Article from

Actuarial Technology Today

May 2020

Guerrilla Automation With R:

R Automation Despite Resource Constraints

By Timothy Quast

Guerrilla warfare is a type of asymmetric warfare wherein a smaller, less powerful military uses unconventional tactics to fight a stronger, more traditional military. It often involves a higher level of mobility, with nimbleness making up for the military's lack of resources. We can draw an analogy in business. When a problem calls for a traditional technological solution, but no such solution is currently in-place, actuaries may require a low-cost, nimble approach to meeting the needs of their stakeholders.

That's where R comes in. R is a free and open source programming language with a powerful and continually expanding set of libraries and packages. It is highly versatile and compatible with a wide variety of other systems. In this article, I will walk through a hypothetical use-case for R that highlights R's advantages. Example code will be interspersed throughout the article. The full code, complete with example excel files can be accessed via Github at the following URL: *https://github.com/TimothyQuast/Guerrilla_Automation*

If you download the repository and would like to follow along, I highly recommend installing RStudio for free from *Rstudio.com*. A rudimentary understanding of R will be helpful in following along, but it is not essential.

THE USE-CASE

Let's suppose that our stakeholder has a large number of files in Excel throughout their network drives with financial information. They would like to be able to summarize all the financial information and trace the lineage of each subcomponent. Specifically, let's say that they want to support numerous account balances using the actuarial workpapers that feed them. They want to break each balance into pieces, with each piece corresponding to one actuarial workpaper, which contributes to that balance. Doing so would be extremely valuable for audit purposes and reasonableness testing.

The actuarial workpapers are stored in Excel in a regular format in different places throughout the stakeholder's network drives, with a variety of teams contributing to the same account balances. Moreover, the workpapers represent an aggregated version of the actuarial results, so they aren't overly granular. But they are granular enough to make a manual solution infeasible. How do we get the data we need to support the account balances?

The proper, traditional method is a big fancy subledger containing the supporting balances along with metadata that traces each balance. But let's say the stakeholder doesn't have a big, fancy subledger yet. Further, let's say that the stakeholder wants the balances supported **soon**—sooner than a big, fancy subledger can be developed. We need a temporary solution to solve the problem quickly. In such a conundrum, one might try automating the task with R!

R is suitable for the task for several reasons:

- **It's free!** It can be used for such a task without requiring investment dollars or licensing.
- **It's open source with a broad user base.** Developers are continually producing marvelous new packages that can solve tough problems, making R extremely versatile.
- **It's highly compatible.** The variety of packages and the non-proprietary nature of the system make it uniquely capable of interacting with other systems, such as Excel.
- **It's suitable for rapid prototyping.** R is elegant and fairly high-level, allowing the user to do a lot with just a little code.

R has some disadvantages too:

- **It's less efficient.** R doesn't do as well as other languages in terms of processing efficiency. That's why it's important that the workpapers are already aggregated. If they were extremely granular, with a high volume of data, then R might struggle.
- **There's a learning curve.** Like all programming languages, R must be learned. It might be difficult for new personnel to learn the language, making the process less portable.
- **It has a copyleft license.** R is usable for any commercial purpose, but the license requires that any derivative works be under a compatible open source license. In other words, you cannot distribute proprietary software that uses R: it would have to be open source. This is not a problem as long as we are internal or we are charging for labor (instead of software licensing).

The drawbacks are not overwhelming, and it won't cost us anything but time to try, so let's figure out how to solve our problem with R. I've constructed an example problem (which you can find in the GitHub repository) using four Excel files (see Figure 1). **Ledger Balances.xlsx** contains the "ledger balances" we are trying to support. Three workpaper files contain the "workpapers" that support the "ledger balances." Pretend that the workpaper files are located in disparate places throughout the stakeholder's network drives!

Figure 1
Example Problem

Excel Files/Ledger Balances.xlsx

Account Number	Amount
A0000	15368
A1111	6973
A2222	13909
A3333	3349
A4444	7725
A5555	9504
A7777	15486
A8888	8472
A9999	2992

Excel Files/Workpaper 1.xlsx

Account Number	Amount
A0000	4747
A1111	0
A2222	5228
A3333	0
A4444	4741
A5555	4445
A7777	9560
A8888	0
A9999	0

Excel Files/Workpaper 2.xlsx

Account Number	Amount
A0000	9836
A1111	0
A2222	8508
A3333	3349
A4444	0
A5555	0
A7777	5926
A8888	8472
A9999	0

Excel Files/Workpaper 3.xlsx

Account Number	Amount
A0000	785
A1111	6973
A2222	173
A3333	0
A4444	2984
A5555	5059
A7777	0
A8888	0
A9999	2992

Figure 2
Input Files Chronicled

Excel Files/Input Control.xlsx

Filepath	Sheet	Range
Excel Files/Workpaper 1.xlsx	Sheet1	B4:C13
Excel Files/Workpaper 2.xlsx	Sheet1	B4:C9001
Excel Files/Workpaper 3.xlsx	An Unusual Sheet Name	C4:D13
Excel Files/Workpaper 3.xlsx	A Mistaken Sheet Name	C4:D13
Excel Files/Workpaper 3 Mistaken Filename.xlsx	An Unusual Sheet Name	C4:D13

You can also manually verify that the account number totals in the three workpapers sum to the ledger balance totals, but then what's the point of automating it? Notice that the workpapers keep the data in a regular format, but **Workpaper 3.xlsx** has the data in different cells and it has an unusual sheet name! These differences are intentional, and we will handle them in our solution.

THE SOLUTION

The first step to solving any problem is simple: get organized! I start by chronicling the input files in the **Excel Files/Input Control.xlsx** spreadsheet (see Figure 2).

I typically prefer to use absolute file paths, but I set this up with relative paths so that it will work on other machines. The first three rows here contain correct information. The last two contain errors in the sheet name and file name, respectively. This can happen if, for example, the process is run on a monthly basis, and the file-naming conventions for a workpaper change from one month to the next. I included them to demonstrate R's ability to deal with these complications. Two things to note:

- The range for Workpaper 2 is unnecessarily large. This is a useful tactic when the numbers of rows changes from month to month and you want to ensure that you capture all the data. We eventually remove the resulting empty rows from our data.
- There are no errors in the Range column. A runtime error in the Range column would be captured in the same way as file/sheet name errors, but a logical error (i.e., if we entered the range incorrectly) could create challenges. It is possible to dynamically determine the correct range for each sheet, but doing so is a bit trickier, and beyond the scope of this article.

Next, we start using R. We rely on three packages: readxl (Wickham & Bryan 2018), writexl (Ooms, 2018), and dplyr (Wickham et al., 2020). If you have Rstudio installed, you can follow along in the GitHub repository by opening the file **Guerrilla Automation.Rproj** and then opening the file **Guerrilla Automation.R** from the same instance of Rstudio. You can run the whole process from start to finish by calling either the **main** or the **output_results** functions. Note that you can install the requisite packages using the following code. You also need to source the R script file:

```
install.packages(c("readxl","writexl",
"dplyr"))
source('Guerrilla Automation.R')
```

Step 1 is to import the Input Control data. We do this in the **import_input_control** function with the following lines:

```
input_control = read_excel(filepath, sheet =
"Input Control", range="B4:D9001")
input_control = data.frame(input_con-
trol[!is.na(input_control$Filepath),])
```

Note that the range parameter is excessively large. This helps if you have to add new files to the input control. The second line converts input_control to a data frame and removes the NA values at the end of the data (which occur because of the excessive range). The input_control data frame now looks as shown in Figure 3:

Figure 3
Input_Control Data

input_control

Filepath	Sheet	Range
Excel Files/Workpaper 1.xlsx	Sheet1	B4:C13
Excel Files/Workpaper 2.xlsx	Sheet1	B4:C9001
Excel Files/Workpaper 3.xlsx	An Unusual Sheet Name	C4:D13
Excel Files/Workpaper 3.xlsx	A Mistaken Sheet Name	C4:D13
Excel Files/Workpaper 3 Mistaken Filename.xlsx	An Unusual Sheet Name	C4:D13

Figure 4
Data Frames

input_list[[“input_control”]]

	A	B	C	D	E	F	G	H	I	J
1	**Filepath**	**Sheet**	**Range**	**File_Is_Good**	**Error_Message**					
2	Excel Files/\	Sheet1	B4:C13	TRUE						
3	Excel Files/\	Sheet1	B4:C9001	TRUE						
4	Excel Files/\	An Unusual	C4:D13	TRUE						
5	Excel Files/\	A Mistaken	C4:D13	FALSE	Error: Sheet 'A Mistaken Sheet Name' not found					
6	Excel Files/\	An Unusual	C4:D13	FALSE	Error: `path` does not exist: ‘Excel Files/Workpaper 3 Mistaken Filename.xlsx’					

Figure 5
Data Frames

input_list[[“input_data”]]

	Filepath	Sheet	Range	Account Number	Amount
1	Excel Files/Input 1.xlsx	Sheet1	B4:C13	A0000	4747
2	Excel Files/Input 1.xlsx	Sheet1	B4:C13	A1111	0
3	Excel Files/Input 1.xlsx	Sheet1	B4:C13	A2222	5228
4	Excel Files/Input 1.xlsx	Sheet1	B4:C13	A3333	0
5	Excel Files/Input 1.xlsx	Sheet1	B4:C13	A4444	4741
6	Excel Files/Input 1.xlsx	Sheet1	B4:C13	A5555	4445
7	Excel Files/Input 1.xlsx	Sheet1	B4:C13	A7777	9560
8	Excel Files/Input 1.xlsx	Sheet1	B4:C13	A8888	0
9	Excel Files/Input 1.xlsx	Sheet1	B4:C13	A9999	0
10	Excel Files/Input 2.xlsx	Sheet1	B4:C9001	A0000	9836

Step 2 is to loop through the input_control data frame and use read_excel on the parameters in each row. We do this in the **gather_input_data** function. We also handle our erroneous example rows in the Input Control using a tryCatch. Similar to how we removed the extraneous rows from the Input Control, we also removed the extraneous rows from Workpaper 2. I've omitted the code for Step 2 because it is rather lengthy, but it results in a list of two data frames. One is an augmented version of input_control and the other is called input_data. I assign these data frames to input_list via:

```
input_list = list()
# Code omitted
input_list["input_control"]] = input_
control
input_list["input_data"]] = input_data
```

Now they are as shown in Figures 4 and 5.

Note the two extra columns in input_control. We can use these to track which files imported successfully and what went wrong with the files that failed.

Step 3 is to summarize the data. We do this in the **aggregate_input_data** function using elegant dplyr functions. We store our results in a list called aggregate_list. We keep the old input_list along with our new summary.

```
aggregate_list = list()
aggregate_list["input_control"]] = input_
list["input_control"]]
aggregate_list["input_data"]] = input_
list["input_data"]]
aggregate_list["summary"]] = data.frame(in-
put_list["input_data"]] %>%
group_by(`Account Number`) %>% sum-
marise(Amount = sum(Amount)))
```

The summary now appears as shown in Figure 6.

Figure 6
Input_Data Summary

aggregate_list[["summary"]]

Account Number	Amount
A0000	15368
A1111	6973
A2222	13909
A3333	3349
A4444	7725
A5555	9504
A7777	15486
A8888	8472
A9999	2992

Note that these values are exactly the same as the ones we started with in **Ledger Balances.xlsx**. Huzzah!

Step 4 is to output the results to Excel. We do this in the output_results function using the writexl package as follows:

```
write_xlsx(aggregate_list, "./Excel Files/R
Output/Results.xlsx")
```

Now, we're back in Excel! In the file **Excel Files/R Output/Results.xlsx**, we have a tab for each data frame. I chose to output the results into a new Excel file. This prevents issues that occur when you are trying to write to a file that is currently open. If you open the **Results.xlsx** file and run the process again, you should get an error. One way to prevent this is to include a time stamp in the file name via:

```
write_xlsx(aggregate_list, paste("./Excel
Files/R Output/Results ",
format(Sys.time(), "%Y%m%d %s"), ".xlsx",
sep=""))
```

Now we can easily see which files were imported successfully, and we can trace each ledger balance back to the files that contributed to it. Our approach to automating data gathering has a lot of flexibility. Here's a few other things we could do with our solution:

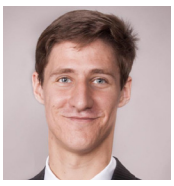
1. Add additional identifier columns to Input Control.xlsx, which can provide helpful splits to our aggregate data.

2. Add functionality that dynamically detects the appropriate range for each input file.

3. Automate the task of comparing the resulting summary to the original ledger balance.

4. Iteratively correct **Input Control.xlsx** using the **input_control** tab in **Results.xlsx** until all of the rows contain correct information.

As you can see, R offers a lot of power and compatibility as a free and open source system. I enjoy using R in both business and as a hobby because of its elegance and versatility. It is a great resource for rapidly developing solutions that can meet the needs of your stakeholder. That's all folks. I hope you enjoyed this "R-ticle" and found it most helpful! ■



Timothy Quast, ASA, is an assistant actuary at AIG. He can be contacted at *timothy.quast@aig.com.*

REFERENCES

Hadley Wickham and Jennifer Bryan (2018). readxl: Read Excel Files. R package version 1.1.0. *https://CRAN.R-project.org/package=readxl*

Jeroen Ooms (2018). writexl: Export Data Frames to Excel 'xlsx' Format. R package version 1.0. *https://CRAN.R-project.org/package=writexl*

Hadley Wickham, Romain François, Lionel Henry and Kirill Müller (2020). dplyr: A Grammar of Data Manipulation. R package version 0.8.5. *https://CRAN.R-project.org/package=dplyr*