---

title: "Lab 8 - Data Manipulation - feedback solutions"

author: "IA"

date: ""

output: html\_document

---

Loading libraries and setting the working directory to be the directory this file is in.

```{r, warning = F}

library(tidyverse)

```

### Character (string)

```{r}

a <- "CMM020"

a

class(a)

```

```{r}

a <- 1234.56

a

class(a)

b <- as.character(a)

b

class(b)

```

#### Substrings

```{r}

c <- "CMM020 Data Visualisation and analysis"

school <- substring(c, 1,2)

level <- substring(c,3,3)

number <- substring(c, 4,6)

title <- substring(c,7)

```

#### Putting strings together

Below we construct 2 strings, which we then concatenate (paste them together). Note the use of different separators below.

```{r}

# pasting strings together

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

```

Concatenating more than 2 strings.

```{r}

# concatenation

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

module

```

#### Replacing text in a string (string substitution)

```{r}

# substitution

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

d

```

### Simple numerical data types

```{r}

# Numeric values

a <- 1234.56

a

class(a)

is.numeric(a)

b <- "this is not numeric"

is.numeric(b)

e <- "1234.56"

is.numeric(e)

g <- as.numeric(e)

is.numeric(g)

```

#### |Integer

```{r}

a <- 1234

class(a)

b <- as.integer(1234)

class(b)

c <- 1234.56

c

d <- as.integer(c)

d

is.integer(a)

is.integer(b)

is.integer(c)

is.integer(d)

```

### Logical (boolean)

```{r}

# Logical (boolean)

a <- 1234.56

b <- is.integer(a)

b

class(b)

isTRUE(b)

```

#### Using logical operators

```{r}

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))

```

### Vectors

```{r}

# Vectors

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

matric <-c( 122345, 023451,072737,092959,075777, 099969)

```

Aggregating vectors.

```{r}

firstname1 <- c("John", "Mary", "Tracy", "Duncan", "Omar","Sania")

firstname2 <- c("Adriana", "Lesley")

firstname <- c(firstname1,firstname2)

firstname

```

Retrieving individual components, or a group of components.

```{r}

firstname[3]

firstname[3:7]

```

Invalid index

```{r}

firstname[3:9]

```

Non-consecutive components

```{r}

firstname[c(2,4,6)]

```

Equivalently

```{r}

indexes <-c(FALSE,TRUE,FALSE,TRUE,FALSE,TRUE,FALSE,FALSE)

firstname[indexes]

```

Coercion when vectors are put together. Numeric values are changed to character string

```{r}

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

matric <- c( 122345, 023451,072737,092959,075777, 099969)

student <- c(firstname, matric)

student

```

Excluding vector members.

```{r}

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

firstname[-3]

```

Updating a vector member.

```{r}

firstname[6] <- "Sannyann"

firstname

```

Checking number of components in a vector

```{r}

length(firstname)

```

Naming components of vectors.

```{r}

modules <- c("CMM020", "Data Vis", "CMM007", "SW Project Eng")

names(modules) <-c("Module1Num", "Module1Name", "Module2Num", "Module2Name")

modules["Module2Num"]

```

```{r}

modules <- c("CMM020", "Data Vis", "CMM007", "SW Project Eng")

names(modules) <-c("Module1Num", "Module1Name", "Module2Num", "Module2Name")

modules["Module2Num"]

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

# accessing named components

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

```

### Factors

The following vector contains character strings

```{r}

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

```

Converting to factors

```{r}

factorstudent <- as.factor(studentname)

factorstudent

```

Checking allowed levels

```{r}

levels(factorstudent)

```

Adding a new value

```{r}

newname <- factor("Chloe")

factorstudent <- c(factorstudent,newname)

factorstudent

```

Incorrect way of adding a new value as a character string - note how factors end up being numbers.

```{r}

newname <- "Chloe"

factorstudent <- c(factorstudent,newname)

factorstudent

```

```{r}

charstudent <- as.character(factorstudent)

charstudent

```

### Matrix

```{r}

vals1 <-c(200,30,24,3000,550,500,600,22,77,430,23,10)

m <- matrix( vals1, nrow=4,ncol=3,byrow=T)

m

```

Accessing element at row 4 column 2.

```{r}

m[4,2]

```

Accessing row 3.

```{r}

m[3,]

```

Transposing rows and columns.

```{r}

mtransposed <- t(m)

mtransposed

```

Combining matrices.

```{r}

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

```

Adding rows instead of columns

```{r}

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

```

Obtaining all the elements in a matrix.

```{r}

vals <- c(o)

vals

```

# Factors

```{r}

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

studentname

factorstudent <- as.factor(studentname)

factorstudent

levels(factorstudent)

```

Adding a new value

```{r}

newname <- factor("Chloe")

factorstudent <- c(factorstudent,newname)

factorstudent

```

Adding a new character string value.

```{r}

newname <- "Chloe"

factorstudent <- c(factorstudent,newname)

factorstudent

```

The above is incorrect, as seen by the results.

Converting to character string

```{r}

charstudent <- as.character(factorstudent)

charstudent

```

# Lists

```{r}

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

```

Retrieving the second component of list info.

```{r}

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

```

Obtaining a list with the second component of list.

```{r}

info[2] ## this is a list

```

Retrieving the the 4th value of the second component in list info.

```{r}

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

```

Naming list components.

```{r}

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

```

Accessing the elements in the list using their name

```{r}

info["modulenames"]

# equivalent to

info$modulenames

# Compare the above with

info[["modulenames"]]

```

Attaching lists.

```{r}

attach(info)

modulenames

```

Detaching after use (this is important!).

```{r}

detach(info)

```

### Data frames

```{r}

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("Jan 18","Sep 17","Jan 18","Sep 17","Sep 17","Jan 18","Sep 17")

students <- data.frame(firstname, matric, course, start)

students

```

Retrieving information about the 3rd student (3rd row).

```{r}

students[3,] # data frame of 1 row

```

Retrieving several rows using a vector of indexes.

```{r}

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

```

Retrieving the 2nd column values.

```{r}

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

```

Retrieving the element at 3rd row, second column.

```{r}

students[3,2]

```

Finding out the number of rows in your data frame.

```{r}

nrow(students)

```

Finding out the number of columns in your dataframe.

```{r}

ncol(students)

```

Selecting specific rows

```{r}

studentsOG <- students |> filter(course == "ITEI")

head(studentsOG)

```

```{r}

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

head(studentsOG)

```

Students from course ITEI with less thatn £150 outstanding

```{r}

studentsOG <- students |> filter(course == "ITEI", outstanding < 150)

head(studentsOG)

```

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

```{r}

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

```

Deleting the birth column.

```{r}

students[5] <- NULL # delete column birth

students

```

An equivalent method to add the column.

```{r}

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

```

#### Aggregation

```{r}

# Creating a new column containing just the year

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

students

# Adding a new column containing outstanding fees for each student

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

```

Checking outstanding fees by course.

```{r}

# Create a dataframe with mean values

meanOutstanding <-aggregate(students$outstanding,

by=list(students$course),

FUN=mean, na.rm=TRUE)

# and getting outstanding fees

colnames(meanOutstanding) <- c("course","meanFeeDue")

View(meanOutstanding)

```

Similar process but taking into account course and start time.

```{r}

meanOutstanding <-aggregate(students$outstanding,

by=list(students$course, students$start),

FUN=mean, na.rm=TRUE)

colnames(meanOutstanding) <- c("course","start","meanFeeDue")

View(meanOutstanding)

```

Obtain mean values - note that there are no subgroups no start is considered, only course.

```{r}

meanOutstanding <-aggregate(students$outstanding,

by=list(students$course),

FUN=mean, na.rm=TRUE)

colnames(meanOutstanding) <- c("course", "meanFeeDue")

View(meanOutstanding)

```

Obtain addition (sum) values.

```{r}

sumOutstanding <-aggregate(students$outstanding,

by=list(students$course),

FUN=sum, na.rm=TRUE)

colnames(sumOutstanding) <- c("course", "sumFeeDue")

View(meanOutstanding)

```

Putting all results together into a data frame

```{r}

statsOutstanding <- data.frame(meanOutstanding$course, meanOutstanding$meanFeeDue, sumOutstanding$sumFeeDue)

colnames(statsOutstanding) <- c("course","mean", "sum")

View(statsOutstanding)

```

#### Transposing

Transpose dataframe.

```{r}

tstatsOutstanding <- t(statsOutstanding)

View(tstatsOutstanding)

```

Renaming the header.

```{r}

colnames(tstatsOutstanding) <- meanOutstanding$course

View(tstatsOutstanding)

```

Removing 1st row, which just contains the course names.

```{r}

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

View(tstatsOutstanding)

```

Renaming row names.

```{r}

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

View(tstatsOutstanding)

```

# Summary

```{r}

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

summary(provenOilReserveWEurope)

```

### Exercise 1

The mean for MT barrels is larger than the median, pointing to a skewed distribution.

The median value is quite low. 50% of values are below the median. The distance between the median and q1 (or even the minumum value!) is a lot lower than the distance to Q3. The maximum is really high when compared to the other values.

### Exercise 2

```{r}

meanOutstanding <-aggregate(students$outstanding,

by=list(students$course),

FUN=mean, na.rm=TRUE)

```

```{r}

medianOutstanding <-aggregate(students$outstanding,

by=list(students$course),

FUN=median, na.rm=TRUE)

```

```{r}

diffStats <- meanOutstanding[2] - medianOutstanding[2]

```

```{r}

statsOutstanding <- data.frame(meanOutstanding[1], meanOutstanding[2], medianOutstanding[2], diffStats[1])

colnames(statsOutstanding) <- c("course", "mean","median", "difference")

```

### Exercise 3

The solution to this exercise depends on the datasets sele