Contents

1 Programming Exercise

1.1 Exercise: Dataset reporting function

For an in-built data set (e.g. iris, mtcars etc), construct a function called `Report` that returns some or all of following outputs:

- dimensions of the data set (`dims`)
- names of components of the data set (`names`)
- summary statistics (`summary`)

Use iris as the default data set.

```
Report <- function(dataset=iris,dims=FALSE,
     cnames=FALSE,smry=TRUE)
 {
 if(dims==TRUE)
  {
   print(dim(dataset))
  }
  if(cnames==TRUE)
  {
   print(names(dataset))
  }
   if(smry==TRUE)
  {
   print(summary(dataset))
  }
 }
Report()
```

Constructing the output for this as a list. (Some lines of code to be added)

```
ReportList <- function(dataset=iris,dims=FALSE,
   cnames=FALSE,smry=TRUE)

 {

 output=list(dims=numeric(),cnames=character(),smry=list())

 if(dims==TRUE)
  {
   output$dims=dim(dataset)
  }

  if(cnames==TRUE)
  {
   output$cnames=names(dataset)
  }


  if(smry==TRUE)
  {
   output$smry=summary(dataset)
  }
  return(output)
 }
```

```
ReportList(iris)
ReportList(mtcars,T,T,F)
ReportList(mtcars,F,T,F)
```

1.2 Data Set for this exercise

The zip file contains 332 comma-separated-value (CSV) files containing pollution monitoring data for fine particulate matter (PM) air pollution at 332 locations in the United States.

Each file contains data from a single monitor and the ID number for each monitor is contained in the file name. For example, data for monitor 200 is contained in the file "200.csv".

Each file contains three variables:

- **Date**: the date of the observation in YYYY-MM-DD format (year-month-day)
- **sulfate**: the level of sulfate PM in the air on that date (measured in micrograms per cubic meter)
- **nitrate**: the level of nitrate PM in the air on that date (measured in micrograms per cubic meter)

1.3 Some Useful Functions

1.3.1 The `nchar()` function

This command returns the number of characters in the argument.

```
> string=c("kevin")
> nchar(string)
[1] 5
> nchar(1001)
[1] 4
```

1.3.2 The `paste()` function

This command creates a string of specified components. The default is to have whitespace between each component. This can be removed with the additional argument `sep=""`.

```
> x=5
> paste("file",x,".csv")
[1] "file 5 .csv"
>
> paste("file",x,".csv",sep="")
[1] "file5.csv"
```

```
> filenm(2)
[1] "file 2 .csv"
```

1.3.3 The `gsub()` function

The `R` command `gsub()` is used to replace a character or piece of text with another in a specified string

```
> string=c("kevin")
> gsub("k","s",string)
[1] "sevin"
```

1.3.4 The `list.files()` function

This command is used to produce a list of files in a specified directory.

```
getwd() # working directory
list.files(getwd()) # List all files in working directory
```

1.3.5 The `sprintf()` function

This command returns a character vector containing a formatted combination of text and variable values. The structure of the command is `sprintf(format, input)`.

```
> sprintf("%f", pi)
[1] "3.141593"
> sprintf("%.3f", pi)   # 3decimal places
[1] "3.142"
> sprintf("%1.0f", pi)   # no decimal places
[1] "3"
> sprintf("%5.1f", pi)  # 5 characters with whitespace
[1] "  3.1"
> sprintf("%05.1f", pi)  #5 characters no whitespace
[1] "003.1"
> sprintf("%+f", pi)
[1] "+3.141593"
```

To express asingle or double digit character integer as three character number

```
> x = 4
> sprintf("%03d", x)
[1] "004"
>
> x=40
> sprintf("%03d", x)
[1] "040"
```

For character data (i.e. strings)

```
> sprintf("%s %d", "test", 1:3)
[1] "test 1" "test 2" "test 3"
```

N.B *s* for string and *d* for integers.

1.4 Programming Assignment Part 1 - `pollutantmean.r`

- Write a function named '`pollutantmean`' that calculates the mean of a pollutant (sulfate or nitrate) across a specified list of monitors.
- The function '`pollutantmean`' takes three arguments: '`directory`', '`pollutant`', and '`id`'.
- Given a vector monitor ID numbers, '`pollutantmean`' reads that monitors' particulate matter data from the directory specified in the 'directory' argument and returns the mean of the pollutant across all of the monitors, ignoring any missing values coded as NA.
- A prototype of the function is as follows

```
pollutantmean <- function(directory, pollutant, id = 1:332) {
## 'directory' is a character vector of length 1 indicating
## the location of the CSV files

## 'pollutant' is a character vector of length 1 indicating
## the name of the pollutant for which we will calculate the
## mean; either "sulfate" or "nitrate".

## 'id' is an integer vector indicating the monitor ID numbers
## to be used

## Return the mean of the pollutant across all monitors list
## in the 'id' vector (ignoring NA values)
}
```

You can see some example output from this function. The function that you write should be able to match this output. Please save your code to a file named `pollutantmean.R`.

1.5 Set Theory Commands

Set theory commands are very useful in simplifying code. The `is.element()` command asks if a specified value is an element in a specified set. If so, the function returns "TRUE", otherwise "FALSE".

```
x <- c(2,4,8)
y <- 1:6
is.element(x,y)
!is.element(x,y)
x %in % y
```

```
> x <- c(2,4,8)
> y <- 1:6
> is.element(x,y)
[1]  TRUE  TRUE FALSE
> !is.element(x,y)
[1] FALSE FALSE  TRUE
>
> x %in% y
[1]  TRUE  TRUE FALSE
```

Example: List of States Abbreviations

(Extract relating to an exercise on a similar R Course)

Let us look at the states (includes some US territories such as Guam, Puerto Rico and the US Virgin Islands). Using the `str()` command, we can see this variable is structured as a **factor**.

```
> summary(Hosp[,7])
AK  AL  AR  AZ  CA  CO  CT  DC  DE  FL  GA  GU  HI  IA  ID  IL  IN  KS
17  98  77  77 341  72  32   8   6 180 132   1  19 109  30 179 124 118
KY  LA  MA  MD  ME  MI  MN  MO  MS  MT  NC  ND  NE  NH  NJ  NM  NV  NY
96 114  68  45  37 134 133 108  83  54 112  36  90  26  65  40  28 185
OH  OK  OR  PA  PR  RI  SC  SD  TN  TX  UT  VA  VI  VT  WA  WI  WV  WY
170 126  59 175  51  12  63  48 116 370  42  87   2  15  88 125  54  29
>
> str(Hosp[,7])
Factor w/ 54 levels "AK","AL","AR",..: 2 2 2 2 2 2 2 2 2 2 ..
```

To generate a list of state names (as in abbreviations) can be generated by using the `unique()` and `as.character()` functions.

```
> as.character(unique(Hosp[,2]))
[1] "AL" "AK" "AZ" "AR" "CA" "CO" "CT" "DE" "DC" "FL" "GA"
[12] "HI" "ID" "IL" "IN" "IA" "KS" "KY" "LA" "ME" "MD" "MA"
```

```
[23] "MI" "MN" "MS" "MO" "MT" "NE" "NV" "NH" "NJ" "NM" "NY"
[34] "NC" "ND" "OH" "OK" "OR" "PA" "PR" "RI" "SC" "SD" "TN"
[45] "TX" "UT" "VT" "VI" "VA" "WA" "WV" "WI" "WY" "GU"
```

We can then sort them using the `sort()` command.

```
StateList <- sort(as.character(unique(Hosp[,7])))
```

```
> StateList <- sort(as.character(unique(Hosp[,7])))
> is.element("XX",StateList)
[1] FALSE
> !is.element("XX",StateList)
[1] TRUE
>
```

1.6 R code to check that inputs are valid

```
# Generate a list of valid state abbreviations from data.

ValidStates <- as.character(unique(Hosp[,2]))

# construct a set of valid outcome names

ValidOutcomes <- c("heart attack","heart failure","pneumonia")


## Check that state is valid
if(!is.element(state,ValidStates)){
stop("invalid state")
}
## Check that outcome is valid
if(!is.element(outcome,ValidOutcomes)){
stop("invalid outcome")
}
```

1.7 Old Part 1 - getmonitor.r

- Write a function named 'getmonitor' that takes three arguments: 'id', 'directory', and 'summarize'.
- Given a monitor ID number, 'getmonitor' reads that monitor's particulate matter data from the directory specified in the 'directory' argument and returns a data frame containing that monitor's data.
- If 'summarize = TRUE', then 'getmonitor' produces a summary of the data frame with the 'summary' function and prints it to the console.
- A prototype of the function is as follows

```
getmonitor <- function(id, directory, summarize = FALSE) {

## 'id' is a vector of length 1 indicating the monitor ID
## number. The user can specify 'id' as either an integer, a
## character, or a numeric.

## 'directory' is a character vector of length 1 indicating
## the location of the CSV files
## 'summarize' is a logical indicating whether a summary of
## the data should be printed to the console; the default is
## FALSE

## Your code here
}
```

1.8 Creating the file name

- `directory` is going to get passed as a string. The correct program will obviously use ”specdata”.
- Using the `paste()` command, we can come up with a filename based on the directory and id.

```
> directory="specdata"
> id = 3
>
> paste(directory, "/", id, ".csv")
[1] "specdata / 3 .csv"
>
```

- We must rectify the problem of the id number. As a string it is written with three characters. The first two characters would be zeros for the first 99 cases. We should have ”`specdata / 003 .csv`”.
- **using** `nchar()` : Before we implement the previous code, we need to replace all values with equivalent three character strings.

```
if (nchar(id) == 1) {
    id <- paste("00", id)
   } else if (nchar(id) == 2) {
    id <- paste("0", id)
   }
```

- Clearly there is whitespace where there shouldn’t be. We will use the `gsub()` command to remove it.

```
> dir <- gsub(" ", "", paste(directory, "/", id, ".csv"))
> dir
[1] "specdata/003.csv"
```

- An alternative approach is to use the `sprintf()` command. (The command `as.numeric()` is to account for irregular inputs)

```
> id=2
> sprintf("%03d", as.numeric(id))
[1] "002"
```

-
  ```
  paste(directory, "/", sprintf("%03d", as.numeric(id)),
    ".csv",sep = "")
  ```

1.9 Workable Programs

```
getmonitor <- function(id, directory, summarize = FALSE) {

  if (nchar(id) == 1) {
   id <- paste("00", id)
  } else if (nchar(id) == 2) {
   id <- paste("0", id)
  }

  filename <- gsub(" ", "",
    paste(directory, "/", id, ".csv"))

  Data <- read.csv(filename, header=TRUE, sep="," )

  if (summarize == TRUE) {
   print(summary(Data))
  }

  return(Data)
}
```

1.10 Another workable program

```
getmonitor <- function(id, directory, summarize = FALSE) {

    filename <- paste(directory, "/", sprintf("%03d",
       as.numeric(id)), ".csv", sep = "")

    Data <- read.csv(filename, header=TRUE, sep="," )

    if (summarize==TRUE) {
        print(summary(Data))
    }

    return(Data)
}
```

1.11 Programming Assignment Part 2 - `complete.r`

- Write a function that reads a directory full of files and reports the number of completely observed cases in each data file.
- The function should return a data frame where the first column is the name of the file and the second column is the number of complete cases.

Prototype:

```
complete <- function(directory, id = 1:332) {
        ## 'directory' is a character vector of length 1
        ## indicating the location of the CSV files

        ## 'id' is an integer vector indicating the monitor
        ## ID numbers to be used

        ## Return a data frame of the form:
        ## id nobs
        ## 1  117
        ## 2  1041
        ## ...
        ## where 'id' is the monitor ID number and
        ## 'nobs' is the number of complete cases
}
```

- input : directory

1.12 The `complete.cases()` command

Consider the ***airquality*** data set (first seven rows for sake of brevity).

```
> airquality[1:7,]
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
7    23     299  8.6   65     5   7
>
> complete.cases(airquality[1:7,])
[1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE
```

How many complete cases are there in the ***airquality*** data set? LEt set up a subset of this data set that comprises only complete cases (lets call it `aq.comp`), and then find the dimensions.

```
> dim(airquality)
[1] 153   6
>
> aq.comp <- airquality[complete.cases(airquality),]
>
> dim(aq.comp)
[1] 111   6
```

1.13 Number of logical-compliant elements

How many elements in a vector fulfil a logical condition? Consider the output of the `complete.cases()` command in the last section. The output tells us which rows are complete cases. Suppose we simply want to find out how many complete cases there are, without resorting to creating new dataframes.

```
> complete.cases(airquality)
  [1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE
 [11] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
............
```

We can use the `as.numeric()` function to turn the logical states into numbers (i.e. 0 and 1). We can then simply sum up the values.

```
as.numeric(complete.cases(airquality))
sum(as.numeric(complete.cases(airquality)))
```

```
> as.numeric(complete.cases(airquality))
  [1] 1 1 1 1 0 0 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1
 [32] 0 0 0 0 0 0 1 0 1 1 0 0 1 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1
.............

> sum(as.numeric(complete.cases(airquality)))
[1] 111
```

This accords with our conclusion from the previous section, that there is 111 complete cases in the ***airquality*** data set.

1.14 Empty Data Frames

We can create an “empty” data frame that can be dynamically populated. Simply specify the name of the data frame, using the `data.frame()` command with no supplied arguments.

```
> mydata = data.frame()
> mydata
data frame with 0 columns and 0 rows
>
> mydata=rbind(mydata,1:6)
> mydata
  X1L X2L X3L X4L X5L X6L
1   1   2   3   4   5   6
>
> mydata=rbind(mydata,7:12)
> mydata
  X1L X2L X3L X4L X5L X6L
1   1   2   3   4   5   6
2   7   8   9  10  11  12
```

We can rename the column names of the dataframe using the `names()` function. The folllowing piece of code demonstrates the two functionalities of this command.

```
> names(mydata)
[1] "X1L" "X2L" "X3L" "X4L" "X5L" "X6L"
> names(mydata) <- c("A","B","C","D","E","F")
> names(mydata)
[1] "A" "B" "C" "D" "E" "F"
```

1.15 Workable Solution

```
complete <- function(directory, id = 1:332) {
    mydata = data.frame()
    for(item in id) {
        file <- sprintf("%s/%03d.csv", directory, item)
        data <- read.csv(file)

        rows <- sum(as.numeric(complete.cases(data)))
        mydata <- rbind(mydata, c(item, rows))
    }
    names(mydata) <- c("id", "nobs")
    return(mydata)
}
```

1.16 Programming Assignment Part 2 : `corr.r`

- Write a function that takes a directory of data files and a threshold for complete cases and calculates the correlation between sulfate and nitrate for monitor locations where the number of completely observed cases (on all variables) is greater than the threshold.
- The reference to completed cases implies that `complete.R` should be called by the function we are going to write.
- The function should return a vector of correlations for the monitors that meet the threshold requirement.
- If no monitors meet the threshold requirement, then the function should return a numeric vector of length 0.

Prototype:

```
corr <- function(directory, threshold = 0) {
    ## 'directory' is a character vector of length 1
    ## indicating the location of the CSV files

    ## 'threshold' is a numeric vector of length 1 indicating
    ## the number of completely observed observations (on
    ## all variables) required to compute the correlation
    ## between nitrate and sulfate; the default is 0

    ## Return a numeric vector of correlations
}
```

1.17 Part 3 Exercise

Input ***directory*** is a character vector of length 1 indicating the location of the CSV files.

Input ***threshold*** is a numeric vector of length 1 indicating the number of completely observed observations (on all variables) required to compute the correlation between nitrate and sulfate;
The default for ***threshold*** is 0.

Output The function return a numeric vector of correlations.

```
corr <- function(directory, threshold = 0) {

    corrsNum <- numeric(0)

    # Get a data frame as ID = 1:332
    CompObs <- complete("specdata")

    # Apply the threshold condition
    # Reduce CompObs to those with enough

    CompObs <- CompObs[CompObs$nobs > threshold, ]

    for (cid in CompObs$id) {
        # Get a data frame as ID in $id
        monDfr <- getmonitor(cid, directory)

        # Calculate correlation between variables
        corrsNum <- c(corrsNum,
          cor(monDfr$sulfate,monDfr$nitrate,
          use = "pairwise.complete.obs"))
        }

    return(corrsNum)
    }
```

Another workable solution

```
corr <- function(directory, threshold = 0) {

    cordata <- c()

    files <- list.files(directory)
    for(file in files){
        # get the file and its data
        file <- sprintf("%s/%s", directory, file)
        data <- read.csv(file)

        # get the complete data (and count) for that file
        good <- data[complete.cases(data),]
        rows <- nrow(good)

        # only calc correlation on files with more than
        # 'threshold' complete rows
        if(rows > threshold){
            correlation <- cor(good$sulfate, good$nitrate)
            cordata <- append(cordata, correlation)
        }
    }

    # if no data (no files with number of rows > threshold

    if(length(cordata) < 1){
        cordata <- vector(mode="numeric", length=0)
    }


    return(cordata)
}
```