Using MonetDB and dplyr to Work with Large HCUP NIS Data Files

Charles DiMaggio
Bellevue-NYU Trauma Service

Charles.DiMaggio@nyumc.org

November 7, 2016

Contents

1 Introduction

This is an addendum to notes related to Nationwide Inpatient Sample analyses and Nationwide Emergency Department Sample analyses found elsewhere on this site. It presents some approaches to dealing with very large data files that result from combining multiple years of administratively collected discharge data like HCUP. This issue is particularly relevant when conducting multi-year analyses of large complex survey data. To most appropriately assess year-to-year trend, it's necessary to combine and adjust for survey sampling, weighting and clustering using the full multi-year data set. This, though, results in a very large data file that taxes even the most robust systems.

As a rule of thumb, an R file with 100 million rows and 5 variables takes up about 4GB. I recently completed an analysis of a 12-year HCUP NIS file [1], which was nearly 100 million rows long with approximately 25 variables for an estimated 250GB. A 7-year NEDS file I worked with had nearly 200 million entries and 45 variables of interest for about double that size. Since R loads everything into RAM, this presents computational memory difficulties on most any machine. [2] One work around is to use an out-of-memory data base. The data reside outside of a computer's RAM, i.e. in a file somewhere on your system, and R only brings results of data manipulations or analyses into your physical memory. This kind of approach has great promise to utilize the robust statistical powers of R in a large-data setting.

2 MonetDB and dplyr

MonetDB is a column-oriented database. As opposed to traditional spreadsheets or relational databases like SQL, column-oriented data bases store each column or variable as a separate file. They are more efficient because you only have to access the data in which you are interested. They are smaller because it is easier to compress categorical data. Column-oriented DBMS like MonetDB have been proposed as a more efficient approach to large and big data sets.

MonetDB can be accessed with the R MonetDB.R and hrefhttps://www.monetdb.org/blog/monetdblite-rMonetDBLite packages. MonetDBLite is intended to provide a simple interface to an out-of-memory MonetDB database via the R DBI package in much the same way the perhaps more familiar RSQLite does. The connection allows the user to submit SQL queries within R. Useful, but not a way to get at the full analytic power of R. But. A MonetDBLiteconnection can be "wrapped" in a dplyr table connection that allows the full functionality of plyr R commands rather than SQL-language queries.

A quick note of caution: MonetDB.R predates MonetDBLite, and was once the only way to access MonetDB with R. But, it was a somewhat complicated process that involved installing a free-standing instance of MonetDB, then setting up a MonetDB server and

[1]

[2] Excepting perhaps computer clusters or a supercomputer.(See https://www.olcf.ornl.gov/titan/)

connecting to it. MonetDBLite has superseded that and made using MonetDB with R pretty painless. But...although MonetDBLite seems to require and depend on MonetDB.R, databases created with MonetDB are not compatible with MonetDBLite. (Ouch.) Bottom line. Just use MonetDBLite.

3 Installation and Test Run

3.1 Install Packages

Install MonetDB.R and MonetDBLite. MonetDB.R installs the client version of MonetDB (client-side means it is on the users machine, as opposed to server-side or out of memory). This also installs the DBI package. DBI is a general interface to DBMS's. You have to load DBI separately and explicitly each time you use MonetDBLite.[3] But, each particular DBMS needs an additional package to handle the specifics for that particular DBMS, e.g. SQLLite. The MonetDBLite package serves that purpose and makes it no longer necessary to to set up and maintain a separate MonetDB server for each session.[4]

The default MonetDB.R and MonetDBLite installation directs to CRAN, but when I tried installing the packages were not there. This may have since been corrected, but the code below sets the repository directly to the MonetDB site and worked the last time I tried in June 2016.

Then install dplyr.

```
# from https://www.monetdb.org/blog/monetdblite-r

install.packages("MonetDB.R", repos="http://dev.monetdb.org/Assets/R/")
install.packages("MonetDBLite", repos="http://dev.monetdb.org/Assets/R/")
# NB: MonetDBLite site set the repository as cran.rstudio,
# but the file was not there.  I found it at the monetdb site.

install.packages(dplyr)
```

3.2 Test Run

3.2.1 Toy Example

The following code takes MonetDB and plyr for a quick ride.[5] We begin by creating a directory in which to hold the MonetDB database we will be writing. In this case, for

[3] At one point it MonetDBLite loaded DBI as a dependency, but that doesn't seem to happen anymore...
[4] This was previously necessary when working with MonetDB.R
[5] This code is taken most shamelessly from the MonetDBLite site itself.

demonstration purposes, we create a temporary directory, but in the real world you will want to create a permanent folder somewhere on your system.

The next step is to specify a database connection within R to the directory or folder that will hold the MonetDB database. This is done with the DBI::dbConnect function and is saved to the object "con". Note that the first argument to dbConnect specifies that this connection is a MonetDB database. The following step actually writes the MonetDB database. Here, we take the venerable mtcars data set that comes with R and use DBI::dbWriteTable to write it to the MonetDB folder using the MonetDB connection. Within the MonetDB directory, we have specified that the table is called "mtcars", but we could have named it anything. The dbListTables function assures that the MonetDB database contains a table called "mtcars"; dbListFields returns the fields (or variables) in that table.

The DBI:dbGetQuery function allows us to pass SQL statements to the database.[6] Depending on your familiarity and comfort with MonetDB and SQL, you could happily use this approach for all your data manipulation needs.

Alternatively, you could use dplyr to connect to the MonetDB table and utilize the increasingly popular plyr-like syntax to work with the database. The dplyr code below uses src_monetdb() to establish a connection to the MonetDB database which is saved as the object "ms", then creates a dplyr table of the "mtcars" MonetDB table in the directory. We can then use dplyr (and plyr) functions on that table.[7]

Begin by loading "dplyr".[8]

```
library(MonetDB.R)
library(MonetDBLite)
library(DBI)
dbdir <- tempdir() # creates a temporary directory
con <- dbConnect(MonetDB.R(), embedded=dbdir) #DBI fx to connect to MonetDB
dbWriteTable(con, "mtcars", mtcars) # writes DBMS "mtcars" from  mtcars
dbListTables(con)
dbListFields(con,"mtcars")
dbGetQuery(con, "SELECT MAX(mpg) FROM mtcars WHERE cyl=8;") # sql query


# library(plyr)
library(dplyr) # run similar query in R using dplyr
ms <- src_monetdb(embedded=dbdir) # connect to a MonetDB database
mt <- tbl(ms, "mtcars") # creates a dplyr version of a dataframe
mt %>% filter(cyl == 8) %>% summarise(max(mpg))
```

[6] The MonetDB dialect of SQL differs in some respects from standard SQL...

[7] When working with databases, dplyr uses an approach called "lazy evaluation". It tries, as much as possible, not to bring anything into R's memory unless and until absolutely necessary. It accomplishes this by translating R commands into SQL and sending a single statement to the out-of-memory database. Practically this means you can work with very large databases without taxing your memory. see https://cran.r-project.org/web/packages/dplyr/vignettes/databases.html

[8] If you plan to use both "plyr" and "dplyr", load "plyr" first, else you will run into problems.

3.2.2 Reading .csv files by writing a schema

The following vignette is stolen from Bob Rudis and is an example of the much more likely scenario of reading in a .csv file into MonetDB. CSV files are by definition comma separated and usually come with headers. MonetDB by default expects no headers and uses the — as a separator. There is a MonetDB.R function to read in .csv files (monetdb.read.csv) but the following general approach creates a MonetDB table with a schema into which to copy the .csv data which can be used for more complex files.

We will again work with the venerable mtcar data set. For demonstration purposes, we will create a .csv version of it. We again create a directory to hold our MonetDB table. This time, rather than create a temporary directory, we first use our operating system (not shown) to create a folder called "testMonet" on our desktop. As before, we then use dbConnect to establish a connection to it.

The next couple of lines of code create a schema or container for the MonetDB table. First read in a 1000 rows from the full .csv file into a smaller representative .csv. Then run the sprintf() on that representative file to return "... a character vector containing a formatted combination of text and variable values."

We then create the table with dbSendQuery, and populate the table with data from the mtcars.csv file using a second dbSendQuery.[9]

We then connect to the out-of-memory directory and create a dplyr table version of the MonetDB table.

```
library(MonetDBLite)
library(MonetDB.R)
library(dplyr)

# convert the mtcars dataframe to a csv file for the example
write.csv(add_rownames(mtcars, "auto"),
"~/testMonet/mtcars.csv", row.names=FALSE)

# MonetDB connection to a permanent file on desktop
mdb <- dbConnect(MonetDBLite(), "~/Desktop/testMonet")

# read in small amount of cvs file to get info for schema
guess <- read.csv("mtcars.csv", stringsAsFactors=FALSE, nrows=1000)
create <- sprintf("CREATE TABLE mtcars ( %s )",
paste0(sprintf('"%s" %s', colnames(guess),
sapply(guess, dbDataType, dbObj=mdb)), collapse=","))
```

[9]The "OFFSET 2" option starts adding data on the second line, i.e. below the header.

```
create

 # "CREATE TABLE mtcars ( \"auto\" STRING,\"mpg\" DOUBLE PRECISION,
 # \"cyl\" INTEGER, # \"disp\" DOUBLE PRECISION,\"hp\" INTEGER,
 # \"drat\" DOUBLE PRECISION,\"wt\" DOUBLE PRECISION,
 # \"qsec\" DOUBLE PRECISION,\"vs\" INTEGER,\"am\" INTEGER,
 # \"gear\" INTEGER,\"carb\" INTEGER )"

dbSendQuery(mdb, create) # create the table using the schema

dbSendQuery(mdb, "COPY OFFSET 2
INTO mtcars
FROM '~/Desktop/testMonet/mtcars.csv' USING  DELIMITERS ','")

dbListTables(mdb)

# connect R via dplyr to the MonetDB database
mdb_src <- src_monetdb(embedded="/Users/charlie/Desktop/testMonet")
#  create dplyr table
mdb_mtcars <- tbl(mdb_src, "mtcars")

count(mdb_mtcars, cyl)
```

3.2.3 Reading .csv files directly

I have found creating a schema to be difficult for more complex files like HCUP NIS and NEDS. The following code presents an example using the monetdb.read.csv() function. As you can see, this is much more straightforward. It has in general worked well for me, and is my preferred approach. I'm using a teaching file of .csv digitalis study data. First, create a connection to the .csv file. Then, read it into a MonetDB data base using the MonetDB.R::monetdb.read.csv() function.

3.3 Working with dplyr

```
library(MonetDBLite)
library(MonetDB.R)

mdb <- dbConnect(MonetDBLite(), "/Users/charlie/Desktop/testMonet")

monetdb.read.csv(mdb, "/Users/charlie/Downloads/dig.csv", "dig")
dbListTables(mdb)
dbGetQuery(mdb, "SELECT COUNT(*) FROM dig")  # 6800
```

Now, connect to the MonetDBLite table with dplyr and run some commands. Note that you must use dplyr commands on the connection object rather than base commands.

```
library(dplyr)

mdb_src <- src_monetdb(embedded = "/Users/charlie/Desktop/testMonet")
mdb_dig <- tbl(mdb_src, "dig")
head(mdb_dig)
nrow(mdb_dig)  # 6800
str(mdb_dig)  # returns structure of the connection object
glimpse(mdb_dig)  # dplyr fx returns neater structure of the table or df
mean(mdb_dig$AGE)  # base fails
summarise(mdb_dig, mean_age = mean(AGE))  # dplyr works

# dplyr command with pipelines (MagrittR)
mdb_dig %>% group_by(TRTMT) %>% summarise(mean_age = mean(AGE))
# TRTMT mean_age (int) (dbl) 1 0 63.54746 2 1 63.41890
```

4 Combining HCUP Files into a single large multi-year file

The following code goes through two approaches to combining large files into a very large out-of-memory multi-year file. The first example illustrates combining 7 base NEDS files by primarily using DBI::dbWriteTable with the appendTRUE option. This puts all the data manipulation outside of R and is my preferred approach. The second example illustrates combining 12-years of NIS data. It involves a fair amount of data processing within R using dplyr. I then present some notes on working with large survey data via the MonetDBLite connection using the sqlsurvey package. Finally, there is an appendix of several failed trial and error attempts, some of which broke files or R or both.

4.1 Combining NEDS files using DBI::dbWriteTable to append

The key here is the "appendTRUE, overwrite=FALSE" option of the DBI function dbWriteTable. This example assumes you have already read the HCUP NEDS .csv files into a separate R dataframe for each year. **Caution: Don't use dots to name your database tables.** MonetDB SQL queries won't work if the sql table name includes a dot. Use underscores.

Begin by creating a folder to hold your large multi-year file, then connect to it using DBI::dbConnect(). Read the first year of data (2006) into R. Then, because MonetDB is very picky about variable names in tables, set all the names to lower case, and change the "year" variable to "yr". (Year is a restricted name for MonetDB). Then read the file into

a MonetDB table called "neds". The gc() (for "garbage collection") command cleans and frees up R memory after memory-intensive operations.

The process is repeated with the R files for 2007 and 2008. This should go smoothly because those files have the same names and variable types as the 2006 file. After each year of data is added, check and note the number of entries in the table.

```
library(DBI)
library(MonetDB.R)
library(MonetDBLite)

library(plyr)
library(dplyr)

# create connection to a MonetDBLite folder
mdb <- dbConnect(MonetDBLite(), "~/NEDSMonet")

# read write yearly NEDS R files into R fix varaible names for sql write
# initial table as individual table append additional years
neds.2006.core <- readRDS("~/NEDS_2006_Core.rds")
names(neds.2006.core) <- tolower(names(neds.2006.core))  # fix names for sql
names(neds.2006.core)[names(neds.2006.core) == "year"] <- "yr"
dbWriteTable(mdb, "neds", neds.2006.core)
rm(neds.2006.core)
gc()

# check results of append
dbListTables(mdb)
dbListFields(mdb, "neds")
dbGetQuery(mdb, "SELECT COUNT(*) FROM neds")  # 25702597, 52330520, 80777668,109638715,
dbGetQuery(mdb, "SELECT AVG(yr) AS AverageYear FROM neds;")  # 2006, 2006.509, 2007.034
commonNames <- dbListFields(mdb, "neds")  # names to which to restric additional years

# read 2007 table, fix names, append to existing table 'neds'
neds.2007.core <- readRDS("~/NEDS_2007_Core.rds")
names(neds.2007.core) <- tolower(names(neds.2007.core))
names(neds.2007.core)[names(neds.2007.core) == "year"] <- "yr"
dbWriteTable(mdb, "neds", neds.2007.core, append = TRUE, overwrite = FALSE)
rm(neds.2007.core)
gc()

# 2008
neds.2008.core <- readRDS("~/NEDS_2008_Core.rds")
names(neds.2008.core) <- tolower(names(neds.2008.core))
names(neds.2008.core)[names(neds.2008.core) == "year"] <- "yr"
```

```
dbWriteTable(mdb, "neds", neds.2008.core, append = TRUE, overwrite = FALSE)
rm(neds.2008.core)
gc()
```

We hit our first wrinkle at the 2009 data. Some of the variables have changed. At this point you have a decision to make. Do you need to create a new variable? Drop an older one? Rename something? It will vary based on your analysis need. In my case, it was sufficient to restrict to the intersection of variable names for the existing and the additional table.

```
# 2009, variables change, need to restrict to intersection of names
neds.2009.core <- readRDS("~/NEDS_2009_Core.rds")
names(neds.2009.core) <- tolower(names(neds.2009.core))
names(neds.2009.core)[names(neds.2009.core) == "year"] <- "yr"
neds.2009.core <- neds.2009.core[commonNames]  # restrict to common variables
dbWriteTable(mdb, "neds", neds.2009.core, append = TRUE, overwrite = FALSE)
rm(neds.2009.core)
gc()

# 2010
neds.2010.core <- readRDS("~/NEDS_2010_Core.rds")
names(neds.2010.core) <- tolower(names(neds.2010.core))
names(neds.2010.core)[names(neds.2010.core) == "year"] <- "yr"
neds.2010.core <- neds.2010.core[commonNames]
dbWriteTable(mdb, "neds", neds.2010.core, append = TRUE, overwrite = FALSE)
rm(neds.2010.core)
gc()
```

We hit another bump in the road at 2011. This time, the variable *names* match, but the variable *types* differ. This was much more problematic as MonetDB is very finicky about data types. After much trial and error (mostly error and very time consuming) I found the only thing that worked was reading the problematic additional year of data into the database as a separate table. Then concatenating this new table of data to the existing multi-year table using SQL UNION. This required dropping an additional field that didn't match up. After ensuring the union operation worked, I removed the single year of data from the database.

The process is repeated for the final year of data, along with a couple of checks to ensure things worked.

```
################### 2011 ######################### differing variables caused errors and
################### crashes following approach only thing tht seemed to work

neds.2011.core <- readRDS("~/NEDS_2011_Core.rds")
names(neds.2011.core) <- tolower(names(neds.2011.core))
names(neds.2011.core)[names(neds.2011.core) == "year"] <- "yr"
neds.2011.core <- neds.2011.core[intersect(commonNames, names(neds.2011.core))]  # resti
dbSendQuery(mdb, "alter table neds drop hosp_region;")  # drop non-matching var hosp_re
```

```
# read 2011 file into monet
dbWriteTable(mdb, "neds_11", neds.2011.core)

# concatenate with existing neds table using sql union (first try returned
# error about non-matching fields that had to be fixed)
setdiff(dbListFields(mdb, "neds"), dbListFields(mdb, "neds_11"))
dbSendQuery(mdb, "alter table neds drop intent_self_harm;")

dbGetQuery(mdb, "CREATE TABLE neds_06_11 AS
\tSELECT *\t
    FROM neds
\tUNION ALL
\tSELECT *
\tFROM neds_11
\tWITH DATA")

dbSendQuery(mdb, "alter table neds drop intent_self_harm;")
setdiff(dbListFields(mdb, "neds"), dbListFields(mdb, "neds_11"))

dbGetQuery(mdb, "SELECT COUNT(*) FROM neds_06_11")  # 167011415
dbGetQuery(mdb, "SELECT AVG(yr) AS AverageYear FROM neds_06_11;")  # 2008.565
dbRemoveTable(mdb, "neds_11")
dbRemoveTable(mdb, "neds")

dbListTables(mdb)
rm(neds.2011.core)
gc()

# append 2012
neds.2012.core <- readRDS("~/NEDS_2012_Core.rds")
names(neds.2012.core) <- tolower(names(neds.2012.core))
names(neds.2012.core)[names(neds.2012.core) == "year"] <- "yr"
neds.2012.core <- neds.2012.core[intersect(dbListFields(mdb, "neds_06_11"),
    names(neds.2012.core))]
dbWriteTable(mdb, "neds_06_11", neds.2012.core, append = TRUE, overwrite = FALSE)

dbListTables(mdb)
dbListFields(mdb, "neds_06_11")
dbGetQuery(mdb, "SELECT COUNT(*) FROM neds_06_11")  # 198102435
dbGetQuery(mdb, "SELECT AVG(yr) AS AverageYear FROM neds_06_11;")  # 2009.104
rm(neds.2012.core)
gc()

# clean up, check
```

```
dbGetQuery(mdb, "CREATE TABLE neds_06_12 AS SELECT * FROM neds_06_11 WITH DATA; DROP TAB

dbListTables(mdb)
dbListFields(mdb, "neds_06_12")
dbGetQuery(mdb, "SELECT COUNT(*) FROM neds_06_12")  # 198102435
dbGetQuery(mdb, "SELECT AVG(yr) AS AverageYear FROM neds_06_12;")  # 2009.104
```

4.1.1 Sample SQL code to add a variable

Finally, here is some code that illustrates working directly with the out-of-memory file using SQL syntax to create a new variable. First, I add a sequential variable that will serve as an identifier that is required by the sqlsurvey package. Next I add a variable set to 1 which can be used for count operations in the survey package. Lastly I create an indicator variable for acute alcohol intoxication based on ICD diagnoses.

```
# need to create an id variable (required by sqlsurvey), and a count
# variable
dbSendQuery(mdb, "alter table neds_06_12 add id serial;")

dbSendQuery(mdb, "alter table neds_06_12 add count int default 1;")
dbGetQuery(mdb, "select sum(count) from neds_06_12;")  # 198102435


# create etoh indicator variable
dbSendQuery(mdb, "alter table neds_06_12 add etoh int default 0;")
dbSendQuery(mdb, "update neds_06_12 set etoh=1 where dx2 = '30300' OR dx2 = '30500' OR
\t\t\tdx3='30300' OR dx3 = '30500' OR dx4 =' 30300' OR dx4 = '30500' OR
\t\t\tdx5 = '30300' OR dx5 = '30500' OR dx6 = '30300' OR dx6 = '30500' OR
\t\t\tdx7 = '30300' OR dx7 = '30500' OR dx8 = '30300' OR dx8 = '30500' OR
\t\t\tdx9 = '30300' OR dx9 = '30500' OR dx10 = '30300' OR dx10 = '30500' OR
\t\t\tdx11 = '30300' OR dx11 = '30500' OR dx12 = '30300' OR dx12 = '30500' OR
\t\t\tdx13 = '30300' OR dx13 = '30500' OR dx14 = '30300' OR dx14 = '30500' OR
\t\t\tdx15 = '30300' OR dx15 = '30500';")

dbGetQuery(mdb, "select sum(etoh) from neds_06_12 ;")  # 1,817,665

dbListFields(mdb, "neds_06_12")
```

4.2 Combining NIS Files with DBI and dplyr

This code illustrates how you can use dplyr to manipulate and combine 12 large files via a MonetDBLite connection. Unlike the previous example where all the data manipulations are conducted outside of R's physical memory constraints, this brings the files into R, and

requires an incremental approach as well as paring down as many variables as possible. In contrast to the previous approach that results in a single, multi-year file with most all the variables, this process results in 4 tables in a MonetDB directory:

- "nis_0003" - single table all rows and all variables 2000 to 2003
- "nis_0407" - single table all rows and all variables 2004 to 2007
- "nis_0811" - single table all rows and all variables 2008 to 2011
- "nis_0011" - single table all rows (94,646,462) and 52 variables from 2000 to 2011 (est about 40GB)

After establishing a connection to a folder that will hold the database, the first part of the code writes each individual-year NIS R file to a MonetDB database as tables by reading in each dataframe, using dbWriteTable() to write to out-of-memory database folder, then removing the R dataframe. Again, **Caution: Don't use dots to name your database tables.** MonetDB SQL queries won't work if the sql table name includes a dot. Use underscores.

Next, create dplyr table versions of the individual-year MonetDB tables. Then, use DBI::dbWriteTable and dplyr::rbind_list to combine the individual-year tables two at a time. Attempting to combine all the files at one time failed. Note that the "n=-1 option" is necessary to combine all the data in a table, else dplyr restricts to just first 100K rows.[10]. Use dbRemoveTable(mdb,tablename) to clean up and remove the individual-year files afterward.

The process is repeated to combine the 2-year files into 4-year files.

The 4-year files are too large to reasonably combine on most machines. It is necessary to pare down the number of variables. This involves first finding the intersection of the variable names between files (they tend to change across years) and then use a select statement to retain only the variables you need for analysis.[11]

NB files are too large as they are, need to (1) find the intersection of the variable names between the files, then (2) pare down the list of variables

```
library(MonetDBLite)
library(MonetDB.R)
# library(plyr)
library(dplyr)

# create connection to folder that will hold the database
mdb <- dbConnect(MonetDBLite(), "~/2000_2011/monet")

# write the individual-year R files into MonetDB database as tables
nis.2000.core<-readRDS("~/nis_2000_core.rds")
```

[10]Thanks to dickoa on StackOverflow for that

[11]This is why I chose to keep the 4-year files in the permanent database. If a future multi-year analysis requires a variable that wasn't included in this first merge, it can be recreated.

```
dbWriteTable(mdb, "nis_00", nis.2000.core)
rm(nis.2000.core)

nis.2001.core<-readRDS("~/nis.2001.core.rds")
dbWriteTable(mdb, "nis_01", nis.2001.core)
rm(nis.2001.core)

nis.2002.core<-readRDS("~/nis_2002_core.rds")
dbWriteTable(mdb, "nis_02", nis.2002.core)
rm(nis.2002.core)

nis.2003.core<-readRDS("~/nis_2003_core.rds")
dbWriteTable(mdb, "nis_03", nis.2003.core)
rm(nis.2003.core)

nis.2004.core<-readRDS("~/nis_2004_core.rds")
dbWriteTable(mdb, "nis_04", nis.2004.core)
rm(nis.2004.core)

nis.2005.core<-readRDS("~/nis_2005_core.rds")
dbWriteTable(mdb, "nis_05", nis.2005.core)
rm(nis.2005.core)

nis.2006.core<-readRDS("~/nis_2006_core_2.rds")
dbWriteTable(mdb, "nis_06", nis.2006.core)
rm(nis.2006.core)

nis.2007.core<-readRDS("~/nis_2007_core_2.rds")
dbWriteTable(mdb, "nis_07", nis.2007.core)
rm(nis.2007.core)

nis.2008.core<-readRDS("~/nis_2008_core_2.rds")
dbWriteTable(mdb, "nis_08", nis.2008.core)
rm(nis.2008.core)

nis.2009.core<-readRDS("~/nis_2009_core_2.rds")
dbWriteTable(mdb, "nis_09", nis.2009.core)
rm(nis.2009.core)

nis.2010.core<-readRDS("~/nis_2010_core_2.rds")
dbWriteTable(mdb, "nis_10", nis.2010.core)
rm(nis.2010.core)

nis.2011.core<-readRDS("~/nis_2011_core_2.rds")
```

```
dbWriteTable(mdb, "nis_11", nis.2011.core)
rm(nis.2011.core)


# create dplyr connection to the database and create dplyr table for each year of data
mdb_src <- src_monetdb(embedded="~/2000_2011/monet")

mdb_nis_00 <- tbl(mdb_src, "nis_00")
mdb_nis_01 <- tbl(mdb_src, "nis_01")
mdb_nis_02 <- tbl(mdb_src, "nis_02")
mdb_nis_03 <- tbl(mdb_src, "nis_03")
mdb_nis_04 <- tbl(mdb_src, "nis_04")
mdb_nis_05 <- tbl(mdb_src, "nis_05")
mdb_nis_06 <- tbl(mdb_src, "nis_06")
mdb_nis_07 <- tbl(mdb_src, "nis_07")
mdb_nis_08 <- tbl(mdb_src, "nis_08")
mdb_nis_09 <- tbl(mdb_src, "nis_09")
mdb_nis_10 <- tbl(mdb_src, "nis_10")
mdb_nis_11 <- tbl(mdb_src, "nis_11")


# combine individual-year tables two at a time
# clean up and remove the individual-year files afterward)

dbWriteTable(mdb, "nis_0001", rbind_list(as.data.frame(mdb_nis_00, n = -1),
                                         as.data.frame(mdb_nis_01, n = -1)))

dbWriteTable(mdb, "nis_0203", rbind_list(as.data.frame(mdb_nis_02, n = -1),
                                         as.data.frame(mdb_nis_03, n = -1)))

dbWriteTable(mdb, "nis_0405", rbind_list(as.data.frame(mdb_nis_04, n = -1),
                                         as.data.frame(mdb_nis_05, n = -1)))

dbWriteTable(mdb, "nis_0607", rbind_list(as.data.frame(mdb_nis_06, n = -1),
                                         as.data.frame(mdb_nis_07, n = -1)))

dbWriteTable(mdb, "nis_0809", rbind_list(as.data.frame(mdb_nis_08, n = -1),
                                         as.data.frame(mdb_nis_09, n = -1)))

dbWriteTable(mdb, "nis_1011", rbind_list(as.data.frame(mdb_nis_10, n = -1),
                                         as.data.frame(mdb_nis_11, n = -1)))

dbRemoveTable(mdb,"nis_00")
dbRemoveTable(mdb,"nis_01")
```

```
dbRemoveTable(mdb,"nis_02")
dbRemoveTable(mdb,"nis_03")
dbRemoveTable(mdb,"nis_04")
dbRemoveTable(mdb,"nis_05")
dbRemoveTable(mdb,"nis_06")
dbRemoveTable(mdb,"nis_07")
dbRemoveTable(mdb,"nis_08")
dbRemoveTable(mdb,"nis_09")
dbRemoveTable(mdb,"nis_10")
dbRemoveTable(mdb,"nis_11")


# combine  2-year tables into 4-year tables

# establish dplyr connection to 2-year tables

mdb_nis_0001 <- tbl(mdb_src, "nis_0001")
mdb_nis_0203 <- tbl(mdb_src, "nis_0203")
mdb_nis_0405 <- tbl(mdb_src, "nis_0405")
mdb_nis_0607 <- tbl(mdb_src, "nis_0607")
mdb_nis_0809 <- tbl(mdb_src, "nis_0809")
mdb_nis_1011 <- tbl(mdb_src, "nis_1011")

# summarise(mdb_nis_1011, mean.age = mean(AGE))
# summarise(mdb_nis_0001, mean.age = mean(AGE))

# combine into 4-year tables
dbWriteTable(mdb, "nis_0003", rbind_list(as.data.frame(mdb_nis_0001, n = -1),
as.data.frame(mdb_nis_0203, n = -1))

dbWriteTable(mdb, "nis_0407", rbind_list(as.data.frame(mdb_nis_0405, n = -1),
as.data.frame(mdb_nis_0607, n = -1)))

dbWriteTable(mdb, "nis_0811", rbind_list(as.data.frame(mdb_nis_0809, n = -1),
as.data.frame(mdb_nis_1011, n = -1)))

#clean up
dbRemoveTable(mdb,"nis_0001")
dbRemoveTable(mdb,"nis_0203")
dbRemoveTable(mdb,"nis_0405")
dbRemoveTable(mdb,"nis_0607")
dbRemoveTable(mdb,"nis_0809")
dbRemoveTable(mdb,"nis_1011")
```

```
# combine the 4 year-files
# create dplyr tables from the 4-year tables
mdb_nis_0003 <- tbl(mdb_src,"nis_0003")
mdb_nis_0407 <- tbl(mdb_src,"nis_0407")
mdb_nis_0811 <- tbl(mdb_src,"nis_0811")

summarise(mdb_nis_0811, mean.age = mean(AGE))
# get intersection of variable names and pare them down
intersect(dbListFields(mdb, "nis_0407"), dbListFields(mdb, "nis_0811"))

# use dplyr select() to create a dplyr table that
# retains only the rows you want in the single 12-year file
nis_0811b<-select(mdb_nis_0811, KEY, AGE, AMONTH, ATYPE, AWEEKEND, DIED,
DISCWT, DISPUNIFORM, DQTR, DSHOSPID, DX1, DX2, DX3, DX4, DX5, DX6,
DX7, DX8, DX9, DX10, DX11, DX12, DX13, DX14, DX15, ECODE1, ECODE2,
ECODE3, ECODE4, FEMALE, HOSPID, HOSPST, LOS,  NECODE, NIS_STRATUM,
PR1, PR2, PR3, PR4, PR5, PR6, PR7, PR8, PR9, PR10, PR11, PR12,
PR13, PR14, PR15, RACE, TOTCHG, YEAR, ZIPINC_QRTL)

summarise(nis_0811b, mean.age = mean(AGE))

# pare down the second 4-year table
nis_0407b<-select(mdb_nis_0407, KEY, AGE, AMONTH, ATYPE, AWEEKEND, DIED,
DISCWT, DISPUNIFORM, DQTR, DSHOSPID, DX1, DX2, DX3, DX4, DX5, DX6,
DX7, DX8, DX9, DX10, DX11, DX12, DX13, DX14, DX15, ECODE1, ECODE2,
ECODE3, ECODE4, FEMALE, HOSPID, HOSPST, LOS,  NECODE, NIS_STRATUM,
PR1, PR2, PR3, PR4, PR5, PR6, PR7, PR8, PR9, PR10, PR11, PR12, PR13,
PR14, PR15, RACE, TOTCHG, YEAR, ZIPINC_QRTL)

# write the 8-year table to the dataframe
dbWriteTable(mdb, "nis_0411", rbind_list(as.data.frame(nis_0407b, n = -1),
                                         as.data.frame(nis_0811b, n = -1)))

dbListTables(mdb)
dbGetQuery(mdb, "SELECT COUNT(*) FROM nis_0411")

# pare down the third -4-year table
nis_0003b<-select(mdb_nis_0003, KEY, AGE, AMONTH, ATYPE, AWEEKEND, DIED,
DISCWT, DISPUNIFORM, DQTR, DSHOSPID, DX1, DX2, DX3, DX4, DX5, DX6, DX7,
DX8, DX9, DX10, DX11, DX12, DX13, DX14, DX15, ECODE1, ECODE2, ECODE3,
ECODE4, FEMALE, HOSPID, HOSPST, LOS,  NECODE, NIS_STRATUM, PR1, PR2,
```

```
PR3, PR4, PR5, PR6, PR7, PR8, PR9, PR10, PR11, PR12, PR13, PR14, PR15,
RACE, TOTCHG, YEAR)

# create dplyr table from the 8-year database table
nis_0411 <- tbl(mdb_src,"nis_0411")

# write 12-year database table from rbind of 8-year and 4-year table
dbWriteTable(mdb, "nis_0011", rbind_list(as.data.frame(nis_0411, n = -1),
                                         as.data.frame(nis_0003b, n = -1)))


dbListTables(mdb)
dbGetQuery(mdb, "SELECT COUNT(*) FROM nis_0011") # 94,646,462


dbListTables(mdb)
```

5 Analyzing Large Muti-Year MonetDB HCUP Survey Files

This section presents two approaches to working with large multi-year complex survey data in an out-of-memory MonetDB database. The first illustrates using Thomas Lumley's sqlsurvey package. The second illustrates the use of dplyr to manipulate data and then use the more complete and mature survey package.

5.1 Using sqlsurvey

The sqlsurvey package is an adaptation of the well-known and wonderful survey package written by Dr. Thomas Lumley that allows for SQL-based analysis of large survey, and thus allows the kind of out-of-memory analyses, in this case for complex surveys, we are aiming for. It is listed as experimental. In my experience, when it works it returns results extremely close to the standard survey package. When it doesn't work it simply fails. The biggest drawback is that it has only a limited number of available functions.

Install and load sqlsurvey.[12] Establish a connection to the out-of-memory Monet database containing the NEDS injury file. Check the file and fields. Note the speed with which the survey object is created.[13]

[12] Note that it's not a standard or default installation.

[13] It should take some matter of minutes. Try creating a standard survey object from a database that large. Assuming you could read the file into R in the first place...

Create a sqlsurvey object from the MonetDB injury table.[14] for details.

```
install.packages("sqlsurvey", repos = "http://R-Forge.R-project.org")
library(sqlsurvey)

mdb <- dbConnect(MonetDBLite(), "~/monetInjury")
dbListTables(mdb)
dbListFields(mdb, "nedsinj0612")
dbGetQuery(mdb, "SELECT COUNT(*) FROM nedsinj0612")

fax <- c("aweekend", "died_visit", "female", "intent_self_harm", "injury", "injury_cut",
    "injury_drown", "injury_fall", "injury_fire", "injury_firearm", "injury_machinery",
    "injury_mvt", "injury_nature", "injury_poison", "injury_severity", "injury_struck",
    "injury_suffocation", "intent_assault", "intent_unintentiona", "multinjury",
    "intent_unintention", "hosp_trauma", "hosp_ur_teach")

system.time(monInj <- sqlsurvey(id = "key_ed", strata = "neds_stratum", weights = "discw
    key = "id", check.factors = fax, database = "~/monetInjury", driver = MonetDBLite(),
    table.name = "nedsinj0612"))
```

Run some survey statistics this very large file.

```
(yrCount <- svytotal(~count, monInj, byvar = ~yr, na.rm = T, se = TRUE))  # yearly surve
(svymean(~age, monInj, na.rm = T, se = T))  # mean age
```

5.2 Using dplyr and survey

The following code illustrates the use of dplyr and the standard survey package to account for year-to-year trends in a large multi-year out-of-memory MonetDB table. The methods are based on work by Bieler, et al.[15] and the US CDC [16]. The code for the actual trend analyses was taken whole-cloth from Anthony D'Amico's very informative site.[17]. I used this approach to report results in Traumatic injury in the United States: In-patient epidemiology 2000-2011.

5.3 Age Trends Adjusting for Year-to-year Variation

Connect to the MonetDB database. Connect and create dplyr tables of both a full data set, and a subset of injury discharges.

[14] See http://rpackages.ianhowson.com/rforge/sqlsurvey/ and http://rpackages.ianhowson.com/rforge/sqlsurvey/man/sqlsurvey.html

[15] Bieler GS, Brown GG, Williams RL, Brogan DJ. Estimating model-adjusted risks, risk differences, and risk ratios from complex survey data. American Journal of Epidemiology 2010;171(5):618-23

[16] http://www.cdc.gov/healthyyouth/yrbs/pdf/yrbs_conducting_trend_analyses.pdf

[17] http://www.asdfree.com/2015/11/statistically-significant-trends-with.html

```
# create connection to folder that holds the database
mdb <- dbConnect(MonetDBLite(), "~/2000_2011/monet")

mdb_src <- src_monetdb(embedded="~/2000_2011/monet")
nis_0011 <- tbl(mdb_src,"nis_0011") # all discharges
inj_0011 <- tbl(mdb_src,"inj_0011") # injury discharges
```

Conduct age-trend analyses.

```
# reduce number variables from full-year injury and nis
# files files using dplyr syntax on the tbl connection objects
inj<-select(inj_0011, KEY, FEMALE, DIED, AGE, count, region, ageGrp)
nis<-select(nis_0011, KEY, DISCWT, NIS_STRATUM, HOSPID, YEAR)

glimpse(inj)
glimpse(nis)

# merge files
inj.long<-full_join(as.data.frame(inj, n=-1), as.data.frame(nis, n=-1),
by="KEY")

class(inj.long) # dataframe
nrow(inj.long) #95049907
table(inj.long$YEAR)

# create linear contrast variable, length 12 (one for each year)
linearContr <- contr.poly( 12 )[ , 1 ]

# create quadratic (squared) contrast variable
quadContr <- contr.poly( 12 )[ , 2 ]

# create cubic contrast variable
cubeContr <- contr.poly( 12 )[ , 3 ]

# tack contrast terms onto dataframe
inj.long$linearContr <- linearContr[ match( inj.long$YEAR ,
        seq( 2000 , 2011 , 1 ) ) ]
inj.long$quadContr <- quadContr[ match( inj.long$YEAR ,
        seq( 2000 , 2011 , 1 ) ) ]
inj.long$cubeContr <- cubeContr[ match( inj.long$YEAR ,
        seq( 2000 , 2011 , 1 ) ) ]


# create survey design accounting for year in nested strata
```

```
 svydes<- svydesign(
                id = ~HOSPID ,
                strata = ~interaction(NIS_STRATUM , YEAR),  # note YEAR interaction
                weights = ~DISCWT ,
                nest = TRUE,
                data = inj.long
        )

# subset survey design to injuries (should use less memory)

injsvy<-subset(svydes, count==1)

# calculate and plot mean age of injured discharges by year

unadjusted<-svyby(~AGE, ~YEAR, injsvy, svymean, na.rm=T,
        vartype = c( 'ci' , 'se' ))
unadjusted

#      YEAR      AGE        se     ci_l     ci_u
# 2000 2000 54.33203 0.7779181 52.80734 55.85673
# 2001 2001 55.28441 0.8792622 53.56108 57.00773
# 2002 2002 54.46620 0.6988638 53.09645 55.83595
# 2003 2003 54.56867 0.7286495 53.14055 55.99680
# 2004 2004 53.60807 0.6943732 52.24713 54.96902
# 2005 2005 55.74838 0.8804398 54.02275 57.47401
# 2006 2006 56.23959 0.7961953 54.67908 57.80011
# 2007 2007 56.96143 0.8854963 55.22589 58.69697
# 2008 2008 58.46062 0.8175963 56.85816 60.06308
# 2009 2009 58.27817 0.7685276 56.77189 59.78446
# 2010 2010 56.32216 0.6514177 55.04540 57.59891
# 2011 2011 60.23681 0.8305331 58.60900 61.86463

my_plot <- data.frame( unadjusted )


my.plot<-ggplot( my_plot , aes( x = YEAR, y = AGE ) ) +
  geom_point() +
  geom_errorbar( aes( ymax = ci_u , ymin = ci_l ) , width = .2 ) +
  geom_line() +
  theme_tufte() +
  ggtitle( "Unadjusted Mean Age Injured Discharges 2000-2011" ) +
  theme( plot.title = element_text( size = 9 , face = "bold" ) )
my.plot
ggsave(file="/Users/charlie/Box Sync/hcup/hcupNotes2/my_plot.jpg", my.plot)
```

```r
# determine how many, if any, joinpoints are necessary for trend analysis
# by running regressions testing the contrasts
# test each contrast in sequence

linyear <- svyglm(AGE ~ FEMALE + linearContr, design=injsvy)

summary(linyear)

# the linear contrast term is statistically significant, so there is at
# least a linear trend (which does not require a joinpoint)

# Call:
# svyglm(formula = AGE ~ FEMALE + linearContr, design = injsvy)
#
# Survey design:
# subset(svydes, count == 1)
#
# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
# (Intercept) 46.42117    0.19437 238.829   <2e-16 ***
# FEMALE      19.59507    0.05964 328.578   <2e-16 ***
# linearContr  5.25335    0.63692   8.248   <2e-16 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# (Dispersion parameter for gaussian family taken to be 606.9774)
#
# Number of Fisher Scoring iterations: 2

# add the quadratic term to the regression

linyear2 <- svyglm(AGE ~ FEMALE + linearContr + quadContr,
        design=injsvy)
summary(linyear2)

# the quadratic contrast is not significant,
# so we could stop here, we don't need a joinpoint

# Call:
# svyglm(formula = AGE ~ FEMALE + linearContr + quadContr, design = injsvy)
#
# Survey design:
# subset(svydes, count == 1)
#
```

```
# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
# (Intercept) 46.42221    0.19448 238.704   <2e-16 ***
# FEMALE      19.59129    0.05899 332.088   <2e-16 ***
# linearContr  5.25985    0.63851   8.238   <2e-16 ***
# quadContr    0.97005    0.66038   1.469    0.142
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# (Dispersion parameter for gaussian family taken to be 606.8987)
#
# Number of Fisher Scoring iterations: 2

# run the regression with the cubic term just to be complete

linyear3 <- svyglm(AGE ~ FEMALE + linearContr + quadContr + cubeContr, design=injsvy)
summary(linyear3)

# as expected, not significant

# Call:
# svyglm(formula = AGE ~ FEMALE + linearContr + quadContr + cubeContr,
#     design = injsvy)
#
# Survey design:
# subset(svydes, count == 1)
#
# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
# (Intercept) 46.41798    0.19403 239.228   <2e-16 ***
# FEMALE      19.59091    0.05921 330.867   <2e-16 ***
# linearContr  5.26371    0.63792   8.251   <2e-16 ***
# quadContr    0.96476    0.66040   1.461    0.144
# cubeContr   -0.68867    0.63917  -1.077    0.281
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# (Dispersion parameter for gaussian family taken to be 606.8593)
#
# Number of Fisher Scoring iterations: 2
```

Since there is a single linear trend in these data, we don't have to run a joinpoint analysis. We can plot the data and present the results of the regression.

```r
meanAGE<-svymean(~AGE, injsvy, na.rm=T)
meanAGE # 56.215 sd=0.2309
confint(meanAGE)
#         2.5 %   97.5 %
# AGE 55.76277 56.66788


p1<-ggplot(data=my_plot, aes(y=AGE, ymin=ci_l, ymax=ci_u, x=YEAR))
             +ylim(0,65)+xlab("Year")+ylab("Mean Age (95% CI)")
p2<-p1+geom_linerange(alpha = I(12/12))
p3<-p2+geom_point(aes(y=AGE, x=YEAR))
p4<-p3+geom_hline(yintercept=56.2, color="grey",alpha = I(12/12))
      + geom_hline(yintercept=c(55.8, 56.7), lty=2, color="grey",
      alpha = I(8/12))
agePlot1<-p4+theme_bw()
ggsave(file="~/agePlot1.jpg", agePlot1)

linyear <- svyglm(AGE ~ FEMALE + YEAR, design=injsvy)
summary(linyear)

# Call:
# svyglm(formula = AGE ~ FEMALE + YEAR, design = injsvy)
#
# Survey design:
# subset(svydes, count == 1)
#
# Coefficients:
#               Estimate Std. Error t value Pr(>|t|)
# (Intercept) -834.60967  106.81186  -7.814 6.04e-15 ***
# FEMALE        19.59507    0.05964 328.578  < 2e-16 ***
# YEAR           0.43931    0.05326   8.248  < 2e-16 ***
# ---
# Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
#
# (Dispersion parameter for gaussian family taken to be 606.9774)
#
# Number of Fisher Scoring iterations: 2

confint(linyear)

#                    2.5 %       97.5 %
# (Intercept) -1043.957076 -625.2622596
# FEMALE         19.478185   19.7119536
# YEAR            0.334916    0.5436986
```

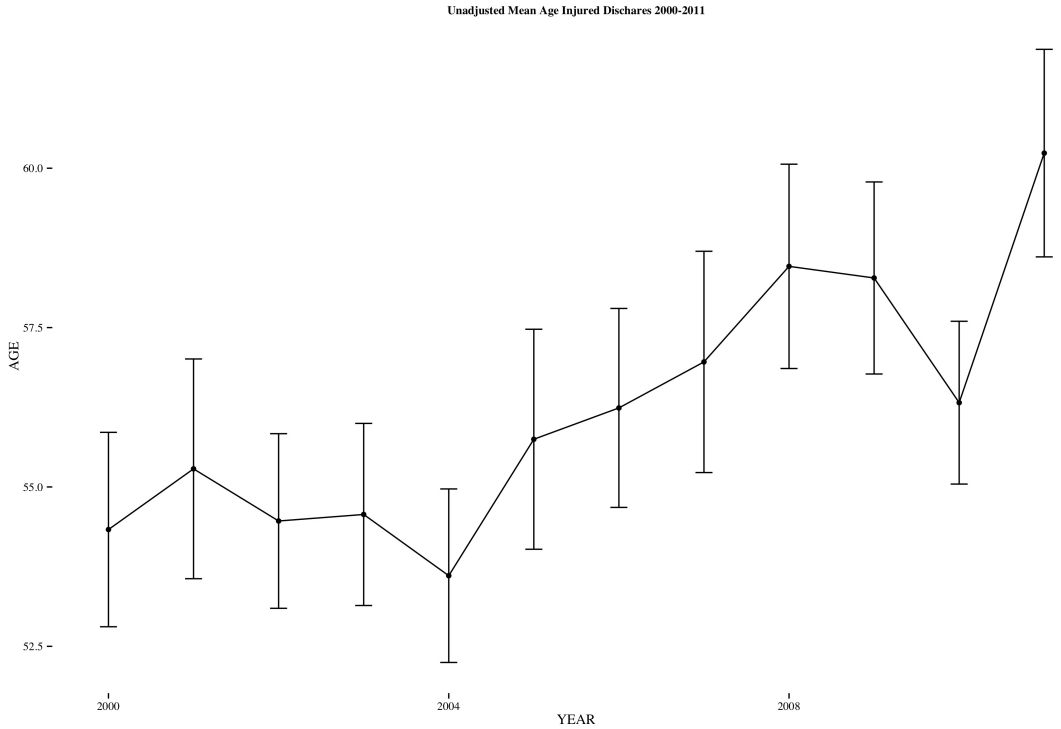

Figure 1: Mean Age by Year, Injury Discharges 2000-2011

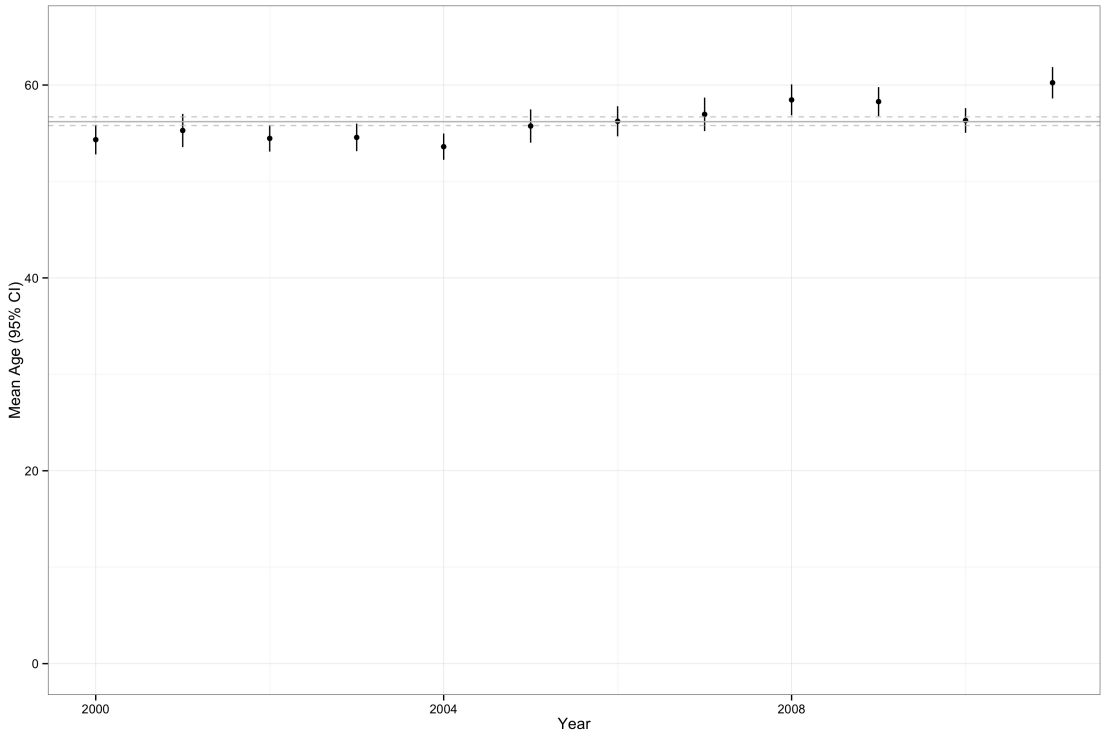

Figure 2: Mean Age by Year, Injury Discharges 2000-2011

5.3.1 Trauma Centers Adjusted for Year-to-year Variation

We now repeat this kind of approach to look at the year-to-year number of trauma centers in the United States based on HCUP NIS survey estimates.

```
# reduce number variables from full-year injury and nis
# files files using dplyr syntax on the tbl connection objects
inj<-select(inj_0011, KEY, FEMALE, DIED, AGE, count, region,
        ageGrp, level.one, HOSP_TEACH, AWEEKEND, severe, Charlson)
nis<-select(nis_0011, KEY, DISCWT, NIS_STRATUM, HOSPID, YEAR)

# merge files
inj.long<-full_join(as.data.frame(inj, n=-1), as.data.frame(nis, n=-1), by="KEY")

# create linear contrast variable, length 12 (one for each year)
linearContr <- contr.poly( 12 )[ , 1 ]

# create quadratic (squared) contrast variable
quadContr <- contr.poly( 12 )[ , 2 ]

# create cubic contrast variable
cubeContr <- contr.poly( 12 )[ , 3 ]

# tack contrast terms onto dataframe
inj.long$linearContr <- linearContr[ match( inj.long$YEAR ,
        seq( 2000 , 2011 , 1 ) ) ]
inj.long$quadContr <- quadContr[ match( inj.long$YEAR ,
        seq( 2000 , 2011 , 1 ) ) ]
inj.long$cubeContr <- cubeContr[ match( inj.long$YEAR ,
        seq( 2000 , 2011 , 1 ) ) ]

svydes<- svydesign(
        id = ~HOSPID ,
        strata = ~interaction(NIS_STRATUM , YEAR),
        weights = ~DISCWT ,
        nest = TRUE,
        data = inj.long
)

injsvy<-subset(svydes, count==1)

(tot.center<-svytotal(~level.one, injsvy, na.rm=T,
        keep.var=TRUE, multicore=T))

#             total     SE
```

```
# level.one 2399216 186904

confint(tot.center)
#              2.5 %  97.5 %
# level.one 2032891 2765541

tot.center/12 # approx 199935 level 1 admissions per year


(center.mean<-svymean(~level.one, injsvy, na.rm=T,
        keep.var=TRUE, multicore=T))
#               mean     SE
# level.one 0.11613 0.0085
confint(center.mean)
#                 2.5 %    97.5 %
# level.one 0.09955505 0.1327056

(centerSevere1<-svyby(~severe, ~level.one, injsvy,
        svymean,na.rm=T, keep.var=TRUE, multicore=T))
#   level.one    severe          se
# 0         0 0.2410288 0.002661688
# 1         1 0.3943349 0.007206799

confint(centerSevere1)
#       2.5 %    97.5 %
# 0 0.2358120 0.2462456
# 1 0.3802098 0.4084600



(centerCFR<-svyby(~DIED, by=~level.one, denominator=~count,
        design=injsvy, svyratio, na.rm=T, keep.var=TRUE, multicore=T))
#   level.one DIED/count se.DIED/count
# 0         0 0.02269223  0.0002096282
# 1         1 0.03444535  0.0007003857

confint(centerCFR)
#        2.5 %     97.5 %
# 0 0.02228137 0.02310309
# 1 0.03307262 0.03581808

centerCFR.glm<-svyglm(DIED~ level.one, injsvy, family=binomial(logit))
summary(centerCFR.glm)
# Call:
```

```
# svyglm(formula = DIED ~ level.one, injsvy, family = binomial(logit))
#
# Survey design:
# subset(svydes, count == 1)
#
# Coefficients:
#              Estimate Std. Error t value Pr(>|t|)
# (Intercept) -3.762779   0.009452 -398.08   <2e-16 ***
# level.one    0.429450   0.023225   18.49   <2e-16 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# (Dispersion parameter for binomial family taken to be 1.000053)
#
# Number of Fisher Scoring iterations: 6

exp(coef(centerCFR.glm))
# (Intercept)   level.one
#  0.02321912  1.53641261
#

exp(confint(centerCFR.glm))
#                  2.5 %    97.5 %
# (Intercept) 0.02279292 0.0236533
# level.one   1.46804142 1.6079680



# test linear contrast for year-to-year variability in trends
# it is statistically significant, so there has been,
# in fact, a statistically significant decline

centerCFR.glmLinContr<-svyglm(DIED~ level.one + AGE + FEMALE + region
        + HOSP_TEACH + AWEEKEND + severe +  Charlson + linearContr,
        injsvy, family=binomial(logit))

summary(centerCFR.glmLinContr)
# Call:
# svyglm(formula = DIED ~ level.one + AGE + FEMALE + region + HOSP_TEACH +
#     AWEEKEND + severe + Charlson + linearContr, injsvy, family = binomial(logit))
#
# Survey design:
# subset(svydes, count == 1)
#
```

```
# Coefficients:
#                   Estimate Std. Error  t value Pr(>|t|)
# (Intercept)     -6.4326594  0.0262064 -245.461  < 2e-16 ***
# level.one        0.2198277  0.0223437    9.838  < 2e-16 ***
# AGE              0.0239986  0.0002884   83.217  < 2e-16 ***
# FEMALE          -0.3935988  0.0089987  -43.740  < 2e-16 ***
# regionNortheast  0.0548135  0.0198443    2.762  0.00575 **
# regionSouth      0.1588829  0.0195357    8.133 4.63e-16 ***
# regionWest       0.0708024  0.0219623    3.224  0.00127 **
# HOSP_TEACH       0.3580994  0.0159756   22.415  < 2e-16 ***
# AWEEKEND         0.0458233  0.0091747    4.995 5.99e-07 ***
# severe           1.9051679  0.0123445  154.333  < 2e-16 ***
# Charlson         0.2620184  0.0028396   92.273  < 2e-16 ***
# linearContr     -0.6185967  0.0260916  -23.709  < 2e-16 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# (Dispersion parameter for binomial family taken to be 0.7989355)
#
# Number of Fisher Scoring iterations: 7

exp(coef(centerCFR.glmLinContr))

centerCFR.glmAdj<-svyglm(DIED~ level.one + AGE + FEMALE + region +
        HOSP_TEACH + AWEEKEND + severe +  Charlson +
        YEAR, injsvy, family=binomial(logit))

summary(centerCFR.glmAdj)
# Call:
# svyglm(formula = DIED ~ level.one + AGE + FEMALE + region + HOSP_TEACH +
#     AWEEKEND + severe + Charlson + YEAR, injsvy, family = binomial(logit))
#
# Survey design:
# subset(svydes, count == 1)
#
# Coefficients:
#                   Estimate Std. Error t value Pr(>|t|)
# (Intercept)     97.3111639  4.3729642  22.253  < 2e-16 ***
# level.one        0.2198277  0.0223437   9.838  < 2e-16 ***
# AGE              0.0239986  0.0002884  83.217  < 2e-16 ***
# FEMALE          -0.3935988  0.0089987 -43.740  < 2e-16 ***
# regionNortheast  0.0548135  0.0198443   2.762  0.00575 **
# regionSouth      0.1588829  0.0195357   8.133 4.63e-16 ***
# regionWest       0.0708024  0.0219623   3.224  0.00127 **
```

```
# HOSP_TEACH       0.3580994  0.0159756  22.415  < 2e-16 ***
# AWEEKEND         0.0458233  0.0091747   4.995 5.99e-07 ***
# severe           1.9051679  0.0123445 154.333  < 2e-16 ***
# Charlson         0.2620184  0.0028396  92.273  < 2e-16 ***
# YEAR            -0.0517297  0.0021819 -23.709  < 2e-16 ***
# ---
# Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#
# (Dispersion parameter for binomial family taken to be 0.7989355)
#
# Number of Fisher Scoring iterations: 7

exp(coef(centerCFR.glmAdj))[-1]

exp(confint(centerCFR.glmAdj))[-1,]

#                      2.5 %    97.5 %
# level.one       1.1924797 1.3016341
# AGE             1.0237101 1.0248680
# FEMALE          0.6628305 0.6866286
# regionNortheast 1.0160468 1.0982387
# regionSouth     1.1281664 1.2179536
# regionWest      1.0281458 1.1205814
# HOSP_TEACH      1.3865073 1.4761111
# AWEEKEND        1.0282325 1.0658849
# severe          6.5598849 6.8851210
# Charlson        1.2923379 1.3068033
# YEAR            0.9455334 0.9536551



# overall case fatality
(tot.CFR<-svyratio(~DIED, ~count, injsvy, na.rm=T))

# Ratio estimator: svyratio.survey.design2(~DIED, ~count, injsvy, na.rm = T)
# Ratios=
#           count
# DIED 0.02405892
# SEs=
#              count
# DIED 0.0002185861

confint(tot.CFR)
#                2.5 %     97.5 %
```

```
# DIED/count 0.0236305 0.02448734
```

6 Addendum: Why not read HCUP .csv directly into MonetDB?

You may ask why I simply don't read the HCUP .csv files directly into MonetDB (like I did with the digitalis example) rather than first creating R files? Well, consider this a cautionary tale.

Reading large .csv files into MonetDB monetdb.read.csv() simply didn't work. So, I turned to reading the files into a MonetDB directly from the .csv's using the schema approach. This was, um, challenging. Creating the schema required extensive debugging. The Booleans all had to be changed to string. Then I needed to change most of the integers to string. I tried keeping some variables like age, length of stay and costs as integers. But this returned an error for costs. Maybe MonetDB wanted a numeric variable? I changed that to double precision. The schema seemed to work at that point and the data dump went on for some while. Until it finally returned an error that "Server says 'SQLException:importTable:Failed to import table Leftover data '4'." After days of trying to suss out what this even meant, various approaches, suggestions from the good folks on StackOverflow, I accepted that I was simply not able to get around this. So I switched over to writing the table directly from the R dataframe as described above.

```
library(MonetDBLite)
library(MonetDB.R)
library(dplyr)

mdb <- dbConnect(MonetDBLite(), "~/1998_2012/monet")

guess <- read.csv("~/nis_1998_core.csv", stringsAsFactors=FALSE, nrows=4000)
create <- sprintf("CREATE TABLE nis_1998_core ( %s )",
paste0(sprintf('"%s" %s', colnames(guess),
sapply(guess, dbDataType, dbObj=mdb)), collapse=","))
create

create <- [1843 chars quoted with '"']

dbSendQuery(mdb, create)

# dbRemoveTable(mdb, "nis_1998_core")

dbSendQuery(mdb, "COPY  6827350 OFFSET 2   RECORDS
                  INTO nis_1998_core
                  FROM '~/nis_1998_core.csv' USING  DELIMITERS ','
```

```
  NULL AS '' ")


                                    #
```

Working with dplyr was also not always a bed of roses. Using plyr::r.bind.fill seemed ideal because the data sets differ in variables from year to year. But, r.bind.fill via DBI:dbWriteTable returns a truncated table. Using dplyr::rbind_all does not seem to work at all. A less useful approach (but still acceptable) involves removing columns to make the tables identical and using a straight-ahead UNION ALL SQL operation. But this requires knowing SQL well enough to parse and address the invariable error messages.

```
# combining dataframes in R memory and writing table failed,
# insufficient memory
nis_core_98_12 <- rbind.fill(nis.1998.core, nis.1999.core,
        nis.2000.core,  nis.2001.core, nis.2002.core, nis.2003.core,
        nis.2004.core, nis.2005.core,   nis.2006.core, nis.2007.core,
        nis.2008.core, nis.2009.core, nis.2010.core,
        nis.2011.core, nis.2012.core)


# SQL CREATE TABLE ... UNION ALL failed differing column names
dbGetQuery(mdb,
        "CREATE TABLE nis_0411 AS
SELECT *
    FROM nis_0407
UNION ALL
SELECT *
FROM nis_0811
WITH DATA")
#NB  need to include the WITH DATA statement at end
# to actually populate the table
# including intersect names failed
intersect(dbListFields(mdb, "nis_0407"), dbListFields(mdb, "nis_0811"))
```