Expanding memory with the bigmemory package

Dr. Zahid Ansari

Contents

```
setwd("D:/AMU Computer Science/Courses/Big Data Analytics/Big Data Analytics Using R/Ch3")
```

- As we know that, the usual importing of data into R using the read.table() family of functions is limited by the available RAM.
- However, we always need to allow additional memory resources for further data manipulations, and other R objects that we create in the process, during the same R session.
- The use of read.table() functions also results in a memory overhead of anything from 30% to 100% of the original data size.
- It simply means that an import of a 1 GB file to an R workspace requires roughly 1.3 GB to 2 GB of available RAM to succeed.
- We already know how to deal with much large data using the ff/ffdf approach.
- The bigmemory package, offers an alternative solution, but again, it's not without its own limitations and disadvantages.
- In general, bigmemory facilitates data import, processing, and analysis by allocating the S4 class objects (matrices) to shared memory and using memory-mapped files.
- One issue with the bigmemory package is, it supports only matrices, and we know that matrices can hold only one type of data.
- Therefore, if your dataset includes a mix of character, numeric, or logical variables, they cannot be used together in bigmemory.
- There are, however, several ways of dealing with this problem. You can:
- 1.Mine and collect your data using methods that only code responses as numeric values (and keep the labels or factor levels in a separate file if necessary), or
- 2.If your original dataset is within the range of the available RAM, import it using read.table() or, even better, through the data.table package and transform the classes of variables to numeric only, or
- 3.For out-of-memory data, use the ff package, change the classes of variables to numeric, and save the resulting data to another file on a disk.
- Let's now prepare some data for the import and further processing with the bigmemory package using the second method.
- In this part we will be using an interesting governmental dataset National Energy Efficiency Data – Framework (NEED) provided by the Department of Energy & Climate Change in the United Kingdom.

- The public use file, which we are going to use here, is very small (7.8 MB) as it only contains a representative sample of 49,815 records, but you are encouraged to test the R code on a much larger sample of 4,086,448 cases (which you can obtain from UK Data Service at https://discover.ukdataservice.ac.uk/catalogue/?sn=7518.
- In short, the data contain information on annual electricity and gas consumption, energy efficiency characteristics, and socio-demographic details of UK-based households over several years, from 2005 until 2012.
- Once you save the data to your working directory, you may import the public use file through a standard read.csv() command:

```
# need0 <- read.csv("need_puf14.csv", header = TRUE, sep = ",")
need0 <- read.csv("need_puf14.csv", header = TRUE, sep = ",", stringsAsFactors = T)
str(need0)
```

```
## 'data.frame':    49815 obs. of  50 variables:
##  $ HH_ID          : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ REGION         : Factor w/ 10 levels "E12000001","E12000002",..: 7 2 2 5 3 7 6 5 7 3 ...
##  $ IMD_ENG        : int  1 4 4 1 1 2 3 5 4 2 ...
##  $ IMD_WALES      : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ Gcons2005      : int  35000 19000 22500 21000 NA NA 12000 18500 35000 28000 ...
##  $ Gcons2005Valid : Factor w/ 5 levels "G","L","M","O",..: 5 5 5 5 3 4 5 5 5 5 ...
##  $ Gcons2006      : int  24500 14900 22500 20500 NA NA 16500 15500 40000 26000 ...
##  $ Gcons2006Valid : Factor w/ 5 levels "G","L","M","O",..: 5 5 5 5 3 4 5 5 5 5 ...
##  $ Gcons2007      : int  22000 16000 22500 18000 NA NA 12300 13900 35000 24000 ...
##  $ Gcons2007Valid : Factor w/ 5 levels "G","L","M","O",..: 5 5 5 5 3 4 5 5 5 5 ...
##  $ Gcons2008      : int  25000 17000 19500 19500 NA NA 12500 16500 35000 29000 ...
##  $ Gcons2008Valid : Factor w/ 5 levels "G","L","M","O",..: 5 5 5 5 3 4 5 5 5 5 ...
##  $ Gcons2009      : int  23000 12800 19500 18500 NA NA 14800 14700 31000 28000 ...
##  $ Gcons2009Valid : Factor w/ 5 levels "G","L","M","O",..: 5 5 5 5 3 4 5 5 5 5 ...
##  $ Gcons2010      : int  20000 13600 19500 19000 NA NA 4000 16500 29000 22000 ...
##  $ Gcons2010Valid : Factor w/ 5 levels "G","L","M","O",..: 5 5 5 5 3 4 5 5 5 5 ...
##  $ Gcons2011      : int  15100 14700 20000 19500 NA NA 4000 18000 28000 26000 ...
##  $ Gcons2011Valid : Factor w/ 5 levels "G","L","M","O",..: 5 5 5 5 3 4 5 5 5 5 ...
##  $ Gcons2012      : int  19500 13200 16500 17500 NA NA 4000 20500 30000 24000 ...
##  $ Gcons2012Valid : Factor w/ 5 levels "G","L","M","O",..: 5 5 5 5 3 4 5 5 5 5 ...
##  $ Econs2005      : int  12500 3100 5600 4900 2500 5000 3450 5650 10300 4350 ...
##  $ Econs2005Valid : Factor w/ 4 levels "G","L","M","V": 4 4 4 4 4 4 4 4 4 4 ...
##  $ Econs2006      : int  10900 2750 4500 4550 2600 4850 4200 6300 7700 4300 ...
##  $ Econs2006Valid : Factor w/ 4 levels "G","L","M","V": 4 4 4 4 4 4 4 4 4 4 ...
##  $ Econs2007      : int  12500 3000 4300 6200 NA 4900 2150 5300 14500 5350 ...
##  $ Econs2007Valid : Factor w/ 4 levels "G","L","M","V": 4 4 4 4 2 4 4 4 4 4 ...
##  $ Econs2008      : int  11000 2200 3800 7100 1200 4100 900 4550 6000 4700 ...
##  $ Econs2008Valid : Factor w/ 4 levels "G","L","M","V": 4 4 4 4 4 4 4 4 4 4 ...
##  $ Econs2009      : int  9500 2450 5600 7400 2300 4300 1650 3850 5650 4800 ...
##  $ Econs2009Valid : Factor w/ 4 levels "G","L","M","V": 4 4 4 4 4 4 4 4 4 4 ...
##  $ Econs2010      : int  10000 2150 4750 7650 2650 800 1500 1500 3050 5150 ...
##  $ Econs2010Valid : Factor w/ 4 levels "G","L","M","V": 4 4 4 4 4 4 4 4 4 4 ...
##  $ Econs2011      : int  7600 3150 5300 8300 2800 2000 1850 3500 3850 5200 ...
##  $ Econs2011Valid : Factor w/ 1 level "V": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Econs2012      : int  6300 3000 4700 7350 1950 3900 2050 5100 4400 5700 ...
##  $ Econs2012Valid : Factor w/ 2 levels "G","V": 2 2 2 2 2 2 2 2 2 2 ...
##  $ E7Flag2012     : int  NA NA NA NA NA 1 NA NA 1 NA ...
```

```
##  $ MAIN_HEAT_FUEL : int  1 1 1 1 1 2 1 1 1 1 ...
##  $ PROP_AGE       : int  101 102 106 101 103 105 104 105 102 102 ...
##  $ PROP_TYPE      : int  104 102 101 104 106 106 103 101 102 102 ...
##  $ FLOOR_AREA_BAND: int  4 2 4 3 1 1 2 4 3 3 ...
##  $ EE_BAND        : int  3 4 3 4 2 1 4 3 3 3 ...
##  $ LOFT_DEPTH     : int  2 2 2 2 99 99 2 2 1 2 ...
##  $ WALL_CONS      : int  2 2 1 2 2 1 1 1 1 1 ...
##  $ CWI            : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ CWI_YEAR       : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ LI             : int  NA NA NA NA NA NA NA NA NA 1 ...
##  $ LI_YEAR        : int  NA NA NA NA NA NA NA NA NA 2009 ...
##  $ BOILER         : int  NA NA 1 NA NA NA NA NA NA NA ...
##  $ BOILER_YEAR    : int  NA NA 2004 NA NA NA NA NA NA NA ...
```

- Notice that our data contains a number of categorical variables (factors).
- You can extract class information for all the variables in the data using the following snippet:

```
classes <- unlist(lapply(colnames(need0), function(x) { class(need0[,x]) }))
classes
```

```
##  [1] "integer" "factor"  "integer" "integer" "integer" "factor"
##  [7] "integer" "factor"  "integer" "factor"  "integer" "factor"
## [13] "integer" "factor"  "integer" "factor"  "integer" "factor"
## [19] "integer" "factor"  "integer" "factor"  "integer" "factor"
## [25] "integer" "factor"  "integer" "factor"  "integer" "factor"
## [31] "integer" "factor"  "integer" "factor"  "integer" "factor"
## [37] "integer" "integer" "integer" "integer" "integer" "integer"
## [43] "integer" "integer" "integer" "integer" "integer" "integer"
## [49] "integer" "integer"
```

- Based on the contents of the classes object, we can now identify indices of factors and use this information to convert them to integers with a for() loop statement:

```
ind <- which(classes=="factor")
for(i in ind) {need0[,i] <- as.integer(need0[, i])}
str(need0)
```

```
## 'data.frame':    49815 obs. of  50 variables:
##  $ HH_ID          : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ REGION         : int  7 2 2 5 3 7 6 5 7 3 ...
##  $ IMD_ENG        : int  1 4 4 1 1 2 3 5 4 2 ...
##  $ IMD_WALES      : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ Gcons2005      : int  35000 19000 22500 21000 NA NA 12000 18500 35000 28000 ...
##  $ Gcons2005Valid : int  5 5 5 5 3 4 5 5 5 5 ...
##  $ Gcons2006      : int  24500 14900 22500 20500 NA NA 16500 15500 40000 26000 ...
##  $ Gcons2006Valid : int  5 5 5 5 3 4 5 5 5 5 ...
##  $ Gcons2007      : int  22000 16000 22500 18000 NA NA 12300 13900 35000 24000 ...
##  $ Gcons2007Valid : int  5 5 5 5 3 4 5 5 5 5 ...
##  $ Gcons2008      : int  25000 17000 19500 19500 NA NA 12500 16500 35000 29000 ...
##  $ Gcons2008Valid : int  5 5 5 5 3 4 5 5 5 5 ...
##  $ Gcons2009      : int  23000 12800 19500 18500 NA NA 14800 14700 31000 28000 ...
##  $ Gcons2009Valid : int  5 5 5 5 3 4 5 5 5 5 ...
```

```
##  $ Gcons2010      : int  20000 13600 19500 19000 NA NA 4000 16500 29000 22000 ...
##  $ Gcons2010Valid : int  5 5 5 5 3 4 5 5 5 5 ...
##  $ Gcons2011      : int  15100 14700 20000 19500 NA NA 4000 18000 28000 26000 ...
##  $ Gcons2011Valid : int  5 5 5 5 3 4 5 5 5 5 ...
##  $ Gcons2012      : int  19500 13200 16500 17500 NA NA 4000 20500 30000 24000 ...
##  $ Gcons2012Valid : int  5 5 5 5 3 4 5 5 5 5 ...
##  $ Econs2005      : int  12500 3100 5600 4900 2500 5000 3450 5650 10300 4350 ...
##  $ Econs2005Valid : int  4 4 4 4 4 4 4 4 4 4 ...
##  $ Econs2006      : int  10900 2750 4500 4550 2600 4850 4200 6300 7700 4300 ...
##  $ Econs2006Valid : int  4 4 4 4 4 4 4 4 4 4 ...
##  $ Econs2007      : int  12500 3000 4300 6200 NA 4900 2150 5300 14500 5350 ...
##  $ Econs2007Valid : int  4 4 4 4 2 4 4 4 4 4 ...
##  $ Econs2008      : int  11000 2200 3800 7100 1200 4100 900 4550 6000 4700 ...
##  $ Econs2008Valid : int  4 4 4 4 4 4 4 4 4 4 ...
##  $ Econs2009      : int  9500 2450 5600 7400 2300 4300 1650 3850 5650 4800 ...
##  $ Econs2009Valid : int  4 4 4 4 4 4 4 4 4 4 ...
##  $ Econs2010      : int  10000 2150 4750 7650 2650 800 1500 1500 3050 5150 ...
##  $ Econs2010Valid : int  4 4 4 4 4 4 4 4 4 4 ...
##  $ Econs2011      : int  7600 3150 5300 8300 2800 2000 1850 3500 3850 5200 ...
##  $ Econs2011Valid : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Econs2012      : int  6300 3000 4700 7350 1950 3900 2050 5100 4400 5700 ...
##  $ Econs2012Valid : int  2 2 2 2 2 2 2 2 2 2 ...
##  $ E7Flag2012     : int  NA NA NA NA NA 1 NA NA 1 NA ...
##  $ MAIN_HEAT_FUEL : int  1 1 1 1 1 2 1 1 1 1 ...
##  $ PROP_AGE       : int  101 102 106 101 103 105 104 105 102 102 ...
##  $ PROP_TYPE      : int  104 102 101 104 106 106 103 101 102 102 ...
##  $ FLOOR_AREA_BAND: int  4 2 4 3 1 1 2 4 3 3 ...
##  $ EE_BAND        : int  3 4 3 4 2 1 4 3 3 3 ...
##  $ LOFT_DEPTH     : int  2 2 2 2 99 99 2 2 1 2 ...
##  $ WALL_CONS      : int  2 2 1 2 2 1 1 1 1 1 ...
##  $ CWI            : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ CWI_YEAR       : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ LI             : int  NA NA NA NA NA NA NA NA NA 1 ...
##  $ LI_YEAR        : int  NA NA NA NA NA NA NA NA NA 2009 ...
##  $ BOILER         : int  NA NA 1 NA NA NA NA NA NA NA ...
##  $ BOILER_YEAR    : int  NA NA 2004 NA NA NA NA NA NA NA ...
```

- You can now export the need0 data.frame to an external file on a disk and import it back using the read.big.matrix() function from the bigmemory package:

```
library(bigmemory)
write.table(need0, "need_data.csv", sep = ",", row.names = FALSE, col.names = TRUE)
need.mat <- read.big.matrix("need_data.csv", header = TRUE, sep = ",", type = "double", backingfile = "r
need.mat
```

```
## An object of class "big.matrix"
## Slot "address":
## <pointer: 0x0000019006f51e50>
```

- The read.big.matrix() call comes with two useful arguments: backingfile and descriptorfile. The first is responsible for holding the raw data on a disk, the latter keeps all metadata describing the data (hence the name).

- By mapping the resulting object to data stored on a disk, they allow users to import large, out-of-memory data into R or attach a cached big.matrix without explicitly reading the whole data again when needed.
- If you didn't set these two arguments, the bigmemory package will still import the file; however, the data will not be stored in a backing file, and hence it will take much longer to load the data in the future.
- This difference can be easily noticed in the following experiment, in which we compared the time taken and RAM used for importing a large sample of over 4 million records of NEED data using read.big.matrix() implementations with and without backingfile and descriptorfile arguments, and the standard read.csv() approach.
- On each occasion the R session was started from fresh and every time the initial memory usage was found to be the same:

```
gc()
```

```
##           used (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells 1045593 55.9    1960528 104.8  1960528 104.8
## Vcells 3697658 28.3   10146329  77.5 10146061  77.5
```

- The first attempt included the following call:

```
# need.big1 <- read.big.matrix("need_big.csv", header = TRUE, sep = ",", type = "double")
need.big1 <- read.big.matrix("need_data.csv", header = TRUE, sep = ",", type = "double")
```

- It took 102 seconds to create the need.big1 object-a big.matrix of only 664 bytes in size.
- The RAM usage did not increase significantly above the base max used values. No files were created in the working directory as a result of this call.
- The second trial contained the backingfile and descriptorfile arguments as follows:

```
need.big2 <- read.big.matrix("need_data.csv", header = TRUE,
                              sep = ",", type = "double",
                              backingfile = "need_big.bin",
                              descriptorfile = "need_big.desc")
```

- Just as in the first trial, the newly created big.matrix was only 664 bytes light and the memory usage remained at the base level.
- It also took exactly 102 seconds to import the data, however two new files named as specified in the backingfile and descriptorfile emerged in the working directory.
- The first was 1.73 GB-heavy and the latter contained only 1 KB of metadata.
- The second step of this trial involved the attachment of big.matrix using the reference to the descriptor file with either the attach.resource() or attach.big.matrix() functions:

```
rm(list=ls())
# need.big2b <- attach.big.matrix("need_big.desc")
need.big2b <- attach.big.matrix("need_data.desc")
```

- This operation took only 0.001 seconds to complete, which clearly shows how useful caching through backingfile and descriptorfile can be, especially if you need to import the same data a few times, or in several separate, but parallel R sessions.
- The third trial compared the previous two with a standard read.csv() function performance. The following call was used:

```r
# need.big3 <- read.csv("need_big.csv", header = TRUE, sep = ",")
need.big3 <- read.csv("need_data.csv", header = TRUE, sep = ",")
```

- As anticipated, this statement engaged the memory resources the most of all three trials, creating a data.frame object of 841.8 MB in size, and using quite substantial amount of RAM in the process:

```r
gc()
```

```
##           used (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells 1045773 55.9    1960528 104.8  1960528 104.8
## Vcells 3698090 28.3   10146329  77.5 10146061  77.5
```

- R also spent almost 191 seconds completing the job, much longer than in the previous attempts.
- The winner is obvious, and, depending on your needs, you can either choose the read.big.matrix() implementation with or without data caching.
- We prefer to store a copy of the data through backingfile and descriptorfile in case we have to import big.matrix again.
- Although it takes a rather generous slice of hard drive space, it saves a great amount of time, especially when you deal with a multi-GB dataset.
- But reading and writing big matrices are not the only selling points of the bigmemory package.
- The library can also perform quite an impressive number of data management and analytical tasks.
- First of all, you can apply generic functions such as dim(), dimnames(), or head() to the big.matrix object:

```r
library(bigmemory)
# need.mat <- read.big.matrix("need_data.csv", header = TRUE, sep = ",", type = "double", backingfile =
need.mat <- attach.big.matrix("need_data.desc")
```

```r
nrow(need.mat)
```

```
## [1] 49815
```

```r
ncol(need.mat)
```

```
## [1] 50
```

```r
dim(need.mat)
```

```
## [1] 49815    50
```

```r
dimnames(need.mat)
```

```
## [[1]]
## NULL
##
## [[2]]
##  [1] "HH_ID"           "REGION"          "IMD_ENG"
##  [4] "IMD_WALES"       "Gcons2005"       "Gcons2005Valid"
##  [7] "Gcons2006"       "Gcons2006Valid"  "Gcons2007"
```

```
## [10] "Gcons2007Valid"  "Gcons2008"       "Gcons2008Valid"
## [13] "Gcons2009"       "Gcons2009Valid"  "Gcons2010"
## [16] "Gcons2010Valid"  "Gcons2011"       "Gcons2011Valid"
## [19] "Gcons2012"       "Gcons2012Valid"  "Econs2005"
## [22] "Econs2005Valid"  "Econs2006"       "Econs2006Valid"
## [25] "Econs2007"       "Econs2007Valid"  "Econs2008"
## [28] "Econs2008Valid"  "Econs2009"       "Econs2009Valid"
## [31] "Econs2010"       "Econs2010Valid"  "Econs2011"
## [34] "Econs2011Valid"  "Econs2012"       "Econs2012Valid"
## [37] "E7Flag2012"      "MAIN_HEAT_FUEL"  "PROP_AGE"
## [40] "PROP_TYPE"       "FLOOR_AREA_BAND" "EE_BAND"
## [43] "LOFT_DEPTH"      "WALL_CONS"       "CWI"
## [46] "CWI_YEAR"        "LI"              "LI_YEAR"
## [49] "BOILER"          "BOILER_YEAR"
```

```
head(need.mat)
```

```
##      HH_ID REGION IMD_ENG IMD_WALES Gcons2005 Gcons2005Valid
## [1,]     1      7       1        NA     35000              5
## [2,]     2      2       4        NA     19000              5
## [3,]     3      2       4        NA     22500              5
## [4,]     4      5       1        NA     21000              5
## [5,]     5      3       1        NA        NA              3
## [6,]     6      7       2        NA        NA              4
##      Gcons2006 Gcons2006Valid Gcons2007 Gcons2007Valid
## [1,]     24500              5     22000              5
## [2,]     14900              5     16000              5
## [3,]     22500              5     22500              5
## [4,]     20500              5     18000              5
## [5,]        NA              3        NA              3
## [6,]        NA              4        NA              4
##      Gcons2008 Gcons2008Valid Gcons2009 Gcons2009Valid
## [1,]     25000              5     23000              5
## [2,]     17000              5     12800              5
## [3,]     19500              5     19500              5
## [4,]     19500              5     18500              5
## [5,]        NA              3        NA              3
## [6,]        NA              4        NA              4
##      Gcons2010 Gcons2010Valid Gcons2011 Gcons2011Valid
## [1,]     20000              5     15100              5
## [2,]     13600              5     14700              5
## [3,]     19500              5     20000              5
## [4,]     19000              5     19500              5
## [5,]        NA              3        NA              3
## [6,]        NA              4        NA              4
##      Gcons2012 Gcons2012Valid Econs2005 Econs2005Valid
## [1,]     19500              5     12500              4
## [2,]     13200              5      3100              4
## [3,]     16500              5      5600              4
## [4,]     17500              5      4900              4
## [5,]        NA              3      2500              4
## [6,]        NA              4      5000              4
##      Econs2006 Econs2006Valid Econs2007 Econs2007Valid
## [1,]     10900              4     12500              4
```

```
## [2,]      2750              4      3000             4
## [3,]      4500              4      4300             4
## [4,]      4550              4      6200             4
## [5,]      2600              4        NA             2
## [6,]      4850              4      4900             4
##      Econs2008 Econs2008Valid Econs2009 Econs2009Valid
## [1,]     11000              4      9500             4
## [2,]      2200              4      2450             4
## [3,]      3800              4      5600             4
## [4,]      7100              4      7400             4
## [5,]      1200              4      2300             4
## [6,]      4100              4      4300             4
##      Econs2010 Econs2010Valid Econs2011 Econs2011Valid
## [1,]     10000              4      7600             1
## [2,]      2150              4      3150             1
## [3,]      4750              4      5300             1
## [4,]      7650              4      8300             1
## [5,]      2650              4      2800             1
## [6,]       800              4      2000             1
##      Econs2012 Econs2012Valid E7Flag2012 MAIN_HEAT_FUEL
## [1,]      6300              2         NA              1
## [2,]      3000              2         NA              1
## [3,]      4700              2         NA              1
## [4,]      7350              2         NA              1
## [5,]      1950              2         NA              1
## [6,]      3900              2          1              2
##      PROP_AGE PROP_TYPE FLOOR_AREA_BAND EE_BAND LOFT_DEPTH
## [1,]      101       104               4       3          2
## [2,]      102       102               2       4          2
## [3,]      106       101               4       3          2
## [4,]      101       104               3       4          2
## [5,]      103       106               1       2         99
## [6,]      105       106               1       1         99
##      WALL_CONS CWI CWI_YEAR LI LI_YEAR BOILER BOILER_YEAR
## [1,]         2  NA       NA NA      NA     NA          NA
## [2,]         2  NA       NA NA      NA     NA          NA
## [3,]         1  NA       NA NA      NA      1        2004
## [4,]         2  NA       NA NA      NA     NA          NA
## [5,]         2  NA       NA NA      NA     NA          NA
## [6,]         1  NA       NA NA      NA     NA          NA
```

- The describe() function prints a description of the backing file, just like the content of a file created with the descriptorfile argument.
- Some basic functions such as ncol() and nrow() have their bigmemory implementation as well.
- But more descriptive statistics, and some serious modeling, can be achieved through two supplementary packages that use big matrices created by the bigmemory package: bigtabulate and biganalytics.
- You can easily obtain contingency tables through bigtable() and bigtabulate() commands (but the base table() will work too), for example:

```
describe(need.mat)
```

```
## An object of class "big.matrix.descriptor"
## Slot "description":
```

```
## $sharedType
## [1] "FileBacked"
##
## $filename
## [1] "need_data.bin"
##
## $dirname
## [1] "./"
##
## $totalRows
## [1] 49815
##
## $totalCols
## [1] 50
##
## $rowOffset
## [1]     0 49815
##
## $colOffset
## [1]  0 50
##
## $nrow
## [1] 49815
##
## $ncol
## [1] 50
##
## $rowNames
## NULL
##
## $colNames
##  [1] "HH_ID"           "REGION"          "IMD_ENG"
##  [4] "IMD_WALES"       "Gcons2005"       "Gcons2005Valid"
##  [7] "Gcons2006"       "Gcons2006Valid"  "Gcons2007"
## [10] "Gcons2007Valid"  "Gcons2008"       "Gcons2008Valid"
## [13] "Gcons2009"       "Gcons2009Valid"  "Gcons2010"
## [16] "Gcons2010Valid"  "Gcons2011"       "Gcons2011Valid"
## [19] "Gcons2012"       "Gcons2012Valid"  "Econs2005"
## [22] "Econs2005Valid"  "Econs2006"       "Econs2006Valid"
## [25] "Econs2007"       "Econs2007Valid"  "Econs2008"
## [28] "Econs2008Valid"  "Econs2009"       "Econs2009Valid"
## [31] "Econs2010"       "Econs2010Valid"  "Econs2011"
## [34] "Econs2011Valid"  "Econs2012"       "Econs2012Valid"
## [37] "E7Flag2012"      "MAIN_HEAT_FUEL"  "PROP_AGE"
## [40] "PROP_TYPE"       "FLOOR_AREA_BAND" "EE_BAND"
## [43] "LOFT_DEPTH"      "WALL_CONS"       "CWI"
## [46] "CWI_YEAR"        "LI"              "LI_YEAR"
## [49] "BOILER"          "BOILER_YEAR"
##
## $type
## [1] "double"
##
## $separated
## [1] FALSE
```

```r
library(bigtabulate)
```

```
## Loading required package: biganalytics

## Loading required package: foreach

## Loading required package: biglm

## Loading required package: DBI
```

```r
library(biganalytics)
bigtable(need.mat, c("PROP_AGE"))
```

```
##   101   102   103   104   105   106
## 13335  7512  8975  9856  5243  4894
```

```r
bigtabulate(need.mat, c("PROP_AGE", "PROP_TYPE"))
```

```
##      101  102  103  104  105  106
## 101 1506 2787 1514 5192  320 2016
## 102  815 3854  605  995  623  620
## 103  856 3084  697 1134 1641 1563
## 104 1737 1790  861 1344 1721 2403
## 105 1439  760  388  550  589 1517
## 106 1557  704  465  642  261 1265
```

- In the preceding listing we first obtained a frequency table of properties belonging to specific age bands (PROP_AGE variable), and then a contingency table of property age band (PROP_AGE) by property type (PROP_TYPE).
- Functions such as summary() or bigtsummary() allow users to calculate basic descriptive statistics about variables or table summaries when conditioned on other variables for example:

```r
summary(need.mat[, "Econs2012"])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##     100    2100    3250    3972    4950   25000     102
```

```r
sum1 <- bigtsummary(need.mat, c(39, 40), cols = 35,  na.rm = TRUE )

sum1[1:length(sum1)]
```

In the first call we have only printed statistics simply describing the total annual electricity consumption in 2012, whereas in the second call we have obtained descriptive statistics for each level of crossed factors: property, age, and property type, using indices of variables rather than their names.

More stat functions such as colmean(), colsum(), colmin(), colmax(), colsd(), and others are also available from the biganalytics package.

The bigmemory approach can also perform a split-apply-combine type of operation, similar to MapReduce known from Hadoop, which we will address very thoroughly in Chapter 4, Hadoop and MapReduce Framework for R.

For example, we may want to split the electricity consumption in 2012 (Econs2012) for each level of the electricity efficiency band (EE_BAND) and then calculate the mean electricity consumption for each band using sapply() function:

```
need.bands <- bigsplit(need.mat, ccols = "EE_BAND", splitcol = "Econs2012")
sapply(need.bands, mean, na.rm=TRUE)
```

```
##        1        2        3        4        5        6 
## 2739.937 3441.517 3921.660 4379.745 5368.460 5596.172
```

A similar job, run on a large big.matrix ,with over 4 millions records, took only one second to run and used as little as 68MB of RAM. Moreover, bigmemory is also capable of running more complex modeling jobs such as generalized linear models through its bigglm.big.matrix() function (which makes use of the previously introduced biglm package) and memory efficient kmeans clustering with the bigkmeans() function.

In the following snippet, we will attempt to perform a multiple linear regression predicting the electricity consumption in 2012 from a number of predictors:

```
library(biglm)
regress1 <- bigglm.big.matrix(Econs2012~PROP_AGE + FLOOR_AREA_BAND +
CWI_YEAR + BOILER_YEAR, data = need.mat, fc = c("PROP_AGE", "FLOOR_AREA_BAND"))
summary(regress1)
```

```
## Large data regression model: bigglm(formula = formula, data = getNextDataFrame, chunksize = chunksize
##     ...)
## Sample size =  49815 
##                       Coef       (95%        CI)        SE
## (Intercept)      10547.0626 -7373.4056 28467.5307 8960.2341
## PROP_AGE102       -119.0011  -196.1487   -41.8535   38.5738
## PROP_AGE103       -139.3637  -212.7222   -66.0052   36.6792
## PROP_AGE104       -127.8420  -201.2732   -54.4109   36.7156
## PROP_AGE105       -285.7916  -375.4228  -196.1605   44.8156
## PROP_AGE106       -269.9347  -441.3419   -98.5275   85.7036
## FLOOR_AREA_BAND2  1041.7280   967.8755  1115.5806   36.9263
## FLOOR_AREA_BAND3  1966.9689  1888.4783  2045.4594   39.2453
## FLOOR_AREA_BAND4  3676.7904  3571.4992  3782.0816   52.6456
## CWI_YEAR            10.7261     4.6191    16.8331    3.0535
## BOILER_YEAR        -14.8711   -22.2074    -7.5348    3.6681
##                       p
## (Intercept)      0.2392
## PROP_AGE102      0.0020
## PROP_AGE103      0.0001
## PROP_AGE104      0.0005
## PROP_AGE105      0.0000
## PROP_AGE106      0.0016
## FLOOR_AREA_BAND2 0.0000
## FLOOR_AREA_BAND3 0.0000
## FLOOR_AREA_BAND4 0.0000
## CWI_YEAR         0.0004
## BOILER_YEAR      0.0001
```

Depending on the original value labelling, the function allows us to indicate which predictors are factors (Thomas Lumley's fc argument). The implementation is quite efficient; it ran for 22 seconds and used only 70 MB of RAM for the dataset with over 4 million cases.

There are two other packages that are part of the bigmemory project. The bigalgebra library, as the name suggests, allows matrix algebra on big.matrix objects, whereas the synchronicity package provides a set of functions supporting synchronisation through mutexes, and thus can be used for multiple thread processes. A package named bigpca, authored and maintained by Nicholas Cooper, offers fast scalable Principle Components Analysis (PCA) and Singular Value Decompositions (SVD) on big.matrix objects.

It also supports multi-core implementation of the apply functionality (through bmcapply() function) and convenient transposing. The data can be imported quickly using a reference to the descriptor file, for example:

```
need.mat2 <- get.big.matrix("need_data.desc")
prv.big.matrix(need.mat2)
```

The bigpca package also enables easy subsetting of big matrices for example:

```
library(bigpca)
need.subset <- big.select(need.mat2, select.cols = c(35, 37:50), pref = "sub")
prv.big.matrix(need.subset)
```

The big.select() function conveniently creates RData, backing, and descriptor files with names specified in the pref argument.

The principal component analysis, and singular value decomposition, on big.matrix objects can be achieved through the big.PCA() and svd() functions respectively.

As these methods go beyond the scope of this book, feel free to follow the examples provided in the help files and the manual of the bigpca package.

In general, the bigmemory approach allows fast and memory-efficient management and processing of large (or even massive and out-of-memory) matrices.

The only serious limitation derives from the definition of a matrix as an R data structure, only holding one type of data. If the original dataset contains various classes of variables and is larger than the available RAM, the only way of converting differing types into a single class will be either through the ff package or by first moving data to a large enough server, or a database, and eventually importing the processed data in the required format back to R.

However, once this initial step is complete, the bigmemory family of packages offers quite an impressive array of data manipulation and processing functions. These can be further extended and speeded up by the multicore support offered by some of the R packages, which allow parallel computing.