


Introduction to the R/RStudio environment (Part 2)

In this session, we continue from part 1 to introduce a few key data structures. **Exercises for you to complete** are marked as 

Part 2 learning objectives:

1. To introduce you to a data structure called list in R
2. To introduce you to one of the most powerful data structures in R - the data frame and how to filter rows and columns and compute totals across rows and columns
3. To show you how to load in data into a data frame from a CSV file

1. Lists

Lists are similar to vectors in that they are *one-dimensional*. However, the difference is that each element can be a *different type*, i.e. different classes of variable (e.g. number or characters). Elements can also be different data structures, such as vectors and matrices (you can even have tables of data as elements). A list does not print to the console like a vector; instead, each element of the list starts on a new line. Elements are accessed by using double brackets `[[]]` or `$`.

```
> a<-list("a", 1, c(4,5,6))
> a
[[1]]
[1] "a"

[[2]]
[1] 1

[[3]]
[1] 4 5 6
```

This is a list that contains three elements: (i) a single character “a”; (ii) a single number 1; and (iii) a numeric vector (4, 5, 6). You can check with using the `str()` function (how?). To access elements, similar to vectors, we can use the element’s index number (or position), but this time **we have to use double brackets `[[]]`**:

```
> a[[1]]
[1] "a"
> a[[3]]
[1] 4 5 6
```

If you want to return multiple elements (this is actually returned as a list) then you need to just use single brackets `[]`:

```
a[c(1,3)]
[[1]]
[1] "a"

[[2]]
[1] 4 5 6
```

Similar to vectors we can give names to the element positions, which is often better as it makes the R code more readable.

```
> names(a)=c("One", "Two", "Three")
> a
$One
[1] "a"

$Two
[1] 1

$Three
[1] 4 5 6
```

We can access the element by `["name"]` and also using `$name` as follows:

```
> a[["One"]]
[1] "a"

> a$One
[1] "a"
```



Can you select the first and third elements using their name?

Note: we can also assign names to the elements when we create the list (and this is the same for other R variable types such as vectors) like this:

```
> a<-list(One="a", Two=1, Three=c(4,5,6))
```

Just to demonstrate the range of data you can represent in a list try this example ('iris' is a dataset that comes in-built in R):

```
> b<-list(text="Data Science is cool", sequence=1:10, data=iris)
```

```
> b
$text
[1] "Data Science is cool"
```

```
$sequence
[1] 1 2 3 4 5 6 7 8 9 10
```

```
$data
      Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1           5.1         3.5         1.4         0.2    setosa
2           4.9         3.0         1.4         0.2    setosa
3           4.7         3.2         1.3         0.2    setosa
4           4.6         3.1         1.5         0.2    setosa
5           5.0         3.6         1.4         0.2    setosa
6           5.4         3.9         1.7         0.4    setosa
```

```

7          4.6          3.4          1.4          0.3      setosa
8          5.0          3.4          1.5          0.2      setosa
...

```

If you just wanted the first row of the data element you could access it like this (similar to how we accessed elements in a matrix):

```

> b$data[1,]
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1          3.5          1.4          0.2  setosa

```

The \$data element in this previous example is a type of R variable called a *data frame*.

2. Data frame

We have arrived at perhaps the most important data type in R for data science - the *data frame* - as this is pretty much the default or de facto data structure for tabular data. You should feel quite comfortable with the notion of a data frame because they are used for storing spreadsheet-like data. They are basically an extension of lists in that the elements can *hold different data types (or classes)*; however, they are also like matrices in that they are *two-dimensional* (they have rows and columns), but are actually lists of vectors (which is handy because most R functions perform over vectors).

This is a useful tutorial on data frames: <http://www.r-tutor.com/r-introduction/data-frame/>

2.1 Create and assess data frame

You create a data frame using the `data.frame()` function. Let's turn back to our Rio 2016 Olympic Medals example (<https://www.cbssports.com/olympics/news/2016-rio-olympics-medal-tracker/>). Last time we named the rows as countries in a matrix but, if we had done this in a data frame we could have simply had a column for country (we couldn't do this with the matrix as all elements must be the same class):

```

> medals <- data.frame(Country=c("USA", "GBR", "CHN"),
  Gold=c(46,27,26),
  Silver=c(37,23,18),
  Bronze=c(38,17,26))

```

In the above R code we have put each column of the data frame on a separate line to make it easier to read (you can just type this all on one line if you like).

```

> medals
  Country Gold Silver Bronze
1     USA   46     37     38
2     GBR   27     23     17
3     CHN   26     18     26

```

Note that each column can have a different data type or class from the other, but within each column the data must be the same type (this is because the columns are vectors). We can access the rows and columns using similar notation for lists and matrices. To access the gold medals column we could do the following:

```

> medals$Gold
[1] 46 27 26

```

```
> medals[[2]]
[1] 46 27 26
```

This returns a numeric vector (check for this yourself using `class()`). However, if you used the following you would find that the object returned is a data frame:

```
> medals["Gold"]
  Gold
1   46
2   27
3   26
> class(medals["Gold"])
[1] "data.frame"

> medals[2]
  Gold
1   46
2   27
3   26
> class(medals[2])
[1] "data.frame"
```

Note of caution: So be aware of the difference between using `[[]]` and `[]` to access the elements as there is a difference in what gets returned. When we use `[]` this is called a *slice* of the data frame (i.e. you slice out a new data frame from an existing one). To slice out the first and third columns we can do the following (using names or index/position numbers):

```
> medals[c("Country", "Gold")]
  Country Gold
1     USA   46
2     GBR   27
3     CHN   26

> medals[c(1,2)]
  Country Gold
1     USA   46
2     GBR   27
3     CHN   26
```

Now let's focus on the rows and we can do the same - slice the data frame. To access the first row (USA) we can use the following (similar to how we accessed rows in a matrix):

```
> medals[1,]
  Country Gold Silver Bronze
1     USA   46     37     38
```



Can you select the first and third rows?



Can you select the first and third rows and the first and third columns?

Now let's turn to how we can select data from a data frame. In the following two sub-sections, I will show you two ways for now, but there are many other ways (remember the R mantra: there is always more than one way to do something in R!).

Select rows and columns - option 1

But what happens if we have more complex questions? Such as “find me the number of gold medals for USA?” We can make use of the logical operators we saw before. Let's start by returning a data frame that contains just the GBR row:

```
> medals[medals$Country=="GBR",]
Country Gold Silver Bronze
2      GBR   27    23    17
```

Now let's use another logical operator ('|' which is logical OR) to select the rows for CHN and GBR:

```
> medals[medals$Country=="CHN" | medals$Country=="GBR",]
Country Gold Silver Bronze
2      GBR   27    23    17
3      CHN   26    18    26
```

Finally, let's select any rows where the number of Gold medals won is 27 or more:

```
> medals[medals$Gold>=27,]
Country Gold Silver Bronze
1      USA   46    37    38
2      GBR   27    23    17
```

Select rows and columns - option 2

In R there is another (and much easier) way of selecting rows and columns of a data frame using the `subset()` function. This can be used in the following way to select columns:

```
subset(df, select=columnname)
subset(df, select=c(columnname, ..., columnnamen))
```

```
> subset(medals, select=Gold)
Gold
1  46
2  27
3  26

> subset(medals, select=c(Country, Gold))
Country Gold
1     USA  46
2     GBR  27
3     CHN  26
```

Note: you don't need to use quotes "" to identify the column names

Rather confusingly you can also use a parameter in the function called `subset` to select rows. This can include expressions to select *specific* rows. For example let's select the row where Country is 'USA':

```
> subset(medals, subset=(Country=="USA"))
  Country Gold Silver Bronze
1    USA   46    37    38
```

What's neat is that you can also use the select and subset together to select rows and columns. So to select the row where Country is 'USA' but only return the Country and Gold columns we can do this:

```
> subset(medals, select=c(Country, Gold), subset=(Country=="USA"))
  Country Gold
1    USA   46
```



Select countries who won 27 or more gold medals and return the Country and Gold columns using the subset() command

2.2 Add rows

Let's suppose we want to add rows to our data frame (e.g. Russia, Germany and Japan from our medals example). We can use the row bind or rbind() function like we did for matrices - this time we use the command as rbind(data frame A, data frame B). To do this we start by creating a new temporary data frame (called newCountry) with the new data and then add this as a new row to the existing data frame medals:

```
> newCountry<-data.frame(Country="RUS", Gold=19, Silver=18, Bronze=19)
> medals<-rbind(medals, newCountry)
> medals
  Country Gold Silver Bronze
1    USA   46    37    38
2    GBR   27    23    17
3    CHN   26    18    26
4    RUS   19    18    19
```



Add rows to the medals data frame for Germany and Japan (you can overwrite the newCountry data with the new values)

2.3 Add Columns

A common task when using data frame (and spreadsheets) is to compute the total sum of rows and columns¹. Suppose we want to create a new column in our data frame that stores the total number of medals won by a country. We can do this using the following where we create new Total column:

```
> medals$Total<-medals$Gold + medals$Silver + medals$Bronze

>medals
  Country Gold Silver Bronze Total
1    USA   46    37    38    121
2    GBR   27    23    17     67
3    CHN   26    18    26     70
4    RUS   19    18    19     56
5    GER   17    10    15     42
```

¹ A helpful discussion of this can be found here:

<https://www.computerworld.com/article/2486425/business-intelligence/business-intelligence-4-data-wrangling-tasks-in-r-for-advanced-beginners.html?page=2>

```
6      JPN    12      8     21    41
```

If you want to remove Total column again you can use the following:

```
> medals<-subset(medals, select=c(Country, Gold, Silver, Bronze))
> medals
  Country Gold Silver Bronze
1     USA   46    37    38
2     GBR   27    23    17
3     CHN   26    18    26
4     RUS   19    18    19
5     GER   17    10    15
6     JPN   12     8    21
```

2.4 Computing totals across rows and columns

One of the hardest aspects of learning R is that you find quickly that there are typically many ways to carry out the same task. So for the previous task of computing totals for each row we could have also used the `transform()` function:

```
> medals<-transform(medals, Total=Gold+Silver+Bronze)
```

Or the `mapply()` function (this stands for ‘multiple apply’ to apply a function to multiple list or vector arguments – see `?mapply`). This is perhaps the most flexible way of doing the task because you can use any function (in this case we use `sum()`). This command applies the `sum()` function to each row:

```
> medals$Total<-mapply(sum, medals$Gold, medals$Silver, medals$Bronze)
```

Finally, another way of doing this is to use the `rowSums()` function that sums values in a row:

```
> medals$Total<-rowSums(medals[c(2,3,4)])
```

We can do the same as above but using the names of the columns instead of index positions:

```
> medals$Total<-rowSums(medals[c("Gold", "Silver", "Bronze")])
```

Before we sum the total of the columns, let’s think about how we could sort data in the columns. For example, suppose we want to sort the data frame rows by the number of Bronze medals won by each country. We could do this as follows:

```
> medals[order(medals$Bronze),] # Sort the medals by Bronze (ascending)
  Country Gold Silver Bronze Total
5     GER   17    10    15     42
2     GBR   27    23    17     67
4     RUS   19    18    19     56
6     JPN   12     8    21     41
3     CHN   26    18    26     70
1     USA   46    37    38    121
```

```
> medals[order(-medals$Bronze),] # Sort the medals by Bronze (descending)
  Country Gold Silver Bronze Total
```

1	USA	46	37	38	121
3	CHN	26	18	26	70
6	JPN	12	8	21	41
4	RUS	19	18	19	56
2	GBR	27	23	17	67
5	GER	17	10	15	42

Now, let's suppose we want to compute the totals of the columns (i.e. the total number of Gold medals, Silver medals, etc.). For this we can use the `colSums()` function:

```
> colSums(medals[c("Gold", "Silver", "Bronze", "Total")])
Gold Silver Bronze Total
147     114    136    397
```

Finally, these are the main functions (or attributes) for data frames:

- `head()` - see first 6 rows
- `tail()` - see last 6 rows
- `dim()` - see dimensions
- `nrow()` - number of rows
- `ncol()` - number of columns
- `str()` - structure of each column
- `names()` - will list the names attribute for a data frame, which gives the column names.

Also, what you will find useful for viewing data frames is the `View()` function. This will display the data structure in the Script window within a new tab (e.g. “medals”):

```
> View(medals)
```



You may find this tutorial helpful regarding computing subsets etc.

<http://www.statmethods.net/management/subset.html>

3. Loading in data from CSV files

Let's scale things up and work with the entire medals dataset for the Rio 2016 Olympic Games (which we used before). To make it easier for you I have already downloaded the dataset in Comma Separated Value (CSV) format which you can find on Blackboard; a file called “Rio2016.csv”. The data is similar to what we have already seen except that the country names are longer (not using abbreviations). There are many ways of importing data into R and we will cover some of these in future sessions, but one of the simplest for CSV files is the `read.csv()` function (NOTE: you may have to replace the file name with the correct file path subject to your own computer):

```
> rio2016Medals<-read.csv("Rio2016.csv", header=TRUE)
```

The `read.csv()` function (and you can check help for more information using `? read.csv`) reads in a spreadsheet-like data stored in CSV format as a data frame (in this case called `rio2016Medals`²). The `header=TRUE` parameter is used to indicate that the CSV file has column headers. You can get a

² It is worth emphasising that you can give your R objects any names but they should be meaningful and also the format you use should be consistent. For example I tend to use a lowercase lowercase character and then a capital letter for subsequent words in the name, e.g. `rio2016Medals`.

quick overview of the data using the `View()` function (shows the table in a new tab in the Script window):

```
> View(rio2016Medals)
```

If you wanted to just look at the first size rows we can use the `head()` function:

```
> head(rio2016Medals)
      Country Gold Silver Bronze
1 United States  46    37    38
2  South Korea   9     3     9
3    France    10    18    14
4  Australia    8    11    10
5   Germany    17    10    15
6    Taiwan     1     0     2
```



Similar to what we did before try and compute the total number of medals won for each country and add this as a fifth column to `rio2016Medals`.



Now try ordering the rows by the total number of medals won.

Note: what's neat with the `order()` function is that we can order first by one column and then another (and another etc.) by simple adding columns to the `order` function. This is really helpful in our case when we have ties (i.e. the same number of total medals won) and want to use other criteria to order the rows:

```
data frame[order(column1, column2, ..., columnn),]
```



Try using `order()` to sort the data first by the number of total medals won (in descending order) and then by the number of gold medals won (also in descending order).



Calculate the total number of gold, silver and bronze medals won at Rio 2016. Also, can you work out the overall number of medals won by all countries in the Olympic Games in 2016?



You may find the following tutorials useful that include sections on data frames. Have a go at working through them:

<http://www.r-tutor.com/r-introduction/data-frame>

<https://www.analyticsvidhya.com/blog/2016/02/complete-tutorial-learn-data-science-scratch/>

4. Further exercises (optional)

If you would like to play more with datasets then you can make use of datasets that are in-built to R. You can also get access to datasets from here:

<https://vincentarelbundock.github.io/Rdatasets/datasets.html>

In particular if you load in a library called MASS (we will be discussing libraries and packages next week so just following the commands given for now) then you can get access to more datasets.

```
> library(MASS)
```

If you type `data()` at the command prompt RStudio will present you with a list of datasets you can choose from. Pressing F1 will provide additional information in the output window. There are datasets of all different kinds. You might want to try and find datasets in the form of data frames to play with (e.g. “women”)

```
> data("women")
> 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
```

Useful R resources

- R home page: <https://www.r-project.org>
- RStudio home page: <https://www.rstudio.com>
- Base R cheat sheet: <https://www.povertyactionlab.org/sites/default/files/r-cheat-sheet.pdf>
- Data Science blog by Dave Langer: <http://www.daveondata.com>)
- Dave Langer's Introduction to Data Science with R – Data Analysis Part 1: <https://youtu.be/32o0DnuRjfg>
- Swirl interactive tutorial on R: https://github.com/swirldev/swirl_courses
- A (very) short introduction to R by Paul Torfs and Claudia Brauer: <https://cran.r-project.org/doc/contrib/Torfs+Brauer-Short-R-Intro.pdf>
- R Tutor: <http://www.r-tutor.com/r-introduction>
- Introduction to data science with R: <https://www.analyticsvidhya.com/blog/2016/02/complete-tutorial-learn-data-science-scratch/>
- R Cookbook (chapters 1 and 2)
- Learning R book (Chapters 1, 2, and 4)
- R for Data Science (Chapters 1, 2 and 16): <http://r4ds.had.co.nz/>