Heterogeneous Data: list and data.frame

Susan Holmes (c)

Lists

A completely heterogeneous set of objects of different types and different sizes can be combined into a list.

We cannot use the `c()` function but have an equivalent, simply the `list()` function, which creates this ordered collection of components.

```
A <- matrix( c(4,2,0,3,1,7,2,8,4,5), nrow=2,ncol=5)
Atypical <- list(name="Susan", byDate=1125, Amatrix=A, size=5.5, urban=FALSE)
Atypical
```

```
## $name
## [1] "Susan"
##
## $byDate
## [1] 1125
##
## $Amatrix
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    4    0    1    2    4
## [2,]    2    3    7    8    5
##
## $size
## [1] 5.5
##
## $urban
## [1] FALSE
```

Addressing elements of a `list`

We access the elements of the list either by the number in the order of the elements:

```
Atypical[[1]]
```

```
## [1] "Susan"
```

```
class(A[[3]])
```

```
## [1] "numeric"
```

```
is.logical(Atypical[[5]])
```

```
## [1] TRUE
```

```
Atypical[[5]]
```

```
## [1] FALSE
```

We can also access the list elements by their label using the $ sign:

```
Atypical$byDate
```

```
## [1] 1125
```

```
Atypical$A
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    4    0    1    2    4
## [2,]    2    3    7    8    5
```

We can ask to see information about a list using either `summary` or `str`

```
summary(Atypical)
```

```
##         Length Class  Mode
## name     1     -none- character
## byDate   1     -none- numeric
## Amatrix 10     -none- numeric
## size     1     -none- numeric
## urban    1     -none- logical
```

```
str(Atypical)
```

```
## List of 5
##  $ name   : chr "Susan"
##  $ byDate : num 1125
##  $ Amatrix: num [1:2, 1:5] 4 2 0 3 1 7 2 8 4 5
##  $ size   : num 5.5
##  $ urban  : logi FALSE
```

We see that summary describes `Atypical$A` with a length attribute equal to 10.

Lists as output

Lists are often used as the output of a complicated function that gives parameters and results of different dimensions as output. Most lists have names for the components, otherwise we can acess them in order with a double square bracket

```
result <- lm(weight~height, data=women)
names(result)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

```
result[[1]]
```

```
## (Intercept)      height
##   -87.51667     3.45000
```

Factor variables

Some variables are encoded as numbers when in fact these numbers are themselves meaningless codes.

For instance we might have a class of 13 male and 11 female students, we could code this using the `rep` function that repeats a value a certain number of times

```
studentg=c(rep(1,13),rep(2,11))
table(studentg)
```

```
## studentg
##  1  2
## 13 11
```

A better solution is to encode the variable gender as a `factor`.

```
gender=factor(c(rep("M",13),rep("F",11)))
gender
```

```
##  [1] M M M M M M M M M M M M M F F F F F F F F F F F
## Levels: F M
```

```
class(gender)
```

```
## [1] "factor"
```

```
table(gender)
```

```
## gender
##  F  M
## 11 13
```

```r
levels(gender)
```

```
## [1] "F" "M"
```

```r
str(gender)
```

```
##  Factor w/ 2 levels "F","M": 2 2 2 2 2 2 2 2 2 2 ...
```

```r
length(gender)
```

```
## [1] 24
```

R Data set example: UScereal

Let's start with an example using the Datasets already available in R.

In the `UScereal` data from the MASS package, the maker is represented by its first initial: G=General Mills, K=Kelloggs, N=Nabisco, P=Post, Q=Quaker Oats, R=Ralston Purina.

```
library(MASS)
?UScereal
UScereal[1:4,1:5]
```

```
##                            mfr calories   protein      fat   sodium
## 100% Bran                    N 212.1212 12.121212 3.030303 393.9394
## All-Bran                     K 212.1212 12.121212 3.030303 787.8788
## All-Bran with Extra Fiber    K 100.0000  8.000000 0.000000 280.0000
## Apple Cinnamon Cheerios      G 146.6667  2.666667 2.666667 240.0000
```

```
table(UScereal[,1])
```

```
##
##  G  K  N  P  Q  R
## 22 21  3  9  5  5
```

```
summary(UScereal[,1])
```

```
##  G  K  N  P  Q  R
## 22 21  3  9  5  5
```

So it is rectangular like a matrix, but some variables are numeric and others are factors.

The first variable is an example of a factor variable. It is because of these different classes of variables that exist often

together as information on the same observations that R needs a richer data structure than vectors, arrays and matrices.

data.frame: A way of combining different type of variables

A data.frame is a list that contains many variables, they can be considered to be in the columns of a table, we won't call this a matrix because the columns can have different types. Let's revisit the `UScereal` data.

We can access the 11th variable as we would in a matrix, this is a factor variable, so a good summary is to tabulate it with a `table` function.

```
table(UScereal[,11])
```

```
##
##     100% enriched     none
##        5       57        3
```

But we could also have used the variable name:

```
table(UScereal$vitamins)
```

```
##
##     100% enriched     none
##        5       57        3
```

```
class(UScereal)
```

```
## [1] "data.frame"
```

```
str(UScereal)
```

```
## 'data.frame':    65 obs. of  11 variables:
##  $ mfr      : Factor w/ 6 levels "G","K","N","P",..: 3 2 2 1 2 1 6 4 5 1 ...
##  $ calories : num  212 212 100 147 110 ...
##  $ protein  : num  12.12 12.12 8 2.67 2 ...
##  $ fat      : num  3.03 3.03 0 2.67 0 ...
##  $ sodium   : num  394 788 280 240 125 ...
##  $ fibre    : num  30.3 27.3 28 2 1 ...
##  $ carbo    : num  15.2 21.2 16 14 11 ...
##  $ sugars   : num  18.2 15.2 0 13.3 14 ...
##  $ shelf    : int  3 3 3 1 2 3 1 3 2 1 ...
##  $ potassium: num  848.5 969.7 660 93.3 30 ...
##  $ vitamins : Factor w/ 3 levels "100%","enriched",..: 2 2 2 2 2 2 2 2 2 2 ...
```

```
summary(UScereal)
```

```
##  mfr       calories        protein            fat            sodium
##  G:22   Min.   : 50.0   Min.   : 0.7519   Min.   :0.000   Min.   :  0.0
##  K:21   1st Qu.:110.0   1st Qu.: 2.0000   1st Qu.:0.000   1st Qu.:180.0
##  N: 3   Median :134.3   Median : 3.0000   Median :1.000   Median :232.0
##  P: 9   Mean   :149.4   Mean   : 3.6837   Mean   :1.423   Mean   :237.8
##  Q: 5   3rd Qu.:179.1   3rd Qu.: 4.4776   3rd Qu.:2.000   3rd Qu.:290.0
##  R: 5   Max.   :440.0   Max.   :12.1212   Max.   :9.091   Max.   :787.9
##      fibre            carbo           sugars          shelf
##  Min.   : 0.000   Min.   :10.53   Min.   : 0.00   Min.   :1.000
##  1st Qu.: 0.000   1st Qu.:15.00   1st Qu.: 4.00   1st Qu.:1.000
##  Median : 2.000   Median :18.67   Median :12.00   Median :2.000
##  Mean   : 3.871   Mean   :19.97   Mean   :10.05   Mean   :2.169
##  3rd Qu.: 4.478   3rd Qu.:22.39   3rd Qu.:14.00   3rd Qu.:3.000
##  Max.   :30.303   Max.   :68.00   Max.   :20.90   Max.   :3.000
##    potassium          vitamins
##  Min.   : 15.00   100%    : 5
##  1st Qu.: 45.00   enriched:57
##  Median : 96.59   none    : 3
##  Mean   :159.12
##  3rd Qu.:220.00
##  Max.   :969.70
```

Question What has the function `summary` shown about the data `UScereal`? The `data.frame` structure or class is a list, we can access the variables using their names as well as using the order they appear in the data.

Question Which of the variables in the `UScereal` data.frame are factors.

```
dim(UScereal)
```

```
## [1] 65 11
```

Question We can use the function `dim` on a data.frame to find out how many variables were measured on how many observations, what other data type can we use `dim` on ?

If we only want to look at the top few observations we can use the function head

```
head(UScereal)
```

```
##                              mfr calories   protein      fat   sodium
## 100% Bran                      N 212.1212 12.121212 3.030303 393.9394
## All-Bran                       K 212.1212 12.121212 3.030303 787.8788
## All-Bran with Extra Fiber      K 100.0000  8.000000 0.000000 280.0000
## Apple Cinnamon Cheerios        G 146.6667  2.666667 2.666667 240.0000
## Apple Jacks                    K 110.0000  2.000000 0.000000 125.0000
## Basic 4                        G 173.3333  4.000000 2.666667 280.0000
##                                  fibre    carbo   sugars shelf potassium
## 100% Bran                    30.303030 15.15152 18.18182     3 848.48485
## All-Bran                     27.272727 21.21212 15.15151     3 969.69697
## All-Bran with Extra Fiber    28.000000 16.00000  0.00000     3 660.00000
## Apple Cinnamon Cheerios       2.000000 14.00000 13.33333     1  93.33333
## Apple Jacks                   1.000000 11.00000 14.00000     2  30.00000
## Basic 4                       2.666667 24.00000 10.66667     3 133.33333
##                              vitamins
## 100% Bran                    enriched
## All-Bran                     enriched
## All-Bran with Extra Fiber    enriched
## Apple Cinnamon Cheerios      enriched
## Apple Jacks                  enriched
## Basic 4                      enriched
```

Some functions know how to behave, whatever the data.

Here is an example of data from the `datasets` package

```
library(datasets)
women
```

```
##    height weight
## 1      58    115
## 2      59    117
## 3      60    120
## 4      61    123
## 5      62    126
## 6      63    129
## 7      64    132
## 8      65    135
## 9      66    139
## 10     67    142
## 11     68    146
## 12     69    150
## 13     70    154
## 14     71    159
## 15     72    164
```

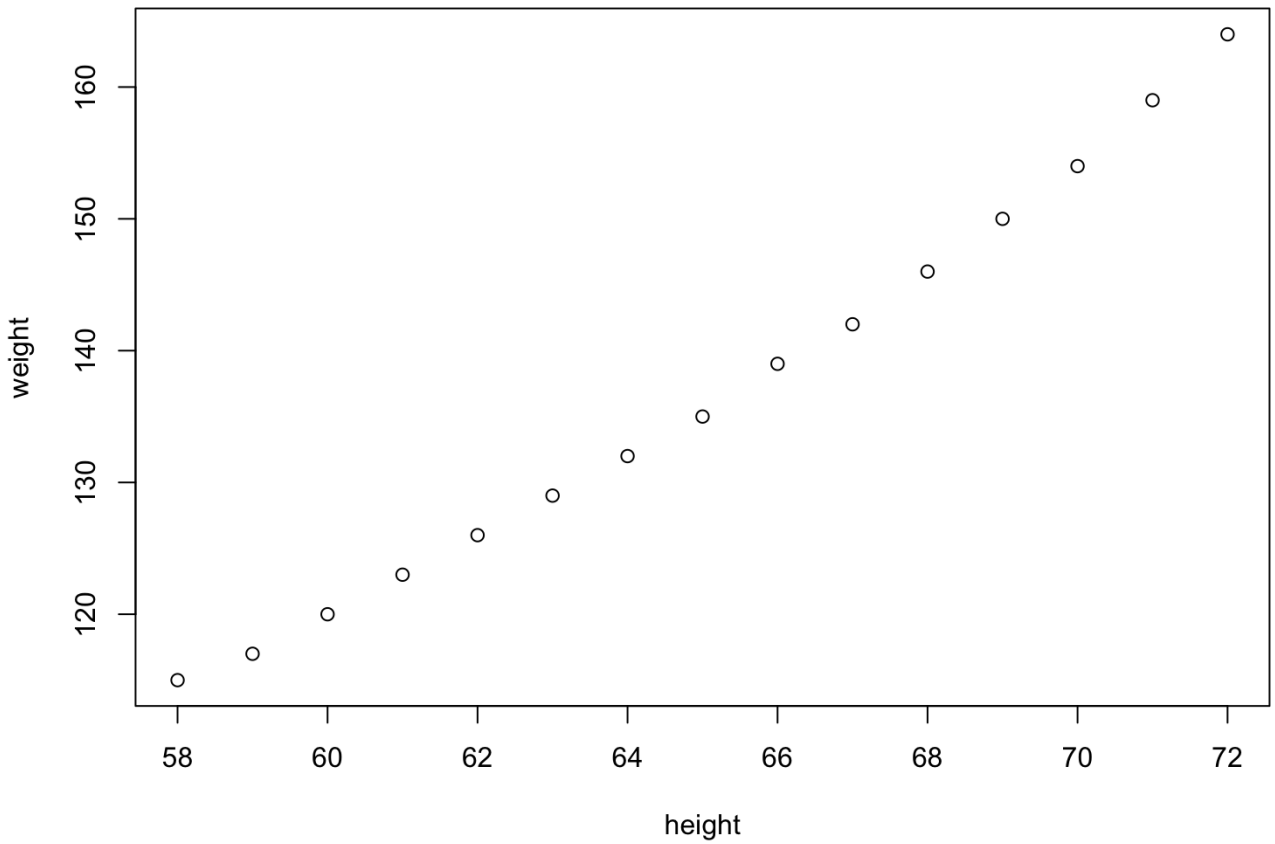
```
class(women)
```

```
## [1] "data.frame"
```

```
dim(women)
```

```
## [1] 15  2
```

```
plot(women)
```



We can change the data class.

Sometimes we may need to go back to matrices, this can be quite easy as is the reverse:

```
matw=as.matrix(women)
class(matw)
```

```
## [1] "matrix"
```

```
women2=as.data.frame(matw)
class(women2)
```

```
## [1] "data.frame"
```

But sometimes it can give surprising results;
Question: Try typing

```
as.matrix(UScereal)
```

and explain what you see.

Summary of our session

- We have learned several new data structures: lists and data.frames.
- We used more extensive addressing than in matrices, instead of [2,3], we saw [[1]] and `$label` addresses.
- We learned of a new type of variable: `factor`
- Some functions we have encountered:
  summary() head() factor() levels() table()
  as.matrix()
  as.data.frame()
  library() length()
- Note that many functions that we have seen work on different classes of data structures, they adapt to the data structure to give an answer.

Question The dataset `mtcars` is very popular as an example in R tutorials available online. Look at the data frame and find out if it has any variables that are factors