-dimensional (row, column) ordered collection of column vectors/factors of the same length – each column (vector) must contain data of the same type but different columns can have different types.
Coercing	The process of changing the data types so that they are all the same
Indexing	The process of selecting an element from a data frame object
Subsetting	Another name for indexing – usually used for where more than one element is obtained from an object
Recycling	The process of extending a smaller vector by repetition to make sense of operations with other vectors – returns a warning if the length of the smaller vector is not a multiple of the length of the larger vector

Key commands

data.frame(<name1>=<vector1>, <name2>=<vector2>, … , stringsAsFactors=TRUE)	Creates a data frame of dimensions nrow by ncol out of the data with column names <name1>, <name2>, … If no name is given the name of the vector is used Last argument is optional By default it assumes that all character data are factors Use dimnames to give names to the rows and columns
is.data.frame(<object>)	Logical test of whether <object> is a data frame Returns TRUE or FALSE
dim(<data frame>)	Gives the dimensions (*ie* rows and columns) of <data frame>
nrow(<data frame>)	Gives the number of row of <data frame>
ncol(<data frame>)	Gives the number of columns of <data frame>
class(<object>)	Gives the class of an object
str(<object>)	Displays the structure of the <object>
cbind(<object1>, <object2>, …)	creates a data frame by combining the different vector objects

rbind(<object1>, <object2>, …)	creates a data frame by combining the objects in rows
names(<data frame>)	Gives the names of the column vectors in the <data frame> Can also be used to assign column names: names(<data frame>) <- c(<name1>, <name2>, …)
rownames(<data frame>)	Gives the names the rows of the <data frame> Can also be used to assign row names: rownames(<data frame>) <- c(<name1>, <name2>, …)
colnames(<data frame>)	Gives the names the column vectors in the <data frame> Can also be used to assign column names: colnames(<data frame>) <- c(<name1>, <name2>, …)
<data frame>[<row>,<column>]	Returns the element from <data frame> given in <row> and <column> Omitting one of the values returns the whole row/column

6 Have a go

You will only get proficient at R by practising.

1. Create a data frame, cars, containing the following data:

$$\begin{pmatrix} \textbf{Ford} & \textbf{Red} & \textbf{9250} \\ \textbf{Skoda} & \textbf{Blue} & \textbf{6500} \\ \textbf{VW} & \textbf{Silver} & \textbf{2300} \end{pmatrix}$$

2. Label the columns, Make, Colour and Price.

3. Add an additional column called Year containing the data: 2017, 2015, 2012.

4. Display only the colours from the data frame.

5. Display all the data except the colours.

You will have ample opportunity to create and manipulate data frames in your study of CS1 or CS2.