Contents

1 Reading data from files **2**
1.1 The working directory (`getwd()` and `setwd()`) 2

2 Exercise 2.1 **2**
2.1 Importing data 2
2.2 Sorting a data frame (`order()`) 4

3 Filtering entries **5**

4 Exercise 2.2 **6**
4.1 Importing fixed width format files `read.fwf()` 6

5 Exercise 2.3 **8**

6 Exercise 2.4 **9**

7 Exercise 2.5 **9**

8 Exercise 2.6 **11**

9 Exercise 2.7 **11**

1 Reading data from files

For the most part, in this course, we will either get data from other packages in R, or read data from files. It is important to be familiar with how to import data, and generally there are only two (three) functions to be aware of, `read.fwf()` and `read.table()` (and `read.csv()`, which is really just `read.table()` with some preset settings).

In general, the first step is to be aware what is the structure of the data. Open the data file you want to import and study its structure. Does it have a header? Are the individual data points in an observation separated by anything? And in general, most files (almost all) should have one observation in one row.

1.1 The working directory (`getwd()` and `setwd()`)

If you encounter any file-related errors (e.g. file not found) when trying to read a file, the first thing you should check is your working directory. Most problems will likely be solved if you check your working directory, because **any functions that require a file (filepath) will try to find the file (filepath) relative to your current working directory.**

To see your current working directory, use `getwd()`. You may, for example, get the following response: `C:/Users/caleb/Documents`. Thus, if I run `read.table("lab1fixed.txt")`, it will look for the file at `C:/Users/caleb/Documents/lab1fixed.txt`. So, check if the file is where you intend it to be, or check that your working directory is where you intend it to be.

To change your working directory, you can use `setwd()` and pass it a path.

Alternatively, in RStudio, use the Files tab, click the 3 dots at the right of the panel and navigate to the folder with your files. Click the cog icon, and find the 'Set As Working Directory' option.

Both methods are equivalent, as they ultimately use the `setwd()` function - it will appear in the R console.

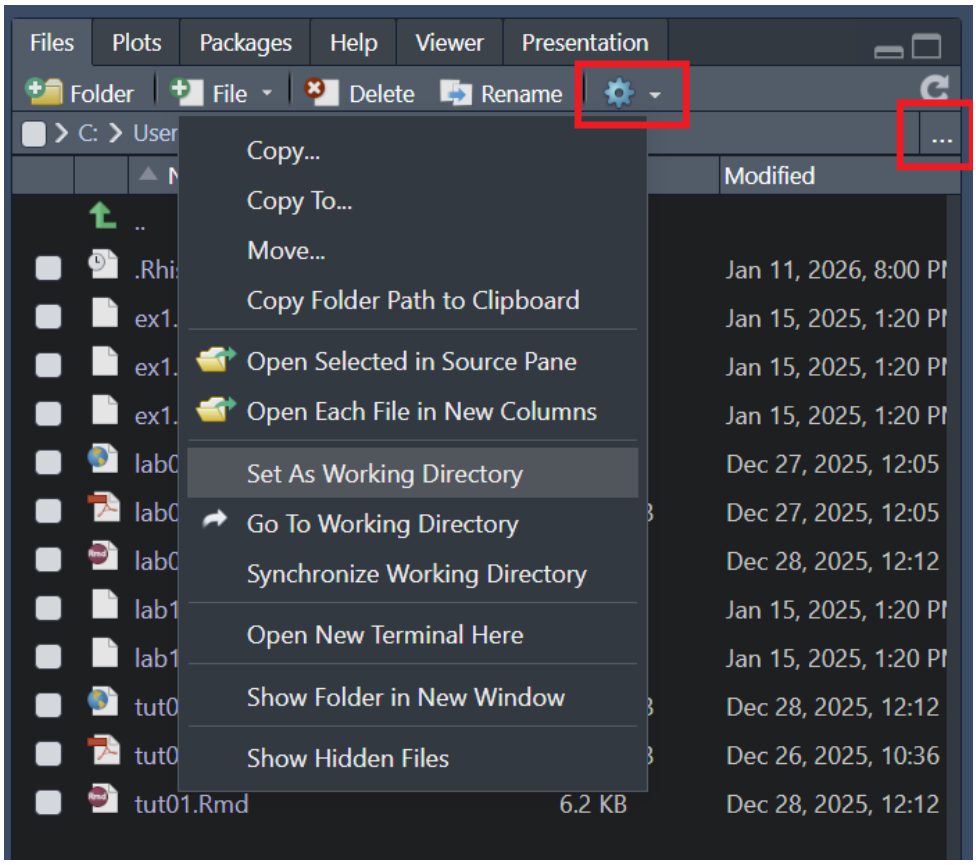
A quick mention for Rmarkdown (`.Rmd`) files: I find that as long as the files are in the same folder (e.g. `Documents/script.Rmd` and `Documents/data.csv`) the working directory is irrelevant.

2 Exercise 2.1

2.1 Importing data

The lab sheet suggests using `read.table()`. To understand how to use it, look at the documentation with `?read.table`. The usage indicates there are many parameters the function can accept, but the first few are the most important: `file`, `header`, and `sep`.

`file` is simply the filename of the data we want to import.



`header` indicates whether the first line of the file contains information on the column headers. In `read.table()`, the default value is `FALSE`. Looking at the data file, we can see that the first line indeed are labels for each column, so we must change `header` to `TRUE`.

`sep` defines the character that separates each column. The default value is an empty string, which will separate by white space, i.e. any amount of space characters, tab characters, line breaks. In this file, it is a comma (`,`), so we change `sep` to `','`.

```
car1 <- read.table("Lab02_Ex1.csv", header=T, sep=",")
car1
```

```
## # A tibble: 12 x 7
##    Car                 MPG Displacement Horsepower Weight Model Origin
##    <chr>             <dbl>        <int>      <int>  <int> <int> <chr>
##  1 Peugeot 304          30           79         70   2074    71 Europe
##  2 Audi 100LS           20          114         91   2582    73 Europe
##  3 Volkswagen Rabbit    29           90         70   1937    75 Europe
##  4 Volvo 245            20          130        102   3150    76 Europe
##  5 Toyota Corolla       25          113         95   2228    71 Japan
##  6 Subaru FT            26          108         93   2391    74 Japan
##  7 Honda Civic CVCC   36.1           91         60   1800    78 Japan
##  8 Datsun 810           22          146         97   2815    77 Japan
##  9 Ford F250            10          360        215   4615    70 US
```

```
## 10 Chevy C20            10          307        200   4376    70 US
## 11 Pontiac Catalina     16          400        170   4668    75 US
## 12 Chevrolet Chevette   30           98         68   2155    78 US
```

Alternatively, one could have used the `read.csv()` function directly, since it has different default values which you can see with `?read.csv`.

```
car1 <- read.csv("Lab02_Ex1.csv")
car1
```

```
## # A tibble: 12 x 7
##    Car                  MPG Displacement Horsepower Weight Model Origin
##    <chr>              <dbl>        <int>      <int>  <int> <int> <chr>
##  1 Peugeot 304         30            79         70   2074    71 Europe
##  2 Audi 100LS          20           114         91   2582    73 Europe
##  3 Volkswagen Rabbit   29            90         70   1937    75 Europe
##  4 Volvo 245           20           130        102   3150    76 Europe
##  5 Toyota Corolla      25           113         95   2228    71 Japan
##  6 Subaru FT           26           108         93   2391    74 Japan
##  7 Honda Civic CVCC    36.1          91         60   1800    78 Japan
##  8 Datsun 810          22           146         97   2815    77 Japan
##  9 Ford F250           10           360        215   4615    70 US
## 10 Chevy C20           10           307        200   4376    70 US
## 11 Pontiac Catalina    16           400        170   4668    75 US
## 12 Chevrolet Chevette  30            98         68   2155    78 US
```

2.2 Sorting a data frame (`order()`)

To sort a data frame, we can use the `order()` function, which just returns a numerical ranking to the data. For example:

```
x <- c(100, 200, 150, 175, 125)
x[c(3,3,2,4,4)] # get the 3rd, 3rd, 2nd, 4th, 4th entries.
```

```
## [1] 150 150 200 175 175
```

```
order(x) # 1, 5, 3, 4, 2
```

```
## [1] 1 5 3 4 2
```

```
x <- x[order(x)]
x
```

```
## [1] 100 125 150 175 200
```

In earlier exercises, we used a boolean condition (true or false, based on some criteria) to determine whether to keep or discard a row (observation). We can also use a numerical index to determine what entries to take, as in line 2 in the above code block.

You can see that `order()` gives the ordering of the numbers from smallest (1) to largest (5). So, when we order x by this sequence of indexes, we will get the elements in x ordered from smallest to largest.

Since we want to order the entries by weight, we use the `order()` function on the weight vector `car1$Weight`. By default, `order()` will return indexes in an increasing order. This behaviour can be controlled by changing the optional parameter `decreasing`, whose default value is `FALSE`.

```
ordering <- order(car1$Weight, decreasing=T)
car1 <- car1[ordering,]
car1
```

```
## # A tibble: 12 x 7
##    Car                  MPG Displacement Horsepower Weight Model Origin
##    <chr>              <dbl>        <int>      <int>  <int> <int> <chr>
##  1 Pontiac Catalina    16            400        170   4668    75 US
##  2 Ford F250           10            360        215   4615    70 US
##  3 Chevy C20           10            307        200   4376    70 US
##  4 Volvo 245           20            130        102   3150    76 Europe
##  5 Datsun 810          22            146         97   2815    77 Japan
##  6 Audi 100LS          20            114         91   2582    73 Europe
##  7 Subaru FT           26            108         93   2391    74 Japan
##  8 Toyota Corolla      25            113         95   2228    71 Japan
##  9 Chevrolet Chevette  30             98         68   2155    78 US
## 10 Peugeot 304         30             79         70   2074    71 Europe
## 11 Volkswagen Rabbit   29             90         70   1937    75 Europe
## 12 Honda Civic CVCC    36.1           91         60   1800    78 Japan
```

3 Filtering entries

We have seen this in action last week. In part (d), we have a compound condition to filter for. To do so, we simply get the two boolean vectors representing the conditions, and do a logical AND operation on them using &.

Below, it is interpreted as “take the rows where weight is greater than 3000 and horsepower is greater than 200, and all the columns.”

```
car1[car1$Weight > 3000 & car1$Horsepower > 200,]
```

```
## # A tibble: 1 x 7
##   Car          MPG Displacement Horsepower Weight Model Origin
##   <chr>      <dbl>        <int>      <int>  <int> <int> <chr>
## 1 Ford F250    10          360        215   4615    70 US
```

The `subset()` function is another way to filter entries, with similar syntax. The columns here can be referred to as variables directly, without specifying the data frame, i.e. `car1$` is not necessary. An optional parameter `select` can be specified to determine what columns to keep.

```
subset(car1, Weight > 3000 & Horsepower > 200)
```

```
## # A tibble: 1 x 7
##   Car          MPG Displacement Horsepower Weight Model Origin
##   <chr>      <dbl>        <int>      <int>  <int> <int> <chr>
## 1 Ford F250    10          360        215   4615    70 US
```

```
subset(car1, Weight > 3000 & Horsepower > 200, select=Car:Weight)
```

```
## # A tibble: 1 x 5
##   Car          MPG Displacement Horsepower Weight
##   <chr>      <dbl>        <int>      <int>  <int>
## 1 Ford F250    10          360        215   4615
```

4 Exercise 2.2

4.1 Importing fixed width format files `read.fwf()`

Open and observe the format of `Lab02fixed.txt`. As the name suggests, it is a **fixed-width format** file, with no separators between each data point in a single observation. Instead, some number of columns corresponds to some information about a single observation, with the details for this file provided in the lab sheet. These columns are passed in under the `widths` parameter.

```
lab2 <- read.fwf("Lab02fixed.txt", widths = c(3,1,3,2,1),
                 col.names = c("id", "gender", "height", "weight",
                 ↪  "siblings"))
str(lab2)
```

```
## 'data.frame':    120 obs. of  5 variables:
##  $ id      : int  201 202 203 204 205 206 207 208 209 210 ...
##  $ gender  : chr  "M" "F" "F" "F" ...
##  $ height  : int  173 158 158 162 169 171 167 157 163 168 ...
##  $ weight  : int  65 47 43 45 58 63 47 44 48 88 ...
##  $ siblings: int  1 1 2 2 1 2 2 2 2 1 ...
```

```
head(lab2, 12)
```

```
## # A tibble: 12 x 5
##       id gender height weight siblings
##    <int> <chr>   <int>  <int>    <int>
##  1   201 M         173     65        1
##  2   202 F         158     47        1
##  3   203 F         158     43        2
##  4   204 F         162     45        2
##  5   205 M         169     58        1
##  6   206 M         171     63        2
##  7   207 F         167     47        2
##  8   208 F         157     44        2
##  9   209 F         163     48        2
## 10   210 M         168     88        1
## 11   211 M         171     99        3
## 12   212 F         163     59        4
```

```
lab2test <- read.table("Lab02test.txt", header=T)
lab2merged <- merge(lab2, lab2test, by.x="id", by.y="id")
head(lab2merged, 12)
```

```
## # A tibble: 12 x 6
##       id gender height weight siblings  test
##    <int> <chr>   <int>  <int>    <int> <int>
##  1   201 M         173     65        1    67
##  2   202 F         158     47        1    64
##  3   203 F         158     43        2    67
##  4   204 F         162     45        2    81
##  5   205 M         169     58        1    86
##  6   206 M         171     63        2    69
##  7   207 F         167     47        2    81
##  8   208 F         157     44        2    82
##  9   209 F         163     48        2    67
## 10   210 M         168     88        1    81
```

```
## 11   211 M          171      99         3     64
## 12   212 F          163      59         4     70
```

```
lab2merged[lab2merged$id == 211, "weight"] <- 77
lab2merged[lab2merged$height > 182, "test"]
```

```
## [1] 55 76 54
```

```
lab2f <- lab2merged[lab2merged$gender == "F",]
ordering <- order(lab2f$height, decreasing=T)
lab2f.ordered <- lab2f[ordering,]
lab2f.ordered[1, c('height', 'weight', 'test')]
```

```
## # A tibble: 1 x 3
##   height weight  test
##    <int>  <dbl> <int>
## 1    178     55    47
```

5 Exercise 2.3

```
number <- 8
if (number < 100)
  { if (number < 10) { result <- "single digit" }
    else {result <- "double digits"}
  } else { result <- "more than 2 digits"}

print(paste(number, "is", result))
```

```
## [1] "8 is single digit"
```

The original code is intensely messy and (to me) difficult to read. With some proper alignment and line breaks, and the modifications asked for in the question:

```
number <- 8
if (number < 10) {
  result <- "single digit"
} else if (number < 100) {
  result <- "double digits"
} else if (number < 1000) {
  result <- "three digits"
} else {
  result <- "more than 3 digits"
}
```

6 Exercise 2.4

```
speed <- 90

while (speed > 60) {
  print(paste("Your speed is", speed, "km/h! Time to slow down!"))
  speed <- speed - 7
}
```

```
## [1] "Your speed is 90 km/h! Time to slow down!"
## [1] "Your speed is 83 km/h! Time to slow down!"
## [1] "Your speed is 76 km/h! Time to slow down!"
## [1] "Your speed is 69 km/h! Time to slow down!"
## [1] "Your speed is 62 km/h! Time to slow down!"
```

```
speed <- 150
while (speed > 60) {
  if (speed > 120) {
    print(paste("Your speed is", "speed", "km/h! Slow down, BIG time!!"))
    speed <- speed - 12
  } else {
    print(paste("Your speed is", speed, "km/h! Time to slow down!"))
    speed <- speed - 7
  }
}
```

```
## [1] "Your speed is speed km/h! Slow down, BIG time!!"
## [1] "Your speed is speed km/h! Slow down, BIG time!!"
## [1] "Your speed is speed km/h! Slow down, BIG time!!"
## [1] "Your speed is 114 km/h! Time to slow down!"
## [1] "Your speed is 107 km/h! Time to slow down!"
## [1] "Your speed is 100 km/h! Time to slow down!"
## [1] "Your speed is 93 km/h! Time to slow down!"
## [1] "Your speed is 86 km/h! Time to slow down!"
## [1] "Your speed is 79 km/h! Time to slow down!"
## [1] "Your speed is 72 km/h! Time to slow down!"
## [1] "Your speed is 65 km/h! Time to slow down!"
```

7 Exercise 2.5

Here, we have a new data type: a `list`. A `list` in R is, as best as I can tell, similar to a dictionary/hash table in other languages.

```
statquote <- "There are three types of lies -- lies, damn lies, and
↪  statistics."
chars <- strsplit(statquote, split="")
class(chars)
```

```
## [1] "list"
```

```
str(chars)
```

```
## List of 1
##  $ : chr [1:65] "T" "h" "e" "r" ...
```

```
chars <- chars[[1]]
```

The second example is kind of gross. It counts the number of 'e's in the string, but is unnecessarily long.

```
ectr <- 0

for (char in chars) {if (char == "e") {ectr <- ectr + 1}}
print(ectr)
```

```
## [1] 9
```

Instead, we can sum over a boolean vector, which basically counts the number of occurrences of TRUE, since `T = 1` and `F = 0`.

```
sum(chars == "e")
```

```
## [1] 9
```

```
sctr <- 0

for (char in chars) {
  if (char == "s") sctr <- sctr + 1
  if (char == "m") break
}
print(sctr)
```

```
## [1] 3
```

8 Exercise 2.6

```
store1 <- c(15, 9, 13, 5, 2, 17, 14)
store2 <- c(16, 7, 5, 16, 8, 13, 14)

store3 <- store1 + store2
store3
```

```
## [1] 31 16 18 21 10 30 28
```

```
mean(store3)
```

```
## [1] 22
```

```
store3 <- store3[order(store3)]
store3
```

```
## [1] 10 16 18 21 28 30 31
```

```
mean(store3, trim=0.2)
```

```
## [1] 22.6
```

The optional `trim` parameter of `mean()` takes out a percentage of the highest and lowest values before calculating the mean. For example, if `trim=0.1`, then the highest 10% and lowest 10% of values are removed.

The number of values trimmed is rounded down. For example, taking the mean of the above vector of 7 items:

- if `trim = 0.1`, trim 10% from each side, or 0.7 values, rounded down to 0 values. No trimming occurs.
- if `trim = 0.2`, trim 20% from each side, or 1.4 values, rounded down to 1 value.
- if `trim = 0.4`, trim 40% from each side, or 2.8 values, rounded down to 2 values.

9 Exercise 2.7

```
m2 <- function(x) {
  y = (x - mean(x))^2
  return (mean(y))
}
```

```r
m4 <- function(x) {
  mean((x - mean(x))^4)
}
```