# Reading, tweaking and using R commands

### 31/10/2018

[1) Introduction 1](#_Toc528959383)

[2) Looking at commands and tweaking them 1](#_Toc528959384)

[3) Examining and using the log file 7](#_Toc528959385)

[4) The data book and other R objects 8](#_Toc528959386)

[5) Using R-Instat’s calculator: a “halfway” dialog 10](#_Toc528959387)

[6) Running a simple R script 12](#_Toc528959388)

[7) Discussion on running this type of R-script 13](#_Toc528959389)

[8) Putting data into an R-Instat data book 13](#_Toc528959390)

[9) Example: Adding a new graph to R-Instat 14](#_Toc528959391)

## Introduction

R is a very popular statistical language. R-Instat provides a simple way of using part of R without having to write R commands.

With many languages, including R, for a beginner it is easier to read than it is to write. Being able to read some R is useful. It also opens the door to being able to adapt, (or tweak!) R commands. Then you may quickly be able to use R commands that others have written.

Reading, adapting and using R commands may be enough for some people. For others it may help them as they also then learn to write.

## Looking at commands and tweaking them

We use the same data and part of the analysis from the first tutorial. This time we also read and examine some of the R commands that have been produced.

\*\* Go to File > Open From Library.

\*\* Click on the From Package dropdown and choose ggplot2.

\*\* Choose the first example data called diamonds.

\*\* Press Ok.

Then one our first actions in the tutorial was to look at all the data, Fig. 1.

|  |
| --- |
| Fig. 1 The diamonds data |
|  |

There are 10 columns, or variables, and 53 thousand rows. The grid in Fig. 1 shows a window onto the first 1000 rows.

|  |  |
| --- | --- |
| Fig. 2 Viewing all the data | Fig. 3 The R viewer |
|  |  |

\*\* To see all the data, right-click on the bottom tab, and choose the last option, Fig. 2. This opens the R-viewer and you can scroll to see all 53 thousand rows of data, Fig. 3.

Fig, 4 shows the output window. R-Instat has issued 2 commands so far. The first was to open the file from the library and the second was to call the R Viewer. The viewer was called through R-Instat issuing the R View command:

View(title="diamonds", x=data\_book$get\_data\_frame(data\_name="diamonds"))

|  |  |
| --- | --- |
| Fig. 4 The R commands so far | Fig. 5 The Summarise menu |
|  |  |

This used the R command called View. This R command provides a spreadsheet-style data viewer. It has just 2 arguments, the main one is x, which dictates what is viewed. In our case it is the diamonds data frame. The second is a title for the dataset.

Now use a dialog:

\*\* Go to Describe > One Variable > Summarise.

\*\* Right-click in the data selector and choose the option to Add all Variables.

\*\* Change the Maximum Factor Levels to 3, Fig. 5.

\*\* Press **OK** to give the results shown in Fig. 6.

The results, and the R command, is also shown in the output window. This dialog has used the summary command.

The results show that the change to 3 of the number of factor levels was not a good one. We don’t now see enough of the details of the 3 factor columns.

\*\* Return to this last dialog, i.e. to the dialog in Fig. 5.

\*\* Click on the To Script button at the bottom of the dialog.

This has opened a new Script window, Fig 7, and typed a copy of the summary command into this window.

|  |  |
| --- | --- |
| Fig. 6 Initial results from the summary | Fig. 7 The script window |
|  |  |

\*\* Change the 3 in the R command to 10 and then click on the Run All button at the top of this window, Fig. 7.

The result, in Fig. 8, is better.

This change in an R command, is what we call “tweaking” the command. You may not understand everything in the R command in Fig. 7. But it is still easy to see the number 3 in the summary command and to make the change. You now know that the *maxsum* argument controls the number of factor levels displayed.

\*\* Now Right-Click in the script window to give the popup menu shown in Fig. 9.

\*\* Choose the bottom option, Clear Script.

\*\* It will check if you really mean this. Click Yes.

|  |  |
| --- | --- |
| Fig. 8 Result after making the change | Fig. 9 Clearing the script window |
|  |  |

\*\* Now press the circular arrow on the toolbar, Fig. 9 or use View > Reset to Default Layout, Fig. 10. to return to the default layout of the windows. This closes the script window.

You could just have easily made this change, from 3 to 10, more simply in the dialog, Fig. 5. The next example shows a larger change to an R command.

|  |  |
| --- | --- |
| Fig. 10 | Fig. 11 |
|  |  |

The first tutorial showed some correlations for these data, as follows.

\*\* Go to the Describe > Multivariate > Correlations dialog. (Note that only the numeric columns are visible for this dialog.)

\*\* Select the **Multiple Columns** button at the top of the dialog, Fig. 11.

\*\* Select the first 2 variables (Carat and Depth) and the last two (y and z), Fig. 11.

\*\* Click on the **Options button** to go to the sub-dialog.

\*\* Select the Pairwise Plot. Then press Return.

\*\* Press Ok to give the results in Fig. 12 and Fig. 13.

|  |  |
| --- | --- |
| Fig. 12 Correlations with R commands | Fig. 13 The pairs plot |
|  |  |

The Output window, Fig. 12, shows this dialog has produced two sets of R commands. The first gave the correlations and the second gave the graph.

\*\* Return to the last dialog***[[1]](#footnote-1)*** and press the To Script button.

|  |
| --- |
| This is a general feature in R-Instat: Every dialog has a To Script button |

The To Script button puts a copy of the commands into the script window.

The command for the graph is ggpairs. It is from the R package called **GGally**, so the command in Fig. 12 and Fig. 14 is given as GGally::ggpairs.

|  |  |
| --- | --- |
| Fig. 14 Tweaking the ggpairs command | Fig. 15 Running the selected commands |
|  |  |

In contrast, the correlation command, cor, is given simply as cor. That’s because cor is from a core R package (stats) is this is always available so it can be specified without the package name.

Now we “tweak” the **ggpairs** command. The correlations dialog is restricted to numeric columns, but the GGally::ggpairs command also provides interesting graphs for factor columns.

\*\* Change the set of columns by adding two factor columns cut and color to the command, Fig. 14.

\*\* Do this very carefully, so it now reads …columns=c(“cut”,”color”,"carat","depth","y","z").

\*\* Now select the 5 lines of R code for the graph as shown in Fig. 15, right-click and choose Run Selected Text, Fig. 15.

You should now get the results as shown in Fig. 16.

|  |  |
| --- | --- |
| Fig. 16 ggpairs for factors and numeric variables | Fig. 17 An error in the variable name |
|  |  |

If you made a mistake in the typing, then try correcting it. If not, then we now make mistakes and run a line again to see the error message. If you are going to use a script file, or R commands in any way, then you are going to make mistakes and you need to be able to read error messages.

\*\* Return to the script window and change cut to Cut, i.e. put the C as a capital letter.

\*\* While you are still on this line, right-click again and use Run Current Line.

The error message is in Fig. 17. R no longer recognises the column name. It also provides you with a list of the names it does recognise, to help with the correction. This is a very informative error message.

\*\* Return to the script file and correct the mistake in the name.

\*\* Now make another mistake, by omitting a comma, i.e. change “cut”,”color” to “cut””color”.

\*\* Run the line again.

|  |  |
| --- | --- |
| Fig. 18 Error message when comma missed | Fig. 19 And when a quote is missed |
|  |  |

The error message is shown in Fig. 18.

\*\* Correct this error, but now miss out a quote (“) symbol, i.e. it is now “cut,”color”.

\*\* Run the line again to give the error message in Fig 19.

These last two messages are less easy to interpret. But you can see the errors mention *strings* and *unexpected symbols*. If you use commands, then you will start getting used to looking at error messages and learning the common mistakes that can help to make corrections.

\*\* Click on the circular arrow in the toolbar (Fig. 9) to return to the default window positions.

## Examining and using the log file

The log file keeps a record of all the R commands in an R-Instat session.

\*\* Click the scroll symbol on the toolbar, to open the log window.

We also show a second method of opening the log window.

\*\* Go to the View menu, as shown in Fig. 10 in Section 2. There is a tick beside the Log Window, signifying it is open. Click to close the Log window. This action also closes the View menu.

\*\* Go to the View menu again and click on the Log Window again. We do want it open!

|  |  |
| --- | --- |
| Fig. 1 Right click in the log window | Fig. 2 |
|  |  |

The log file is a record of all the commands in an R-Instat session. As such you may want to save this file. This is either through the right-click menu, Fig. 1, or using File > Save As > Save Log As, Fig. 2.

Once you become more skilled in using R you may also sometimes start an analysis in R-Instat and then continue with RStudio.

If you do not have RStudio you can download it here https://www.rstudio.com/, or just follow by reading this section.

|  |  |
| --- | --- |
| Fig. 3 | Fig. 4 |
|  |  |

\*\* In the log window in R-Instat select from the top to the first comment line that says #Code run from script window, Fig. 3.

|  |
| --- |
| A general feature of R-Instat is that every dialog has a Comment field. When you use a dialog, you may accept the default comment or type your own more meaningful comments. We encourage users to use comments to makes your log file clearer and meaningful. |

\*\* Right-Click and choose Copy.

\*\* Open RStudio. It will look something like Fig. 4.

\*\* In RStudio use File > New File > RScript, Fig 4.

\*\* In RStudio either use <ctrl> V to paste or use Edit > Paste.

The R-Instat commands are now available in RStudio.

\*\* In RStudio Select the part of this R script up to the View line, Fig. 5 and click on Run.

The result is in Fig. 6. You can scroll through the data, just as you did earlier in R-Instat.

|  |  |
| --- | --- |
| Fig. 5 | Fig. 6 |
|  |  |

\*\* Now, still in RStudio, return to the script file.

\*\* Select the rest of the script file, Fig. 7, and run this.

The results are in Fig. 8, just as in R-Instat.

|  |  |
| --- | --- |
| Fig. 7 | Fig. 8 |
|  |  |

The R commands in the log file provide a record of what you have done. In addition, you can share it with others.

## The data book and other R objects

\*\* Now return to R-Instat

\*\* Click on the circular arrow in the toolbar, or use View > Reset to Default Layout to return to the two default windows.

In R-Instat the data are stored in a *data book*. We now explain a data book. This is both to help in using R-Instat and to explain the R commands that are generated.

In R-Instat the data are in data frames. A data frame is roughly what Excel calls a list and many database packages call a table. It is a rectangle of columns. Each column has a name and the data in each column (or variable) is of a single type.

The column type can be numeric, or factor, or logical or date, or character, or others. If you have a number column into which there is accidentally a letter “o”, instead of a zero, then the whole column will automatically become a character (text) column.

Data frames also have metadata associated with the columns (variables) of data. For example, as well as a name, each variable can also have a label.

\*\* Use View > Column Metadata (or click on the i symbol in the toolbar).

This shows the metadata currently associated with the diamonds data frame, Fig. 1

|  |  |
| --- | --- |
| Fig. 1 | Fig. 2 |
|  |  |

\*\* Now use Prepare > R Objects > View, Fig. 2.

This View dialog is in Fig. 3. It shows there are 2 objects, called last\_model and last\_graph. They came from the correlations dialog that produced both some summaries and a graph.

|  |
| --- |
| This is a general feature. Every dialog that produces a graph includes an option to save the corresponding object. If not, then the most recent graph is always saved by R-Instat, and given the name last\_graph.  The same is true for all the dialogs that produce a model or summary. |

\*\* Select the last\_model and the option to print. You now again get the correlation matrix in the output window.

|  |  |
| --- | --- |
| Fig. 3 | Fig. 4 |
|  |  |

In R-Instat these objects are also part of the metadata associated with the corresponding data frame. They form part of the *data sheet*. A data sheet is a data frame with (often) quite a lot of metadata and associated objects (e.g. graphs and models).

We now produce a second, summary data frame.

\*\* Go to Prepare > Column: Reshape > Column Summaries.

We choose to summarise a numeric column, *carat*, by the 2 factors called **cut** and **color**.

\*\* Complete the main dialog as shown in Fig. 4. Then click on the Summaries button.

|  |  |
| --- | --- |
| Fig. 5 | Fig. 6 |
|  |  |

\*\* Complete this sub-dialog as shown in Fig. 5.

\*\* Press Return to go back to the main dialog and then Ok.

This has produced a second data frame, Fig. 6. It has 35 rows, because the 2 factors have 5 and 7 levels.

This dialog has also produced more metadata that link these 2 data frames. Further summaries from the diamonds data that use the same factor then go automatically into the same summary data frame.

In R-Instat, a data sheet is a single R data frame with the associated meta data and objects. A data book is a collection of data sheets. Usually the data frames in a data book are linked.

The data book is “behind the scenes” in almost all the R-Instat dialogs. R-Instat users gain a lot from the data book, particularly when the analysis involves more than one data frame. But it does make the R commands a little harder to read. The **data\_book** structure was created specifically for R-Instat so you will not find in other examples of R code.

With this extra information we explain more about R-Instat’s R commands

\*\* Return to the Describe > Multivariate > Correlations dialog again.

\*\* Click on the Options button and change the Graphics to None. Press Return.

\*\* Click on the To Script button.

\*\* Click OK to see the output from this dialog again.

Fig. 7 shows the code from the script window

|  |
| --- |
| Fig 7 (revise with the current script) |
| # Code generated by the dialog, Correlation   1. last\_model <- cor(x=data\_book$get\_columns\_from\_data(data\_name="diamonds", col\_names=c("carat","depth","y","z")), use="complete.obs") 2. InstatDataObject$add\_model(data\_name="diamonds", model=last\_model, model\_name="last\_model") 3. InstatDataObject$get\_models(model\_name="last\_model", data\_name="diamonds") 4. rm(list=c("diamonds", )) |

Line 1 is important. It runs the cor command on the specified columns of data. The data are from the relevant data sheet and the part of the command

“data\_book$get\_columns\_from\_data(data\_name="diamonds", col\_names=c("carat","depth","y","z"))”

has simply taken the 4 columns from the diamonds data frame that is in that ***data*** sheet and put them into a second data frame. The results are not printed, but are put into the last\_model object.

Line 2 has added the model into that data sheet.

Line 3 has simply printed the model, i.e. the correlations in this case.

Finally line 4 has removed the diamonds data from R’s memory.

\*\* In the script window make a new line just after line 1 and type (or copy) last\_model.

\*\* Select lines 1 and the line 2 that you just typed, right-click and choose Run Selected Text. You should get the correlation matrix again.

\*\* Even simpler, delete the left-hand-side from the first line, so it starts with cor. Right-click again and choose Run Current Line.

This shows that getting the result in R, or in RStudio is simpler than in R-Instat. The apparent complexity in reading R-Instat’s R commands is just that they always retrieve what is needed from the relevant data book at the start and then to update the data book at the end.

There are now two linked data frames, i.e. two data sheets. Together they make up a data book.

## Using R-Instat’s calculator: a “halfway” dialog

Most of R-Instat’s dialogs are designed for you to input (File menu), organise (Prepare menu) and analyse (Describe and Model menu) your data without necessarily knowing any R commands.

Each R-Instat dialog produces one, or more, R commands. The commands are usually included in the output window. But if you prefer, right-click in the Output window and choose to Hide (Future) Commands. Then you are not reminded that the R statistical package is behind it all!

In contrast, when you use RStudio, you will use a set of R commands, collected into a script.

A few R-Instat dialogs do not protect you quite so much from R. We call them “halfway” dialogs. They involve you, with assistance, in typing a single R command (or part of a command). We first introduce the most used halfway command, namely R-Instat’s calculator.

\*\* Go to Prepare > Column: Calculate > Calculations.

|  |  |
| --- | --- |
| Fig. 1 | Fig. 2 |
|  |  |

You appear to have a simple calculator of the sort on your mobile phone.

\*\* Use the keyboard to type 12 \* 9.

\*\* Untick Save Result into and press the Try button, Fig. 1.

It says 108, which is not surprising!

\*\* Press Ok and it copies this simple R command into the output window and again says 108.

These halfway dialogs are very powerful – because R is so powerful. But they are also risky!

\*\* Add a zero, so it is 12 \* 90 and press Try again.

\*\* Now change the number 0 for the letter o. It reads 12 \* 9o. Press Try again, Fig. 2.

It now says this command produced an error – which is true. But Ok is still enabled.

\*\* Press Ok

|  |  |
| --- | --- |
| Fig. 3 | Fig. 4 |
|  |  |

It gives an error as we knew it would, see Fig. 3! This time it is quite clear, namely there is a symbol R can’t recognise.

This shows why the halfway dialogs are more dangerous. In most dialogs R-Instat checks that a sensible set of R commands is generated before enabling the Ok button. But not in these halfway dialogs.

You can check for yourself, which is why there is also a Try button.

The positive side is partly the power of these halfway dialogs. They can also help by preparing you gently to write single commands. This can be a useful stepping-stone, to writing scripts later.

We now exploit more of this dialog.

\*\* Check that the diamonds data sheet is the selected one in the dialog, i.e. the primary data and not the summary data, Fig. 4.

\*\* Press the Clear button.

\*\* Click to add the price variable, then the divide, i.e. the / sign and then add the carat variable.

\*\* Check that the formula is sensible by pressing the Try button.

\*\* Check the Save Result into checkbox and give the new column the name pricepc.

\*\* Press Ok

This produced a new column of price per carat in the data frame. It did this by executing the R command pricepc <- price/carat.

The commands are shown in Fig. 5, where we have again added line numbers.

|  |
| --- |
| Fig. 5 |
| 1. diamonds <- data\_book$get\_data\_frame(use\_current\_filter=FALSE, data\_name="diamonds") 2. attach(what=diamonds) 3. pricepc <- price/carat 4. data\_book$add\_columns\_to\_data(col\_data=pricepc, col\_name="pricepc", data\_name="diamonds") 5. rm(pricepc) 6. detach(unload=TRUE, name= diamonds) |

Lines 1-2 gets the data and “attaches” it.

Line 3 does the calculation.

Line 4 puts the new column into the R-Instat data book, and it is then shown in the data frame, Fig. 6

Lines 5-6 is the usual housekeeping and removes the new column from R’s memory and “detaches” the data frame.

|  |  |
| --- | --- |
| Fig. 6 | Fig. 7 |
|  | Many keyboards and showing the logical one completed |

This last example has started to show a little of R’s power. The simple command pricepc <- price/carat has produced a new column with 53 thousand rows.

R has many functions that produce new columns, and hence R-Instat’s calculator has many keyboards, see Fig. 7. It is a logical calculator, a maths calculator, and can also manipulate dates and character columns etc.

Some of these calculations can alternatively also be done with special R-Instat dialogs. For example, R-Instat also has a Prepare > Column: Dates menu to simplify operations with dates.

\*\* Complete the calculation shown in Fig. 7 to add a logical column that indicates whether the diamonds are less than 1 carat or not.

\*\* Use Prepare > Check Data > One Variable Summarise with this new column to show that 36,438 values in this column are TRUE, i.e. are less than 1 carat.

In the R-Instat tutorial the violin plot showed the variable called table was odd in that almost all the cases were recorded as whole numbers. The next calculation therefore rounds the column to whole numbers and then checks how many of the 53 thousand cases were originally recorded with a decimal place.

\*\* Return to the calculator and use the Clear button, Fig. 8. Change to the Maths keyboard and use the round key, Fig. 8.

\*\* Use Try to check the formula and save the result in a column called table0, Fig. 8.

\*\* Press Ok.

We now introduce a more powerful function – even though it isn’t really needed here!

\*\* Return to the calculator and press Clear. Go back to the logical and symbols keyboard, Fig 9, and press the Show Arguments checkbox.

\*\* Press the ifelse key. See Fig. 9.

|  |  |
| --- | --- |
| Fig. 8 | Fig. 9 |
|  |  |

\*\* Change the function to ifelse(test=(table==table0) , TRUE , FALSE)

\*\* Press Try and put the result into a column called tablediff.

\*\* If Try worked, then press Ok. Otherwise correct the formula.

\*\* Use the One Variable Summarise dialog again to show there are only 924 cases.

So, the variable called table was given with a decimal less than one case (row) in 50. The table0 variable is therefore probably the more appropriate variable to analyse.

\*\* Finally, a small exercise. Calculate the **tablediff** variable with a simpler formula than was used above.[[2]](#footnote-2)

## Running a simple R script

R-Instat includes well over 100 R packages. One of these is called agricolae. The instruction guide for each package is accessible from R-Instat’s help menu.

The instruction guide for the package usually gives an example of how each function works. Many of these examples use data sets provided by the package and these data sets are available through the File > Open from Library dialog in R-Instat. However, some examples use a set of R-commands to generate and then analyse the data. We consider one such example.

\*\* Go to Help > R-Packages and choose the agricolae package. It will open in a browser – though you don’t need to be online.

We run an example of an analysis for Balanced Incomplete Blocks (BIBs). Here we are simply interested in the process of running a simple script file of R commands. The scenario is that you would like to understand the potential use of this function by seeing the results from an example of the analysis – though that is not our particular interest here!

\*\* In the agricolae guide go to the examples for the BIB.test command. It is on about page 15. Copy the first 9 lines of the example, to the line, print(out). Or copy the text from Fig. 1.

|  |
| --- |
| Fig. 1 |
| library(agricolae)  # Example Design of Experiments. Robert O. Kuehl. 2nd. Edition. 2001  run<- gl(10,3)  psi<- c(250,325,475,250,475,550,325,400,550,400,475,550,325,475,550,250,400,475,250,325,400,250,400,550,250,325,550,325,400,475)  monovinyl<- c(16,18,32,19,46,45,26,39,61,21,35,55,19,47,48,20,33,31,13,13,34,21, 30,52,24,10,50,24,31,37)  out<- BIB.test(run, psi, monovinyl, test="waller", group=FALSE)  print(out) |

\*\* (Optional, with RStudio) if you still have RStudio open, then copy the text into a window in RStudio and run it.

This is very simple – it is what RStudio is meant to do!

\*\* In R-Instat use the View menu and click to open the script file.

\*\* If the script file is not empty then right-click and use Clear Script.

\*\* Paste the commands above into the script file.

\*\* Check that the data on the lines starting psi and monovinyl are all on 1 line. If not, then delete the end of line so this is the case.

(RStudio has no problem executing R commands that extend over multiple lines. Currently R-Instat accepts one command per line.)

\*\* Click to run this set of commands.

It should run and provide some output Fig. 2.

|  |  |
| --- | --- |
| Fig. 2 | Fig. 3 |
|  |  |

\*\* In R-Instat delete the first (library) line. Instead change the line “out<- BIB.test(…” to

“out<- agricolae::BIB.test(…”, i.e. add the name of the package, followed by 2 colons.

\*\* Run the script window again. It should give the same result as before.

It would be good to see the data in a spreadsheet type window.

\*\* Add the line View(data.frame(run,psi,monovinyl)) to the bottom of the script window.

\*\* Right-click and choose Run Current Line.

It should show the 3 variables in the R-viewer. If not, then an error was made in the names of the variables, or the brackets, or perhaps view was typed (all in lower case) instead of View.

It would be good also to add a graph of the resulting model.

\*\* Add the line agricolae::plot.group(out)

The last lines of the script are shown in Fig. 3.

\*\* Select this new line, then right-click and choose Run Selected Text.

This gives a plot of the means as shown in Fig. 4.

|  |
| --- |
| Fig. 4 |
|  |

## Discussion on running this type of R-script

One reason for wanting to run this type of script in R-Instat would be to look at the resulting data in the R-Instat data window, and then perhaps to use other R-Instat dialogs on these same data. This is only possible if the data are in an R-Instat data book, or are added to an existing book. This did not happen here. The data in the 3 columns are floating in R but are not usable by any other dialogs. Nor are they visible in the data window.

The running of the script in Section 6 may have appeared satisfactory, but it is not. If the results of section 6 are all that is needed, then it is much easier to use RStudio than R-Instat.

In RStudio the editor (the equivalent of the script window) is much more powerful and helps with the construction of the command lines. Like a phone, it includes auto-complete, so it is much simpler to construct or correct commands. Some commands are also easier. For example, the plot command above can be given simply as:

agricolae::plot(out), or just plot(out) if the agricolae package was loaded earlier.

## Putting data into an R-Instat data book

We use the same data as earlier, Section 6, with another “halfway” dialog.

\*\* Use File > New. Choose the first tab, which says Construct, Fig. 1.

\*\* Copy the 3 data lines from the script in Section 6 into the corresponding fields in the dialog, Fig. 2.

\*\* Copy the details of the description into the ***Comment field***, Fig. 2

\*\* Call the data frame ***BIB1***.

\*\* Press OK

|  |  |
| --- | --- |
| Fig. 1 | Fig. 2 |
|  |  |

The data for analysis are now in the data frame shown in Fig. 3.

|  |  |
| --- | --- |
| Fig. 3 | Fig. 4 |
|  |  |

The next step is the command from the script for the analysis. This uses yet another “halfway” dialog:

\*\* Use Model > Hypothesis Tests.

\*\* Tick the checkbox to include arguments and change the package to agricolae Fig. 4.

\*\* Press the BIB key, Fig. 4.

\*\* Complete the dialog as shown in Fig. 5, so it is consistent with the command given in the script file.

\*\* Click on Try

\*\* Check the box to save the object as out.

\*\* Click Ok.

|  |  |
| --- | --- |
| Fig. 5 | Fig. 6 |
|  |  |

## Example: Adding a new graph to R-Instat

R includes over 13,000 packages. In this last section we show how to include an analysis in R-Instat that is not currently provided as a menu option. One way is simply to do this analysis in RStudio, but we assume here you would prefer to remain in R-Instat.

The example is to produce an adjusted boxplot, i.e. one that is designed for skew data. We illustrate by looking at daily rainfall data. These are very skew and the ordinary boxplot is not particularly helpful.

The function is in the R package called robustbase and is called adjbox. It produces a graph, but not one through the ggplot2 system. The robustbase package is already used by R-Instat[[3]](#footnote-3) for some other functions, and hence is available.

We illustrate with climatic data from the R-Instat library.

\*\* Use File > Open from Library.

\*\* Choose the option to Load From Instat Collection and then press Browse.

\*\* Select the Climatic directory and the file called ***Dodoma.rds***

\*\* Click Ok on the Import Dataset dialog that has opened.

Fig. 1 shows the data file that has been opened. The rainfall variable is daily and is from 1935 onwards.

|  |  |
| --- | --- |
| Fig. 1 | Fig. 2 |
|  |  |

Our aim is to do an adjusted boxplot on the non-zero rainfall data. We start with an ordinary boxplot, which provides a similar set of commands. We then change the script to provide the adjusted boxplot.

The first task is to filter out the zero rainfalls.

\*\* Right-click in the name field and choose Filter, or use Prepare > Data Frame > Filter.

\*\* Choose the option to Define New Filter.

\*\* On the sub-dialog use the condition Rain > 0.85.

\*\* On returning to the main dialog, accept this condition.

The data should now look as shown in Fig. 2.

\*\* Use the menu item Describe > Specific > Boxplot.

|  |  |
| --- | --- |
| Fig. 3 Ordinary boxplot | Fig 4 Results |
| Dialog for Rain, by Month\_abbr with variable width. Put the red boxes round the 3 components. |  |

\*\* Complete the dialog as shown in Fig. 3.

The result is in Fig. 4. It is of some interest, but the ordinary boxplot is not intended for data that are as skew as these.

\*\* Check the script window is empty

\*\* Return to the dialog and press the Script button.

\*\*

|  |
| --- |
| Fig. 5 Script window for the boxplot commands |
| 1. Dodoma <- data\_book$get\_data\_frame(data\_name="Dodoma") 2. last\_graph <- ggplot2::ggplot(data=Dodoma, mapping=ggplot2::aes(y=Rain, x=month\_abbr)) + ggplot2::geom\_boxplot(varwidth=TRUE, outlier.colour="red") + theme\_grey() 3. data\_book$add\_graph(graph\_name="last\_graph", graph=last\_graph, data\_name="Dodoma") 4. data\_book$get\_graphs(data\_name="Dodoma", graph\_name="last\_graph") 5. rm(list=c("last\_graph", "Dodoma")) |

Line numbers have been added in Fig. 5, line 2 has to be changed. From the manual for the robustbase package we find that a possible command is:

last\_graph <- robustbase::adjbox(Rain ~ month\_abbr, data =Dodoma, col=”red”, varwidth=TRUE)

\*\* Make this the new line 2 in Fig 5.

\*\* Press Run All at the top of the script window.

You should get the graph shown in Fig. 6. The code has done more than this, because it has also saved the resulting object in the R-Instat data book.

\*\* Use Prepare > R Objects > View

\*\* Take the option to Print the last\_graph.

Part of the results are shown in Fig. 7.

|  |  |
| --- | --- |
| Fig. 6 | Fig. 7 |
|  |  |

They show there were just 3 outliers, 119.8, 41.7 and 90.4 in February, May and November.

1. You can choose Describe > Multivariate > Correlations again, but the toolbar has a shortcut to recall the last dialog, or any of the last 10 dialogs used. [↑](#footnote-ref-1)
2. The ifelse is very powerful, like the IFELSE function in Excel. That’s why we introduced it. But here just doing table == table0 gives the same answer. [↑](#footnote-ref-2)
3. If you wish to use a command from a package that is not currently loaded into R-Instat, then one way is to add the package in RStudio. That’s a simple operation. Then, start R-Instat again and the package will also be available in R-Instat. [↑](#footnote-ref-3)