

CHAPTER 7

Connecting R to Excel with the Openxlsx Package

OBJECTIVE

This chapter shows how you can use the **openxlsx** R package for creating, writing, styling and editing Excel worksheets within RStudio. This package provides a very useful high-level interface for automating various Excel tasks with R.

Contents

- 7.1 *Introduction* 272
- 7.2 *Reading an Excel Workbook into an R Data Frame* 272
- 7.3 *Writing an R Object to Excel* 278
 - 7.3.1 *Writing an R Object to an Excel Worksheet* 279
 - 7.3.2 *Changing the Color of Worksheet Tabs* 284
 - 7.3.3 *Alternative Methods for Writing a Data Frame to Excel* . . 288
- 7.4 *Formatting Individual Cells* 295
- 7.5 *Inserting R Graphs Into an Excel Worksheet* 305
- 7.6 *Concluding Remarks* 308

7.1 Introduction

The `openxlsx` package is one option for creating, formatting and interacting with Excel files. It provides a high-level interface for writing, styling and editing Excel worksheets from within R. One reason why `openxlsx` is my preferred package for interacting with Excel is that, unlike its competitors, it does not depend on another software such as Java. Moreover, I found its use to be more intuitive.

Before you can use this package for the first time, you need to install it from the R console as follows:

```
> install.packages("openxlsx", dependencies = TRUE)
```

If you want to determine which version of the package has been installed, key in the command `> packageVersion("openxlsx")` on the console. After doing it, I obtained the following outcome: `[1] '4.2.5'`. It indicates that `openxlsx` 4.2.5 is installed on my system.

Once the package installation is complete, load it into your R environment to make all of its functions available for use by your R scripts. This task is accomplished as follows:

```
> library(openxlsx)
```

Throughout this chapter, I will review the most important functions that you will need to create workbooks or edit worksheets.

7.2 Reading an Excel Workbook into an R Data Frame

Consider an Excel file named `chap7data.xlsx`, which you can download from the link <https://bit.ly/3CBdVqm>. This Excel workbook contains 5 worksheets “mtcars,” “iris,” “IrrData,” “IccData,” and “CacData.” With the `openxlsx` package, you can either read the entire content of a given worksheet,

or read a specific range of data of a chosen worksheet. Reading a worksheet in its entirety, assumes that its content is a well-organized data table (e.g. "mtcars" or "iris"). The worksheet named "IrrData" on the other hand, contains 3 independent tables. Therefore, each of the them should be read by specifying the range of cells that defines its content. Let us consider the following example:

Example 7.1

Suppose that you want to read the `iris` worksheet (see Figure 7.1) in its entirety into an R data frame named `iris.df`. To perform this task, the `openxlsx` package offers the 2 functions `read.xlsx()` and `readWorkbook()`, both of which have a similar functionality.

I wrote Script 7.1, a script file that performs this task. I assume that this script file is located in the root directory of an RStudio project folder¹. This project folder also has a subfolder called `data`, where the Excel file `chap7data.xlsx` is located.

The first 2 lines of this script file loads the `tidyverse` and `openxlsx` packages. The 3rd line uses the `read.xlsx()` function to read the `iris` worksheet of the Excel file into a data frame named `iris.df`. This data frame is then converted to the tibble format in line #05, for convenience and to make it more compatible with the `tidyverse` collection of packages. Executing this script file will yield the `iris.df` data frame, an extract of which is shown in Table 7.1 .

- The `read.xlsx()` function uses 2 arguments. The "`xlsxFile`=" argument refers to the full Excel file path name, which includes its directory. The `sheet`= argument contains the name associated with the worksheet you want to read.
- The "`xlsxFile`=" argument can also be an R workbook object. Example 7.2 shows you how R workbook objects are created.

¹To learn more about project directories, it could be helpful to review the material in section 3.3 of chapter 3

- The `sheet=` argument can also be the worksheet sequential number (1,2,...) associated with the order in which they appear in the workbook. For example, the argument `sheet=2` will still refer to the `iris` worksheet, because it is the second worksheet in the workbook. This feature gives you access to worksheets without knowing their respective names.

	A	B	C	D	E
1	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
2	5.1	3.5	1.4	0.2	setosa
3	4.9	3	1.4	0.2	setosa
4	4.7	3.2	1.3	0.2	setosa
5	4.6	3.1	1.5	0.2	setosa
6	5	3.6	1.4	0.2	setosa
7	5.4	3.9	1.7	0.4	setosa
8	4.6	3.4	1.4	0.3	setosa
9	5	3.4	1.5	0.2	setosa
10	4.4	2.9	1.4	0.2	setosa
11	4.9	3.1	1.5	0.1	setosa
12	5.4	3.7	1.5	0.2	setosa
13	4.8	3.4	1.6	0.2	setosa
14	4.8	3	1.4	0.1	setosa
15	4.3	3	1.1	0.1	setosa
16	5.8	4	1.2	0.2	setosa
17	5.7	4.4	1.5	0.4	setosa
18	5.4	3.9	1.3	0.4	setosa
19	5.1	3.5	1.4	0.3	setosa
20	5.7	3.8	1.7	0.3	setosa

Figure 7.1: Extract of the `iris` worksheet^a from the workbook `chap6datasets.xlsx`

^aThe `iris` worksheet has 150 rows of data.

Script 7.1. Reading the iris

```
01 library(tidyverse)
02 library(openxlsx)
03 iris.df <- read.xlsx(xlsxFile = "../data/chap7data.xlsx",
04   sheet = "iris")
05 iris.df <- as_tibble(iris.df)
06 iris.df
```

End of Script

Note that R packages other than **openxlsx** offer functions with the same name **read.xlsx()**. This will create a conflict. If this happens, then the solution would be to call the **read.xlsx()** functions as follows: **openxlsx::read.xlsx()**.

Table 7.1 : Extract of the iris.df R Dataset

# A tibble: 150 x 5					
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
	<dbl>	<dbl>	<dbl>	<dbl>	<chr>
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa
# ... with 140 more rows					

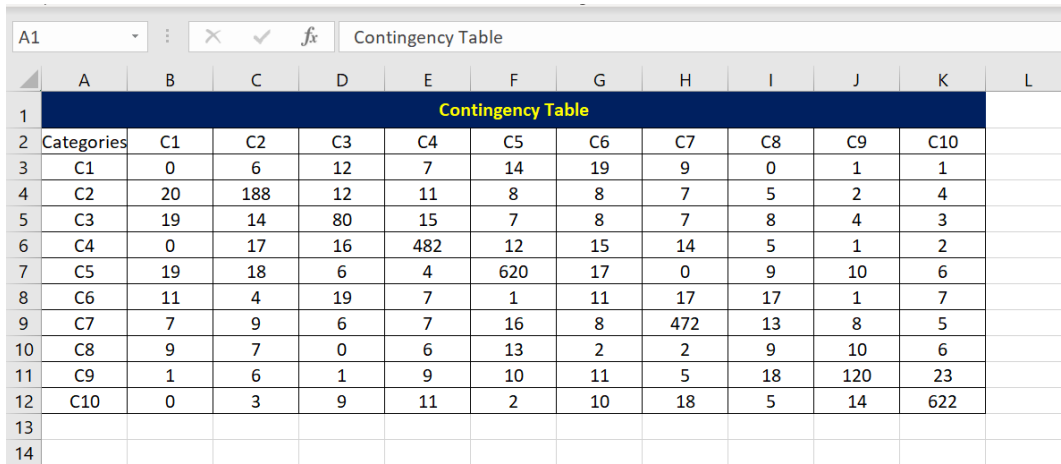
Example 7.2

Reading the contingency table in the IrrData worksheet

As previously mentioned, the `IrrData` worksheet contains 3 data tables. These are the “Contingency Table,” the “Categorical Ratings,” and the “Quantitative Ratings.” To only read the “Contingency Table”, shown in Figure 7.2, you need to formulate a more descriptive request in your R script.

- Script 7.2 reads the Contingency Table portion of the `IrrData` worksheet into the `irrdata.df` data frame. Table 7.3 shows the content of `irrdata.df`.
 - The first 2 lines of this script file load the `tidyverse` and `openxlsx` packages.
 - In line #03, the `loadWorkbook()` function of the `openxlsx` package is used to create an R workbook object named `wb.df1`, which points to the Excel workbook `chap7data.xlsx`.
 - Line #04 uses the `names()` function to list all worksheets included in the workbook identified by the workbook object `wb.df1`. This function can be useful for verification purposes.
 - Line #05 reads the Excel workbook into a data frame named `irrdata.df`, using the `readWorkbook()` function. This time, I have not used the `read.xlsx()` function as before. As a matter of fact, both functions can be used interchangeably. The `"xlsxFile="` argument of this function is assigned the workbook object previously created by the `loadWorkbook()` function. The `"sheet="` argument specified the worksheet you want to read. The 2 arguments `rows = 2:12`, and `cols = 1:11` tell R to read the range of cells defined by rows 2 through 12 and columns 1 through 11 (see Figure 7.2). You can actually select non-contiguous rows. For example, `rows = c(2,3,6,8)`, and `cols = 1:11` will yield the small data frame of Table 7.2
-

Example 7.2 shows how convenient it is to manipulate Excel data from within R, without touching the original data. A manual extraction of specific rows, columns or cells from within Excel would be tedious and error-prone. Even if you were to use Excel VBA, which is Excel’s macro programming language, this task would still be laborious.



	A	B	C	D	E	F	G	H	I	J	K	L
1	Contingency Table											
2	Categories	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	
3	C1	0	6	12	7	14	19	9	0	1	1	
4	C2	20	188	12	11	8	8	7	5	2	4	
5	C3	19	14	80	15	7	8	7	8	4	3	
6	C4	0	17	16	482	12	15	14	5	1	2	
7	C5	19	18	6	4	620	17	0	9	10	6	
8	C6	11	4	19	7	1	11	17	17	1	7	
9	C7	7	9	6	7	16	8	472	13	8	5	
10	C8	9	7	0	6	13	2	2	9	10	6	
11	C9	1	6	1	9	10	11	5	18	120	23	
12	C10	0	3	9	11	2	10	18	5	14	622	
13												
14												

Figure 7.2: Distribution of 3380 subjects by category

Script 7.2. Reading a contingency table from the IrrData worksheet in the chap7data.xlsx workbook

```
01 library(tidyverse)
02 library(openxlsx)
03 wb.df1 <- loadWorkbook(file="./data/chap7data.xlsx")
04 names(wb.df1)
05 irrdata.df <- readWorkbook(xlsxFile=wb.df1,sheet="IrrData",
06                             rows = 2:12,cols = 1:11)
07 irrdata.df <- as_tibble(irrdata.df)
08 irrdata.df
```

_____ End of Script _____

Table 7.2 : Selected Rows from the Contingency Table in the IrrData Worksheet

```
> irrdata.df
# A tibble: 3 x 11
  Categories C1 C2 C3 C4 C5 C6 C7 C8 C9 C10
  <chr>      <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 C1         0     6    12     7    14    19     9     0     1     1
2 C4         0    17    16   482    12    15    14     5     1     2
3 C6        11     4    19     7     1    11    17    17     1     7
```

Table 7.3 : Contingency Table from the IrrData Worksheet

```
> irrdata.df
# A tibble: 10 x 11
  Categories C1 C2 C3 C4 C5 C6 C7 C8 C9 C10
  <chr>      <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 C1         0     6    12     7    14    19     9     0     1     1
2 C2        20   188    12    11     8     8     7     5     2     4
3 C3        19    14    80    15     7     8     7     8     4     3
4 C4         0    17    16   482    12    15    14     5     1     2
5 C5        19    18     6     4   620    17     0     9    10     6
6 C6        11     4    19     7     1    11    17    17     1     7
7 C7         7     9     6     7    16     8   472    13     8     5
8 C8         9     7     0     6    13     2     2     9    10     6
9 C9         1     6     1     9    10    11     5    18   120    23
10 C10        0     3     9    11     2    10    18     5    14   622
```

7.3 Writing an R Object to Excel

After learning how to read Excel data into R, you will now learn to write an R object to Excel. Note that the primary goal for connecting R and Excel is to use the powerful R functions that are effective for performing complex analyses, and to write the analysis results back to Excel. For example cleaning and reorganizing a data worksheet with tens of thousands rows will be done

more effectively with R.

Here are 3 situations you may encounter in practice:

- *Writing an R Object to an Excel Worksheet:* The `writeData()` function takes an R object (vector, matrix, or data frame) and write it to a worksheet starting from a specific cell defined by a row and a column number. In section 7.3.1, you will see with a concrete example how this function should be used. Moreover, if you need to create several Excel worksheets, then you could enhance the readability of your workbook by changing the color of each worksheet tab as discussed in section 7.3.2.
- *Writing an R Data Frame to a Worksheet as an Excel Table:* The Excel table is a special range of cells that is set up to make managing and analyzing a group of related data easier. You can write an R data frame to Excel and format it as an Excel table using the `writeDataTable()` function. The use of this function is illustrated in section 7.3.3.
- *Writing a Data Frame or a List to a New Excel Workbook to be Created.* This task is performed with the `write.xlsx()` function. This function is known to be offered by R packages other than `openxlsx`, which may create a conflict. If in doubt, then call the `write.xlsx()` functions as `openxlsx::write.xlsx()`, specifying the package it originates from. A concrete example in section 7.3.3 will show you how to use this function.

7.3.1 Writing an R Object to an Excel Worksheet

Let us consider the `chap7data.xlsx` workbook once again². The objective is to read the “Quantitative Ratings” table of the `IrrData` worksheet (c.f. Figure 7.3) and to create by its side a summary table containing the following information:

²You may download it with the link <https://bit.ly/3CBdVqm>

- The mean value of each of the variables J1, J2, J3, and J4 calculated at the Group-Target level.
- The standard deviation of all Group-Target means, calculated at the Group level, and separately for each of the variables J1, J2, J3, and J4.
- The title “Mean Ratings by Group and Target, and Standard Deviations by Group” is added at the top of the summary table.

For example, if `Group="A"` and `Target="1"`, the summary table shows `J1=5.5`, which is the mean value $(6+6.5+4)/3$. Moreover, for `Group="A"`, `gsd.J1=0.471`, representing the standard deviation of $\{5.5, 8\}$, the 2 mean values associated with group "A". Script 7.3 shows you the commands you need to perform this task, and produce an output depicted in Figure 7.4.

Script 7.3. R Script that illustrates the `writeData()` and `saveWorkbook()` functions. You may download it with the link: <https://bit.ly/3wytXx6>

```
01 library(tidyverse)
02 library(openxlsx)
03 wb.df1 <- loadWorkbook(file="./data/chap7data.xlsx")
04 names(wb.df1)
05 quant.df <- read.xlsx(xlsxFile=wb.df1, sheet = "IrrData",
06   rows = c(2:17),cols = 19:24)
07 summary.df <- as_tibble(quant.df) %>%
08   group_by(Group,Target) %>%
09   summarise(across(c(J1,J2,J3,J4),mean)) %>%
10   mutate(across(c(J1,J2,J3,J4),sd,.names="gsd..col")) %>%
11   ungroup() %>%
12   mutate(across(!c(Group,Target),round,3))
13
14 #- Write R object to an Excel Worksheet
15 writeData(wb = wb.df1,sheet = "IrrData",
16   x = "Mean Ratings by Group and Target,",
17   startCol = 26, startRow = 1)
```

```
18 writeData(wb = wb.df1, sheet = "IrrData",
19           x = "and Standard Deviations by Group",
20           startCol = 26, startRow = 2)
21 writeData(wb = wb.df1, sheet = "IrrData", x=summary.df,
22           startCol = 26, startRow = 4, rowNames = FALSE)
23 saveWorkbook(wb = wb.df1, file = "../data/chap7Output1.xlsx",
24             overwrite = TRUE)
```

End of Script

I am going to review this script file and make a few comments about what it does:

- Lines #01 and #02 load the 2 packages `tidyverse` (I always use it for data analysis) and `openxlsx` (needed to manipulate Excel workbooks).
- Line #03 creates an Excel workbook object named `wb.df1` that R will use from now on to interact with the `chap7data.xlsx` workbook.
- Line #04 list all worksheets contained in the `chap7data.xlsx` workbook. This list is useful since you need these names to access specific datasets.
- Lines #05 and #06 use the `read.xlsx()` function of the `openxlsx` package to read a worksheet named `IrrData`³, and extract all values in the range of cells defined by rows 2 through 17 and columns 19 through 24, into the data frame `quant.df`.
- Lines #07 through #12 perform many tasks. In line #07, the `as_tibble()` function converts the standard `quant.df` data frame to a `tidyverse`-friendly tibble. The `group_by()` function is used in line #08 to tell R that the subsequent analysis will be performed at the `Group-Target` level. In line #09, the mean value of each variable J1, J2, J3 and J4 is calculated at the `Group-Target` level. This produces a summary table for each `Group-Target` combination.

³Note that worksheet names in R are case sensitive. That is, `IrrData` is different from `irrData`.

Note that the `summary()` function takes the input data frame `quant.df` and creates a smaller summary dataset where each record represents a **Group-Target** level. However, this summary dataset will still be “grouped” by **Group** only. That is, the subsequent analysis on the summary file will be carried out at the **Group** level. If instead, you want it to be carried out at the row or record level, then you must “ungroup” the summary data file using the `ungroup()` function.

In line #10, I use the `mutate()` function to add 4 columns named `gsd.J1`, `gsd.J2`, `gsd.J3` and `gsd.J4` that contains the standard deviation of the J-variables evaluated at the **Group** level.

In line #11, the `ungroup()` function instructs R to perform subsequent analysis at the row level so that the `mutate()` function of line #12 can be used to round all numbers to 3 digits after the decimal points.

Without the use of the `ungroup()` function in line #11, the `mutate()` function of line #12 will not work.

Now is the time to export the results to Excel.

- In lines #15, #16, and #17, I use the `writeData()` function to write the title “Mean Ratings by Group and Target” to the `IrrData` worksheet starting from column #26 and row #1.
- In lines #18, #19, and #20, I use the `writeData()` function one more time to write the second part of the title “and Standard Deviations by Group”, to the `IrrData` worksheet starting from column #26 and row #2.
- In lines #21, and #22, I use the `writeData()` function to write the `summary.df` data frame to the `IrrData` worksheet, starting from column #26 and row #4.
- In lines #23 and #24, I use the `saveWorkbook()` function to write the workbook object `wb.df1` to a physical location on your hard drive as an actual Excel workbook named `chap7Output1.xlsx`.

By using the argument `overwrite=TRUE`, I opted to overwrite any existing Excel workbook with the same name `chap70Output1.xlsx`. You could also have specified the exact name of the input file `chap7data.xlsx`, in which case it would have been overwritten. The output file `chap70Output1.xlsx`, can be downloaded using the link <https://bit.ly/3ASmQSZ>.

If creating a single summary table such as that of Figure 7.4 is your primary objective, you would not need to connect R to Excel to obtain it. You can create it from within Excel without R. The key benefit of connecting R and Excel lies in the ability to perform multiple data manipulations involving grabbing pieces of data from various worksheets, different workbooks and possibly external non-Excel data sources, combining them and updating existing Excel workbooks with new data. The new enriched workbook(s) can be further analyzed more conveniently from within Excel. The current example shows you some of the possibilities that are available to you with the `openxlsx` package.

	S	T	U	V	W	X
	Quantitative Ratings					
Group	Target	J1	J2	J3	J4	
A	1	6	1	3	2	
A	1	6.5	3	3	4	
A	1	4	3	5.5	4	
B	5	10	5	6	9	
B	5	9	4.5	5	9	
B	5	9.5	4	6.6	8	
B	4	6	2	4	7	
B	4	7	1	3	6	
B	4	8	2.5	4	5	
A	2	9	2	5	8	
A	2	7	1	2	6	
A	2	8	2	2	7	
B	3	10	5	6	9	
B	3	7	4	6	5	
B	3	8	4	6	8	

Figure 7.3: Quantitative Ratings from 4 Judges

S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI
Quantitative Ratings						Mean Ratings by Group and Target, and Standard Deviations by Group										
Group	Target	J1	J2	J3	J4											
A	1	6	1	3	2											
A	1	6.5	3	3	4	Group	Target	J1	J2	J3	J4	gsd.J1	gsd.J2	gsd.J3	gsd.J4	
A	1	4	3	5.5	4	A	1	5.5	2.333	3.833	3.333	1.768	0.471	0.589	2.593	
B	5	10	5	6	9	A	2	8	1.667	3	7	1.768	0.471	0.589	2.593	
B	5	9	4.5	5	9	B	3	8.333	4.333	6	7.333	1.251	1.494	1.31	1.333	
B	5	9.5	4	6.6	8	B	4	7	1.833	3.667	6	1.251	1.494	1.31	1.333	
B	4	6	2	4	7	B	5	9.5	4.5	5.867	8.667	1.251	1.494	1.31	1.333	
B	4	7	1	3	6											
B	4	8	2.5	4	5											
A	2	9	2	5	8											
A	2	7	1	2	6											
A	2	8	2	2	7											
B	3	10	5	6	9											
B	3	7	4	6	5											
B	3	8	4	6	8											

Figure 7.4: Quantitative Ratings along with a Summary Table of Means by Group and Target, as well as Group Standard Deviations of Means

You may have noticed that the `openxlsx` package makes it more convenient to assign R objects to an Excel range of cells than the `xlsx` package, which requires that you first define cell objects.

7.3.2 Changing the Color of Worksheet Tabs

In this section, you will learn 2 things. First, you will see how you can use R to analyze a large number of variables in a data frame without having to list them all explicitly. Second, you will learn how with R, you would create multiple Excel worksheets in the same workbook, while assigning a different color to each worksheet tab to make them stand out.

Let us consider the Excel workbook `chap7data.xlsx` used in previous examples, and which you can download with the link <https://bit.ly/3CBdVqm>.

The focus in this section, will be on a specific worksheet named `IccData`. It contains the largest dataset of the `chap7data.xlsx` workbook with 2160 records and 26 variables, 5 of which are categorical and the remaining 21 are quantitative.

- *Categorical Variables*

You will choose the level of aggregation of your analysis along dimensions that are defined by the following 5 categorical variables:

`VSOF`T, `VID_Subj`, `VMOD`, `vSoftMod` and `RestStress_Rater`.

- *Quantitative Variables*

The 21 quantitative variables on which the calculations will be performed, are given by:

`vglobal`, `vlad`, `vlc`x, `vrca`,
`vseg01`, `vseg02`, `vseg03`, `vseg04`, `vseg05`, `vseg06`, `vseg07`,
`vseg08`, `vseg09`, `vseg10`, `vseg11`, `vseg12`, `vseg13`, `vseg14`,
`vseg15`, `vseg16`, `vseg17`.

To conduct my analysis, I need to read the Excel worksheet `IccData` included in the workbook `chap7data.xlsx`, and to create in that same workbook, 2 worksheets named `summaryStats1` and `summaryStats2`. The `summaryStats1` worksheet, whose tab will be colored in red, should contain the summation of each of the variables `vglobal`, `vlad`, `vlc`x, and `vrca`, calculated by `VSOF`T level. The columns of this table will be named as `tot_vglobal`, `tot_vlad`, `tot_vlc`x, and `tot_vrca` (see Figure 7.5). The `summaryStats2` worksheet, whose tab will be colored in blue, should contain the summation of each of the 17 variables `vseg01-vseg17`, calculated by `VSOF`T. The columns of this table will be named as `tot_vseg01-tot_vseg17` (see Figure 7.6).

Script 7.4 is a 28-line script file that I wrote to implement this analysis. In lines #01-#16, the content of the `IccData` worksheet is read into a data frame named `icc.df`, and the 2 summary data frames of interest `vsoft1` and `vsoft2` are created. Note that `vsoft1` and `vsoft2` are the 2 data frames that are later written to the `summaryStats1` and `summaryStats2` worksheets respectively.

Script 7.4. R Script for creating the `summaryStats1` and `summaryStats2` worksheets of Figures 7.5 and 7.6. Download this script with the link: <https://bit.ly/3Tz9SRs>

```
01 library(tidyverse)
02 library(openxlsx)
03 wb.df1 <- loadWorkbook(file="./data/chap7data.xlsx")
04 names(wb.df1)
05 icc.df <- read.xlsx(xlsxFile=wb.df1, sheet = "IccData")
06 icc.df <- as_tibble(icc.df)
07 vsoft1 <- icc.df %>%
08   group_by(VSOFT) %>%
09   summarize(across(c(vglobal,vlad,vlcx),sum,
10     .names="tot_.col"))
11 vsoft2 <- icc.df %>%
12   group_by(VSOFT) %>%
13   summarize(across(contains("vseg"),sum,
14     .names="tot_.col"))
15 print.data.frame(vsoft1)
16 print.data.frame(vsoft2)
17
18 addWorksheet(wb=wb.df1,sheetName="summaryStats1",
19   tabColour = "red")
20 addWorksheet(wb=wb.df1,sheetName="summaryStats2",
21   tabColour = "blue")
22 #- Write R object to an Excel Worksheet
23 writeData(wb=wb.df1, sheet="summaryStats1", x=vsoft1,
24   startCol = 1, startRow = 2, rowNames = FALSE)
25 writeData(wb=wb.df1, sheet="summaryStats2", x=vsoft2,
26   startCol = 1, startRow = 2, rowNames = FALSE)
27 saveWorkbook(wb.df1, "./data/summaryFile.xlsx",
28   overwrite = TRUE)
```

End of Script

Look at lines #10 and #14 where the column names are created by specifying `.names="tot_.col"` as argument of the `summarize()` function. The original column name is represented by `col`, whereas the `"tot_"` is the prefix that is added to the column name.

7.3. Writing an R Object to Excel

	A	B	C	D	E	F
1						
2	VSOFIT	tot_vglobal	tot_vlad	tot_vlcx		
3	1	272.569597	265.769453	266.38632		
4	2	274.621596	270.17	270.71		
5	3	547.991511	555.030293	553.32374		
6	4	531.318	529.508	501.773		
7	5	289.177	288.599	277.564		
8	6	929.621783	921.835471	905.323094		
9	7	258.234971	261.302686	249.06978		
10	8	285.233	283.854	272.883		
11	9	245.992056	206.240652	205.885995		
12	11	497.136266	494.035067	492.113282		

Ready Accessibility: Investigate

Figure 7.5: Totals by VSOFIT of the 3 variables vglobal, vlad and vlcx.

	A	B	C	D	E	F	G
1							
2	VSOFIT	tot_vseg01	tot_vseg02	tot_vseg03	tot_vseg04	tot_vseg05	tot_vseg06
3	1	247.871825	271.715726	320.244532	287.355567	258.276853	247.460413
4	2	287.895577	310.452052	306.176068	290.850844	267.558457	282.207293
5	3	525.474849	518.354174	500.440563	514.616549	512.925048	498.42716
6	4	491.017	580.854	606.073	529.778	487.519	481.382
7	5	262.517	270.007	274.595	277.58	257.346	253.902
8	6	773.951022	787.338442	806.854233	885.500552	817.014523	774.536472
9	7	264.4103	263.9239	240.0571	256.2554	238.2825	224.5542
10	8	238.473	246.36	257.831	272.328	257.125	245.908
11	9	217.384887	215.384768	215.340497	211.940933	205.31458	212.024111
12	11	502.186977	478.350359	461.74652	485.449629	447.691113	478.211591

Ready Accessibility: Investigate

Figure 7.6: Totals by VSOFIT of all variables that start with vseg.

Lines #18-#21 are used to create the 2 worksheets `summaryStats1`, and `summaryStats2` with colored tabs, in the workbook object `wb.df1` created in line #03. Note that so far, all the work is being done on the virtual workbook