The BigMemory Suite of Packages

Contents

1 Introduction

As put by Kane et al. (2013), it was quite puzzling when very few of the competitors, for the Million dollars prize in the Netflix challenge, were statisticians. Perhaps this is because the statistical community has always used SAS SPSS and R. The first two well-equipped tools for dealing with big data, but they are not user-friendly when trying to implement a new method. On the other hand, R is very supportive of innovation, but it was not equipped to handle large flix datasets (big data). A lot has changed in R since 2006.

The choice to work on this chapter is due to its importance in our future as data scientist with a master's degree in Big Data.

The main purpose of the `bigmemory` package is to allow us to work with big data, potentially more than the RAM available in our devices.

Alternatively, you can read only part of the X matrix, check all the variables in that part, and then read another part. To do this, simply read array X using `read.big.matrix` from the `bigmemory` package. We can also create, store, access and manipulate massive arrays.

Many specialized packages accompany the bigmemory package such as biganalytics, synchronicity, bigalgebra, and bigtabulate

The `big.matrix` class is designed for matrices containing elements of type double, integer, short, or char which is similar to the traditional R matrix but with wiser, more structured memory consumption better than the latter.

We will now explain the workflow of the bigmemory package. We will see together that `bigmemory`, with the large matrix object `big.matrix` that creates it, is a very powerful mechanism. If you are dealing with large digital matrices, you will find it very useful, but if you are dealing with big data frames, or any other non-digital matrix, Bigmemory might not be the right tool, and you should try ff, or the commercial package, RevoScaleR.

2 Data

The dataset that has the data from $CMS2010 Medicare carrier claims$ can be downloaded from the following path,

The file contains eleven (11) variables:

- Gender (`BENE_SEX_IDENT_CD`): (1) male or (2) female.
- Age (`BENE_AGE_CAT_CD`): The age of the beneficiary, declared in six categories: (1) under 65, (2) 65-69, (3) 70-74, (4) 75-79, (5) 80-84, (6) 85 and more.
- ICD-9-CM diagnostic code (`CAR_LINE_ICD9_DGNS_CD`): International Classification of Diseases, Clinical Modification Version 9 (ICD-9-CM) is a three-digit code (four digits for Diagnostic codes ICD-9-CM "E") .7 In this PUF, 926 codes of this type are observed.
- HCPCS code (`CAR_LINE_HCPCS_CD`): These are Healthcare Common Procedure Classification Coding System (HCPCS) codes (level I and level II) and take 4900 possible values in the PUF of CMS 2010 BSA carrier line items.
- BETOS code (`CAR_LINE_BETOS_CD`): This is the type of service Berenson-Eggers codes based on clinically meaningful categories of procedures and services. In this PUF, 98 codes of this type are observed.
- Number of services (`CAR_LINE_SRVC_CNT`): It is the count of the total number of services associated with the line item, between 1 and 999.

- Type of supplier code (`CAR_LINE_PRVDR_TYPE_CD`): This variable identifies the type of supplier providing the service. Examples of types of providers include clinics, physicians, and independent laboratories. In this PUF, 6 of these codes are observed.
- Type of service code (`CAR_LINE_CMS_TYPE_SRVC_CD`): identifies the type of service. Examples of types of services include medical care, surgery, and consultation. In this PUF, 20 such codes are observed.
- Place of service code (`CAR_LINE_PLACE_OF_SRVC_CD`): This variable identifies the place of service. Examples of service locations include the office, independent laboratory, inpatient hospital, outpatient hospital, and emergency room. In this PUF, 28 codes of this type are observed.
- Medicare payment amount (`CAR_HCPCS_PMT_AMT`): Medicare payment is rounded according to the rules. Note that a payment amount between $ 0 and $ 2.49 is rounded to $ 0 according to the rounding rules.

For more information, you can consult the pdf files 2010_*BSA_Carrier_Line_Items_PUF_DataDic.pdf* and 2010_*BSA_Carrier_Line_Items_PUF_GenDoc.pdf* by following this link 2010 Basic Stand Alone Carrier Line Items PUF .

3 Big memory

3.1 `read.big.matrix`

First of all, we start by creating a memory-mapped files using the bigmemory library. This command could be used only once.

`read.big.matrix` results in 2 files, a binary file-backing .bin and a shared descriptor file .desc. These 2 files, helps the user to call the data immediately for future use so we don't have to recreate the memory-mapped file all over again and waste unnecessary time.

3.2 `attach.big.matrix`

To call the memory mapped data directly, we use,

```
library(bigmemory)

x<-dget("medicares.desc")
system.time(medicares <- attach.big.matrix(x))
```

```
##    user  system elapsed
##       0       0       0
```

We can see that the time needed to read the file is almost equal to 0.

While the time to read from the .csv file directly is very much longer,

```
system.time(csvF<-read.csv('2010_BSA_Carrier_PUF.csv'))
```

```
##    user  system elapsed
##    4.63    0.28    4.91
```

3.3 description

The description including the name of columns, number of rows and columns, the binary backup file's name, of the big.matrix object medicares is as follows,

```
describe(medicares)
```

```
## An object of class "big.matrix.descriptor"
## Slot "description":
## $sharedType
## [1] "FileBacked"
##
## $filename
## [1] "medicares.bin"
##
## $dirname
## [1] "C:/Users/gabym/Desktop/Semestre 3 UPPA/Machine Learning/Projet Final/"
##
## $totalRows
## [1] 2801660
##
## $totalCols
## [1] 11
##
## $rowOffset
## [1]       0 2801660
##
## $colOffset
## [1]  0 11
##
## $nrow
## [1] 2801660
##
## $ncol
## [1] 11
##
## $rowNames
## NULL
##
## $colNames
##  [1] "BENE_SEX_IDENT_CD"         "BENE_AGE_CAT_CD"
##  [3] "CAR_LINE_ICD9_DGNS_CD"     "CAR_LINE_HCPCS_CD"
##  [5] "CAR_LINE_BETOS_CD"         "CAR_LINE_SRVC_CNT"
##  [7] "CAR_LINE_PRVDR_TYPE_CD"    "CAR_LINE_CMS_TYPE_SRVC_CD"
##  [9] "CAR_LINE_PLACE_OF_SRVC_CD" "CAR_HCPS_PMT_AMT"
## [11] "CAR_LINE_CNT"
##
## $type
## [1] "integer"
##
## $separated
## [1] FALSE
```

The dimension of the x object are similar to the dimensions of the main data set, which assures us that

nothing has been lost during the transformation as seen here,

```
dim(medicares)
```

```
## [1] 2801660      11
```

```
dim(View(read.csv('2010_BSA_Carrier_PUF.csv')))
```

```
## NULL
```

3.4 sub.big.matrix

We can now see, that x has an insignificant size and is classed as a `big.matrix` object that has a pointer to a C++ matrix on the disk,

```
pryr::object_size(medicares)
```

```
## Registered S3 method overwritten by 'pryr':
##   method      from
##   print.bytes Rcpp
```

```
## 696 B
```

```
sub.big.matrix(medicares) #or just medicares
```

```
## An object of class "big.matrix"
## Slot "address":
## <pointer: 0x0000000011e63850>
```

Also this object is backed and shared on the memory,

```
cat(sprintf('is filebacked ?: %s \n',is.filebacked(medicares)))
```

```
## is filebacked ?: TRUE
```

```
cat(sprintf('is big.matrix ? :%s \n',is.big.matrix(medicares)))
```

```
## is big.matrix ? :TRUE
```

```
cat(sprintf('is shared ?:%s \n',is.shared(medicares)))
```

```
## is shared ?:TRUE
```

3.5 summary

Now, we're going to see a summary of statistics of what is inside the medicares variable we just created by calling,

```
summary(head(medicares))
```

```
##  BENE_SEX_IDENT_CD BENE_AGE_CAT_CD CAR_LINE_ICD9_DGNS_CD CAR_LINE_HCPCS_CD
##  Min.   :1.000     Min.   :1       Min.   : NA           Min.   :99213
##  1st Qu.:1.000     1st Qu.:1       1st Qu.: NA           1st Qu.:99213
##  Median :1.000     Median :1       Median : NA           Median :99213
##  Mean   :1.667     Mean   :1       Mean   :NaN           Mean   :99213
##  3rd Qu.:1.000     3rd Qu.:1       3rd Qu.: NA           3rd Qu.:99213
##  Max.   :5.000     Max.   :1       Max.   : NA           Max.   :99213
##                                    NA's   :6             NA's   :5
##  CAR_LINE_BETOS_CD CAR_LINE_SRVC_CNT CAR_LINE_PRVDR_TYPE_CD
##  Min.   : NA       Min.   :1.000     Min.   :1.000
##  1st Qu.: NA       1st Qu.:1.000     1st Qu.:3.000
##  Median : NA       Median :1.500     Median :3.000
##  Mean   :NaN       Mean   :1.667     Mean   :2.667
##  3rd Qu.: NA       3rd Qu.:2.000     3rd Qu.:3.000
##  Max.   : NA       Max.   :3.000     Max.   :3.000
##  NA's   :6
##  CAR_LINE_CMS_TYPE_SRVC_CD CAR_LINE_PLACE_OF_SRVC_CD CAR_HCPS_PMT_AMT
##  Min.   :1                 Min.   :11                Min.   : 5.00
##  1st Qu.:1                 1st Qu.:41                1st Qu.:10.00
##  Median :1                 Median :41                Median :12.50
##  Mean   :1                 Mean   :36                Mean   :18.33
##  3rd Qu.:1                 3rd Qu.:41                3rd Qu.:15.00
##  Max.   :1                 Max.   :41                Max.   :55.00
##  NA's   :5
##   CAR_LINE_CNT
##  Min.   :  2.0
##  1st Qu.: 20.5
##  Median : 46.5
##  Mean   :102.3
##  3rd Qu.:119.8
##  Max.   :363.0
##
```

The range return the minimum and maximum between the parameters included, so while using `na.rm=TRUE`, it returns if the `na` should be omitted.

```
range(medicares[,11], na.rm = TRUE)
```

```
## [1]     1 65200
```

Which is not the case in this column.

4 Functions

Many functions support the big.matrix objects including the ones we've already used such as: * `big.matrix()` * `is.big.matrix()` * `as.big.matrix()` * `hash.mat()` * `nrow()` * `ncol()` * `dim()` * `dimnames()` * `tail()` * `head()` * `print()` * `mwhich()` * `read.big.matrix()` * `write.big.matrix()` * `rownames()` * `colnames()` * `add.cols()` * `rm.cols()` * `"["` and `"[<-"` * `deepcopy()` * `typeof()`

While some others are specific to the shared-memory functionality such as: `shared.big.matrix()` `is.shared()` `attach.big.matrix()` `describe()` `shared.deepcopy()` `rw.mutex()` `attach.rw.mutex()`

And of course some of the basic functions can also be used: `colmin()` `min()` `max()` `colmax()` `colrange()` `range()` `colmean()` `mean()` `colvar()` `colsd()` `colsum()` `sum()` `colprod()` `prod()` `kmeans.big.matrix()` `summary()` `biglm.big.matrix()` `bigglm.big.matrix()`

We will try to demonstrates the use of most of the rest in what follows.

4.1 `deepcopy()`

To duplicate the big.matrix object, Kane & Emerson provided us with `deepcopy()` function because traditional syntax would only copy the object that means the pointer to the big.matrix rather than the `big.matrix` itself as we can see here,

```
medicarescopy <- medicares
medicarescopy
```

```
## An object of class "big.matrix"
## Slot "address":
## <pointer: 0x0000000011e70c10>
```

```
medicares
```

```
## An object of class "big.matrix"
## Slot "address":
## <pointer: 0x0000000011e70c10>
```

```
# medicares pointer : 0x0000023e0521ba90
```

That's why the importance of using the `deepcopy()` function,

```
# deepcopy(x, # big.matrix
# cols = NULL,
# rows = NULL,
# y = NULL, #optional destination object
# type = NULL,
# separated = NULL, # use separated column organization of the data
# backingfile = NULL, # binary file-backing name
# backingpath = NULL, # path to the file
# descriptorfile = NULL, # .desc file-backing
# binarydescriptor = FALSE,
# shared = options()$bigmemory.default.shared # true by default
# )

medicaresdc <- deepcopy(medicares,1:10,backingfile = "medicaresdc.bin",descriptorfile = "medicaresdc.des
dim(medicaresdc)
```

```
## [1] 2801660      10
```

Now we can see, that the new copy has a different pointer and is a totally different big.matrix object,

```
medicaresdc
```

```
## An object of class "big.matrix"
## Slot "address":
## <pointer: 0x0000000014c2cd80>
```

4.2 dimnames

dimnames can retrieve or set the dimnames of an object

```
dimnames(medicares)
```

```
## [[1]]
## NULL
##
## [[2]]
##  [1] "BENE_SEX_IDENT_CD"          "BENE_AGE_CAT_CD"
##  [3] "CAR_LINE_ICD9_DGNS_CD"      "CAR_LINE_HCPCS_CD"
##  [5] "CAR_LINE_BETOS_CD"          "CAR_LINE_SRVC_CNT"
##  [7] "CAR_LINE_PRVDR_TYPE_CD"     "CAR_LINE_CMS_TYPE_SRVC_CD"
##  [9] "CAR_LINE_PLACE_OF_SRVC_CD" "CAR_HCPS_PMT_AMT"
## [11] "CAR_LINE_CNT"
```

```
# dimnames(x) <- value #a possibel value
```

4.3 extract or replace elements

To extract or replace big.matrix elements we use

```
# medicares[i, j, drop]

medicares[1, 1, drop=FALSE]
```

```
##      BENE_SEX_IDENT_CD
## [1,]                 5
```

i and j are the row and column respectively, while the use of drop is to reduce the dimensions of the array-like R object returned.

```
medicares[1, 1, drop=TRUE] # reduces the dimensions with drop=TRUE
```

```
## [1] 5
```

4.4 flush()

flush() writes any modified information or data (in RAM) to the file-backing (in disk) and returns a TRUE or FAlSE to indicate the successfulness of the operation

```
medicarescopy[1,1] <- 5
```

```
## Warning in SetElements.bm(x, i, j, value): Assignment will down cast from double to integer
## Hint: To remove this warning type:  options(bigmemory.typecast.warning=FALSE)
```

```
# flush(medicarescopy)
medicarescopy[1,1]
```

```
## [1] 5
```

```
medicares[1,1]
```

```
## [1] 5
```

It might seem not a big deal in our example here, but in fact it could be useful for improving performance in cases where allowing the operating system to decide on flushing creates a bottleneck (likely near the threshold of available RAM).

```
# GetMatrixSize returns the size of the created matrix in bytes
GetMatrixSize(medicarescopy)
```

```
## [1] 0
```

```
# Length of a big.matrix object
length(medicarescopy)
```

```
## [1] 30818260
```

4.5 morder and mpermute

`morder` (based on the R's function `order`) and `mpermute` do permute and order or sort the row indices to rearrange the `big.matrix` object the way we ask for.

`morder` returns an ordering vector while the `mpermute` does not return anything but makes changes in the object. As we can see, we can also do that on the columns with `morderCols` and `mpermuteCols`.

4.6 mwhich

The `mwhich` function for "multi-which" which is based on R's `which` function and it provides us with efficient row selections for the big.matrix objects and high perfomance comparisons between `big.matrix`. It can be used as follow,

```
# mwhich(x, cols, vals, comps, op = 'AND')
indices <- mwhich(medicares, 'CAR_LINE_CNT', 46.5, "le")

cat(sprintf('number of CAR_LINE_CNT < 46,5: %s \n',length(indices)))
```

```
## number of CAR_LINE_CNT < 46,5: 2613998
```

```
cat(sprintf('The percentage of this number from the total is: %s %% \n',100*round(length(indices)/dim(m
```

```
## The percentage of this number from the total is: 93 %
```

Where `x` is the big.matrix object `cols` is the column that we're studying (could've been 11 instead of `CAR_LINE_CNT`) `vals` is the value we're comparing to and `comps` is the logical operator (eq,le,ge, ...). `op` can be used to do comparisons on multiple columns (with `AND` or `OR`) as a union.

There are many other uses to the `mwhich` function.

4.7 write.big.matrix

The write.big.matrix helps us write the big.matrix object into any kind of file we specify,

```
# write.big.matrix(x, filename, row.names = FALSE, col.names = FALSE, sep = ",")
system.time(write.big.matrix(medicaresdc,"medicaresdc.txt"))
```

```
##    user  system elapsed
##    9.95    0.10   10.08
```

5 biglm.big.matrix() and biglm() for Linear Regression

One of the basic functions, is Using the Lumley's `biglm.big.matrix()` package. `biglm` stands for "bounded memory linear regression" which can also be used by calling `bigglm.big.matrix()`

```
library(biglm)
```

```
## Loading required package: DBI
```

```
require(foreach)
```

```
## Loading required package: foreach
```

```
require(DBI)
library(biganalytics)
```

```
lm.1 = biglm.big.matrix( CAR_LINE_ICD9_DGNS_CD ~ BENE_AGE_CAT_CD, data = medicares, fc = "BENE_AGE_CAT_

print(summary(lm.1))
```

```
## Large data regression model: biglm(formula = formula, data = data, ...)
## Sample size =  2670499
##                       Coef     (95%      CI)     SE p
## (Intercept)      539.8049 539.1525 540.4573 0.3262 0
## BENE_AGE_CAT_CD2 -18.5830 -19.5162 -17.6498 0.4666 0
## BENE_AGE_CAT_CD3 -20.3844 -21.3184 -19.4504 0.4670 0
## BENE_AGE_CAT_CD4 -19.4645 -20.4163 -18.5127 0.4759 0
## BENE_AGE_CAT_CD5 -16.6272 -17.6011 -15.6533 0.4869 0
## BENE_AGE_CAT_CD6  -6.8453  -7.8188  -5.8717 0.4868 0
```

In this example, we try to use Age of the beneficiary to try to predict the most affected by the diseases. As it appears, those who are 85 and older(6) have more deseases than the others.

```
lm.2 = biglm.big.matrix( CAR_HCPS_PMT_AMT ~ BENE_AGE_CAT_CD, data = medicares, fc = "BENE_AGE_CAT_CD")
#biglm
print(summary(lm.2))
```

```
## Large data regression model: biglm(formula = formula, data = data, ...)
## Sample size =  2801660
##                      Coef    (95%     CI)     SE     p
## (Intercept)      80.1114 79.5439 80.6788 0.2837 0e+00
## BENE_AGE_CAT_CD2  4.4180  3.6065  5.2296 0.4058 0e+00
## BENE_AGE_CAT_CD3  5.2403  4.4277  6.0528 0.4063 0e+00
## BENE_AGE_CAT_CD4  4.3875  3.5591  5.2159 0.4142 0e+00
## BENE_AGE_CAT_CD5  1.4129  0.5646  2.2611 0.4241 9e-04
## BENE_AGE_CAT_CD6 -4.8883 -5.7369 -4.0396 0.4243 0e+00
```

while in this example, it appears that those who have between 70 and 74 years of age (3) have the biggest amount of paiments.

`shared.big.matrix` can help us create a `big.matrix` object on a shared memory

5.1 `biglm` mathematics based equations

The mathematics behind biglm are based on considering the linear model n > p:

$$y = X\beta + \epsilon$$

so the least square estimation is

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

That if we encounter a tall data, the R's basic lm.fit use $O(np + p^2)$ of the memory. This is where the Biglm methods come for rescue, as it tries to compute the decomposition of $X = QR$ and $Q^T y$ which leads us to the value of β

$$R\beta = Q^T y$$

The function uses only $O(p^2)$ in front of p variables and then, by calling update, the fitted object can be updated with more data.

6 bigkmeans

We were also able to use bigkmeans that can apply the kmeans method on the specified data,

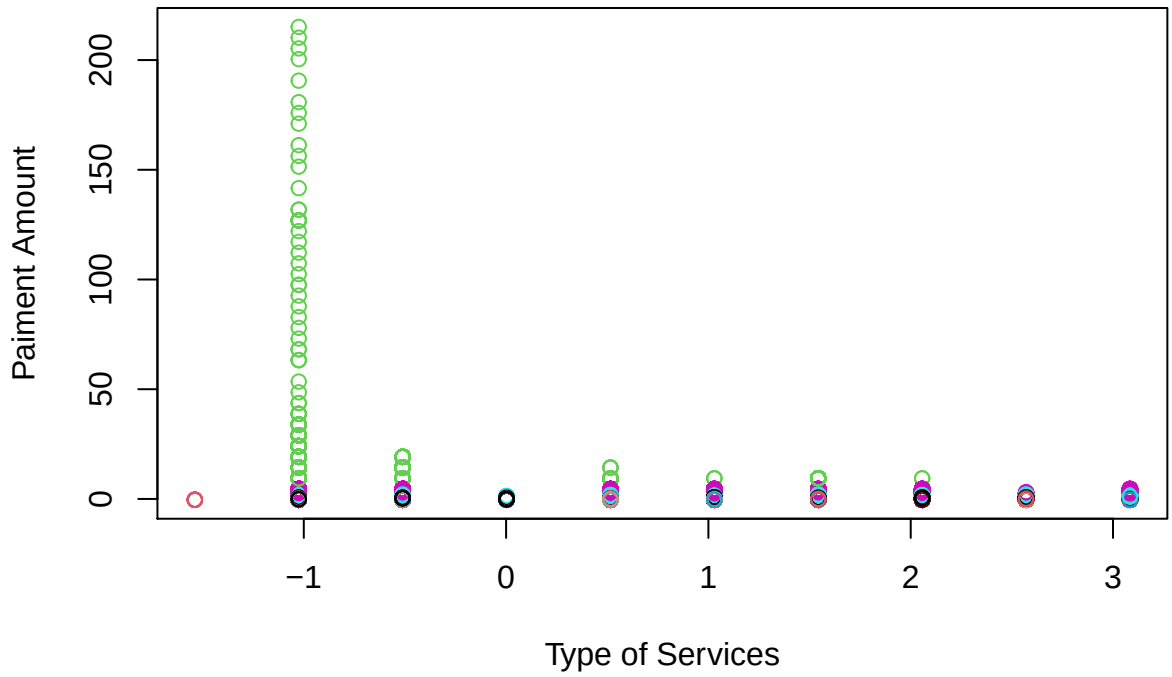
```
# install.packages('factoextra')
library(factoextra)
```

```
## Loading required package: ggplot2
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```r
# bigkmeans(medicares, 2, iter.max = 10, nstart = 1)
bkm<- bigkmeans(scale(medicares[,10]), 10)
#
# fviz_cluster(bkm, data = bkm$centers,
#              palette = c("#2E9FDF", "#00AFBB"),
#              geom = "point",
#              ellipse.type = "convex",
#              ggtheme = theme_bw()
#              )

plot(x = scale(medicares[,8]), y = scale(medicares[,10]), xlab = 'Type of Services', ylab= 'Paiment Amou
```



7 Parallelization

`big.matrix` also provides us with a user friendly parallelizing methods as * Its structure support the shared memory for parallel computing * using reference to avoid uncalled for copies * column-major format that can work under the legacy linear algebra packages

7.1 `foreach`

At first, we call the parallel library and check the number of cores on the pc,

```
library(parallel)
cores <- detectCores() ## How many cores do I have?
print(cores)
```

```
## [1] 12
```

Than we call the `foreach` function to decompose the data on multiple `chunks` and `cluster` controlled by the different cores,

```
dataX <- attach.big.matrix(dget("medicares.desc"))
Xdes <- describe(dataX)

XtX.big <- function(X.des, ng = 1) {
readchunk <- function(X, g, size.chunk) {
rows <- ((g - 1) * size.chunk + 1):(g * size.chunk)
chunk <- X[rows,]
}
res <- foreach(g = 1:ng, .packages = c("bigmemory"), .combine = "+") %dopar% {
X <- attach.big.matrix(X.des)
size.chunk <- nrow(X) / ng
chunk.X <- readchunk(X, g, size.chunk)
# chunk.Y <- readchunk(Y, g, size.chunk)
term <- t(chunk.X) %*% chunk.X
}
return(res)
}
library(doSNOW)
```

```
## Loading required package: iterators
```

```
## Loading required package: snow
```

```
##
## Attaching package: 'snow'
```

```
## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, clusterSplit, makeCluster, parApply,
##     parCapply, parLapply, parRapply, parSapply, splitIndices,
##     stopCluster
```

```
cl <- makeCluster(4)
registerDoSNOW(cl)
cl <- makeCluster(4)
res.big <- XtX.big(Xdes,ng=10)
stopCluster(cl)
summary(res.big)
```

```
##  BENE_SEX_IDENT_CD   BENE_AGE_CAT_CD     CAR_LINE_ICD9_DGNS_CD
```

```
##  Min.   :  6946483   Min.   : 14682025   Min.   : NA
##  1st Qu.:  8180718   1st Qu.: 17264762   1st Qu.: NA
##  Median : 14682025   Median : 39816029   Median : NA
##  Mean   : 86394888   Mean   :188620480   Mean   :NaN
##  3rd Qu.:106979985   3rd Qu.:232639660   3rd Qu.: NA
##  Max.   :352814300   Max.   :766036460   Max.   : NA
##  NA's   :4           NA's   :4           NA's   :11
##  CAR_LINE_HCPCS_CD CAR_LINE_BETOS_CD CAR_LINE_SRVC_CNT   CAR_LINE_PRVDR_TYPE_CD
##  Min.   : NA       Min.   : NA       Min.   :8.946e+06   Min.   :  6946483
##  1st Qu.: NA       1st Qu.: NA       1st Qu.:1.431e+07   1st Qu.: 12146754
##  Median : NA       Median : NA       Median :1.055e+08   Median : 15378801
##  Mean   :NaN       Mean   :NaN       Mean   :2.976e+08   Mean   : 96337457
##  3rd Qu.: NA       3rd Qu.: NA       3rd Qu.:2.262e+08   3rd Qu.:147356360
##  Max.   : NA       Max.   : NA       Max.   :1.488e+09   Max.   :333030685
##  NA's   :11        NA's   :11        NA's   :4           NA's   :4
##  CAR_LINE_CMS_TYPE_SRVC_CD CAR_LINE_PLACE_OF_SRVC_CD CAR_HCPS_PMT_AMT
##  Min.   : NA               Min.   :1.029e+08         Min.   :3.330e+08
##  1st Qu.: NA               1st Qu.:1.460e+08         1st Qu.:5.594e+08
##  Median : NA               Median :2.233e+08         Median :1.488e+09
##  Mean   :NaN               Mean   :1.429e+09         Mean   :2.102e+10
##  3rd Qu.: NA               3rd Qu.:2.319e+09         3rd Qu.:4.295e+09
##  Max.   : NA               Max.   :4.746e+09         Max.   :1.356e+11
##  NA's   :11                NA's   :4                 NA's   :4
##   CAR_LINE_CNT
##  Min.   :1.055e+08
##  1st Qu.:1.203e+08
##  Median :2.419e+08
##  Mean   :2.880e+10
##  3rd Qu.:2.881e+09
##  Max.   :1.953e+11
##  NA's   :4
```

7.2 `foreach` mathematics based equation

This parallel computing with `foreach` is based in its mathematical background on the computation by chunk of the big.matrix objects X and Y

$$(X^T Y) = \sum_{g=1}^{G} X_{(g)}^T Y_{(g)}$$

8 Conclusion

To conclude, the `bigmemory` package has proven to be very useful with optimized memory consumption. The problems encountered were not finding all the functions listed in the package when called in the R script.

Another good impression, is that this library can apply many important methods from those we have learned in other courses such as PCA (principal component analysis) by installing the `bigpca` library, which can also be said for the `bigrf` (RadomForest), `biglars` (least regression and LASSO), `biglasso` (lasso model fitting) and `bigstatr` (for statistical purposes).

The downside of this package is that it only works with matrices, which means a faster way of calculating for the pc but a more difficult way to understand and visualize well from the user's point of view, this is where the `ff` and the `RevoScaleR` packages have the upper hand in the battle.