

# Reading, tweaking and using R commands

Draft: 18/02/2019

1) Introduction .....	1
2) Looking at commands and tweaking them .....	1
3) Examining and using the log file.....	7
4) The data book and other R objects .....	9
5) Using R-Instat's calculator: a "halfway" dialogue.....	11
6) Running a simple R script .....	15
7) Discussion on running this type of R-script .....	17
8) Putting data into an R-Instat data book .....	17
9) Example: Adding a new graph to R-Instat .....	19

## 1) 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, it is easier to read than to write. Being able to read some R is useful. It also opens the door to being able to adapt, (or tweak!) the 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.

## 2) Looking at commands and tweaking them

We use the same data and part of the analysis from the first R-Instat 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.

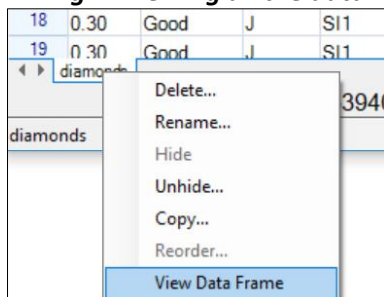
**Fig. 1 The diamonds data**

Data View										
	carat	cut (o.f)	color (o.f)	clarity (o.f)	depth	table	price	x	y	z
1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
6	0.24	Very Good	J	VVS2	62.8	57.0	336	3.94	3.96	2.48
7	0.24	Very Good	I	VVS1	62.3	57.0	336	3.95	3.98	2.47
8	0.26	Very Good	H	SI1	61.9	55.0	337	4.07	4.11	2.53
9	0.22	Fair	E	VS2	65.1	61.0	337	3.87	3.78	2.49
10	0.23	Very Good	H	VS1	59.4	61.0	338	4.00	4.05	2.39
11	0.30	Good	J	SI1	64.0	55.0	339	4.25	4.28	2.73
12	0.23	Ideal	J	VS1	62.8	56.0	340	3.93	3.90	2.46
13	0.22	Premium	F	SI1	60.4	61.0	342	3.88	3.84	2.33
14	0.31	Ideal	J	SI2	62.2	54.0	344	4.35	4.37	2.71
15	0.20	Premium	E	SI2	60.2	62.0	345	3.79	3.75	2.27
16	0.32	Premium	E	I1	60.9	58.0	345	4.38	4.42	2.68
17	0.30	Ideal	I	SI2	62.0	54.0	348	4.31	4.34	2.68
18	0.30	Good	J	SI1	63.4	54.0	351	4.23	4.29	2.70
19	0.30	Good	J	SI1	63.8	56.0	351	4.23	4.26	2.71

Showing 1000 of 53940 rows | Showing 10 of 10 columns

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**

	carat	cut	color	clarity	depth	table	price	x	y	z
53922	0.70	Very Good	E	VS2	62.8	60.0	2755	5.59	5.65	3.53
53923	0.70	Very Good	D	VS1	63.1	59.0	2755	5.67	5.58	3.55
53924	0.73	Ideal	I	VS2	61.3	56.0	2756	5.80	5.84	3.57
53925	0.73	Ideal	I	VS2	61.6	55.0	2756	5.82	5.84	3.59
53926	0.79	Ideal	I	SI1	61.6	56.0	2756	5.95	5.97	3.67
53927	0.71	Ideal	E	SI1	61.9	56.0	2756	5.71	5.73	3.54
53928	0.79	Good	F	SI1	58.1	59.0	2756	6.06	6.13	3.54
53929	0.79	Premium	E	SI2	61.4	58.0	2756	6.03	5.96	3.68
53930	0.71	Ideal	G	VS1	61.4	56.0	2756	5.76	5.73	3.53
53931	0.71	Premium	E	SI1	60.5	55.0	2756	5.79	5.74	3.49
53932	0.71	Premium	F	SI1	59.8	62.0	2756	5.74	5.73	3.43
53933	0.70	Very Good	E	VS2	60.5	59.0	2757	5.71	5.76	3.47
53934	0.70	Very Good	E	VS2	61.2	59.0	2757	5.69	5.72	3.49
53935	0.72	Premium	D	SI1	62.7	59.0	2757	5.69	5.73	3.58
53936	0.72	Ideal	D	SI1	60.8	57.0	2757	5.75	5.76	3.50
53937	0.72	Good	D	SI1	63.1	55.0	2757	5.69	5.75	3.61
53938	0.70	Very Good	D	SI1	62.8	60.0	2757	5.66	5.68	3.56
53939	0.86	Premium	H	SI2	61.0	58.0	2757	6.15	6.12	3.74
53940	0.75	Ideal	D	SI2	62.2	55.0	2757	5.83	5.87	3.64

\*\* 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**

```

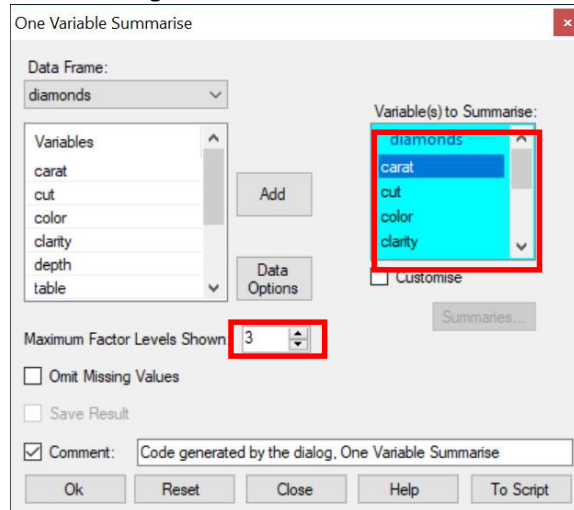
Output Window
# Code generated by the dialog, Open Dataset from Library
utils::data(package="ggplot2", X=diamonds)

diamonds <- diamonds
InstatDataObject$import_data(data_tables=list(diamonds=diamonds))

rm(diamonds)

# Right Click Menu: View R Data Frame
View(title="diamonds", x=InstatDataObject$get_data_frame(data_name="diamonds"))
    
```

**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 dialogue:

- \*\* 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 dialogue 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 dialogue, i.e. to the dialogue in Fig. 5.
- \*\* Click on the **To Script** button at the bottom of the dialogue.

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**

```

# Code generated by the dialog, One Variable Summarise
summary(object=InstatDataObject$get_columns_from_data(data_name="diamonds", col_names=c(
"carat","cut","color","clarity","depth","table","price","x","y","z")), na.rm=FALSE, maxsum=3)

  carat      cut      color      clarity      depth
Min.   :0.200  Ideal :I1551  G       :I1292  SI1     :I3069  Min.   :43.0
1st Qu.:0.400  Premium:13791 E       : 9787  VS2     :12259  1st Qu.:61.0
Median :0.700  (Other):18598 (Other):32851 (Other):28617 Median :61.8
Mean   :0.798                                     Mean   :61.8
3rd Qu.:1.040                                     3rd Qu.:62.5
Max.   :5.010                                     Max.   :79.0

  table      price      x      y      z
Min.   :43.0  Min.   : 326  Min.   : 0.00  Min.   : 0.00  Min.   : 0.00
1st Qu.:56.0  1st Qu.: 950  1st Qu.: 4.71  1st Qu.: 4.72  1st Qu.: 2.91
Median :57.0  Median :2401  Median : 5.70  Median : 5.71  Median : 3.53
Mean   :57.5  Mean   :3933  Mean   : 5.73  Mean   : 5.73  Mean   : 3.54
3rd Qu.:59.0  3rd Qu.:5324  3rd Qu.: 6.54  3rd Qu.: 6.54  3rd Qu.: 4.04
Max.   :95.0  Max.  :18823  Max.   :10.74  Max.   :58.90  Max.   :31.80
    
```

**Fig. 7 The script window**

```

Script
Run All

names=c("carat","cut","color","clarity","depth","table","price","x","y","z"), na.rm=FALSE, maxsum=10)
    
```

- \*\* 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.

\*\* 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**.

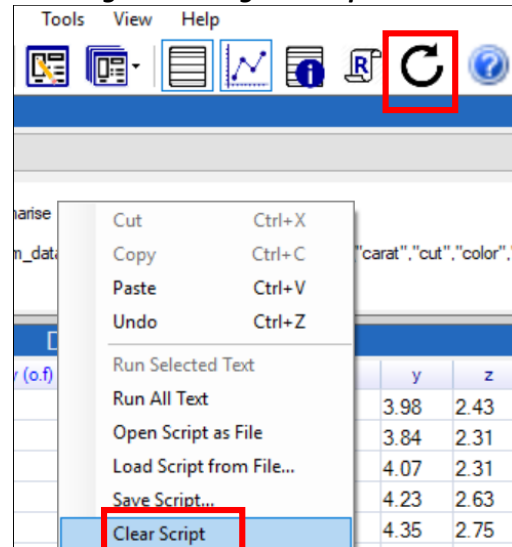
**Fig. 8 Result after making the change**

```
# Code run from Script Window
summary(object=InstatDataObject$get_columns_from_data(data_name="diamonds", column_names=c(
"carat","cut","color","clarity","depth","table","price","x","y","z")), na.rm=FALSE, maxsum=10)
```

carat	cut	color	clarity	depth
Min. :0.200	Fair : 1610	D: 6775	I1 : 741	Min. :43.0
1st Qu.:0.400	Good : 4906	E: 9797	SI2 : 9194	1st Qu.:61.0
Median :0.700	Very Good:12082	F: 9542	SI1 :13065	Median :61.8
Mean :0.798	Premium :13791	G:11292	VS2 :12258	Mean :61.8
3rd Qu.:1.040	Ideal : 21551	H: 8304	VS1 : 8171	3rd Qu.:62.5
Max. :5.010		I: 5422	VVS2: 5066	Max. :79.0
		J: 2808	VVS1: 3655	
			IF : 1790	

table	price	x	y	z
Min. :43.0	Min. : 326	Min. : 0.00	Min. : 0.00	Min. : 0.00
1st Qu.:56.0	1st Qu.: 950	1st Qu.: 4.71	1st Qu.: 4.72	1st Qu.: 2.91
Median :57.0	Median : 2401	Median : 5.70	Median : 5.71	Median : 3.53
Mean :57.5	Mean : 3933	Mean : 5.73	Mean : 5.73	Mean : 3.54
3rd Qu.:59.0	3rd Qu.: 5324	3rd Qu.: 6.54	3rd Qu.: 6.54	3rd Qu.: 4.04
Max. :95.0	Max. :18823	Max. :10.74	Max. :10.74	Max. :31.80

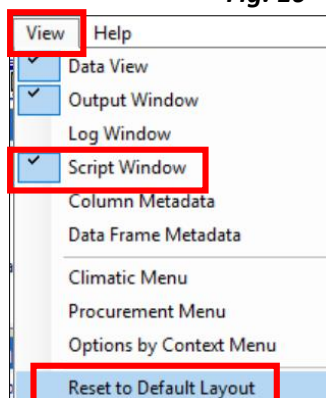
**Fig. 9 Clearing the script window**



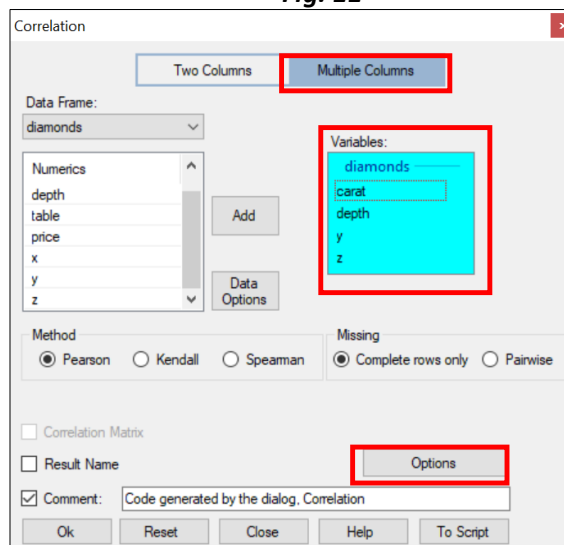
\*\* Now **press the curly arrow**, Fig. 9 or use **View > Reset to Default Layout**, Fig. 10. to return to the default layout of the windows. This closes the script window.

You could have made this change, from 3 to 10, more simply in the dialogue, 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 dialogue, 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-dialogue.

\*\* 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**

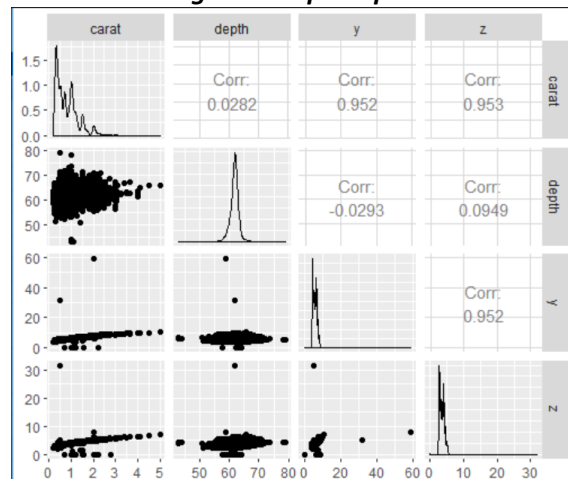
```

Output Window
# Code generated by the dialog, Correlation
last_model <- cor(x=InstatDataObject$get_columns_from_data(data_name="diamonds",
col_names=c("carat","depth","y","z")), use="complete.obs")
InstatDataObject$add_model(data_name="diamonds", model=last_model,
model_name="last_model")
InstatDataObject$get_models(model_name="last_model", data_name="diamonds")

  carat   depth     y     z
carat 1.00000 0.02822 0.95172 0.95339
depth 0.02822 1.00000 -0.02934 0.09492
y      0.95172 -0.02934 1.00000 0.95201
z      0.95339 0.09492 0.95201 1.00000

diamonds <- InstatDataObject$get_data_frame(data_name="diamonds", remove_attr=TRUE)
last_graph <- GGally::ggpairs(data=diamonds, columns=c("carat","depth","y","z"), upper=list
(continuous=GGally::wrap("points")), lower=list(continuous=GGally::wrap("points")))
InstatDataObject$add_graph(data_name="diamonds", graph=last_graph,
graph_name="last_graph")
InstatDataObject$get_graphs(graph_name="last_graph", data_name="diamonds")
  
```

**Fig. 13 The pairs plot**



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

\*\* **Return to the last dialogue**<sup>1</sup> and press the **To Script** button.

This is a general feature in R-Instat: Every dialogue 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**

```

# Code generated by the dialog, Correlation
last_model <- cor(x=InstatDataObject$get_columns_from_data(data_name="diamonds",
col_names=c("carat","depth","y","z")), use="complete.obs")
InstatDataObject$add_model(data_name="diamonds", model=last_model, model_name="last_model")
InstatDataObject$get_models(model_name="last_model", data_name="diamonds")

diamonds <- InstatDataObject$get_data_frame(data_name="diamonds", remove_attr=TRUE)
last_graph <- GGally::ggpairs(data=diamonds, columns=c("cut","color","carat",
  
```

**Fig. 15 Running the selected commands**

```

# Code generated by the dialog, Correlation
last_model <- cor(x=InstatDataObject$get_columns_from_data(data_name="diamonds", col_names=c("carat","depth","y","z")), use="complete.obs")
InstatDataObject$add_model(data_name="diamonds", model=last_model, model_name="last_model")
InstatDataObject$get_models(model_name="last_model", data_name="diamonds")

diamonds <- InstatDataObject$get_data_frame(data_name="diamonds", remove_attr=TRUE)
last_graph <- GGally::ggpairs(data=diamonds, columns=c("cut","color","carat","depth","y","z"), lower=list(
continuous=GGally::wrap("points")), upper=list(continuous=GGally::wrap("points")))
InstatDataObject$add_graph(data_name="diamonds", graph=last_graph, graph_name="last_graph")
InstatDataObject$get_graphs(graph_name="last_graph", data_name="diamonds")
  
```

In contrast, the correlation command, **cor**, which is from the R stats package is given simply as **cor**, i.e. not as **stats::cor**. That's because the stats package is produced by the core R team. It is loaded and available whenever R (or R-Instat) is loaded. Other packages are only loaded by R-Instat when they are needed. So the package name in **GGally::ggpairs** includes the name of the R package.

Now we "tweak" the ggpairs command. The correlations dialogue in R-Instat 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.

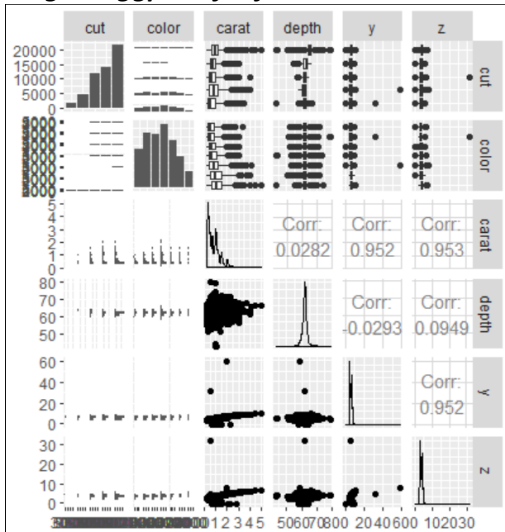
\*\* Do this very carefully, so it now reads ... columns=c("cut","color","carat","depth","y","z").

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

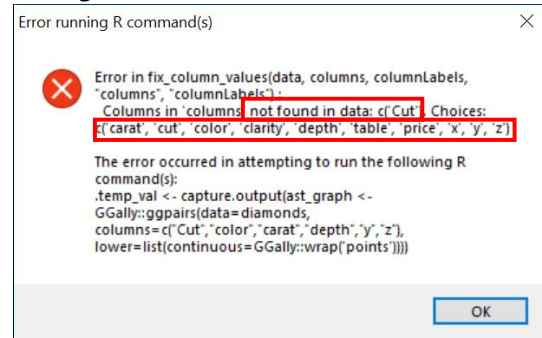
You should now get the results as shown in Fig. 16.

<sup>1</sup> You can choose **Describe > Multivariate > Correlations** again, but the toolbar has a shortcut to recall the last dialogue, or any of the last 10 dialogues used.

**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 deliberate mistakes and run the one line again to see the error messages. If you are going to use a script file, or R commands in any way, then (unless you never make mistakes!) 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*.

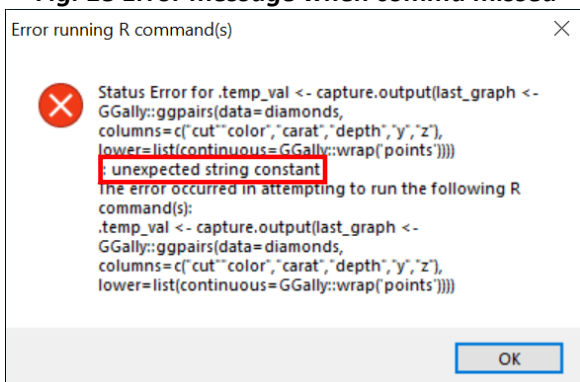
The error message is in Fig. 17. R no longer recognises the column name. It also provides you with a list of the names it does recognise, to help with the correction.

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

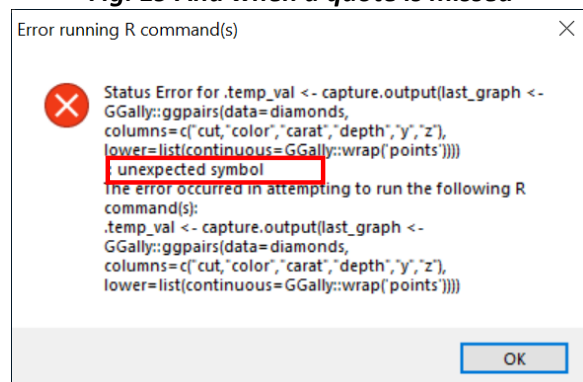
\*\* Now make another mistake, by *omitting a comma*, i.e. change “cut”, “color” to “cut””color”.

\*\* *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 messages are sensible, though that is not always the case. If you use commands, then you will have to get used to looking at error messages and using them to help make corrections.

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

### 3) Examining and using the log file

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

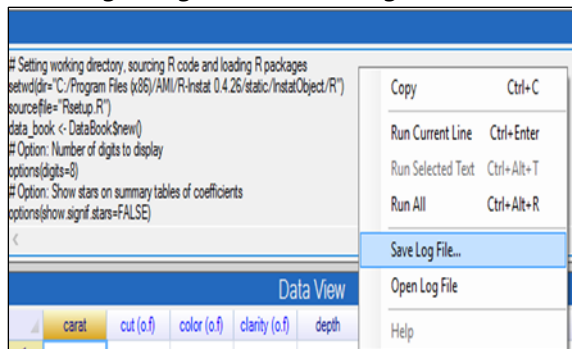
**\*\* Click the scroll symbol on the toolbar, to open the log window.**

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

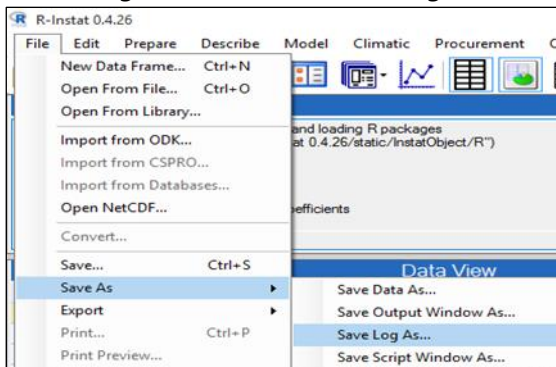
**\*\* Go to the *View menu*, as shown in Fig. 10 in Section 2. There is a tick beside the Log Window, signifying it is open. Click to close the Log window. This action also closes the View menu.**

**\*\* 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 File>Save As>Save Log As**

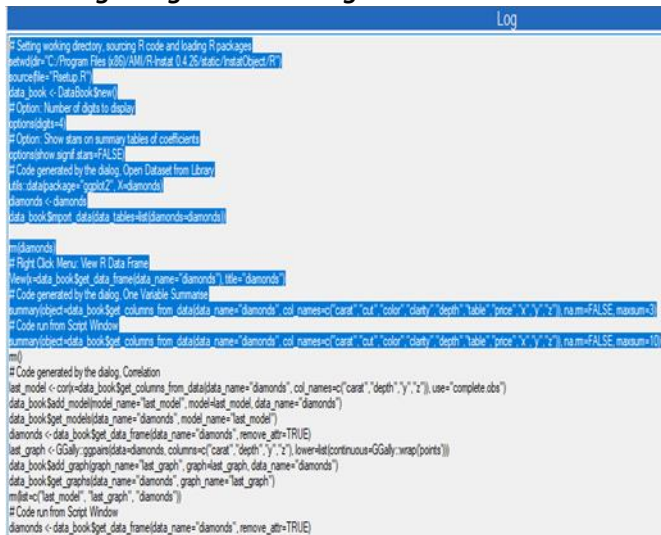


The log file is a record of all the commands in an R-Instat session. As such you may want to save this file. This is either through the right-click menu, Fig. 1, or using **File > Save As > Save Log As**, Fig. 2.

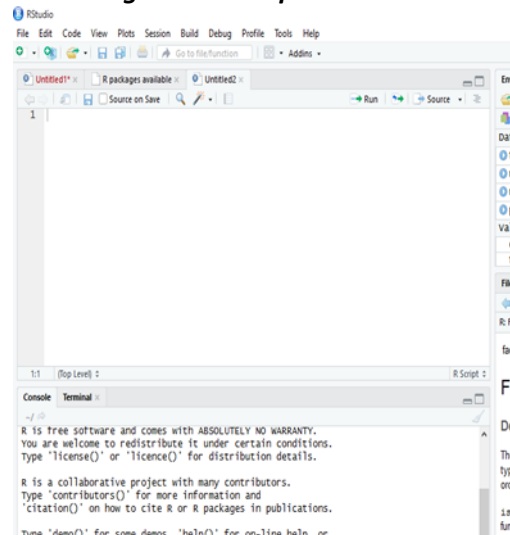
Once you become more skilled in using R you may also sometimes start an analysis in R-Instat and then continue with RStudio.

If you do not have RStudio installed, then this is just for you to read.

**Fig. 3 Fig. 3 Code in Log Window in R-Instat**



**Fig. 4 New Script in RStudio**



**\*\* 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 dialogue has a Comment field. When you use a dialogue, you may accept the default comment or type your own more meaningful comments. That makes the log file clearer.

- \*\* **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 R-Instat Code in Rstudio Source Window**

```

1 | Setting working directory, sourcing R code and loading R packages
2 | setwd(dir="c:/Program Files (x86)/AMI/R-Instat 0.4.26/static/Instatobject/R")
3 | source(file="Rsetup.R")
4 | data_book <- DataBook$new()
5 | # Option: Number of digits to display
6 | options(digits=4)
7 | # Option: Show stars on summary tables of coefficients
8 | options(show.signif.stars=FALSE)
9 | # Code generated by the dialog, Open Dataset from Library
10 | utils::data(package="ggplot2", x=diamonds)
11 | diamonds <- diamonds
12 | data_book$import_data(data_tables=list(diamonds=diamonds))
13 |
14 | rm(diamonds)
15 | # Right Click Menu: View R Data Frame
16 | View(x=data_book$get_data_frame(data_name="diamonds"), title="diamonds")

```

**Fig. 6 Results of the highlighted code**

	carat	cut	color	clarity	depth	table	price	x	y	z
1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
3	0.23	Good	E	VSI1	56.9	65.0	327	4.05	4.07	2.31
4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
6	0.24	Very Good	J	VVS2	62.8	57.0	336	3.94	3.96	2.48
7	0.24	Very Good	I	VVS1	62.3	57.0	336	3.95	3.98	2.47
8	0.26	Very Good	H	SI1	61.9	55.0	337	4.07	4.11	2.53
9	0.22	Fair	E	VS2	65.1	61.0	337	3.87	3.78	2.49
10	0.23	Very Good	H	VSI1	59.4	61.0	338	4.00	4.05	2.39
11	0.30	Good	J	SI1	64.0	55.0	339	4.25	4.28	2.73
12	0.23	Ideal	J	VSI1	62.8	56.0	340	3.93	3.90	2.46
13	0.22	Premium	F	SI1	60.4	61.0	342	3.88	3.84	2.33
14	0.31	Ideal	J	SI2	62.2	54.0	344	4.35	4.37	2.71
15	0.20	Premium	E	SI2	60.2	62.0	345	3.79	3.75	2.27
16	0.32	Premium	E	I1	60.9	58.0	345	4.38	4.42	2.68
17	0.30	Ideal	I	SI2	62.0	54.0	348	4.31	4.34	2.68

- \*\* 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 Remainder of the code**

```

# Code generated by the dialog, One Variable Summarise
summary(object=data_book$get_columns_from_data(data_name="diamonds", col_names=c("carat", "c
# Code run from script window
summary(object=data_book$get_columns_from_data(data_name="diamonds", col_names=c("carat", "c

```

**Fig. 8 Results**

```

carat      cut      color      clarity      depth
Min.   :0.200  Ideal :21551  G       :11292  SI1    :13065  Min.   :43.0
1st Qu.:0.400  Premium:13791 E       : 9797  VS2    :12258  1st Qu.:61.0
Median :0.700  (Other):18598 (Other):32851 (Other):28617 Median :61.8
Mean   :0.798
3rd Qu.:1.040
Max.   :5.010

table      price      x      y      z
Min.   :43.0  Min.   : 326  Min.   : 0.00  Min.   : 0.00  Min.   : 0.00
1st Qu.:56.0  1st Qu.: 950  1st Qu.: 4.71  1st Qu.: 4.72  1st Qu.: 2.91
Median :57.0  Median :2401  Median : 5.70  Median : 5.71  Median : 3.53
Mean   :57.5  Mean   :3933  Mean   : 5.73  Mean   : 5.73  Mean   : 3.54
3rd Qu.:59.0  3rd Qu.:5324  3rd Qu.: 6.54  3rd Qu.: 6.54  3rd Qu.: 4.04
Max.   :95.0  Max.  :18823  Max.  :10.74  Max.  :10.74  Max.  :131.80

# Code run from script window
summary(object=data_book$get_columns_from_data(data_name="diamonds", col_names=c("carat", "c
carat      cut      color      clarity      depth      table
Min.   :0.200  Fair   : 1610  D: 6775  I1   : 741  Min.   :43.0  Min.   :43.0
1st Qu.:0.400  Good   : 4906  E: 9797  SI2  :9194  1st Qu.:61.0  1st Qu.:56.0
Median :0.700  Very Good:12082 F: 9342  SI1  :13065 Median :61.8  Median :57.0
Mean   :0.798  Premium :13791 G:11292 VS2  :12258 Mean   :61.8  Mean   :57.5
3rd Qu.:1.040  Ideal  :21551 H: 8304  VSI1 :8171  3rd Qu.:62.5  3rd Qu.:59.0
Max.   :5.010          I: 5422  VVS2 :5066 Max.   :79.0  Max.   :95.0
          J: 2808  VS1  :3655
          IF :1790

price      x      y      z
Min.   : 326  Min.   : 0.00  Min.   : 0.00  Min.   : 0.00
1st Qu.: 950  1st Qu.: 4.71  1st Qu.: 4.72  1st Qu.: 2.91
Median :2401  Median : 5.70  Median : 5.71  Median : 3.53
Mean   :3933  Mean   : 5.73  Mean   : 5.73  Mean   : 3.54
3rd Qu.:5324  3rd Qu.: 6.54  3rd Qu.: 6.54  3rd Qu.: 4.04
Max.  :18823  Max.  :10.74  Max.  :10.74  Max.  :131.80

```

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



#### 4) The data book and other R objects

\*\* Now return to R-Instat

\*\* Click on the **curly 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, and so on. 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.

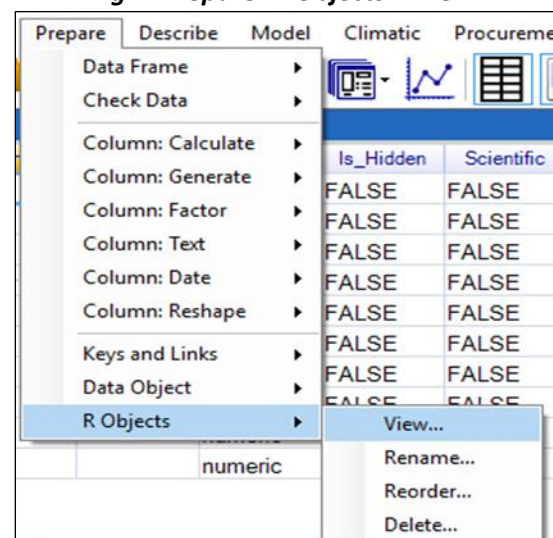
\*\* Use **View > Column Metadata** (or **click on the i symbol** in the toolbar).

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

**Fig. 1 Column Metadata**

	Name	label	class	Is_Hidden	Scientific	Signif_Figure
1	carat		numeric	FALSE	FALSE	3
2	cut		ordered, fact	FALSE	FALSE	NA
3	color		ordered, fact	FALSE	FALSE	NA
4	clarity		ordered, fact	FALSE	FALSE	NA
5	depth		numeric	FALSE	FALSE	3
6	table		numeric	FALSE	FALSE	3
7	price		integer	FALSE	FALSE	3
8	x		numeric	FALSE	FALSE	3
9	y		numeric	FALSE	FALSE	3
10	z		numeric	FALSE	FALSE	3

**Fig. 2 Prepare>R Objects > View**



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

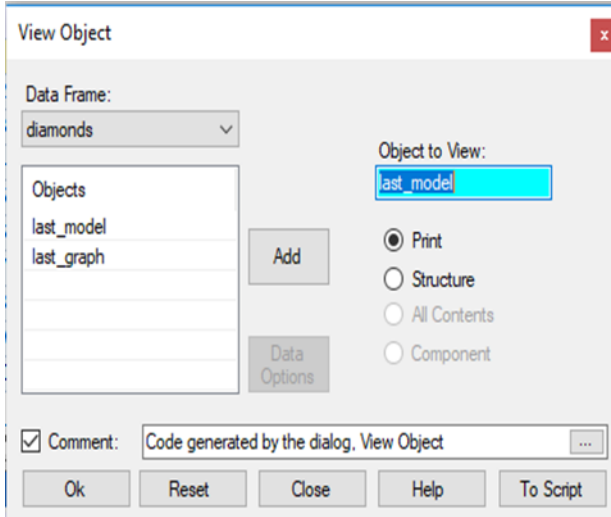
This View dialogue is in Fig. 3. It shows there are 2 objects, called last\_model and last\_graph. They came from the correlations dialogue that produced both some summaries and a graph.

This is a general feature of R-Instat. Every dialogue 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.

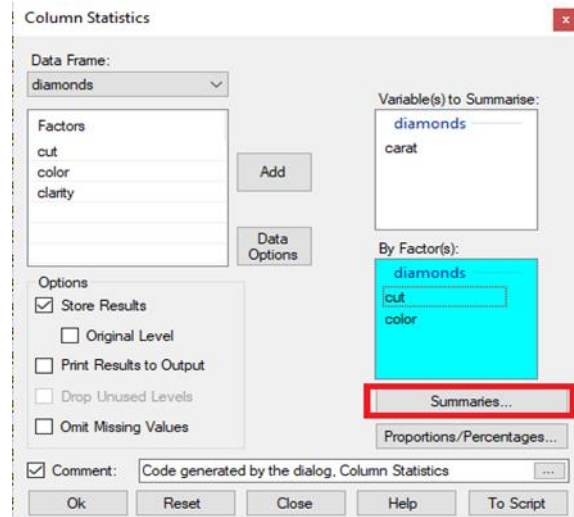
The same is true for all the dialogues that produce a model or summary.

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

**Fig. 3 View Object Dialog**



**Fig. 4 Column Summaries dialog**



In R-Instat these objects are also part of the metadata associated with the corresponding data frame. They form part of the Instat data object. An **Instat data sheet** is a data frame with (often) quite a lot of metadata.

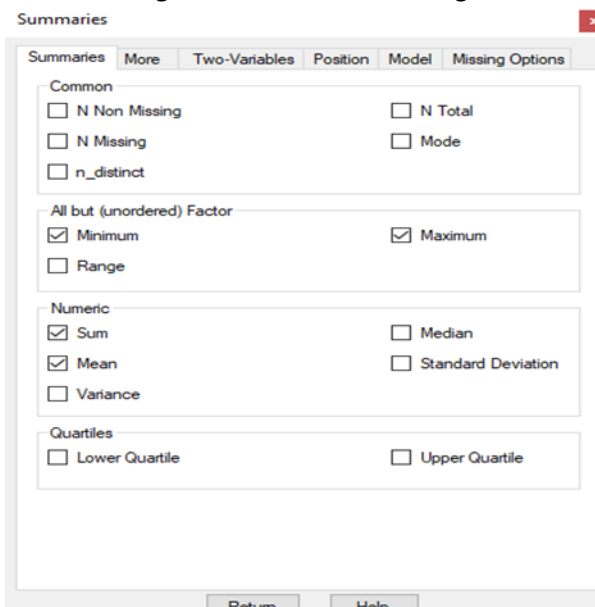
We now produce a second, summary data frame

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

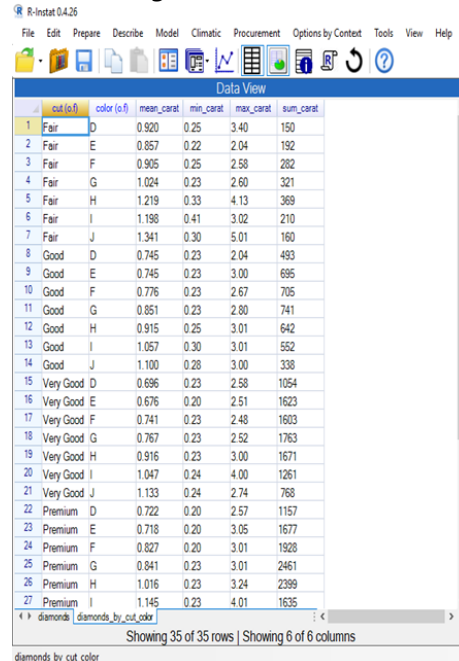
We choose to summarise a numeric column, carat, by the 2 factors called cut and color.

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

**Fig. 5 Summaries Subdialog**



**Fig. 6 Diamonds Summaries**



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

\*\* Press **Return** to go back to the main dialogue 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 dialogue 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. A **data book** is a collection of **data sheets**. Usually it is a set of linked data frames.

The **data book** is “behind the scenes” in almost all the R-Instat dialogues. 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.

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

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

Fig. 7 shows the code from the script window

**Fig 7 Script for correlations**

```
# Code generated by the dialog, Correlation
1. last_model <- cor(x=data_book$get_columns_from_data(data_name="diamonds", col_names=c("carat","depth","y","z")),
  use="complete.obs")
2. data_book$add_model(model_name="last_model", model=last_model, data_name="diamonds")
3. data_book$get_models(data_name="diamonds", model_name="last_model")
4. rm(last_model)
```

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 R-Instat 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**.

## 5) Using R-Instat’s calculator: a “halfway” dialogue

Most of R-Instat’s dialogues 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.

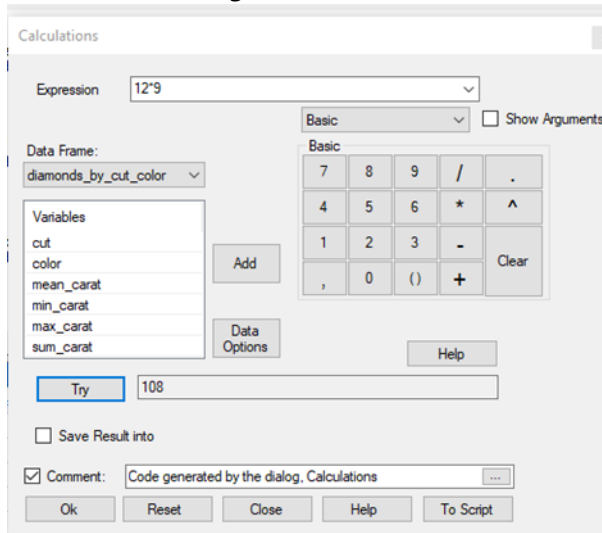
Each R-Instat dialogue produces one, or more, R commands. The commands are usually included in the output window. But if you prefer, **right-click in the Output window** and choose to **Hide (Future) Commands**. Then you are not reminded that the R statistical package is behind it all!

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

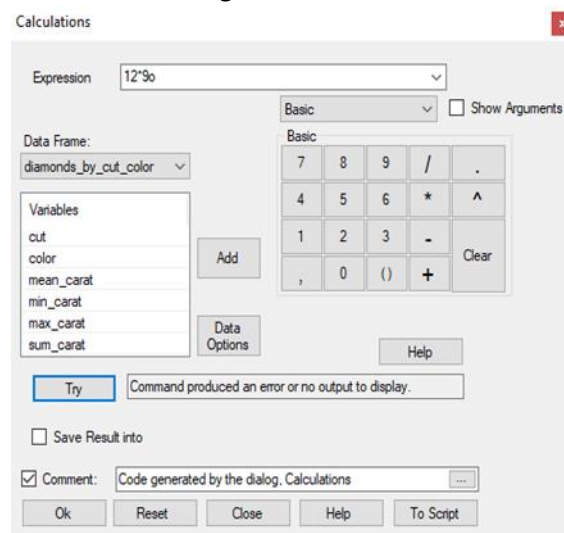
A few R-Instat dialogues do not protect you quite so much from R. We call them “halfway” dialogues. They involve you, with assistance, typing a single R command (or part of a command). We first introduce the most used halfway command, namely R-Instat’s calculator.

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

**Fig. 1 Calculator**



**Fig. 2 Calculator**



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 dialogues are very powerful – because R is so powerful. But they are also risky!

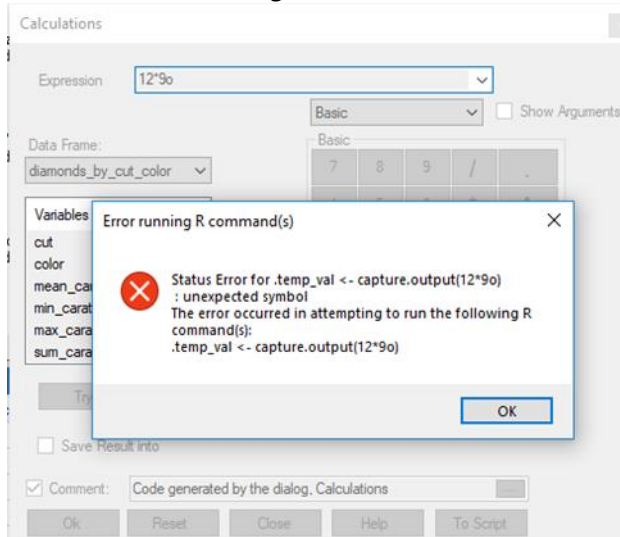
\*\* Add a **zero**, so it is **12 \* 90** and press **Try** again.

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

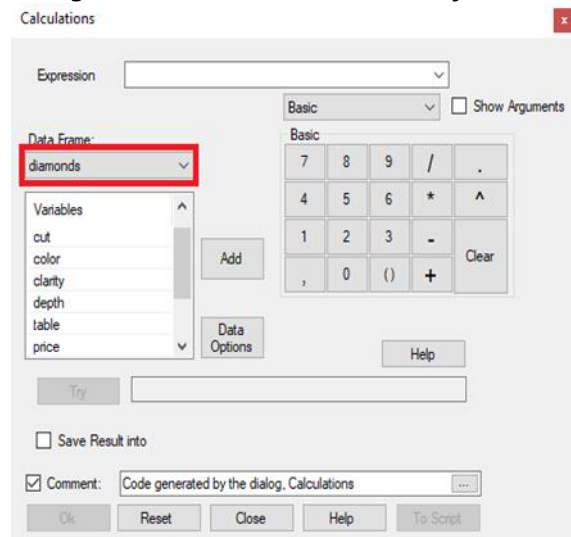
It now says this command produced an error – which is true. But **Ok** is still enabled.

\*\* Press **Ok**

**Fig. 3 Error**



**Fig. 4 Choose the Diamonds data frame**



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

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

The positive side is partly the power of these halfway dialogues. 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 dialogue.

\*\* Check the **diamonds** data frame is the active one in the dialogue, 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 R commands from the calculator**

1. pricepc <- price/carat
2. data\_book\$add\_columns\_to\_data(col\_data=pricepc, col\_name="pricepc", data\_name="diamonds")
3. rm(pricepc)

Line 1 does the calculation.

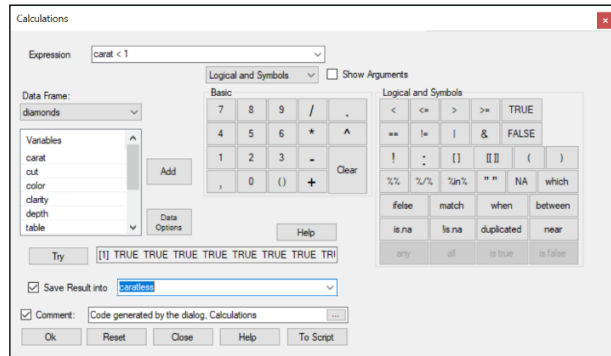
Line 2 puts the new column into the R-Instat data book, and it is then automatically shown in the data frame, Fig. 6

Line 3 is the usual housekeeping and removes the new column from R's memory.

**Fig. 6 pricepc column**

(o.f)	depth	table	price	x	y	z	pricepc
1	61.5	55.0	326	3.95	3.98	2.43	1417
2	59.8	61.0	326	3.89	3.84	2.31	1552
3	56.9	65.0	327	4.05	4.07	2.31	1422
4	62.4	58.0	334	4.20	4.23	2.63	1152
5	63.3	58.0	335	4.34	4.35	2.75	1081
6	62.8	57.0	336	3.94	3.96	2.48	1400
7	62.3	57.0	336	3.95	3.98	2.47	1400
8	61.9	55.0	337	4.07	4.11	2.53	1296
9	65.1	61.0	337	3.87	3.78	2.49	1532
10	59.4	61.0	338	4.00	4.05	2.39	1470
11	64.0	55.0	339	4.25	4.28	2.73	1130
12	62.8	56.0	340	3.93	3.90	2.46	1478
13	60.4	61.0	342	3.88	3.84	2.33	1555
14	62.2	54.0	344	4.35	4.37	2.71	1110
15	60.2	62.0	345	3.79	3.75	2.27	1725
16	60.9	58.0	345	4.38	4.42	2.68	1078
17	62.0	54.0	348	4.31	4.34	2.68	1160
18	63.4	54.0	351	4.23	4.29	2.70	1170
19	63.8	56.0	351	4.23	4.26	2.71	1170
20	62.7	59.0	351	4.21	4.27	2.66	1170
21	63.3	56.0	351	4.26	4.30	2.71	1170

**Fig. 7 Logical Calculator**



This last example has started to show a little of R's power. The 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 dialogues. 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.

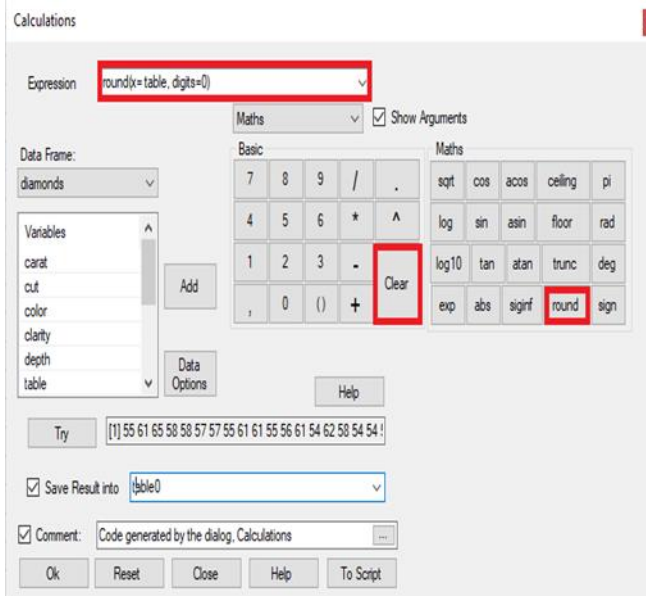
\*\* Press **Ok**.

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

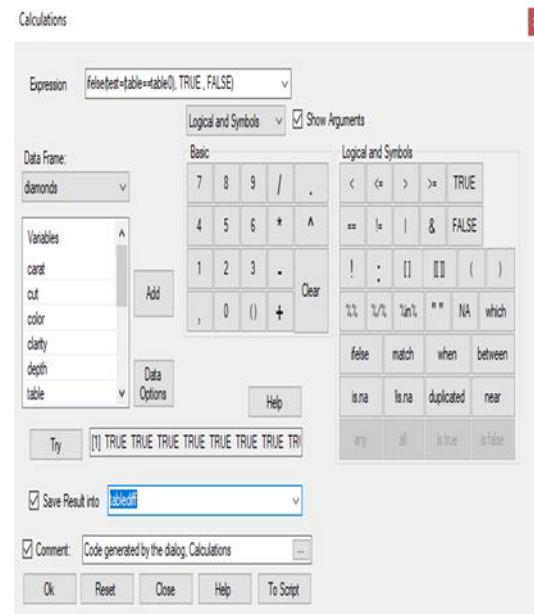
\*\* **Return to the calculator** and press **Clear**. Go back to the **logical and symbols keyboard**, Fig 9, and press the **Show Arguments** checkbox.

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

**Fig. 8 Rounding off**



**Fig. 9 ifelse**



\*\* 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** dialogue again to show there are only 924 cases that are FALSE.

So, the variable called **table** was given without a decimal on 98% of the rows of data. 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.<sup>2</sup>

## 6) 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** dialogue 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 particular function by seeing the results from an example of the analysis – though that is not our 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**.

<sup>2</sup> 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.

**Fig. 1 R commands for a BIB analysis from the agricolae package**

```
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)
monovinyll<- 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, monovinyll, 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 monovinyll 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 expects one command per line.)

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

It should run and provide some output as shown in Fig. 2.

**Fig. 2 The Output**

```
$parameters
lambda treatmeans blockSize blocks r alpha test
3          5          3          10 6 0.05 BIB

$statistics
Mean Efficiency CV
31.67      0.8333 17.54

$comparison
Difference significant
250 - 325      2.933      FALSE
250 - 400     -10.400      TRUE
250 - 475     -18.333      TRUE
250 - 550     -30.200      TRUE
325 - 400     -13.333      TRUE
325 - 475     -21.267      TRUE
325 - 550     -33.133      TRUE
400 - 475      -7.933      TRUE
400 - 550     -19.800      TRUE
475 - 550     -11.867      TRUE

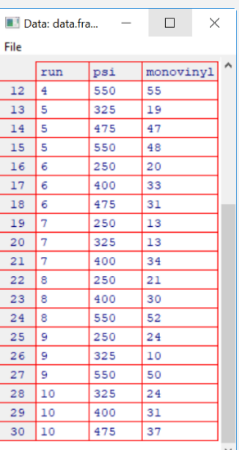
$means
monovinyll mean.adj SE r std Min Max Q25 Q50 Q75
250      18.83      20.47 2.442 6 3.869 13 24 14.75 19.5 20.75
325      18.33      17.53 2.442 6 6.154 10 26 14.25 18.5 22.75
400      31.33      30.87 2.442 6 5.955 21 39 30.25 32.0 33.75
475      38.00      38.80 2.442 6 6.928 31 47 32.75 36.0 43.75
550      51.83      50.67 2.442 6 5.636 45 61 48.50 51.0 54.25

$groups
NULL

attr(,"class")
[1] "group"
```

**Fig. 3 Additional commands with View of the data**

```
View(data.frame(run,psi,monovinyll))
agricolae::bar.err(out$means,variation="range",ylim=c(0,60),bar=FALSE,col=0)
```



	run	psi	monovinyll
12	4	550	55
13	5	325	19
14	5	475	47
15	5	550	48
16	6	250	20
17	6	400	33
18	6	475	31
19	7	250	13
20	7	325	13
21	7	400	34
22	8	250	21
23	8	400	30
24	8	550	52
25	9	250	24
26	9	325	10
27	9	550	50
28	10	325	24
29	10	400	31
30	10	475	37

\*\* 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,monovinyll))** 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.

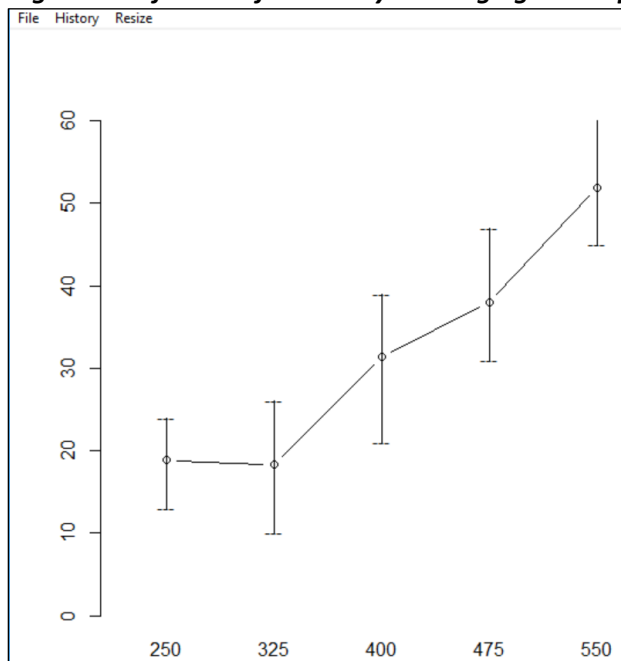
**\*\* Add** the line `agricolae::bar.err(out$means,variation="range",ylim=c(0,60),bar=FALSE,col=0)`

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

**\*\* Select this new line**, then either **right-click** and choose **Run Current Line**, or press <Ctrl><Enter>.

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

**Fig. 4 Plot of means from analysis using agricolae package**



## 7) 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 dialogues 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 with the script above. The data in the 3 columns are floating in R but are not usable by any other dialogues. Nor are they visible in the R-Instat data window.

The running of the script in Section 6 may have seemed 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 simpler to construct or correct commands. Some commands are also easier.

## 8) Putting data into an R-Instat data book

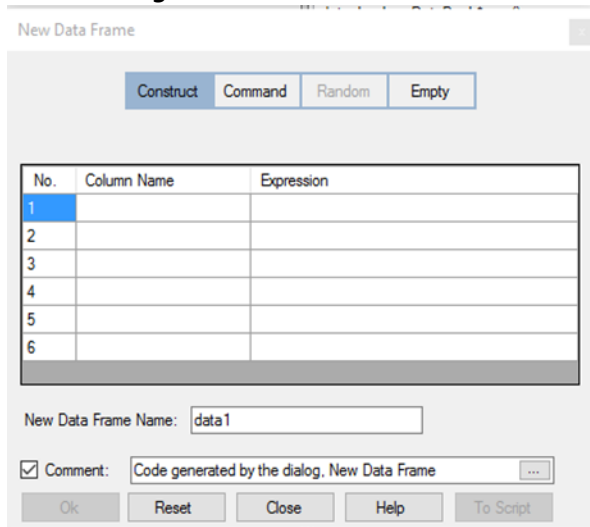
We use the same data as earlier, Section 6, with another “halfway” dialogue.

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

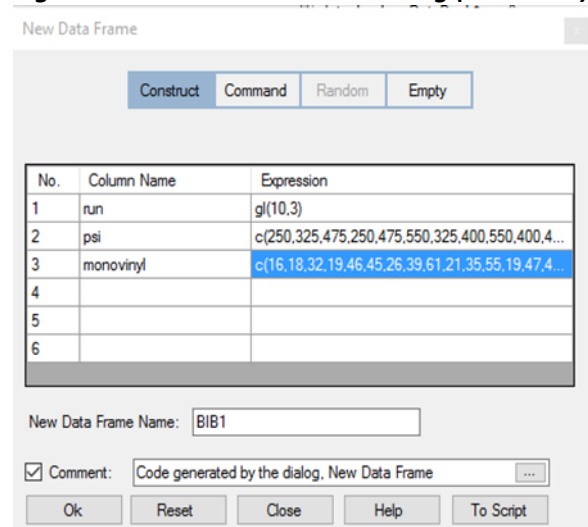
**\*\* Copy the 3 data lines** from the script in Section 6 into the corresponding fields in the dialogue, Fig. 2.

- \*\* Copy the details of the description into the **Comment field**, Fig. 2
- \*\* Call the data frame **BIB1**.
- \*\* Press OK

**Fig. 1 File > New Data Frame**



**Fig. 2 Filled New Data Frame Dialog (Construct)**

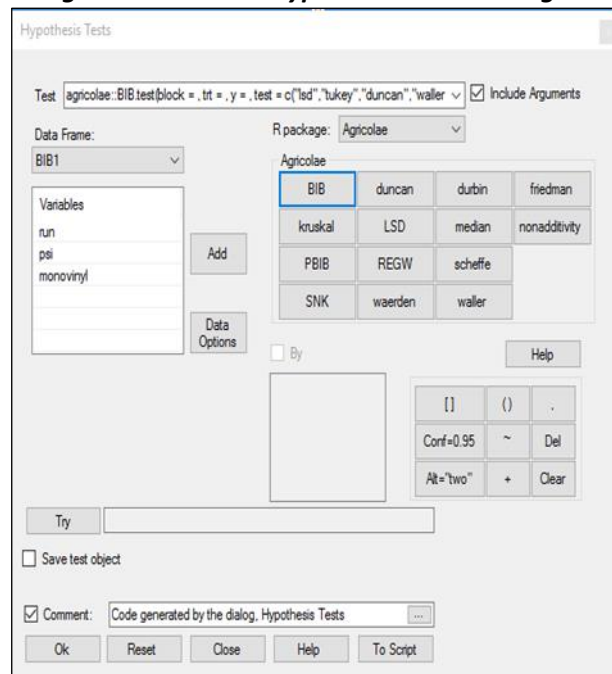


The data for analysis are now in the data frame shown in Fig. 3.

**Fig. 3 BIB1 data frame**

	run (f)	psi	monovinyl
1	1	250	16
2	1	325	18
3	1	475	32
4	2	250	19
5	2	475	46
6	2	550	45
7	3	325	26
8	3	400	39
9	3	550	61
10	4	400	21
11	4	475	35
12	4	550	55
13	5	325	19
14	5	475	47
15	5	550	48
16	6	250	20
17	6	400	33
18	6	475	31
19	7	250	13
20	7	325	13
21	7	400	34
22	8	250	21
23	8	400	30
24	8	550	52
25	9	250	24
26	9	325	10
27	9	550	50

**Fig. 4 The Model > Hypothesis Tests dialogue**

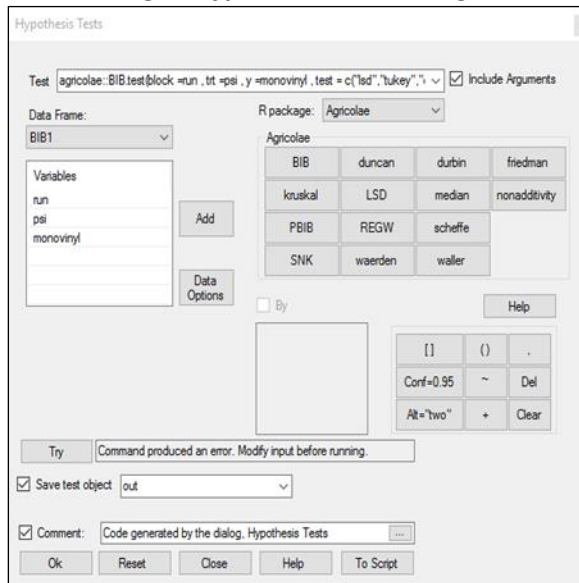


The next step is the command from the script for the analysis. This uses yet another “halfway” dialogue:

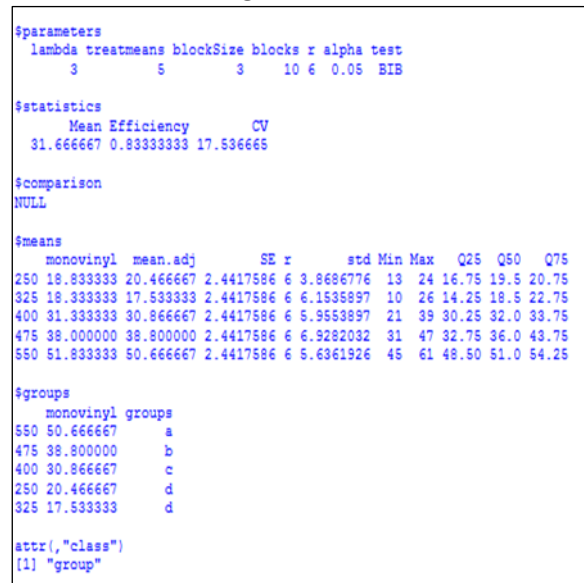
- \*\* Use **Model > Hypothesis Tests**.
- \*\* Tick the checkbox to **include arguments** and change the package to **agricolae** Fig. 4.
- \*\* Press the **BIB** key, Fig. 4.
- \*\* Complete the dialogue as shown in Fig. 5, so it is consistent with the command given in the script file.

- \*\* Click on **Try**
- \*\* Check the box to **save** the object as **out**.
- \*\* Click **Ok**.

**Fig. 5 Hypothesis Test Dialog**



**Fig. 6 Results**



## 9) 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<sup>3</sup> 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 dialogue that has opened.

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

<sup>3</sup> 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.

**Fig. 1 Dodoma Data Frame**

	Year	Month (f)	Day	Date (D)	month_abbr	doy_366	Rain	Tmax	Tmin
1	1935	1	1	1935-01-01	Jan	1	0.0	NA	NA
2	1935	1	2	1935-01-02	Jan	2	6.3	NA	NA
3	1935	1	3	1935-01-03	Jan	3	1.8	NA	NA
4	1935	1	4	1935-01-04	Jan	4	0.0	NA	NA
5	1935	1	5	1935-01-05	Jan	5	0.0	NA	NA
6	1935	1	6	1935-01-06	Jan	6	0.0	NA	NA
7	1935	1	7	1935-01-07	Jan	7	0.0	NA	NA
8	1935	1	8	1935-01-08	Jan	8	0.5	NA	NA
9	1935	1	9	1935-01-09	Jan	9	0.0	NA	NA
10	1935	1	10	1935-01-10	Jan	10	0.0	NA	NA
11	1935	1	11	1935-01-11	Jan	11	0.0	NA	NA
12	1935	1	12	1935-01-12	Jan	12	0.0	NA	NA
13	1935	1	13	1935-01-13	Jan	13	0.0	NA	NA
14	1935	1	14	1935-01-14	Jan	14	0.0	NA	NA
15	1935	1	15	1935-01-15	Jan	15	0.0	NA	NA
16	1935	1	16	1935-01-16	Jan	16	0.0	NA	NA
17	1935	1	17	1935-01-17	Jan	17	0.0	NA	NA
18	1935	1	18	1935-01-18	Jan	18	0.0	NA	NA
19	1935	1	19	1935-01-19	Jan	19	0.0	NA	NA
20	1935	1	20	1935-01-20	Jan	20	0.0	NA	NA
21	1935	1	21	1935-01-21	Jan	21	0.0	NA	NA
22	1935	1	22	1935-01-22	Jan	22	0.0	NA	NA
23	1935	1	23	1935-01-23	Jan	23	0.0	NA	NA
24	1935	1	24	1935-01-24	Jan	24	0.0	NA	NA
25	1935	1	25	1935-01-25	Jan	25	0.0	NA	NA
26	1935	1	26	1935-01-26	Jan	26	0.0	NA	NA
27	1935	1	27	1935-01-27	Jan	27	0.0	NA	NA

Showing 28855 of 28855 rows | Showing 10 of 10 columns

**Fig. 2 Dodoma Filtered**

	Year	Month (f)	Day	Date (D)	month_abbr	doy_366	Rain	Tmax	Tmin
2	1935	1	2	1935-01-02	Jan	2	6.3	NA	NA
3	1935	1	3	1935-01-03	Jan	3	1.8	NA	NA
28	1935	1	28	1935-01-28	Jan	28	78.5	NA	NA
29	1935	1	29	1935-01-29	Jan	29	1.8	NA	NA
30	1935	1	30	1935-01-30	Jan	30	2.5	NA	NA
36	1935	2	5	1935-02-05	Feb	36	28.2	NA	NA
42	1935	2	11	1935-02-11	Feb	42	3.0	NA	NA
47	1935	2	16	1935-02-16	Feb	47	13.2	NA	NA
49	1935	2	18	1935-02-18	Feb	49	55.1	NA	NA
51	1935	2	20	1935-02-20	Feb	51	17.8	NA	NA
53	1935	2	22	1935-02-22	Feb	53	4.8	NA	NA
56	1935	2	25	1935-02-25	Feb	56	3.8	NA	NA
57	1935	2	26	1935-02-26	Feb	57	2.3	NA	NA
58	1935	2	27	1935-02-27	Feb	58	1.3	NA	NA
60	1935	3	1	1935-03-01	Mar	61	34.8	NA	NA
66	1935	3	7	1935-03-07	Mar	67	3.6	NA	NA
97	1935	4	7	1935-04-07	Apr	98	2.8	NA	NA
100	1935	4	10	1935-04-10	Apr	101	4.6	NA	NA
101	1935	4	11	1935-04-11	Apr	102	21.3	NA	NA
102	1935	4	12	1935-04-12	Apr	103	3.3	NA	NA
103	1935	4	13	1935-04-13	Apr	104	1.3	NA	NA
111	1935	4	21	1935-04-21	Apr	112	2.0	NA	NA
112	1935	4	22	1935-04-22	Apr	113	8.1	NA	NA
121	1935	5	1	1935-05-01	May	122	1.0	NA	NA
137	1935	5	17	1935-05-17	May	138	41.7	NA	NA
335	1935	12	1	1935-12-01	Dec	336	2.3	NA	NA
336	1935	12	2	1935-12-02	Dec	337	5.8	NA	NA

Showing 3379 of 3379 rows (28855) | Showing 10 of 10 columns

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.**

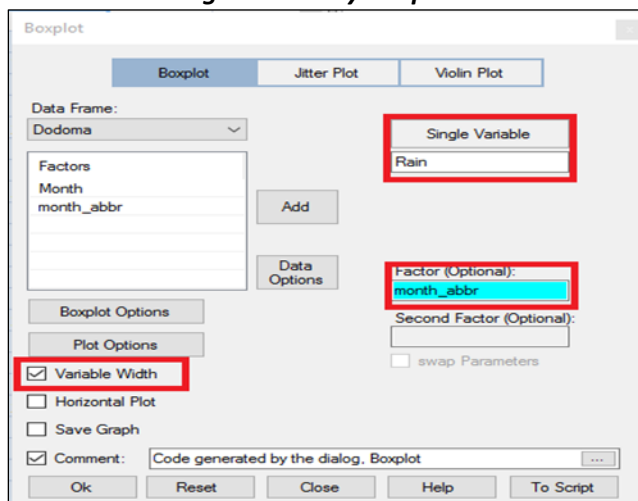
**\*\* On the sub-dialogue use the condition  $Rain > 0.85$ .**

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

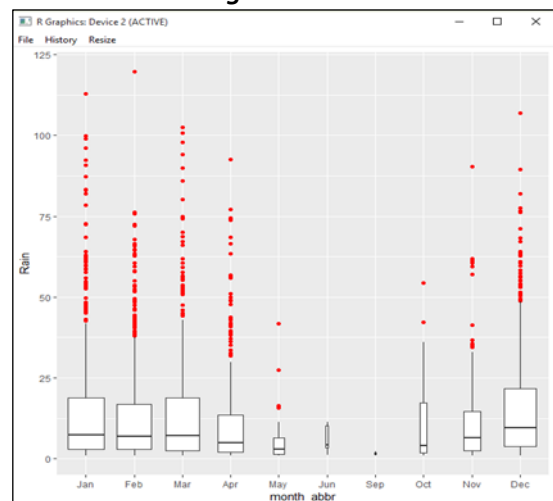
The data should now look as shown in Fig. 2.

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

**Fig. 3 Ordinary boxplot**



**Fig 4 Results**



\*\* Complete the dialogue 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 dialogue and **press the Script button**.

\*\*

**Fig. 5 Script window for the boxplot commands**

```
1. Dodoma <- data_book$get_data_frame(data_name="Dodoma")
2. last_graph <- ggplot2::ggplot(data=Dodoma, mapping=ggplot2::aes(y=Rain, x=month_abbrev)) +
  ggplot2::geom_boxplot(varwidth=TRUE, outlier.colour="red") + theme_grey()
3. data_book$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 and it is line 2 that has to change. From the manual for the robustbase package we find that a possible command is:

```
last_graph <- robustbase::adjbox(Rain ~ month_abbrev, data =Dodoma, col="red", varwidth=TRUE)
```

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

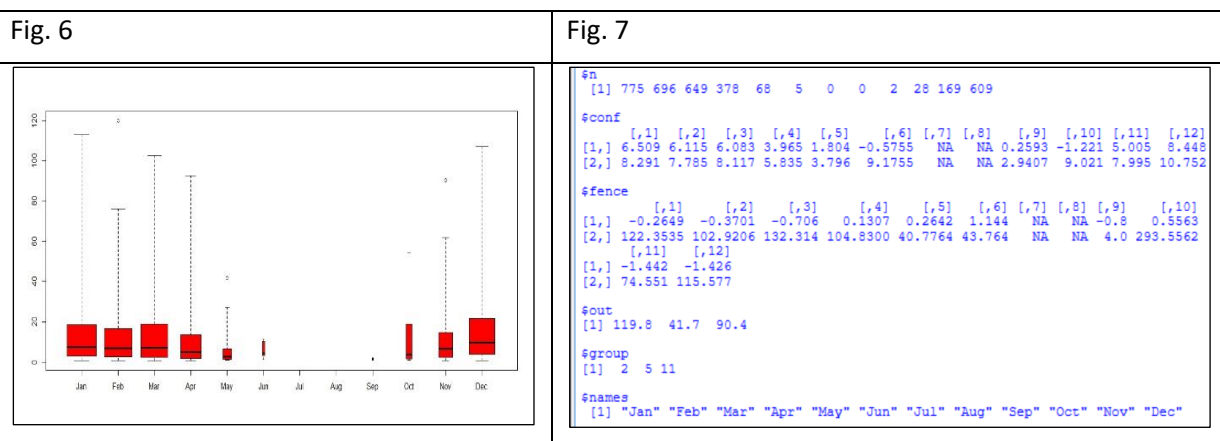
\*\* Press **Run All** at the top of the script window.

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.



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