**Programming for Big Data**

**Assignment 3**

**D10126532 John Warde DT230B**

**NOTE: Must be exported/saved to a PDF before submission!!!**

# 1. Tax Underpayment

Imagine you are an analyst with the Revenue Commissioners. Part of your job is identifying characteristics of people who are likely to have underpaid tax.

Imagine you have been asked to design a solution to automate the process in R and generate regular reports.

You can assume that

* Revenue has a record of everyone's tax information, such as historical tax payments, income, what income tax band they are in, whether they have paid their property tax, whether they have underpaid tax before, etc.
* Revenue has detailed personal information about each taxpayer, such as age, full address, family status, employment status, profession and so on.
* Revenue maintains all of its data in one giant multi-gigabyte database table with one row per taxpayer.

Note: This exercise is about the data analytics process with R, but you do \*not\* need to write any R code in your answers.

Proposed Fields

|  |  |
| --- | --- |
| Name | Description |
| * name\_first * name\_last * dob * Revenue has a record of everyone's tax information, such as historical tax payments, income, what income tax band they are in, whether they have paid their property tax, whether they have underpaid tax before, etc. * Revenue has detailed personal information about each taxpayer, such as age, full address, family status, employment status, profession and so on. |  |

## Part A. Suitable Analyses

Part A. Suitable Analyses: Propose at least three hypotheses about the data which may help find cases of underpayment of tax. For instance you might consider whether the self-employed pay less tax than PAYE workers, or the relationship between the amount of tax paid and other characteristics of each taxpayer. You can assume any attributes you like are available in the database, so be creative. Which graphical plots and which statistical tests that R provides are suitable to investigate each hypothesis you have proposed? Give concrete examples.

## Part B. Data Storage

Part B. Data storage Comment briefly on the suitability (or otherwise) for this task of each of the out-of-memory data access methods which we have seen in the lecture (SQL, bigmemory, RHadoop). Suggest some ways in which the data set could be broken down into chunks for incremental processing with an out-of-memory storage technology such as MySQL or bigmemory.

# 2. Stock Performance

For the non-parallel solution, I coded the functionality into self-contained re-usable functions as much as possible.

* Chose to read in the numeric data and apply the stock names using factors – allows more flexibility if stocks are added/removed.

For parallel solution,

* In the getAveragesPerStock() function, I filtered the stockData on the current stock code within the parallelised foreach/%dopar% loop because the cluster is on the same machine. If a cluster of physical machine was available to me I would consider taking the following line out of the   
    
  dfForStock <- dfStock[dfStock$stock==stockNamesAsLevels[i],]  
    
  to possibly allow the transfer of a small amount of data to the other machines, however I would benchmark the function before and after to get a clear picture of performance.

For out of memory solution:

* Would consider using the SQL AVG() function to the an average on the database side rather that bringing all the rows back to R and using the mean() function there. Again, I would benchmark duration again to make a decision on the best option.  
    
  "SELECT AVG(gain) FROM stock\_gains WHERE stock = '%s' AND day <= %d ORDER BY day"
* Used cat(“Custom Message: ”, SomeValue, “\n") to track down problems but this caused problems when running code in parallel (%dopar%)

# 3. R and Hadoop

Blah

We know the benefits of Hadoop:

Scalable, parallel, piecewise execution of tasks

Single and multi-node clusters in a common interface

Why use R as a Hadoop front-end?

You’re doing stats on big data sets

You have analyses coded up that you’d like to use on bigger

data sets

You like how little code you need for M/R programs in R

to.dfs and from.dfs access serialised HDFS storage

groups = rbinom(n=50, size=32, prob = 0.4)

table(groups)

groupsHandle = to.dfs(groups)

resultsHandle <-

mapreduce(

input = groupsHandle,

map = function(., v) keyval(v, 1),

reduce =

function(k, vv)

keyval(k, length(vv))))

results <- as.data.frame(from.dfs(resultsHandle))

Say you have functions of the following form

f <- function(input) {... ,

mapreduce(input,

map=...

reduce=...

...)}

where input is a HDFS handle

These can be chained, since mapreduce() returns a

HDFS handle

If f1 and f2 are such functions, f2(f1(input)) is a chained MapReduce job

The output of f1 is the input to f2

MapReduce key,value outputs can be retrieved into R memory with from.dfs

So we can continue processing the data within R. . .

... doing statistical tests

... plotting graphs

This works when the results of the MapReduce job are significantly smaller than the input data