Data structures

2

R is designed to handle experimental data

- Although the basic unit of R is a vector, we usually handle data in **data frames**.
- A data frame is a set of observations of a set of variables – in other words, the outcome of an experiment.
- For example, we might want to analyse information about a set of patients. To start with, let's say we have ten patients and for each one we know their name, sex, age, weight and whether they give consent for their data to be made public.

The patients data frame

We are going to create a data frame called 'patients', which will have ten rows (observations) and seven columns (variables). The columns must all be equal lengths.

```
   First_Name Second_Name      Full_Name    Sex Age Weight Consent
1        Adam       Jones     Adam Jones   Male  50   70.8    TRUE
2         Eve      Parker     Eve Parker Female  21   67.9    TRUE
3        John       Evans     John Evans   Male  35   75.3   FALSE
4        Mary       Davis     Mary Davis Female  45   61.9    TRUE
5       Peter       Baker    Peter Baker   Male  28   72.4   FALSE
6        Paul     Daniels   Paul Daniels   Male  31   69.9   FALSE
7      Joanna     Edwards Joanna Edwards Female  42   63.5   FALSE
8     Matthew       Smith  Matthew Smith   Male  33   71.5    TRUE
9       David     Roberts  David Roberts   Male  57   73.2   FALSE
10      Sally      Wilson   Sally Wilson Female  62   64.8    TRUE
```

Let's see how we can construct this from scratch.

Character, numeric and logical data types

- Each column is a vector, like previous vectors we have seen, for example:

```
> age<-c(50, 21, 35, 45, 28, 31, 42, 33, 57, 62)
> weight<-c(70.8, 67.9, 75.3, 61.9, 72.4, 69.9, 63.5, 71.5, 73.2, 64.8)
```

- But we also have two new types of vector. A character vector includes values that are strings of characters, which we put in quotation marks:

```
> firstName<- c("Adam", "Eve", "John", "Mary", "Peter", "Paul", "Joanna",
"Matthew", "David", "Sally")
> secondName<-c("Jones", "Parker", "Evans", "Davis", "Baker", "Daniels",
"Edwards", "Smith", "Roberts", "Wilson")
```

- A logical vector only contains the values TRUE and FALSE:

```
> consent<-c(TRUE,TRUE,FALSE,TRUE,FALSE,FALSE,FALSE,TRUE,FALSE,TRUE)
```

Character, numeric and logical data types

- Vectors can only contain one type of data; we cannot mix numbers, characters and logical values in the same vector. If we try this, R will convert everything to characters:

```
> c(20, "a string", TRUE)
[1] "20"       "a string" "TRUE"
```

- We can see the type of a particular vector using the **mode** function:

```
> mode(firstName)
[1] "character"
> mode(age)
[1] "numeric"
> mode(weight)
[1] "numeric"
> mode(consent)
[1] "logical"
```

Factors

- Character vectors are fine for some variables, like names.
- But sometimes we have categorical data and we want R to recognize this.
- A factor is R's data structure for categorical data.

```
   > sex<-c("Male", "Female", "Male", "Female", "Male", "Male", "Female",
"Male", "Male", "Female")
   > sex
 [1] "Male"   "Female" "Male"   "Female" "Male"   "Male"   "Female" "Male"
"Male"   "Female"
   > factor(sex)
 [1] Male   Female Male   Female Male   Male   Female Male   Male   Female
Levels: Female Male
```

- R has converted the strings of the sex character vector into two **levels**, which are the categories in the data.
- Note the values of this factor are not character strings, but levels.
- We can use this factor to compare data for males and females.

Creating a data frame (first attempt)

- We can construct a data frame from other objects:

```
> patients<-data.frame(firstName, secondName, paste(firstName,secondName),
     sex, age, weight, consent)
> patients
   firstName secondName paste.firstName..secondName.    sex age weight consent
1       Adam      Jones                   Adam Jones   Male  50   70.8    TRUE
2        Eve     Parker                   Eve Parker Female  21   67.9    TRUE
3       John      Evans                   John Evans   Male  35   75.3   FALSE
4       Mary      Davis                   Mary Davis Female  45   61.9    TRUE
5      Peter      Baker                  Peter Baker   Male  28   72.4   FALSE
6       Paul    Daniels                 Paul Daniels   Male  31   69.9   FALSE
7     Joanna    Edwards               Joanna Edwards Female  42   63.5   FALSE
8    Matthew      Smith                Matthew Smith   Male  33   71.5    TRUE
9      David    Roberts                David Roberts   Male  57   73.2   FALSE
10     Sally     Wilson                 Sally Wilson Female  62   64.8    TRUE
```

- The **paste** function joins character vectors together.
- We can access particular variables using the **dollar** operator:

```
> patients$age
 [1] 50 21 35 45 28 31 42 33 57 62
```

Naming data frame variables

- R has inferred the names of our data frame variables from the names of the vectors or the commands (eg the paste command).
- We can name the variables after we have created a data frame using the **names** function, and we can use the same function to see the names:

-
  ```
  > names(patients)<-c("First_Name", "Second_Name", "Full_Name", "Sex",
  "Age", "Weight", "Consent")
  ```

```
   > names(patients)
   [1] "First_Name"   "Second_Name" "Full_Name"    "Sex"            "Age"
"Weight"       "Consent"
```

- Or we can name the variables when we define the data frame:

```
   > patients<-data.frame(First_Name=firstName, Second_Name=secondName,
Full_Name=paste(firstName,secondName), Sex=sex, Age=age, Weight=weight,
Consent=consent)
```

```
   > names(patients)
   [1] "First_Name"   "Second_Name" "Full_Name"    "Sex"            "Age"
"Weight"       "Consent"
```

Factors in data frames

- When creating a data frame, R assumes all character vectors should be categorical variables and converts them to factors. This is not always what we want:

```
   > patients$firstName
 [1] Adam    Eve     John    Mary    Peter   Paul    Joanna  Matthew David   Sally
Levels: Adam David Eve Joanna John Mary Matthew Paul Peter Sally
```

- We can avoid this by asking R not to treat strings as factors, and then explicitly stating when we want a factor by using **factor**:

```
   > patients<-data.frame(First_Name=firstName, Second_Name=secondName,
Full_Name=paste(firstName,secondName), Sex=factor(sex), Age=age,
Weight=weight, Consent=consent, stringsAsFactors=FALSE)
   > patients$Sex
     [1] Male   Female Male   Female Male   Male   Female Male   Male   Female
Levels: Female Male
   > patients$First_Name

   [1] "Adam"    "Eve"     "John"    "Mary"    "Peter"   "Paul"    "Joanna"
"Matthew" "David"   "Sally"
```

Matrices

`matrix(..., ncol=…, nrow=…)`

- Data frames are R's speciality, but R also handles matrices:

```
> e <- matrix(1:10, nrow=5, ncol=2)
> e
     [,1] [,2]
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
[5,]    5   10
> f ← matrix(1:10, nrow=2, ncol=5)
> f
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
> f %*% e
     [,1] [,2]
[1,]   95  220
[2,]  110  260
```

The `%*%` operator is the matrix multiplication operator, not the standard multiplication operator.

Lists

`list(name1=obj1, name2=obj2, …)`

- We have seen that vectors can only hold data of one type. How can we store data of multiple types? Or vectors of different lengths in one object?
- We can use lists. A list can contain objects of any type.

```
a <- 1:10
b <- matrix(runif(100),ncol=10,nrow=10)
c <- data.frame(a, month.name[1:10])

myList<-list( ls.obj.1=a, ls.obj.2=b,ls.obj.3=c )
summary(myList)
names(myList)
```

- We can use the dollar syntax to access list items (in fact, a data frame is a special type of list):

```
myList$ls.obj.1
```

- We can also use `myList[[1]]` to get the first item in the list.
- (For the curious: this double indexing is necessary because lists are in fact just like vectors – they can only contain one type of object. But one of the types they can contain is a list. So any list like the above is actually a list of lists; the first element `myList[1]` is a list containing a vector, and so we need double indexing to actually get the vector.)

Indexing data frames and matrices

Special cases:
a[*i*,] i-th row
a[,*j*] j-th column

- You can index multidimensional data structures like matrices and data frames using commas. If you don't provide an index for either rows or columns, all of the rows or columns will be returned.

```
object [ rows , columns ]
```

```
> e[1,2]
[1] 6
> e[1,]
[1] 1 6
> patients[1,2]
[1] "Jones"
> patients[1,]
  First_Name Second_Name  Full_Name  Sex Age Weight Consent
1       Adam       Jones Adam Jones Male  50   70.8    TRUE
```

Advanced indexing

- As values in R are really vectors, so indices are actually vectors, and can be numeric or logical:

```
> s <- letters[1:5]
> s[c(1,3)]
[1] "a" "c"
> s[c(TRUE, FALSE, TRUE, FALSE, FALSE)]
[1] "a" "c"
> a<-1:5
> a<3
[1]  TRUE  TRUE FALSE FALSE FALSE
> s[a<3]
[1] "a" "b"
> s[a>1 & a<3]
[1] "b”
> s[a==2]
[1] "b"
```

Operators

- arithmetic

  +, -, *, /, ^

- comparison

  <, >, =<, >=, ==, !=

  (equal to, not equal to)

- logical

  !, &, |, xor

these always return logical values ! (TRUE, FALSE)

Exercise

- Create a data frame called **my.patients** using the instructions in the slides. Change the data if you like.
- Check you have created the data frame correctly by loading the original version from this file in the *Day_1_scripts* folder using **source**:

```
> source("05_patients.R")
```

- Remake your data frame with three new variables: country, continent, and height. Make up the data. Make country a character vector but continent a factor.
- Try the **summary** function on your data frame. What does it do? How does it treat vectors (numeric, character, logical) and factors? (What does it do for matrices?)
- Use logical indexing to select the following patients:
  - Patients under 40
  - Patients who give consent to share their data
  - Men who weight as much or more than the average European male (70.8 kg)

Logical indexing answers

- Patients under 40:

```
> patients[patients$Age<40,]
```

- Patients who give consent to share their data:

```
> patients[patients$Consent==TRUE,]
```

- Men who weigh as much or more than the average European male (70.8 kg):

```
> patients[patients$Sex=="Male" & patients$Weight<=70.8,]
```