STA 250 Assignment 1

Matthew Meisner
ID: 997477746

23 January 2014

1 Description of Methods

1.1 Method 1: Column Extraction in the Shell

The first method can be broken down into the following steps:

1. Extract the bz2 file into its component .csv files.
2. In R, loop over the file names.
3. For each file, call the shell from R using system(), and use cut in the shell to extract the column containing arrival delays (note: this column differs depending on the format of the data, so we need to keep track of which file we're working on so we can select the column number accordingly).
4. After extracting the correct column from each file, make a frequency table of the delay times. Make a separate table for each year. This facilitates calculating year-specific means later, if desired. Also, it is advantageous to create a table after reading each file so that we don't need to keep all the individual observations around, which would take up a lot of memory.
5. After a frequency table has been computed for each year, merge them together into one frequency table that contains the total counts of the delay times across the entire dataset.
6. Use the combined frequency table to calculate the desired summary statistics: the mean, median, and standard deviation.
    - To find the mean, multiply the values in the table by the names of the table (element by element), add up the resulting vector, and divide by the total number of flights for which an arrival delay time was recorded.
    - To find the median, sort the names of the table from smallest delay time (i.e. early arrival) to largest delay time. Start with the smallest delay time, and sequentially move to larger delay times, keeping a cumulative sum of the number of flights with shorter delays than those at the current position in the table. As soon as this cumulative sum reaches or exceeds half of the total number of flights with recorded delays (a value we can obtain by summing the entries of the merged table), we know that the median lies in the current "bin" of delay times.
    - To find the standard deviation, I first found the variance and then took the square root. To find the sample variance, I went through each entry in the merged table, squared the difference between the name of that entry in the table (i.e. the delay time) and the mean, multiplied that by the value at that entry in the table (i.e. the number of flights with that delay time), summed up the resulting vector, and divided by the total number of flights.

1.2 Method 2: Frequency Table in the Shell

This method is relatively similar to the first. The primary difference is that we use the shell to compute the frequency table for each file instead of reading the columns into R and using R to make the frequency tables. Specifically, the steps of this method are:

1. Extract the bz2 file into its component .csv files.
2. In R, loop over the file names.
3. For each file, call the shell from R using system(), and use cut in the shell to extract the column containing arrival delays.
4. Pipe the output from cut to sort.
5. Pipe the output from sort to "uniq -c", which creates a frequency table.
6. The shell output of the frequency table is just a bunch of strings with counts and delays on each line, so, in R, we need to split up these strings and make it into an actual R numeric object so that we can calculate the desired statistics.
7. After a frequency table has been computed for each year, merge them together into one frequency table that contains the total counts of the delay times across the entire dataset.
8. Use the combined frequency table to calculate the desired summary statistics: the mean, median, and standard deviation, as described in the first method

1.3 Method 3: PostgreSQL Database

This method was quite an experiment for me, since I've never worked with SQL (perhaps it makes up for the fact that the first two methods are fairly similar). The steps to run this method are as follows:

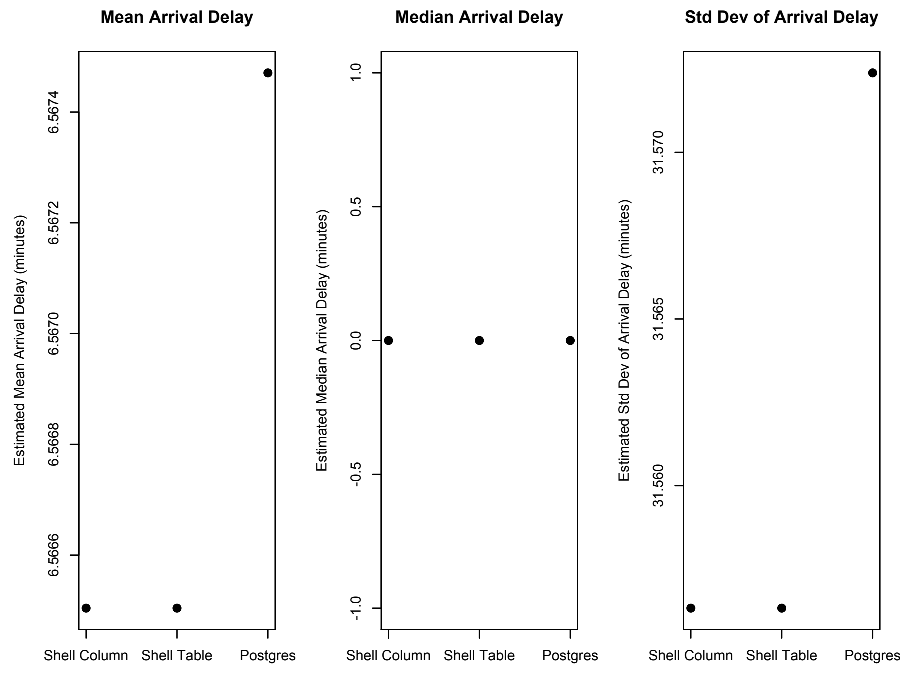
1. Unzip files.
2. Install postgres using wget, configure, make, and make install, as specified on the course website.
3. Run the postgres shell, and use that to create empty tables with the desired schema. Since in my trials it seemed faster to load fewer columns into the dataset, I only loaded 4 columns: the delay times, the origin airport, the destination airport, and the carrier. I selected these columns because I was interested in exploring how delay times depend on origin/destination airports and the carrier.
4. Create a separate table for 1987 to 2007 (the first format of the data) and 2008 to 2012; this was necessary because the 4 desired columns are in a different order in the two formats, and cut in the shell is only able to extract columns in the order that they appear.
5. Use "cat" in the shell to select all of the .csv files from 1987 through 2007, pipe the lines to "cut" to extract the columns we want, pipe the result to "grep -v" to remove the column headers, and then pipe the result to postgres to copy the data from STDIN into the database.
6. Do the same as the previous step for the 2008 to 2012 data, but add an intermediate piping to "sed" to replace the blank arrival delay entries with NAs, since otherwise postgres gave an error when trying to load blanks into a float or int column. This wasn't an issue with the earlier data format, since here the missing values were already denoted by NAs, not blanks.
7. We will need a frequency table to calculate the median, since there's no built-in median function. There are built-in mean and variance functions in Postgres, but I elected not to use them because it was faster not to do so. Why? Because we have to compute the frequency table anyway in order to find the median, and it is extremely fast to find the mean and standard deviation once we have the frequency table (faster than finding the mean and variance using built-in functions). So, using "COUNT(*)" and "GROUP BY arrdelay" we get a frequency table of delay times.

8. A few boring technical acrobatics are needed to convert the frequency table to the same form as the ones from the previous two methods, but, once this has been done, we can use the same functions from before to find the statistics of interest.

2 Results

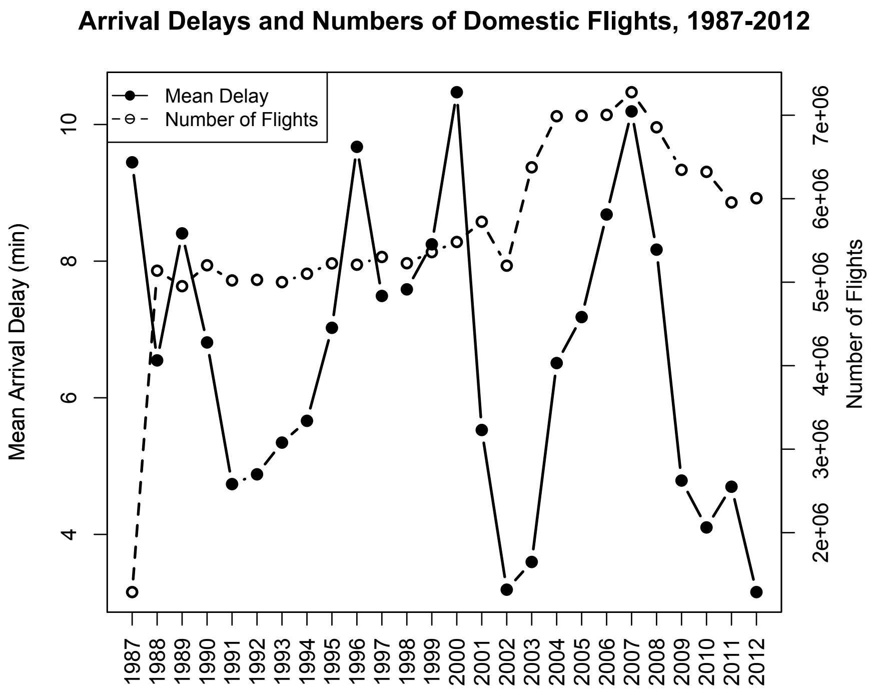
2.1 Summary Statistics

In keeping with your stern warnings against tables, I plotted the mean, median, and standard deviation obtained using all three methods. The estimated median, 0, was identical using all three methods. The mean was estimated to be 6.5665 using the first two methods, but was estimated to be ever so slightly higher, 6.5675, using the third method. The standard deviation was estimated to be 31.55633 using both the first and second methods, but was again slightly higher, 31.57238, using the database method. Note: these plots are very misleading, and imply that there are huge differences in the estimated mean and standard deviation using the different methods; in reality, the differences are extremely small.



2.2 A Little Exploratory Data Analysis

I also was curious to see if there have been trends for flight delays over time, so I found the mean arrival delay by year (and the number of total flights with recorded delay times in each year). This computation was practically free given that the frequency tables of delay times were already made for each year using the first two methods.

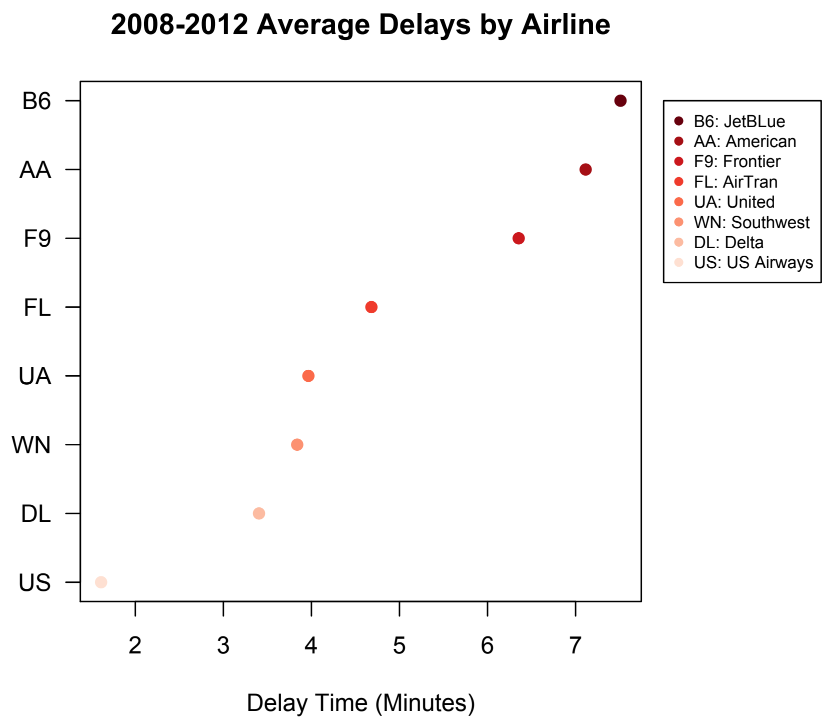


There seems to be a peak in delays in the late 1980s, the late 1990s, and mid 2000s. However, in recent years, the airlines' behavior seems to be improving! It's my cynical suspicion that they're just getting better at padding flight schedules with extra times to appear to be more on time, though.

The dip in delay times in 2002 interestingly corresponds with the dramatic reduction in flight schedules precipitated by the reduced demand for air travel in the wake of the 9/11 terrorist attacks. We can see from the dashed line, which displays the total number of domestic flights each year (for which a delay time was recorded), that there was indeed a noticeable drop in air travel in 2002. Also, at least for 1997-2012, there appears to be a correlation wherein years with more flights had greater mean delay times (a result that makes complete sense given the congested US airspace and lack of extra air traffic control capacity, especially at major airports). However, we can't definitively conclude from eyeballing the plot that this correlation is actually significant or that it indicates a causal relationship. This plot also suggests that we don't have complete data from 1987, since the number of flights in the dataset from that year is quite small. It's also interesting that the observed decline in flight numbers the last few years, in contrast to 2005-2006, corresponds with the global financial disaster which wreaked economic havoc on the airline industry and I know led the airlines to reduce flight capacity to help increase fares and profits.

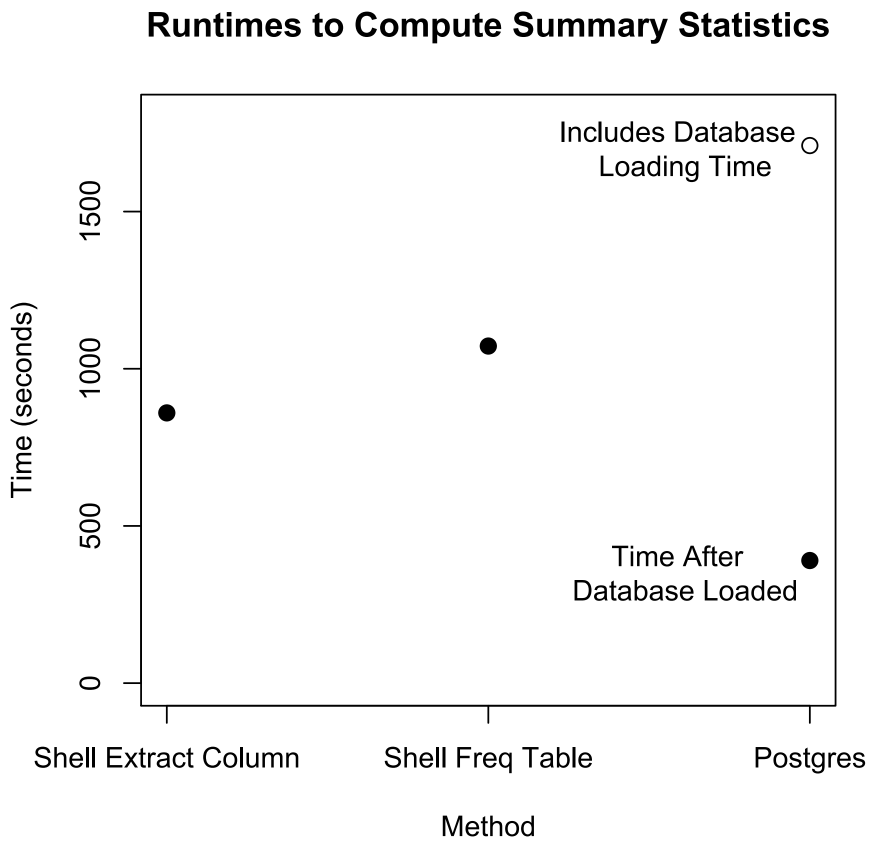
Next, I was interested in seeing whether or not certain airlines tend to have flights that are more delayed. Below, I plotted the mean arrival delay time for 8 large US airlines, for the 2008-2012 dataset. I was surprised that JetBlue had the highest average delay time, since they tend to score well in customer satisfaction surveys. This could be due to the fact that they primarily operate out of congested East Coast airports like JFK and BOS that tend to have more severe delays than airports in smaller cities. I wasn't surprised to see American and Frontier as runners up for the glorious title of most delayed airline, since I've never liked these airlines anyway! I was please that Southwest and Delta, my two favorite domestic airlines, performed quite well.

I also looked at the delay times by a few major airports. Specifically, in the bay area, SFO had the highest average delay times (about 11 minutes), whereas OAK had the lowest (2.8 mintes). SMF's mean arrival delay time was about 4.5 minutes.



3 Time Comparison

Again, I've eschewed a table in favor of a graph. The fastest method depends on how you run the timer. If we include the time to load the data into the database (about 20 minutes), then Postgres is by far the slowest method. However, if we only include the time to performe the relevant queries and calculations *after* the data have already been loaded into the database, then Postgres is by far the fastest method, taking only about 6 minutes. Using the shell to extract columns and then using R to make frequency tables took about 14 minutes, whereas using the shell to both extract the correct column and create the frequency table took about 18 minutes.



4 Discussion

All three methods returned practically identical estimates of the three parameters. More specifically, all three methods returned the same value for the median, the first two methods resulted in identical mean and sd estimates, and the Postgres method led to mean and sd estimates that were very slightly higher than the estimates from the first two methods. I'm not sure what caused this difference. I get the same elevated sd estimate using the Postgres frequency table even if I use the "correct" mean from the first two methods, and I get the same elevated mean even if I use the same find-mean-from-a-frequency-table function that I used before instead of AVERAGE in Postgres. So, this indicates that the frequency table I got from Postgres isn't exactly correct. The most likely explanation is that I made some error in loading the data into the database. I'm not sure what that might have been, but I'm not that familiar with grep and sed so it's possible that I messed something up when using those functions to remove column headers and deal with missing values. I suppose it is also possible that COUNT(*) arrdelay and GROUP BY arrdelay didn't make an exact frequency table, but it seems unlikely to me that the database would only give an approximate count.

In terms of time, I think the most important take-home message is that the database method requires a large initial time investment to set up the table with the correct types and to load the data, but then, once it is built, can return the statistics of interest much more quickly than the two shell and R-based methods. Moreover, not only was Postgres faster to return the summary statistics, but it also provided us the flexibility to address additional questions in the dataset, such as looking at average delay times by carrier or destination airport. In contrast, it would not have been possible to address these questions using the first two methods without revising the computational strategy, since the first two methods involved only storing frequency tables of delay times – no other information, such as carrier or airport information, was acquired.

So, if I were going to be querying this database frequently, and asking many different questions of it (i.e. grouping by different, unanticipated factors), then the Postgres approach seems optimal, as it provides the flexibility to quickly answer new questions about the data. However, if the goal were really just to find the mean, median, and sd and then never look at the data again, then the time investment required to create the database outweighs the speed gains when actually calculating the statistics.

It was also interesting that the method of using the shell to extraxt the correct column and then R to create the frequency table was about 20% faster than the method of using the shell to both extract the column and create the frequency table. So, it appears that "sort — uniq -c" is slower than R's table() function. I don't know enough about how these shell commands work to really speculate on why R's table() function is substantially faster. However, it's possible that R's table() function just calls C code, and this explains why it is quite fast. Another possible reason for the disparity is that the "sort — uniq -c" commands are operating by line, as the result of cut is piped to them. In contrast, the first and faster method involves waiting until the entire column from each file has been returned, and *then* making a table in R. Again, I'm not sure of the details of how these functions work, but it is possible that making the table all at once, as opposed to line by line, contributes to the observed greater speed of making the frequency tables in R than in the shell.

5 Code

```
###### METHOD 1: Loop over files, use shell to extract column of interest and
    then R to make frequency table for each
files = system('ls /Users/matthewmeisner/Downloads/Delays1987_2013',intern=
    TRUE)
runtime1 = system.time({
tables1 = sapply(1:length(files),function(i){
  filename = files[i]
  cat('currently working on file ',filename, '\n')
  filepath = paste0('/Users/matthewmeisner/Downloads/Delays1987_2013/',
      filename)
  # need to find what column we want, since it's annoyingly not the same in
      each file
  if(i<=21){
    col_number = 15
  }else{
    col_number = 45 # this gets the "ARR_DEL15" column; the ARR_DELAY column
        values make no sense! but the ARR_DEL15 values *seem* reasonable...
  }
  shell_command = paste('export LANG=C; cut -f',col_number, '-d, ',filepath)
  delays = system(shell_command,intern=TRUE)
  table(delays[-1]) # -1 removes the column header
})

# need to change names of the tables to all be integers (they are in 3.00 form
     for the later years)
# also need to makes sure NA and 'NA' are called the same thing (some were
    characters and some were actually NAs; this was causing problems when
    merging tables)
for(i in 22:length(tables1)){
  names(tables1[[i]]) = as.character(as.integer(names(tables1[[i]])))
  names(tables1[[i]])[is.na(names(tables1[[i]]))]= 'NA'
}

merged_table1 = mergeFreqTable(tables1,na.rm=T)
```

```
mean1 = meanFreqTable(merged_table1)
median1 = medianFreqTable(merged_table1)
sd1 = sdFreqTable(merged_table1,mean1)

})


###### METHOD 2: Loop over files, use shell to make frequency table for each:

runtime2 = system.time({
tables2 = sapply(1:length(files),function(i){
  filename = files[i]
  cat('currently working on file',filename,'\n')
  filepath = paste0('/Users/matthewmeisner/Downloads/Delays1987_2013/',
      filename)
  # need to find what column we want, since it's annoyingly not the same in
      each file
  if(i<=21){
    col_number = 15
  }else{
    col_number = 45 # this gets the "ARR_DEL15" column; the ARR_DELAY column
        values make no sense! but the ARR_DEL15 values *seem* reasonable...
  }
  shell_command = paste('export LANG=C; cut -f',col_number,'-d,',filepath,'|
      sort|uniq -c')
  t = system(shell_command,intern=TRUE)

  # now, convert this to a table (currently just has strings with both delay
      and count)
  del = unlist(sapply(1:length(t),function(i){
    split = strsplit(t[i],' ')[[1]]
      is_num = grepl('^-|^[0-9]',split) # regular expression weeds out the
          blank/NA column and the ARR_DELAY header

    if(sum(is_num)==2){ # this is 2 if and only if this line of the frequency
        table actually had a delay on it (not NA or a column header)
      delay = as.numeric(split[is_num][2])
      n = split[is_num][1]
      return(c(delay,n))
    }else{
      return(NULL)
    }
  }))

  # we have vector of length (2*number of unique delay times); every other
      entry is the delay and the count
  delays = del[seq(from=1,to=length(del)-1,by=2)]
  counts = as.integer(del[seq(from=2,to=length(del),by=2)])
  names(counts)=delays
  counts
})

merged_table2 = mergeFreqTable(tables2)
```

```
mean2 = meanFreqTable(merged_table2)
median2 = medianFreqTable(merged_table2)
sd2 = sdFreqTable(merged_table2,mean2)


})


##### Method 3: PostgreSQL Database
library(RPostgreSQL)
drv <- dbDriver("PostgreSQL")
con <- dbConnect(drv, dbname="postgres")
# create table for first format of data (1987-2007)
system('~/local/bin/psql -U matthewmeisner postgres -c "CREATE TABLE
    delays1987to2007(carrier CHARACTER(10), arrdelay FLOAT, origin CHARACTER(3)
    ,dest CHARACTER(3));"')

# run this in the shell. Can't use system (at least I don't know how we could)
     since we need both single and double quotes in the shell command
cat 19*.csv 2000.csv 2001.csv 2002.csv 2003.csv 2004.csv 2005.csv 2006.csv
    2007.csv | cut -f 9,15,17,18 -d,| grep -v ArrDelay | ~/local/bin/psql -U
    matthewmeisner postgres -c "COPY delays1987to2007 FROM STDIN DELIMITER ','
     CSV HEADER null 'NA';" # took about 12 minutes

# create table for second format of data (2008-2012) (had to use second table
    since order of columns differs in later dataset, and cut can't rearrange
    columns)
system('~/local/bin/psql -U matthewmeisner postgres -c "CREATE TABLE
    delays2008to2012(carrier CHARACTER(10),origin CHARACTER(3),dest CHARACTER
    (3), arrdelay FLOAT);"')

# run this in the shell:
cat 2008*.csv 2009*.csv 2010*.csv 2011*.csv 2012*.csv  | cut -f 9,15,25,45 -d
    ,| grep -v ARR_DEL15 | sed 's/,$/,NA/g' | ~/local/bin/psql -U
    matthewmeisner postgres -c "COPY delays2008to2012 FROM STDIN DELIMITER ','
     CSV HEADER null 'NA';" # took 10 minutes


# mean - need to do weighted average of the means of the two - but see below
    for a more efficient way to do this, since I have to compute a frequency
    table in order to find the median anyway.
n1 = dbGetQuery(con, "select count(*) from delays1987to2007")
m1 = dbGetQuery(con, "select avg(arrdelay) from delays1987to2007")
n2 = dbGetQuery(con, "select count(*) from delays2008to2012")
m2 = dbGetQuery(con, "select avg(arrdelay) from delays2008to2012")
mean3 =(n1*m1+n2*m2)/(n1+n2)

runtime3 = system.time({
# will need to get a frequency table for sd and median -- no built in command
    for median, and no easy way to merge sd calculated separately on the two
    databases.
# get delays times:
u1 = dbGetQuery(con, "SELECT DISTINCT arrdelay FROM delays1987to2007")
u2 = dbGetQuery(con, "SELECT DISTINCT arrdelay FROM delays2008to2012")
```

```
# get counts of said delay times:
t1 = dbGetQuery(con, "SELECT count(*) FROM delays1987to2007 GROUP BY arrdelay"
    )
t2 = dbGetQuery(con, "SELECT count(*) FROM delays2008to2012 GROUP BY arrdelay"
    )

# creates named numerics (i.e. tables) to be merged, so that we can find
    median and sd:
tab1 = t1[,1]
names(tab1) = u1[,1]
tab2 = t2[,1]
names(tab2) = u2[,1]

# merge tables
merged_table3 = mergeFreqTable(list(tab1,tab2),na.rm=T)

# find median/sd
mean3 = meanFreqTable(merged_table3)
median3 = medianFreqTable(merged_table3)
sd3 = sdFreqTable(merged_table3,mean3)
})

##### make plots of summary statistics and times

par(mfrow=c(1,3),xpd=T)
plot(unlist(c(mean1,mean2,mean3)),pch=19,cex=1.2,xaxt='n',xlab='',main='Mean 
    Arrival Delay',ylab='Estimated Mean Arrival Delay (minutes)')
axis(1,c(1,2,3),c('Shell Column','Shell Table','Postgres'))
plot(c(median1,median2,median3),pch=19,cex=1.2,xaxt='n',xlab='',main='Median 
    Arrival Delay',ylab='Estimated Median Arrival Delay (minutes)')
axis(1,c(1,2,3),c('Shell Column','Shell Table','Postgres'))
plot(c(sd1,sd2,sd3),pch=19,cex=1.2,xaxt='n',xlab='',main='Std Dev of Arrival 
    Delay',ylab='Estimated Std Dev of Arrival Delay (minutes)')
axis(1,c(1,2,3),c('Shell Column','Shell Table','Postgres'))

plot(c(runtime1[3],runtime2[3],runtime3[3]),ylim=c(0,1800),xaxt='n',xlab='
    Method',pch=19,cex=1.2,ylab='Time (seconds)',main='Runtimes to Compute 
    Summary Statistics')
axis(1,c(1,2,3),c('Shell Extract Column','Shell Freq Table','Postgres'))
points(3,runtime3[3]+22*60,cex=1.2)
text(2.6,1700,'Includes Database \n Loading Time')
text(2.6,350,'Time After \n Database Loaded')

############## Make some extra/fun plots:
# get means by year!
means_1987_2007 = sapply(1:21,function(year){
  meanFreqTable(mergeFreqTable(tables[year],na.rm=T))
})
monthly_files_2008_2012 = sapply(2008:2012,function(y){grepl(y,files)})
means_2008_2012 = sapply(1:5,function(y){
  w = which(monthly_files_2008_2012[,y])
  meanFreqTable(mergeFreqTable(tables[w],na.rm=T))
})
yearly_mns = c(means_1987_2007,means_2008_2012)
```

```
par(mar=c(4,4,4,5))
plot(yearly_mns,type='b',lwd=2,pch=19,xaxt='n',xlab='',ylab='Mean Arrival 
    Delay (min)',main='Arrival Delays and Numbers of Domestic Flights, 
    1987-2012')
axis(1,at=1:length(yearly_mns),labels=c(1987:2012),las=2)

# also find number of flights in each year and add that to the graph
nflights_1987_2007 = sapply(1:21,function(year){
  sum(mergeFreqTable(tables[year],na.rm=T))
})
monthly_files_2008_2012 = sapply(2008:2012,function(y){grepl(y,files)})
nflights_2008_2012 = sapply(1:5,function(y){
  w = which(monthly_files_2008_2012[,y])
  sum(mergeFreqTable(tables[w],na.rm=T))
})
yearly_n = c(nflights_1987_2007,nflights_2008_2012)
par(new=T)
plot(yearly_n,type='b',lwd=2,lty=2,xaxt='n',xlab='',ylab='Mean Arrival Delay (
    min)',yaxt='n')
axis(4)
mtext('Number of Flights',side=4,outer=T)
text(30,4.5e6,'Number of Flights',xpd=T,srt=90)
legend('topleft',legend=c('Mean Delay','Number of Flights'),pch=c(19,1),lty=c
    (1,2),cex=.9)

# look at delays by carrier
bc_mean = dbGetQuery(con, "SELECT AVG(arrdelay) FROM delays2008to2012 GROUP BY
     carrier")[,1]
bc_sd = sqrt(dbGetQuery(con, "SELECT VARIANCE(arrdelay) FROM delays2008to2012 
    GROUP BY carrier")[,1])
bc_count = dbGetQuery(con, "SELECT COUNT(*) FROM delays2008to2012 GROUP BY 
    carrier")[,1]
bc_se = bc_sd/sqrt(bc_count)
c = dbGetQuery(con, "SELECT DISTINCT carrier FROM delays2008to2012")
c = gsub(' ','',c[,1])
# airlines to look at: FL (AirTran), US (US Airways), B6 (JetBlue),AA (
    American),UA (United), WN (Southwest), DL (Delta), F9 (frontier)
airlines = c('FL','US','B6','AA','UA','WN','DL','F9')
s = sapply(airlines,function(a){
  c(bc_mean[c==a],bc_se[c==a])
})

o = order(s[1,])

# std errors are too small to draw error bars...weird, maybe I did something
    wrong
col = brewer.pal(9,'Reds')[2:9]
par(mar=c(5,4,4,8),xpd=T)
plot(s[1,][o],1:8,xlab='Delay Time (Minutes)',yaxt='n',ylab='',main='2008-2012
     Average Delays by Airline',pch=19,col=col)
axis(2,at=1:8,airlines[o],las=2)
legend(8,8,legend = c('B6: JetBLue','AA: American','F9: Frontier','FL: AirTran
    ','UA: United','WN: Southwest','DL: Delta','US: US Airways'),cex=.7,pch
    =19,col=rev(col))
```

```
sfo = dbGetQuery(con, "SELECT AVG(arrdelay) FROM delays2008to2012 WHERE dest =
    'SFO'")
oak = dbGetQuery(con, "SELECT AVG(arrdelay) FROM delays2008to2012 WHERE dest =
    'OAK'")
sjc = dbGetQuery(con, "SELECT AVG(arrdelay) FROM delays2008to2012 WHERE dest =
    'SJC'")
smf = dbGetQuery(con, "SELECT AVG(arrdelay) FROM delays2008to2012 WHERE dest =
    'SMF'")
lax = dbGetQuery(con, "SELECT AVG(arrdelay) FROM delays2008to2012 WHERE dest =
    'LAX'")
jfk = dbGetQuery(con, "SELECT AVG(arrdelay) FROM delays2008to2012 WHERE dest =
    'JFK'")
bos = dbGetQuery(con, "SELECT AVG(arrdelay) FROM delays2008to2012 WHERE dest =
    'BOS'")
ord = dbGetQuery(con, "SELECT AVG(arrdelay) FROM delays2008to2012 WHERE dest =
    'ORD'")
den = dbGetQuery(con, "SELECT AVG(arrdelay) FROM delays2008to2012 WHERE dest =
    'DEN'")
mia = dbGetQuery(con, "SELECT AVG(arrdelay) FROM delays2008to2012 WHERE dest =
    'MIA'")
slc = dbGetQuery(con, "SELECT AVG(arrdelay) FROM delays2008to2012 WHERE dest =
    'SLC'")
dfw = dbGetQuery(con, "SELECT AVG(arrdelay) FROM delays2008to2012 WHERE dest =
    'DFW'")
phx = dbGetQuery(con, "SELECT AVG(arrdelay) FROM delays2008to2012 WHERE dest =
    'PHX'")

# didn't quite have time to make a plot!
d = data.frame(delays = c(sfo[1,1],oak[1,1],sjc[1,1],smf[1,1],lax[1,1],jfk
    [1,1],bos[1,1],ord[1,1],den[1,1],mia[1,1],slc[1,1],dfw[1,1],phx[1,1]))
d$grp = cut(d$delays,9)

# plot bay area airports and delay times
##### functions for calculating summmary statistics from tables

# function to merge tables (we have a table from each year and want a combined
    one with values summed across all years)
mergeFreqTable = function(tt,na.rm=FALSE){
  # tt needs to be a list of tables to be merged
  # returns a named integer vector; names are delay times and value are counts
  # na.rm deterines in NAs are included in the final table

  # first, find all the unique values in all the tables combined
  all_names = unlist(lapply(tt,function(t){names(t)}))
  unique_names = unique(all_names)

  # loop over all possible values; within that loop over all years and extract
      corresponding counts. Then sum them.
  merged = sapply(unique_names,function(delay){
    sum(sapply(tt,function(t){t[delay]}),na.rm=T)
  })
  # remove NAs if desired
  if(na.rm){
    w = which(is.na(names(merged)))
    merged = merged[-w]
  }
  merged
}

# next, need functions for mean, median, and sd from freq table
meanFreqTable = function(t){
  # takes a table (or named vector) and finds the mean of all the names,
      assuming each name is replicated the number of times corresponding to
      the entry for that name
  sum(as.integer(names(t))*t)/sum(t)
}

medianFreqTable = function(t,debug=F){
  # takes a table (or named vector) and finds the median of all the names,
      assuming each name is replicated the number of times corresponding to
      the entry for that name
  # this won't get the right answer if the median need to be an average of two
       values. (This fxn will return the lower of those 2 numbers). However,
      given the strong tendency for delays to be near 0, it's exceptionally
      unlikely that for this application this averaging will be needed.  I
      also checked with the debug option in my function, and the cumulative
      sum at the value before the median is reached is much less than half,
      and the cumulative sum right after the median is reached is much more
      than half. So, it's not an issue for these data.
  n = sum(t)
  half = ceiling(n/2)
  # sort the names; we will start from the lowest and keep a cumulative sum of
       counts until we reach the midway point.
  sorted_names = sort(as.integer(names(t)))
  cumul_sum = 0
  i = 1
  while(cumul_sum<half){
    if(debug){cat('cumulative sum so far is:',cumul_sum,'\n')}
    current_number = sorted_names[i]
    cumul_sum = cumul_sum + t[as.character(current_number)]
    i = i+1
  }
  if(debug){cat('cumulative sum after final bin is:',cumul_sum,'\n')}
  current_number
}

sdFreqTable = function(t,mean){
    # takes a table (or named vector) and its mean, and finds the sample SD (
        MLE, biased estimate) of all the names, assuming each name is
        replicated the number of times corresponding to the entry for that
        name
  var_mle = sum(t*(as.integer(names(t))-mean)**2)/sum(t)
  sd_mle = sqrt(var_mle)
  sd_mle
}
```

```
########### save results
info = list(sessionInfo(),Sys.info())
names(info) = c('sessionInfo','systemInfo')

runtimes = list(runtime1,runtime2,runtime3)
means = list(mean1,mean2,mean3)
medians = list(median1,median2,median3)
sds = list(sd1,sd2,sd3)

save(info,means,medians,sds,file='~/Documents/STA250Winter2014/results_final.
    rda')
```