CHAPTER 8

Connecting R to Excel with the xlsx Package

OBJECTIVE

This chapter shows how you can use the **xlsx** R package for creating, writing, styling and editing Excel worksheets within RStudio. This package is an alternative to the `openxlsx` package of chapter #7, and provides another high-level interface for automating various Excel tasks with R. While the xlsx package offers some important features not found in `openxlsx`, it also comes with its own limitations. This chapter provides a detailed account of what you can do and cannot do with the xlsx package.

Contents

8.1 Introduction

Before you can use the `xlsx` package for the first time, you need to install and load it to your R environment. These 2 simple tasks are accomplished as follows:

```
> install.packages("xlsx")
> library(xlsx)
```

If this installation fails, then you may need to install Java[1] on your system, and also make it work with RStudio. Depending on your system, you may have to do some "googling" to find the tip that can help you resolve your problem. The good news is that Java is a very popular programming language. Therefore, a Java-related problem that you encounter has most likely been previously resolved by others. But some Google search will generally be necessary to find that solution. You do not need to know Java at all. But the `xlsx` package needs to use it in the background.

For my Windows 11 system, I found useful instructions for installing Java by following the link https://www.windows11.dev/ce7in/java-55a9. However, after installing Java, I still could not make the R command `library(xlsx)` work. It turned out that a patch needed to resolve this problem was made available and could be downloaded from the link https://cran.r-project.org/bin/windows/base/rpatched.html. Again doing some Google search was essential for finding the solution.

To illustrate how the `xlsx` package connects R to Excel, I will use data included in the `chap8data.xlsx` Excel workbook, which you can download with the link https://bit.ly/3BcTgri. Note that this workbook is password-protected, and can only be opened with the pasword `"Mircrosoft365"`. You will see in the next few paragraphs, how its content can be explored using some powerful functions from the `xlsx` package. You will also see how easy it is to

[1] As previously mentioned, the `xlsx` package is dependent upon the Java programming language.

create new workbooks and populate them with data.

As previously indicated, all scripts are assumed to be saved in a project directory. If needed, you may review section 3.3 of chapter 3, to learn how project directories work in R.

8.2 Analysis of Excel Data with R

In this section, you will learn how to read Excel data into R, analyze it within R, and write the analysis results back to Excel. All examples in this section are assumed to use datasets and script files that are organized in a project directory named `xlsx` and structured as in Figure 8.1. If you are going to experiment with these examples, I advise that you first create such a project directory. For more information regarding project directories in R, please refer to section 3.3.2 or chapter 3.

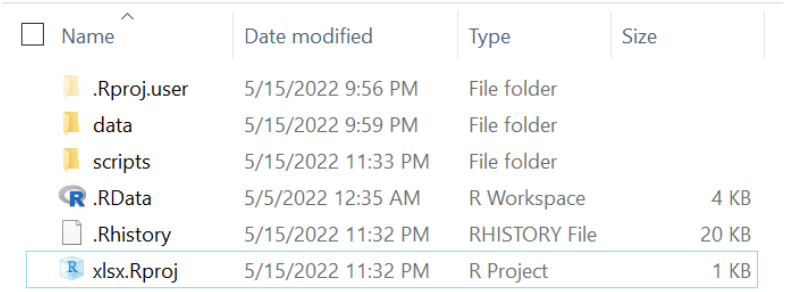

Figure 8.1: Structure of the `xlsx` directory

The `data` subdirectory of project directory `xlsx` should contain an Excel workbook `chap6datasets.xlsx`, which can be downloaded with link `https://bit.ly/3giWj9m`. This workbook is password protected. The password `Microsoft365` will be needed to unprotect it.

Script 8.1 starts by reading the "Quantitative Ratings" data table from the `IrrData` worksheet of the `chap6datasets.xlsx` workbook. Figure 7.3 of chapter 7 shows you what that dataset looks like. It is used afterwards in

the script to compute the summary table `summary.df` that shows the average rating by `Group` and `Target` for each of the 4 variables, as well as the standard deviation of these averages by `Group` only. This summary table (see the right hand side of Figure 7.4 of chapter 7) is written to worksheet `summaryStats1` of a new Excel workbook `chap8data1.xlsx`. The default worksheet location to start writing this summary data is the top-left cell `A1`. The same summary table is written again to another worksheet named `summaryStats2`, starting this time from cell `C5` in order to illustrate how to change the positioning of your table. You can download the Excel workbook `chap8data1.xlsx`[2] to see the final dataset that Script 8.1 has produced.

I am now going to review the different segments of this script file, will introduce the different functions it uses, and explain how they accomplish various tasks.

Script 8.1. R Script for reading the `chap6datasets.xlsx` Excel workbook, analyzing its data, and writing summary statistics to specific worksheets. (*You may download this script with the link:* `https://bit.ly/3g6unFE`)

```
01 library(tidyverse)
02 library(xlsx)
03 passWD <- "Microsoft365"
04 wb.df1 <- loadWorkbook(file="./data/chap6datasets.xlsx",
05                         password = passWD)
06 sht.names <- names(getSheets(wb.df1))
07 print(sht.names)
08
09 #-- Reading the password-protected Excel workbook -
10 irrData.df <- read.xlsx(file="./data/chap6datasets.xlsx",
11                          sheetName = "IrrData",
12                          rowIndex = c(2:17),colIndex = 19:24,
13                          password = passWD)
14
15 #-- Compute the summary table
16 summary.df <- as_tibble(irrData.df) %>%
```

[2]Here is the download link: `https://bit.ly/3TcIuIk`

```
17   group_by(Group,Target) %>%
18   summarise(across(c(J1,J2,J3,J4),mean)) %>%
19   mutate(across(c(J1,J2,J3,J4),sd,.names="gsd.{.col}")) %>%
20   ungroup() %>%
21   mutate(across(!c(Group,Target),round,3))
22 print(summary.df)
23
24 #-- Write the summary table to the input workbook
25 write.xlsx(x = as.data.frame(summary.df),
26              file="./data/chap8data1.xlsx",
27              sheetName = "summaryStats1",append = FALSE,
28              row.names = FALSE, password = passWD)
29
30 #- Write the summary.df at a specific location of the worksheet
31 wb.df1 <- loadWorkbook(file="./data/chap6datasets.xlsx",
32                         password = "Microsoft365")
33 my.sheet <- createSheet(wb=wb.df1, sheetName="summaryStats2")
34 addDataFrame(x = as.data.frame(summary.df), sheet = my.sheet,
35               col.names = TRUE,row.names = FALSE,
36               startRow=3, startColumn=6)
37 saveWorkbook(wb.df1, file="./data/chap8data1.xlsx",
38               password = passWD)
39
40 #-- Delete the summaryStats2 worksheet if necessary
41 removeSheet(wb=wb.df1, sheetName = "summaryStats2")
42 saveWorkbook(wb.df1, file="./data/chap8delete.xlsx",
43               password = passWD)
```

End of Script

I am now going to review Script 8.1 by splitting it into chunks of code and discussing how the analysis is done within each chunk. Script 8.2 contains the first 6 lines of code from Script 8.1. Lines `#01` and `#02` load the 2 packages I will need. The `tidyverse` package is needed to create the summary table, whereas `xlsx` is used to manipulate Excel files.

Line `#03` assigns the workbook password to variable `passWD` for later use. Lines `#04` and `#05` create a workbook object (i.e. java object) named `wb.df1` and which points to the Excel workbook I want to analyze. Note that the

`loadWorkbook()` function needs the password argument, since the workbook is password-protected and the `"file="` argument provides the path to the Excel workbook.

Script 8.2. Create a workbook object that points to `chap8data.xlsx`

```
01  library(tidyverse)
02  library(xlsx)
03  passWD <- "Microsoft365"
04  wb.df1 <- loadWorkbook(file="./data/chap6datasets.xlsx",
05                         password = passWD)
06  sht.names <- names(getSheets(wb.df1))
07  print(sht.names)
```

End of Script

> Note that the `openxlsx` package that is covered in chapter 7 also offers a version of the `loadWorkbook()` function. Although the function name is identical, both versions of this function have different functionalities. In case you decide to load both packages (`openxlsx` and `xlsx`) in the same script, always refer to the `xlsx` version of the function as `xlsx::loadWorkbook()` and to the `openxlsx` version as `openxlsx::loadWorkbook()`.

In line `#06`, the `getSheets()` function is used to obtain a named list of java objects that point to the worksheets contained in the workbook. The `names()` function returns a character vector containing all worksheet names. Therefore, executing line `#07` will yield the following outcome:

```
> print(sht.names)
[1] "mtcars"  "iris"    "IrrData" "IccData" "CacData"
>
```

The second segment of Script 8.1 going from line `#09` through `#22` aims at reading the `Quantitative Ratings` data table from the `IrrData` worksheet and using it to create the summary table named `summary.df`.

In lines #10 through #13, I use the `read.xlsx()` function to read the `"Quantitative Ratings"` data into an R data frame called `irrData.df`. The `"file="` argument specifies the full Excel file pathname. The `"sheetName="` argument provides the name of the worksheet that contains the input data. `"rowIndex="` and `"colIndex="` specify respectively, the rows and columns defining the data range of interest. If you look at the `IrrData` worksheet, you will see that the `Quantitative Ratings` data table is located precisely in that same range. Again, the `passWD` argument is needed only because, the workbook `chap6datasets.xlsx` is password-protected.

The creation of the summary table `summary.df` in lines #16 through #21 was previously covered in chapter 4 (particularly in section 4.2.4). You can play with this code to get a closer look at what is being done.

Script 8.3. Reading the workbook `chap6datasets.xlsx` and creating the summary table `summary.df`

```
09  #-- Reading the password-protected Excel workbook -
10  irrData.df <- read.xlsx(file="./data/chap6datasets.xlsx",
11                          sheetName = "IrrData",
12                          rowIndex=c(2:17),colIndex=c(19:24),
13                          password = passWD)
14
15  #-- Compute the summary table
16  summary.df <- as_tibble(irrData.df) %>%
17    group_by(Group,Target) %>%
18    summarise(across(c(J1,J2,J3,J4),mean)) %>%
19    mutate(across(c(J1,J2,J3,J4),sd,.names="gsd..col")) %>%
20    ungroup() %>%
21    mutate(across(!c(Group,Target),round,3))
22  print(summary.df)
```

End of Script

The next segment of Script 8.1 uses the `write.xlsx()` function to write the

`summary.df` data frame to the `summaryStats1` worksheet included in an Excel workbook named `chap8data1.xlsx`. If this workbook does not exist, then it will be created. If it exists, you will need to close it before it can be updated. Otherwise, R will generate an error message.

Script 8.4. Writing the `summary.df` dataset to the `summaryStats1` worksheet

```
24  #-- Write the summary table to the input workbook
25  write.xlsx(x = as.data.frame(summary.df),
26         file="./data/chap8data1.xlsx",
27         sheetName = "summaryStats1",append = FALSE,
28         row.names = FALSE, password = passWD)
```

End of Script

There are a few additional things you must know when using the `write.xlsx()` function of the `xlsx` package:

- The `x` argument in line `#25`, specifies the data frame that you want to write to Excel. However, the `write.xlsx()` function works best when this argument is an R data frame, and not a tibble. Since `summary.df` was created by the `tidyverse` package, it is necessarily a tibble and needed to be converted to a data frame object with the `as.data.frame()` function, before it is passed to the function.

- What is done in line `#27`, must be well understood. I used the 2 arguments `"sheetName="` and `"append="`. Note that by default, the `"append="` argument is set to `FALSE`. In this case, if the workbook defined by the `"file="` argument already exists, then *it will be destroyed* and replaced by a new one containing the only sheet defined by the `"sheetName="` argument. If it does not exist, then it will be created, and the `"append="` argument will not play any role.

  When `append=TRUE`, then if the worksheet defined by the `sheetName=` argument does not already exist, it will be added to the other worksheets

already present in the workbook. If it already exists, then R will produce an error message because it cannot add a new worksheet with a name that already exists in the same workbook.

> *The `write.xlsx()` function is practical for writing a data frame to a worksheet. However, it does not allow you to position that data at a location of your choice on the worksheet. That data will have to be written starting from cell `A1`. The `addDataFrame()` function is a more flexible alternative.*

In the next segment of the main script file labeled as Script 8.5, I will write the `summary.df` data frame to a new worksheet named `summaryStats2` in such a way that the top-left corner of the data table is positioned at the cell defined by row #3 and column #6. This task is accomplished using a new function `addDataFrame()`. The objective is to show you how the positioning of a data table in a worksheet can be modified.

Script 8.5. Writing data to a specific cell range on the worksheet

```
30  #- Write the summary.df at a specific worksheet location
31  wb.df1 <- loadWorkbook(file="./data/chap6datasets.xlsx",
32          password = "Microsoft365")
33  my.sheet <- createSheet(wb=wb.df1,sheetName="summaryStats2")
34  addDataFrame(x = as.data.frame(summary.df), sheet=my.sheet,
35          col.names = TRUE,row.names = FALSE,
36          startRow=3, startColumn=6)
37  saveWorkbook(wb.df1, file="./data/chap8data1.xlsx",
38          password = passWD)
```

End of Script

In lines #31 and #32 of Script 8.5, I update the workbook object `wb.df1` by reloading the Excel file `chap6datasets.xlsx`. Otherwise, `wb.df1` will continue pointing to the old version of the workbook. Remember that this workbook was modified by Script 8.4 after the `summaryStats1` worksheet was created. In line

#33, the `createSheet()` function is used to create a worksheet object named `my.sheet`, which points to a newly-created worksheet named `summaryStats2`.

In lines #34 through #36, the `addDataFrame()` function is used to write the `summary.df` data frame to the newly-created worksheet object `my.sheet`. Note that the `x` argument representing the data frame is assigned the value `as.data.frame(summary.df)` instead of `summary.df`. It is because `summary.df` was created as a tibble in line #16, and must be converted to the R data frame format to make it compatible with the `addDataFrame()` function. The `"startRow="` and `"startColumn="` arguments are needed to define the location of the left-top corner of the data table on the worksheet.

So far, your work has been done using workbook and worksheet objects, and not actual files. Therefore, you need to save it to the actual Excel workbook `chap8data1.xlsx`. This task is performed in lines #37 and #38. Assigning `passWD` to the `password=` argument will password-protect your workbook. To remove the password protection, replace `passWD` with `NULL`. Alternatively, you can use an arbitrary string value such as `"newPassWord"` to create a new password.

Finally, Script 8.6 is the segment of the main Script 8.1 that shows you how to delete a specific worksheet from a workbook. Line #41 shows you how to use the `removeSheet()` function to delete the `summaryStats2` worksheet. Since this function acts upon a workbook object and not on an actual Excel workbook, it is necessary to save the workbook object as an actual Excel workbook after deletion, as shown in lines #42 and #43. After deleting the `summaryStats2` worksheet, the associated workbook is saved in line #42 as `chap8delete.xlsx`. This workbook can be downloaded using the link https://bit.ly/3TrG7Sd.

Script 8.6. Deleting a specific worksheet

```
40  #-- Delete the summaryStats2 worksheet if necessary
41  removeSheet(wb=wb.df1, sheetName = "summaryStats2")
42  saveWorkbook(wb.df1, file="./data/chap8delete.xlsx",
43          password = passWD)
```

End of Script

> *So far, you have used existing Excel workbooks from which you could read data into R, and to which you could write data frames. Even if you do not have Excel data to read, you may still want to export an R data frame to Excel. In this case, you need to create a workbook object in R as follows:* `my.workbook <- createWorkbook()`*. This creates a workbook object named* `my.workbook`*, which you can used to export your data to Excel.*

8.3 Manipulating Individual Cells

In section 8.2, you learned to export R data frames to Excel. However, there are times when you need to write various R objects to specific cells or range of cells on a worksheet. You may also want to read values from specific cells for further processing. When preparing your analysis report in Excel, some cell formatting with the use of special fonts, colors and border types is often necessary. For example, you may need to add a caption to a data table by assigning a string of characters to a specific cell, before doing some formatting. Therefore, you need to know how to define an Excel cell in R, assign a value to it and save the workbook on the disk.

In section 8.3.1, you will learn to write to and read from Excel worksheet cells. Section 8.3.2 will show you how to add formatting styles to cells when preparing a report.

8.3.1 *Writing to and Reading from Cells*

A typical *cell* belongs to a *row* and a *column*, both of which are part of a *worksheet* included with other worksheets in a *workbook*. Therefore, the definition of an Excel cell in R depends on a prior definition of a workbook object, a worksheet object, and a row object. The cell object will then be defined by specifying which column number or column numbers within a given row object are used to identify the cells the object will point to.

Consider the Excel workbook `chap8data1.xlsx`[3], which contains 2 worksheets named `summaryStats1` and `summaryStats2`. Note that these are the same 2 worksheets that were previously created after running Script 8.1. They both contain the same data table. Only the positioning of the table on the worksheets differ. In the `summaryStats2` worksheet, the leftmost corner of the table is located in cell `C5` as shown in Figure 8.2. This Excel workbook is used as input file in the next example.

	A	B	C	D	E	F	G	H	I	J	K	L
1												
2												
3												
4												
5			Group	Target	J1	J2	J3	J4	gsd.J1	gsd.J2	gsd.J3	gsd.J4
6			A	1	5.5	2.333	3.833	3.333	1.768	0.471	0.589	2.593
7			A	2	8	1.667	3	7	1.768	0.471	0.589	2.593
8			B	3	8.333	4.333	6	7.333	1.251	1.494	1.31	1.333
9			B	4	7	1.833	3.667	6	1.251	1.494	1.31	1.333
10			B	5	9.5	4.5	5.867	8.667	1.251	1.494	1.31	1.333

Figure 8.2: Data table without caption in the `summaryStats2` worksheet of workbook `chap8data1.xlsx`

I like to develop an R script that will achieve the following objectives:

- A caption must be added to the summary table in the `"summaryStats2"` worksheet in a location that is determined by the range of cells `C1:H4`

[3]It is downloadable with the link: https://bit.ly/3TcIuIk

(see Figure 8.3). The workbook will be saved under the new name `chap8data2.xlsx`. You will see how to write text to an empty cell.

- The second objective of the R script is to add to a comment in cell `C12` of worksheet `summaryStats2`, which says "Score that Judge3 assigned to subject 1 is: 3.833." (see Figure 8.4) The revised workbook will be saved under the name `chap8data3.xlsx`. This example shows you how to read the content of a specific cell without having to read the entire worksheet into an R data frame.

	A	B	C	D	E	F	G	H	I	J	K	L
1			Table 1: Mean Ratings by Group and Target, and									
2				Group-level standard deviations								
3			(Author: Kilem L. Gwet)									
4												
5			Group	Target	J1	J2	J3	J4	gsd.J1	gsd.J2	gsd.J3	gsd.J4
6			A	1	5.5	2.333	3.833	3.333	1.768	0.471	0.589	2.593
7			A	2	8	1.667	3	7	1.768	0.471	0.589	2.593
8			B	3	8.333	4.333	6	7.333	1.251	1.494	1.31	1.333
9			B	4	7	1.833	3.667	6	1.251	1.494	1.31	1.333
10			B	5	9.5	4.5	5.867	8.667	1.251	1.494	1.31	1.333

Figure 8.3: Data table with caption in the `summaryStats2` worksheet of workbook `chap8data2.xlsx`

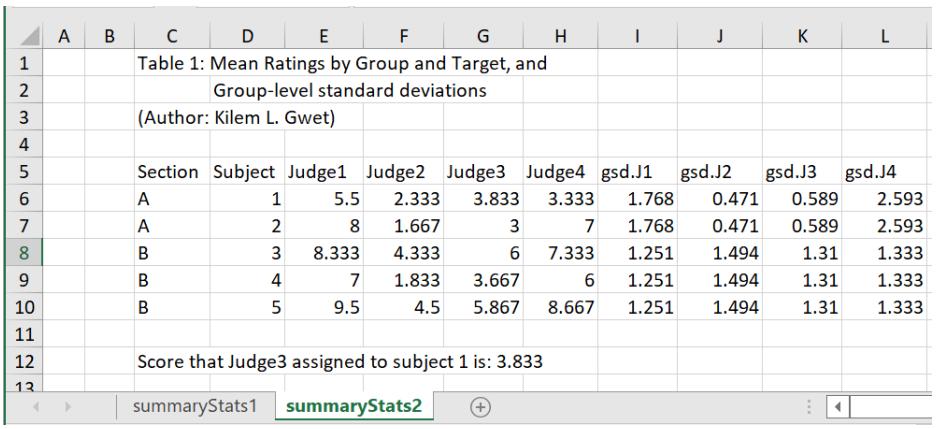

Figure 8.4: Data table with caption and revised column labels, in the `summaryStats2` worksheet of workbook `chap8data3.xlsx`

Here is the R script file that allows you to achieve the 2 goals set above:

Script 8.7. R Script for adding a caption to a data table (*Download this script with the link:* https://bit.ly/3TRAkpo)

```
01 library(xlsx)
02 passWD <- "Microsoft365"
03 wb.df <- loadWorkbook(file="./data/chap8data1.xlsx",
04                        password = passWD)
05 my.sheets <- getSheets(wb.df)
06 print(names(my.sheets))
07
08  #-- create a range of cells to receive the title
09
10 tit.rows <- createRow(my.sheets[[2]], rowIndex=1:4)
11 tit.cells <- createCell(row=tit.rows, colIndex=3:8)
12
13  #-- write title in the first 3 rows and save workbook
14
15 setCellValue(tit.cells[[1,1]],
16              "Table 1: Mean Ratings by Group and Target, and")
17 setCellValue(tit.cells[[2,2]],
18              "Group-level standard deviations")
19 setCellValue(tit.cells[[3,1]],"(Author: Kilem L. Gwet)")
20
21 saveWorkbook(wb.df, file="./data/chap8data2.xlsx")
22
23  #-- change the first 6 column labels in the first row
24
25 wb.df <- loadWorkbook(file="./data/chap8data2.xlsx")
26 my.sheets <- getSheets(wb.df)
27 print(names(my.sheets))
28 clabels.rows <- getRows(sheet = my.sheets[["summaryStats2"]],
29                         rowIndex=5)
30 clabs.cells <- getCells(row=clabels.rows,colIndex = 3:8)
31 setCellValue(clabs.cells[["5.3"]],"Section")
32 setCellValue(clabs.cells[["5.4"]],"Subject")
33 setCellValue(clabs.cells[["5.5"]],"Judge1")
34 setCellValue(clabs.cells[["5.6"]],"Judge2")
```

```
35 setCellValue(clabs.cells[["5.7"]],"Judge3")
36 setCellValue(clabs.cells[["5.8"]],"Judge4")
37
38  #-- Add comment line in row #12
39
40 sub1.rows <- getRows(sheet = my.sheets[[2]], rowIndex=6)
41 sub1.cells <- getCells(row=sub1.rows,colIndex = 3:12)
42 ct.rows <- createRow(my.sheets[[2]], rowIndex=12)
43 ct.cells <- createCell(row=ct.rows, colIndex=1:8)
44 j3score <- getCellValue(sub1.cells[["6.7"]])
45 footnote = paste0("Score that Judge3 assigned to subject 1 is: ",
46                   j3score)
47 setCellValue(ct.cells[[1,3]],footnote)
48
49 saveWorkbook(wb.df, file="./data/chap8data3.xlsx")
_____________________________ End of Script _____________________________
```

In line #01, the `xlsx` package is loaded to your R environment (it is assumed that this package has previously been installed on your machine), and in line #02, the input workbook's password is assigned to the `passWD` variable. Lines #03 and #04 use the `loadWorkbook()` function to create a workbook object that points to the input Excel workbook `chap8data1.xlsx`.

In line #05, the `getSheets()` function returns the following named list of worksheet java objects pointing to the 2 worksheets of interest:

```
> my.sheets
$summaryStats1
[1] "Java-Object{Name: /xl/worksheets/sheet1.xml - Content Type:
application/vnd.openxmlformats-officedocument.spreadsheetml.worksheet+xml}"

$summaryStats2
[1] "Java-Object{Name: /xl/worksheets/sheet2.xml - Content Type:
application/vnd.openxmlformats-officedocument.spreadsheetml.worksheet+xml}"
```

Therefore, `my.sheets` is a two-element named list. The description of each of these elements starts with the dollar ($) sign followed by the worksheet name. For example, `$summaryStats1` indicates that the first list element is named `summaryStats1`. The second part of the list element is the java object

description in a special lingo that only the `xlsx` package needs to understand. Line `#06` uses the `names()` function to extract only the names associated with the named list elements and return a vector of worksheet names. This function can prove useful if the number of worksheets in your workbook is very large to be manipulated manually.

`Writing the table caption in the first 3 rows`

Before I explain what line `#10` does, remember that the table caption must be written in the range of cells `C1:H4` (see Figure 8.2). Therefore, you must create a cell object that points to that range. This is precisely the task that is undertaken in lines `#10` and `#11` in a 2-step process where a list of row objects `tit.rows` pointing to the 4 rows `1:4` is first created in lines `#10` using the `createRow()` function. In line `#11`, I used the `createCells()` function to create a named list of cell objects `tit.cells` based on the `tit.rows` list of row objects, by specifying the columns of interest `3:8` in the `colIndex=` argument. It is the `tit.cells` object that points to the target range of cells where the table caption will be written. For a more comprehensive list of functions needed to create new Excel objects or update cell objects, see Figure 8.6. For a list of functions often used to access existing Excel objects, see Figure 8.5.

Now that the range of cells needed to write the table caption has been defined by the list of the cell objects `tit.cells`, you need a way of accessing these individual cells. Note that `tit.cells` is a matrix of Java objects, each of which pointing to one of the 24 cells (`4 rows` × `6 columns`) you have "booked" for the caption. Here is an extract of what R will display on the console if you print out the content of the `tit.cells` variable:

```
> tit.cells
  3
1 <S4 class 'jobjRef' [package "rJava"] with 2 slots>
2 <S4 class 'jobjRef' [package "rJava"] with 2 slots>
3 <S4 class 'jobjRef' [package "rJava"] with 2 slots>
4 <S4 class 'jobjRef' [package "rJava"] with 2 slots>
  4
1 <S4 class 'jobjRef' [package "rJava"] with 2 slots>
```

```
2 <S4 class 'jobjRef' [package "rJava"] with 2 slots>
3 <S4 class 'jobjRef' [package "rJava"] with 2 slots>
4 <S4 class 'jobjRef' [package "rJava"] with 2 slots>
```

In total, you will have 6 such blocks of 4 objects each (only 2 are displayed here). Each block represents one of the columns of the `4×6` matrix of cell objects. Note that these blocks are numbered 3, 4, 5, 6, 7 and 8 corresponding to the actual column numbers on your Excel worksheet. However, cell `C1` will be referred to as `tit.cells[[1,1]]`. It is because column `#3` of the worksheet represents the first column in the range of cells defined by `tit.cells`.

In lines `#15` and `#16`, I use the `setCellValue()` function to assign the first title line to cell `C1`. You can see from Figure 8.3 that cell `C1` is associated with the matrix element `tit.cells[[1,1]]`. The same function is used again in lines `#17-#19` to write the remaining title lines.

Note that the `setCellValue()` function writes the title to an Excel object located in the computer memory. Therefore, you need to save that information on the disk so that it becomes accessible outside of the R environment. The `saveWorkbook()` function is used in line `#21` to export the content of the `wb.df` workbook object to an Excel workbook named `chap8data2.xlsx`. Figure 8.3 shows you what the `ssummaryStats2` worksheet of the `chap8data2.xlsx` workbook looks like.

Changing the first 6 column labels

Suppose that you want to change the first 6 column labels of the summary table of Figure 8.3 to what is shown in Figure 8.4. That is, you want the labels `Group, Target, J1, J2, J3, J4` to be changed to `Section, Subject, Judge1, Judge2, Judge3, Judge4` without having to read `ssummaryStats2` data. This task is accomplished in lines `#25-#36`.

What is done in lines `#25-#27` has already been previously discussed, and will not be covered here. The purpose of lines `#28-#30` is to define the range of cells `"C5:H5"` as an Excel object that can further be manipulated. Since these

cells have already been used and are therefore not empty, you will create the Excel objects using the `getRows()` and the `getCells()` functions as opposed to the `createRow()` and `createCell()` used previously.

> *I must stress out that the* `createCell()` *function always returns a matrix of cell objects, whose elements are referenced as* `cells[[3,2]]` *for example. However, the* `getCells()` *function always returns a named list of cell objects, whose elements are referenced as* `cells[["3.2"]]` *for example.*

If you print out the Excel object `clabs.cells` created in line `#30`, you will obtain the following named list of 6 elements:

```
> clabs.cells
$`5.3`
[1] "Java-Object{Section}"
$`5.4`
[1] "Java-Object{Subject}"
$`5.5`
[1] "Java-Object{Judge1}"
$`5.6`
[1] "Java-Object{Judge2}"
$`5.7`
[1] "Java-Object{Judge3}"
$`5.8`
[1] "Java-Object{Judge4}"
```

These elements are named `"5.3", "5.4", "5.5", "5.6", "5.7", "5.8"` and are used in lines `#31-#36` to make the changes that you were after.

`Adding a comment line in row #12`

To finalize the summary table of Figure 8.4, I need to add the comment line in row `#12`, which says `"Score that Judge3 assigned to subject 1 is: 3.833"`. To this end, you first need to read the score the Judge3 assigned to subject 1 in row `#6`. Therefore, you need to define 2 Excel objects. One object named `sub1.cells`, is defined in line `#41` of Script 8.7 and points to the range

of cells `"C6:L6"`, which contains all subject 1's scores (including judge3's)[4]. The second object named `ct.cells` is defined in line `#43` and points to the range of cells `"A12:H12"`, where the command line will be written[5].

Function	Description	Example
loadWorkbook(file, xlsxFile = NULL, isUnzipped = FALSE)	loadWorkbook creates a java object reference corresponding to the Excel workbook to load.	wb <- loadWorkbook(file = "./data/wbexample.xlsx")
getSheets(wb)	getSheets returns a list of java object references each pointing to an worksheet. The list is named with the sheet names	sheets <- getSheets(wb)
getRows(sheet, rowIndex = NULL)	getRows returns a list of java object references each pointing to a row. The list is named with the row number	rows <- getRows(sheet) # get all the rows
getCells(row, colIndex = NULL, simplify = TRUE)	getCells returns a list of java object references for all the cells in the row if colIndex is NULL. If you want to extract only a specific columns, set colIndex to the column indices you are interested.	cells <- getCells(rows) # returns all non empty cells
getCellValue(cell, keepFormulas = FALSE, encoding = "unknown")	getCellValue returns the value in the cell as an R object. Type conversions are done behind the scene. This function is not vectorized.	value <- getCellValue(cell)
removeSheet(wb object, sheetName)	Delete a specific worksheet from a given workbook	removeSheet(wb, sheetName = "Sheet1")

Figure 8.5: Functions that allow you to access existing Excel objects

The `sub1.cells` object is created with the `getCells()` function because row `#6` already exists (i.e. is not empty) and needs not be created. However, the situation is different with row `#12`, which is empty and must be created. Therefore the `ct.cells` object must be created with the `createCell()` function.

In line `#44`, the score that subject 1 received from judge3 is assigned to

[4] Technically speaking, you can define `sub1.cells` to point to cell `"G6"` only.

[5] Since the comment line is written in cell `"C12"`, you could well points this object directly to that cell.

the `j3score` variable. This variable is used in line `#46` during the creation of the `footnote` variable, which contains the comment line that you want. The content of this footnote variable is written to cell `"C12"` in line `#47` using the `setCellValue()` function. Finally the revised spreadsheet is saved in a new Excel workbook named `chap8data3.xlsx` in line `#49`.

Function	Description	Example
createWorkbook()	createWorkbook returns a java object reference pointing to an empty workbook object	wb <- createWorkbook()
createSheet()	createSheet returns a worksheet object	sheet1 <- createSheet(wb, "Sheet1")
createRow(sheet, rowIndex = 1:5)	createRow() creates a list of java objects, each pointing to a row.	rows <- createRow(sheet1, rowIndex=1:10) # 10 rows
createCell(row, colIndex = 1:5)	createCell creates a matrix of lists, each element of the list being a java object reference to an object of type Cell representing an empty cell.	cells <- createCell(rows, colIndex=1:8) # 8 columns
setCellValue(cell, value, richTextString = FALSE, showNA = TRUE)	setCellValue writes the content of an R variable into the cell.	setCellValue(cell1, 1) # add value 1 to cell A1

Figure 8.6: Functions to create Excel objects and update cell objects

Script 8.8 shows how you can use the functions in Figure 8.6 to create a new Excel workbook named `"New Excel Workbook.xlsx"` from within R. This script also adds a worksheet named `"WorkSheet-1"`, and writes the text `"Workbook created from within R"` in cell `"C2"`.

In line `#02`, I creates an empty workbook object named `my.wb` with the `createWorkbook()` function. Line `#03` uses the `createSheet()` function to create a worksheet object named `my.sheet` that points to a single worksheet named `"WorkSheet-1"` in the workbook object `my.wb`. Line `#04` creates a list of 3 row objects, whereas line `#05` creates a 3×15 matrix named `my.cell`,

each element of which is a cell object. For example, `my.cell[[2,3]]` allows you to access the cell defined by row 2 and column 3, which is cell `"C2"`. Therefore, you would assign the string `"Workbook created from within R"` to cell `"C2"` using the `setCellValue()` function as shown in line `#06`. The workbook created at this stage exists only inside the computer memory. The `saveWorkbook()` function of line `#07` is what you need to write the workbook object `my.wb` to the disk and give it the name `"New Workbook.xlsx."`

Script 8.8. R Script for creating a new Excel worksheet from within R *(Download this script with the link: https://bit.ly/3DcxSUn)*

```
01 library(xlsx)
02 my.wb <- createWorkbook()
03 my.sheet  <- createSheet(wb=my.wb, sheetName="WorkSheet-1")
04 my.row <- createRow(sheet=my.sheet, rowIndex = 1:3)
05 my.cell <- createCell(row=my.row, colIndex = 1:15)
06 setCellValue(my.cell[[2,3]],"Workbook created from within R")
07 saveWorkbook(wb=my.wb, file="./data/New Excel Workbook.xlsx")
_____________________________ End of Script _____________________________
```

8.3.2 *Formatting Individual Cells*

In section 8.3.1, you learned how to assign values to specific cells, and write captions to data tables. In section 8.2, the focus was on writing R data frames to Excel worksheets. In this section, I will show you 2 things:

1. You will learn how to write R vectors and matrices to Excel. This requires that you become familiar with the manipulation of ranges of cells or blocks of cells.

2. I will show you how you can do basic cell formatting to highlight specific cells conditionally upon their content.

Let us start with a concrete example and create a 10×6 matrix of random

numbers[6]. Ultimately, I want to print that matrix to an Excel worksheet within the range of cells `B3:K19`. The final appearance of the worksheet is shown in Figure 8.7. All negative numbers in the matrix are automatically written in yellow and highlighted in blue. The sequential numbers in row #3 and in column `B` delineate the initial range of cells (or cell block in R jargon) within which I decided to work.

	A	B	C	D	E	F	G	H	I	J	K
1											
2											
3		1	2	3	4	5	6	7	8	9	10
4		2									
5		3									
6		4		7.173	5.325	1.670	1.628	9.004	3.485		
7		5		1.293	-0.456	3.896	0.491	4.575	1.106		
8		6		7.420	-1.746	2.993	0.897	0.825	11.533		
9		7		7.208	1.429	5.476	0.097	-0.579	-0.443		
10		8		4.034	1.393	2.289	5.189	-1.816	2.297		
11		9		-3.198	0.977	4.363	6.762	-1.443	3.426		
12		10		-0.936	3.433	6.517	6.171	-3.286	4.788		
13		11		1.410	-0.800	-0.057	0.911	6.779	1.861		
14		12		2.479	4.112	-2.253	7.082	5.579	-5.728		
15		13		11.397	-2.079	2.673	1.466	1.659	-2.175		
16		14									
17		15									
18		16									
19		17									

Figure 8.7: Worksheet cells formatted programmatically with the `xlsx` package

To create the worksheet of Figure 8.7, one option is to execute Script 8.9. I am now going to review step by step what this script file does.

- Line #01 loads the `xlsx` package into your R environment.
- In line #02, I create a workbook object named `my.wb`, which you can see as a workbook that resides inside the computer memory.

[6]These random numbers are generated from the Normal distribution with mean 2.5 and standard deviation 3.7. The Normal distribution plays no special role here. What matters is the 10×6 table of numbers.