Lab

Data Manipulation

Lab objectives:

- To revise the data types which have been introduced and used in previous labs
- To manipulate data structures to prepare data for analysis

Introduction

Before we manipulate data, we will revise the main data types available in R.

Note that some of the content of this lab was covered during lab 1 when the basics of R was covered.

You will normally be working with data frames (end of this lab) and vectors as well as the basic data types (character, numeric and logical) but list and matrices are included for completeness.

Load library tidyverse

```
library(tidyverse)
```

Character (string)

String values are of type character in R. They must be quoted (both single and double quotes work). Try

```
a <- "CMM020"
a
```

You can check that indeed **a** is a of character type with the **class f**unction. Try

```
class(a)
```

If an object is not of type character, you can convert it to a character using function as.character. Try

```
a <- 1234.56
a
class(a)
b <- as.character(a)
b
class(b)
```

You can get parts of a character value using function **substring.** By default it uses one blank space as separator. Try

```
c <- "CMM020 Data Visualisation and analysis"
school <- substring(c, 1,2)
## CM is the code for computing science and digital media
level <- substring(c,3,3)
## level M is for Master
number <- substring(c, 4,6)
title <- substring(c,7)
```

You can put strings together using the function **paste.** Try

```
a <- "CMM020"
b <- "Data visualisation and analysis"

c <- paste(a,b)  # default separator is one blank space
c

d <- paste(a,b, sep="")   # no blank space
d

d <- paste(a,b, sep=": ")   # colon and blank space as separator
d
```

You can concatenate a number of character strings together. Try

```
module <- paste("This module's school code is", school,
                "with level code",level)
module
```

To substitute the first occurrence of the word "Data" with the word "information" try

```
d <- sub("Data", "information", c)
d
```

Note that it is case sensitive.

Numeric (real numbers)

Numeric is the default data type in R. Decimal values are of type numeric in R. Try

```
a <- 1234.56
a
```

You can check that indeed **a** is numeric with the **class** function

```
class(a)
```

You can also test whether a variable is numeric with function **is.numeric.** Try

```
is.numeric(a)
b <- "this is not numeric"
is.numeric(b)
```

You can force a string (or other data types|) to be of type numeric (if appropriate) using as.numeric. For example

```
e <- "1234.56"
is.numeric(e)
g <- as.numeric(e)
is.numeric(g)
```

Integer

The numeric data type is the default, for R. Try

```
a <- 1234
class(a)
```

Despite variable **a** having an integer value, it is of class numeric.

To assign a variable an integer data type, this needs to be specified using the **as.integer** function. Try

```
b <- as.integer(1234)
class(b)
```

If you have a variable of type numeric or string, you can force its value to be an integer (including rounding) using **as.integer.** Try

```
c <- 1234.56
c
d <- as.integer(c)
d
```

You can check if a variable is an integer. Try

```
is.integer(a)
is.integer(b)
```

```
is.integer(c)
is.integer(d)
```

Logical (Boolean)

In R, Booleans are of **logica**l data type. Booleans have values **`TRUE`** or **`FALSE.`**

```
a <- 1234.56
b <- is.integer(a)
b
class(b)
```

To check that a logical has value TRUE use function `isTRUE`

```
isTRUE(b)
```

R supports a number of fucntions which return Booleans. For example, comparisons between numeric or integer values such as **`>, <, >=, <=, ==.`**
R supports the standard logical operators:

- And: **`&`**
- Or: **`|`**
- Not: **`!`**
- Exclusive or: **`xor(expr1,expr2)`**

Try

```
a <- 20
b <- 40
c <- 50
(a > b)
(a < b) & (b < c)
(a > b) | (c > b)
xor((a < b) , (b < c))
xor((a < b) , (b > c))
```

Vector (list of basic components, all of the same type)

A vector is a list of values (components) of the same basic type. Vectors are defined with function c. You covered vectors in lab one.

Try the following 2 vectors, the first one containing character values and the second one numeric values.

```
firstname<- c("John", "Mary", "Tracy", "Duncan", "Omar","Sania")
matric <-c( 122345, 023451,072737,092959,075777, 099969)
```

Larger vectors, with more components, can be obtained by aggregating shorter ones. Try

```
firstname1 <- c("John", "Mary", "Tracy", "Duncan", "Omar","Sania")
firstname2 <- c("Adriana", "Lesley")
firstname <- c(firstname1,firstname2)
firstname
```

To retrieve the 3rd component in firstname

```
firstname[3]
```

To retrieve components between position 3 and position 5

```
firstname[3:5]
```

If the index is out of range, NA is returned for all components with an invalid index. Try

```
firstname[3:9]
```

To retrieve a list of non consecutive components, e.g. components in positions 2, 4 and 6

```
firstname[c(2,4,6)]
```

The above is equivalent to

```
indexes <-c(FALSE,TRUE,FALSE,TRUE,FALSE,TRUE,FALSE,FALSE)
firstname[indexes]
```

Note that indexes do not need to be in ascending order and can be duplicated. Try

```
firstname[c(8,4,6,4)]
```

Note that when 2 vectors are put together, their values are coerced to be of the same type. Try

```
firstname<- c("John", "Mary", "Tracy", "Duncan", "Omar","Sania")
matric <- c( 122345, 023451,072737,092959,075777, 099969)
student <- c(firstname, matric)
student
```

As you can see, the matriculation numbers (previously of type numeric) are of type character in student.

To exclude a vector component, negate the index of the component.

```
Firstname <- c("John", "Mary", "Tracy", "Duncan", "Omar","Sania")
firstname[-3]
```

To update the name of the 6th element

```
firstname[6] <- "Sannyann"
firstname
```

The number of components in a vector can be obtained using **`length`**. Try

```
length(firstname)
```

You can name each component and subsequently use names to access components with function **`names`**.

```
modules <- c("CMM020", "Data Vis", "CMM007", "SW Project Eng")
names(modules) <-c("Module1Num", "Module1Name",
"Module2Num", "Module2Name")
modules["Module2Num"]
```

You can access several components

```
modules[c("Module1Num", "Module2Num")]
```

Factors

Factors are used to store data which have a limited number of different values or categories which are called levels. They are often used with datasets which are stored in data frames (see below).

For example, assume you have the following vector of character strings

```
studentname<- c("John", "Mary", "Tracy", "John, Duncan",
"Omar","Sania", "Mary")
```

We can obtain a vector of factors using the `as.factor` function which converts to factors.

```
factorstudent <- as.factor(studentname)
```

To check the allowed values use `levels`

```
levels(factorstudent)
```

To add a new value "Chloe", it needs to be of type factor

```
newname <- factor("Chloe")
factorstudent <- c(factorstudent,newname)
factorstudent
```

Check what happens if you do not have the new name as a factor, i.e.

```
newname <- "Chloe"
factorstudent <- c(factorstudent,newname)
factorstudent
```

To convert a factor to a character use as.character

```
charstudent <- as.character(factorstudent)
charstudent
```

Matrix

A **matrix** is a 2D list of elements of the same basic type.

```
vals <-c(200,30,24,3000,550,500,600,22,77,430,23,10)
m <- matrix( vals, nrow=4,ncol=3,byrow=T)
m
```

To access element at row 4, column 2

```
m[4,2]
```

To access row 3

```
m[3,]
```

Function **t** can be used to transpose (change rows into columns) a matrix. Try

```
mtransposed <- t(m)
mtransposed
```

Function **cbind** can be used to combine matrices with the same number of rows. Try

```
vals1 <-c(200,30,24,3000,550,500,600,22,77,430,23,10)
m <- matrix( vals1, nrow=4,ncol=3,byrow=T)
m
vals2 <-c(2,3,4,5,6,7,8,9)
```

```
n <- matrix(vals2, nrow=4,byrow=T)
n
o <- cbind(m,n)    # o combines the columns of  m and n
o
```

To add rows instead of columns use **rbind**

```
vals1 <-c(200,30,24,3000,550,500,600,22,77,430,23,10)
m <- matrix( vals1, nrow=4,ncol=3,byrow=T)
m
vals2 <-c(2,3,4,5,6,7)
n <- matrix(vals2, ncol=3,byrow=T)
n
o <- rbind(m,n)    # o combines the rows of m and n
o
```

To obtain all the elements in a matrix use **c**

```
vals <- c(o)
vals
```

You can assign names to a matrix rows and columns using **dimnames.** Use the help to find out more

```
?dimnames
```

List

A list is a vector of (non-basic) objects.

```
modules <- c("CMM020","CMM007")
students <-c("John", "Mary", "Tracy", "Duncan", "Omar","Sania")
rooms <- c(1,2,3,4,5,6,7,8)
info <-list(modules, students, rooms,01224262700)
info
```

To retrieve the second component of list info

```
info[[2]] ## this is a vector of character values
```

To obtain a list with just the second component of list info

```
info[2] ## this is a list
```

To retrieve the 4th value of the second component in list info

```
info[[2]][4] # get the second component, then find its 4th value
```

You can name each list component. For example, try

```
modules <- c("CMM020","CMM007")
students <-c("John", "Mary", "Tracy", "Duncan", "Omar","Sania")
rooms <- c(1,2,3,4,5,6,7,8)
info <-list(modulenames=modules, studentnames = students,
rooomnumbers=rooms,tel=01224262700)
info
```

This allows you to access the elements in the list using their name. Try

```
info["modulenames"]
```

This is equivalent to

```
info$modulenames
```

Compare the above with

```
info[["modulenames"]]
```

*You can **attach** a list so that its elements can be used without mentioning the list it comes from.* ***You should always detach the list when you finish using it.***

```
attach(info)
modulenames
```

When you have finished using info detach it with

```
detach(info)
```

Data frame

You have been using data frames to store your datasets. A data frame is a list of vectors, all of them of the same length. Function **data.frame** is used for data frame construction. For example try

```
module <- paste("module school code is", school, "with level
code",level)
firstname<- c("louise", "John", "Mary", "Tracy", "Duncan",
"Omar","Sania")
matric <-c( 222222,122345, 023451,072737,092959,075777, 099969)
course <- c("ITEI", "ITBI", "ITCY", "DS" ,"ITEI", "ITCY", "ITEI")
start <- c("Jan18","Sep 17","Jan 18","Sep 17","Sep 17","Jan 18","Sep
17")
students <- data.frame(firstname, matric, course, start,
```

```
                  stringsAsFactors=T)
students
```

Note how the first line has been composed of the names of the variables we have used to build the data frame. This is the header. Also note the `stringsAsFactor=T` part. This means strings will be factors, not character strings.

To retrieve information about the 3rd student (3rd row)

```
students[3,]  # data frame of 1 row
```

To **retrieve several rows** use a vector of indexes. For example, to obtain the 3rd, 5th and 6th rows

```
students[c(3,5,6),]
```

To **retrieve** the 2nd **column values**

```
students[,2] # vector of numeric values
```

Equivalent to

```
students$matric # vector
```

Also equivalent to

```
students[[2]]  # vector
```

Compare the above to

```
students[2]  # data frame
```

Which is the same as

```
students["matric"] # data frame
```

To **retrieve the element at** 3rd **row**, second **column**

```
students[3,2]
```

To find out the number of rows in your data frame you can use the nrow function

```
nrow(students)
```

For number of columns use ncol

```
ncol(students)
```

Selecting specific rows, e.g. the students who study in course ITEI:

```
studentsITEI <- students |> filter(course == "ITEI")
head(studentsITEI)
```

Selecting specific rows using 2 conditions, e.g. the students who study in course ITEI who started in Sep 17:

```
studentsITEISep17 <- students |> filter(course == "ITEI", start="Sep 17")
head(studentsITEISep17)
```

Similarly selecting students who studied ITEI who have less than £150 outstanding

```
studentsITEILess150 <- students |>
     filter(course ==  "ITEI", outstanding < 150)
head(studentsITEILess150)
```

Adding a new column can be done by creating a vector of values and assigning it to the new column name. Try

```
birthdate <- c("06-11-1990", "02-04-2000","14-06-1993","29-01-1989","01-01-1991", "16-10-1976","20-12-1999")
students$birth <-birthdate
students
```

Before testing an equivalent method, delete the birth column

```
students[5] <- NULL  # delete column birth
students
```

An equivalent method to add the column is

```
birthdate <- c("06-11-1990", "02-04-2000","14-06-1993","29-01-1989","01-01-1991", "16-10-1976","20-12-1999")
```

```
students[5] <-birthdate  # assign the values of birthdate to the 5th column
colnames(students)[5] <- "birth"   # rename the new column
students
```

Adding a new column based on existing columns, e.g. adding a column with the first name and the matriculation number concatenated together.

```
students$reference <- paste(students$matric, students$firstname,
                            sep= "-")
```

In the paste function, sep= "-" indicates that a hyphen will be used to separate the various components being pasted together.

Adding two data frames together

Before we did obtain all the students in ITEI as follows.

```
studentsOG <- students[course == "ITEI",]
```

We can obtain another dataframe with the students in ITBI or ITCY.

```
studentsBICY <- students[course == "ITBI" | course =="ITCY",]
View(studentsBICY)
```

Use `rbind` to add rows. Both dataframes must have the same number of columns. For example, we combine the students in ITEI with the students in ITBI and ITCY to obtain a new dataframe:

```
studentsIT <- rbind(studentsOG,studentsBICY)
```

Use `cbind` to add columns. Both dataframes must have the same number of rows.

```
#dataframe with matriculation number and first name
details1<- students[,c("matric","firstname")]

#dataframe with matriculation number and course
details2 <- students[, c("matric","course")]

# adding the two dataframes together
details <- cbind(details1, details2)
```

```
View(details)
```

The matric column is repeated. We can **delete** one of the **column**s as follows:

```
details$matric <- NULL
```

Processing columns in data frames

Assume that you are only interested in the year students were born. The year corresponds to positions 7-10 in the birth column.

You could create a new column containing just the year as follows

```
students$year <- substring(students$birth,7,11)
students
```

Add a new column containing outstanding fees for each student

```
students$outstanding <- c(1200, 0,3200,500,0,0,100)
```

We want to check outstanding fees by course. We can create a dataframe with mean values as follows:

```
meanOutstanding <-aggregate(students$outstanding,
                    by=list(students$course),
                    FUN=mean, na.rm=TRUE)
colnames(meanOutstanding) <- c("course","meanFeeDue")
View(meanOutstanding)
```

Compare this with a similar process but taking into account course and start time

```
meanOutstanding <-aggregate(students$outstanding,
                    by=list(students$course, student$start),
                    FUN=mean, na.rm=TRUE)
colnames(meanOutstanding) <- c("course","start","meanFeeDue")
View(meanOutstanding)
```

You could follow the same process to obtain the sum of outstanding fees per course (and start time) by replacing “mean” with “sum” in the function (FUN) field.

Transposing data frames

You can use function t to transpose data frames, but beware, the column names will need to be sorted. First lets get the mean and the sum

```
# obtain mean and sum values – note that there are no subgroups
# no start considered, only course)

meanOutstanding <-aggregate(students$outstanding,
                    by=list(students$course),
                    FUN=mean, na.rm=TRUE)
colnames(meanOutstanding) <- c("course", "meanFeeDue")
View(meanOutstanding)

sumOutstanding <-aggregate(students$outstanding,
                    by=list(students$course),
                    FUN=sum, na.rm=TRUE)
colnames(sumOutstanding) <- c("course", "sumFeeDue")
View(meanOutstanding)
```

Put together into a data frame

```
statsOutstanding <- data.frame(meanOutstanding$course,
meanOutstanding$meanFeeDue, sumOutstanding$sumFeeDue)
colnames(statsOutstanding) <- c("course","mean", "sum")

View(statsOutstanding)
```

We would like to transpose this data frame

```
tstatsOutstanding <- t(statsOutstanding)
View(tstatsOutstanding)
```

To rename the header

```
colnames(tstatsOutstanding) <- meanOutstanding$course
View(tstatsOutstanding)
```

To remove 1st row, which just contains the course names

```
tstatsOutstanding <- tstatsOutstanding[c(-1),]
View(tstatsOutstanding)
```

To rename row names you could use

```
rownames(tstatsOutstanding) <- c("average", "aggregate")
View(tstatsOutstanding)
```

Obtaining summary statistics

The summary function can be used to obtain summary statistics for data frames. Try the following (note that provenOilreserves.csv was given for a previous lab).

```
provenOilReserveWEurope <- read.csv("provenOilReserveWEurope.csv",
header=T, stringsAsFactors=T)
summary(provenOilReserveWEurope)
```

You will see that for numeric values, you get:

- Minimum
- Q1
- Median = Q2
- Mean
- Q3
- Maximum

For categorical values (factors), you get the number of times each value appears.

Exercises

1. Look at MT.Barrels in the summary function you used above. What can you say about its distribution?
2. Use functions to obtain a data frame which contains the mean, the medianand the difference between the mean and median from the outstanding fees of the student dataset which you prepared earlier. The function names for the calculation of the mean and median are `mean` and `median`.
3. Apply the data transformation functions which you have learnt to some of the datasets that you have seen in earlier labs.