Dealing with Large CSV Files in R

By Jeff Heaton

The R programming language is becoming a common tool for actuaries and data scientists to examine and model a variety of different data types. A number of useful functions are provided to load data into memory, process the dataset, and then write results to another file. Unfortunately, an additional complication can enter the picture when these files become large. If R is commanded to read a CSV that is larger than the computer's memory, an error will be returned. If you experience this, don't worry, there are a number of solutions available.

The most obvious solution is to obtain more memory. This could be more physical memory. This could also mean using part of the hard drive as virtual memory.

The most obvious solution is to obtain more memory. This could be more physical memory. This could also mean using part of the hard drive as virtual memory. These are certainly viable solutions. There are also a host of "Big Data" solutions. A multi-node Hadoop or Spark solution could be installed that allows many computers to work together to process the file. Again, this is certainly a solution, but it might not be necessary. There definitely is a class of data that are so large that "Big Data" technology is required to process them in any reasonable amount of time. However, this is often neither necessary nor economical.

Most R functions simply load an entire file into memory. This is the simplest way of handling the file, but it means that the program can only handle data up to a certain size. Consider a simple example that illustrates the process. A large data file has been provided that contains the premium payments for customers over a potentially large span of time. Such a file might appear as follows.

```
policy_number,product,premium,month,year
J10234,term15,110,1,2010
Z10400,term10,100,1,2010
J10523,term15,110,1,2010
Z10624,term10,100,1,2010
J10234,term15,110,2,2010
Z10400,term10,100,2,2010
J10523,term15,110,2,2010
Z10624,term10,100,2,2010
...
```

To see how to handle a file of any length, consider a simple example where it is necessary to bin/roll up the premium amount by product and month. This would produce a result file similar to the following:

```
product,premium
term15,220
term10,200
...
```

A simple R program to perform this task is provided:

```
data <- read.csv("c:\\test\\sample.csv ")
result <- aggregate(data$premium,
  by=list(product=data$product,month=-
  data$month,year=data$year),
  FUN=sum)
write.csv(result,c:\\test\\sample_output.
csv ")
```

This program begins by reading the entire file "sample.csv" into the variable named "data." If this file fits into memory, everything works well and the output file is written. However, if the file does not fit into memory, an error occurs and there is no output. An alternative approach is to read the file line by line and perform the aggregation by the program. This approach is a bit more complex, but it will work on very large files.

```
# Hold all of the bins
bins <- list()

# Open the file.
fp  <-  file(“c:\\test\\sample.csv”,  open  =
“r”)

# Skip header
readLines(fp, n = 1)

# Loop over entire file
while (length(line <- readLines(fp, n = 1))
> 0) {

 # Read a single line from the file
 line <- unlist((strsplit(line, “,”)))

 # Extract the columns we care about
 product <- line[2]
 premium <- as.numeric(line[3])
 month <- as.integer(line[4])
 year <-as.integer(line[5])

 # Produce a key that holds all values we
want to “group by”

 # Is this the first time we’ve seen this
combination of month/year/product?
 key <- paste(product,month,year)
 if (key %in% names(bins)) {
  # Add to our running premium bin
  binprem <- as.numeric(bins[[key]][4])
  bins[[key]] <- list( product, year, month,
premium + binprem )
 } else {
  # Create a new premium bin
  bins[[key]] <- list( product, year, month,
premium )
 }
}
close(fp)

# Transform the bin’s list into a dataframe
for output
bins  <-  as.data.frame(matrix(unlist(bins),
nrow=4, byrow = T))
colnames(bins)       <-       c(‘product’,
‘year’,’month’,’premium’)

bins
```

The above code uses named lists, called “bins” to hold the value of each of our bins that aggregate product, year and month. A key is created to find the correct bin. This key is nothing more than a string, such as “term15 2010 1” to represent the bin for January 2010’s term15 premiums. Comments are provided to demonstrate the process. This short program could be a great starting point for any other situations where it is necessary to iterate over a very large file. Similar techniques can be very useful for other types of files, such as XML, JSON or even raw text.

In conclusion, each of the solutions outlined above should be reviewed within the common context of any problems solving activity, including money, time and available human and hardware resource capacity. I hope this discussion provides some meaningful alternatives in the increasing landscape of widespread utilization of R within the financial services vertical industry. ■

Jeff Heaton is a senior data scientist for RGA. He can be contacted at *jheaton@rgare.com.*