Airline Data Streaming

Intro

Data streaming has become an increasingly important technique in recent years due to both size and the dynamic nature of data. In this assignment, we work with data in "blocks". In other words, we don't read the entire data into memory but rather read a subset of the data, process it immediately, discard it right away and continue with the next subsets of the data. Therefore, we can have very large scale of data streaming into our memory and extract statistical information at the same time.

We will compare the speed of processing data using Unix shell tools, pure R tools and combination of both tools. We found that a connection between R and Unix is an efficient way to stream and process the data. Database tools such as SQLite is also explored to improve the efficiency of processing large scale of data. We will compute the number of flights leaving each of the airports (LAX, OAK, SFO, SMF) from 1987 to 2008. We will also compute the mean and standard deviation of the arrival delay times for flights departing from these four airports.

Data Structure

The data sets are constructed by 22 csv files (totally 12 Gigabytes), which contain domestic airline flights summaries as well as arrival and departure performance each year from 1987 to 2008. We have 29 columns in each data set for different years. We have particular interests in the 15th column for arrival delay (ARR_DELAY) and 17th column for departure airport (ORIGIN). Other relevant time variable incudes YEAR, MONTH, DAY_OF_MONTH, DAY_OF_WEEK.

The data are available at http://eeyore.ucdavis.edu/stat242/data/. Please see Appendix I for R codes to download and uncompressed files programmatically.

Tools Exploration

The tools used in this assignment include Unix shell tools, R tools and database engine. We compare the speed of counting flights leaving each of the airports, specifically LAX, OAK, SFO and SMF for year 2008 to find the most efficient way to obtain statistical information (e.g. counts, means and standard deviations) for the entire data sets. Following is the system time needed to for counting:

	Shell		R Streaming		Database
	i. Multiple passes (wc -l)	**ii. One pass (sort|uniq -c)**	**i. readLines (file(csv))**	**ii. readLines (pipe(egrep))**	**RSQLite**
user	67.933	62.077	268.294	19.476	0.577
system	0.833	0.242	1.229	0.061	0.016
elapsed	55.612	61.230	269.525	57.478	0.576

Table -1. System.time for counts of flight leaving LAX, OAK, SFO and SMF in 2008

We found that Unix shell tools are extremely fast in taking subsets of data using "cut" and "grep" commands, which select the specific column and search with regular expression. "sort" and "uniq -c" is easier to write without doing multiple passes but it takes longer to sort given the data set is larger enough.

```
system.time(SHcounts.wc<-system("for airport in LAX OAK SFO SMF; do cut -f 17
-d , 2008.csv | grep $airport | wc -l; done",intern=TRUE))

system.time(SHcounts.uniq<-system("egrep '([0-9]|NA),(LAX|OAK|SFO|SMF),[A-
Z]' 2008.csv | cut -f 17 -d , | sort | uniq -c",intern=TRUE))
```

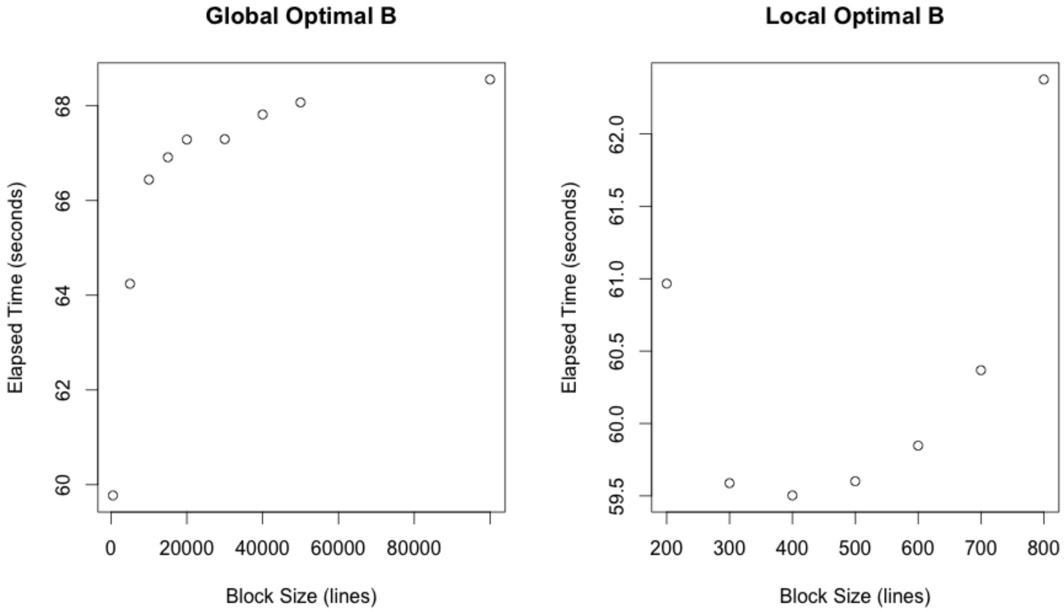
Reading data in blocks directly from csv file in R is the slowest way to stream and process the data. However, prepare subset data in the Unix environment with regular expression significantly reduce the processing time from 269.525 seconds to 57.478 seconds. This is close enough to counting lines in shell scripting. Codes are available in Appendix II.

```
con<-pipe("egrep '([0-9]|NA),(LAX|OAK|SFO|SMF),[A-Z]' 2008.csv","r")

rcount<-function(B,con,year){
 B=as.integer(B)
 # set up counters
 Rcounts<-structure(c(data=rep(0,4)),names=c("LAX","OAK","SFO","SMF"))
 # reading blocks
 while(TRUE){
  txt=readLines(con,n=B)
  if (length(txt)==0)
    break
  # counting
  temp=sapply(strsplit(txt,","),"[[",17)
  update<-as.numeric(table(temp)[names(Rcounts)])
  update[is.na(update)]<-0
  Rcounts<-Rcounts+update
 }
 Rcounts
}

system.time(RSHcount<-rcount(400L,con,"2008"))
```

We are interested in exploring the best block size that is highly relevant to the performance for reading data while Unix is processing data line by line. The parameterized function and plot function in R are handy for us to find the optimal block size for streaming data from Unix to R. The optimal block size for reading lines directly from csv files and connection from Unix are different, specifically 12000L versus 4000L, because R is slower than Unix shell tools in taking the subsets of data and thus would be better of in dealing with larger block size in counting. But if we stream data indirectly from the connection of

Unix using pipe(), we are more efficient in counting targeted observations with smaller block size than that in reading directly from csv files.

Following are plots for the exploration of optimal block size in the case of streaming prepared data from Unix to R :



The optimal B is 400 lines for an elapsed time less than 60 seconds if we pre-process data in Unix and then stream to R. We can expect to improve the efficiency by interfacing to C code because we could do faster while loops.

With the guarantee of speed, we would like to use R for its good extensibility with handy statistical tools. Functions such as table(), sum() and var() will help us to obtain the counts, mean and standard deviation.

Statistics: counts, means and standard deviation

Grounded on the data streaming in R reading from Unix, a function rstats() is developed to get the counts, obtain the sum and cumulate the variance for airlines leaving LAX, OAK, SFO and SMF. The mean is the sum divided by counts while the standard deviation is the square root of variance divided by counts.

(I) Outliers
Notice that the counts here are different from the counts obtained in the previous section (Tools Exploration) because we exclude the observations with NA on arrival delay. We will obtain more accurate mean and standard deviation for the arrival delay time by excluding the NA observations.

The counts of ORIGIN for each airport in 2008:
> LAX 184048; > OAK 53679; > SFO 118635; > SMF 45364; #NA included
> LAX 181308; > OAK 52818; > SFO 116029; > SMF 44907; #NA excluded

The counts of ORIGIN for each airport from 1987 to 2008:

```
> LAX 3947365; > OAK 1134520; > SFO 2605579; > SMF 790378; #NA included
> LAX 3879885; > OAK 1121028; > SFO 2551563; > SMF 782171;  #NA excluded
```

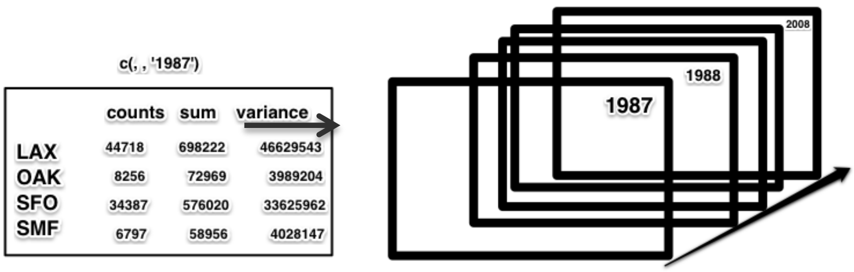
(II) AWK Validation

We use awk with under UNIX to do a quick validation of the sum of arrival delay time. It would print the counts and cumulated arrival delay time. The statistical results in R are validated by this AWK validation for 1987, 2001 and 2008.

```
system.time(cmean<-system("egrep '([0-9]|NA),LAX,[A-Z]' 2008.csv | cut -f 15 -d , | awk 'BEGIN {s=0; c=0}; {s=s+$1;c=c+1}; END {print c,s, s/c}'",intern=TRUE))
```

(III) Results

The statistical results are constructed as arrays. Columns are the statistics including counts, sum, variance; rows are the airports of our interests, specifically for LAX, OAK, SFO and SMF; the third dimension is the year from 1987 to 2008. Please see Appendix III for the statistics obtained from data streaming.



We can use apply function to obtain the mean and std.dev for each year. Results are available in Appendix IV.

```
#counts
apply(RS.ALL.ARRAY,c(3),function(x){x[,1]})
#means
apply(RS.ALL.ARRAY,c(3),function(x){x[,2]/x[,1]})
#standard deviation
apply(RS.ALL.ARRAY,c(3),function(x){sqrt(x[,3])/x[,1]})
```

To obtain the statistics for the entire data set from 1987 to 2008, we can use apply function to sum up the counts, sum and variance through out the time.

```
TOTAL<-apply(RS.ALL.ARRAY,c(1,2),sum)
TOTAL[,1]#counts
TOTAL[,2]/TOTAL[,1]  #mean
sqrt(TOTAL[,3]/TOTAL[,1]) #std.dev
```

The statistical results for the entire data set from 1987 to 2008 are shown in the table below. Statistics for individual year are available in Appendix V and VI.

	Counts	Means	Std.Dev
LAX	3879885	6.011921	26.31361
OAK	1121028	5.027054	21.09526
SFO	255156	8.008065	29.06354
SMF	782171	5.342205	24.14323

Conclusion

In this assignment, we work with data in "blocks" to break through the limitation of memory and improve speed of data processing. Learning data streaming is an important step to learn parallel computing which carry out the calculations simultaneously. If data streaming is processing data in an time series way, parallel computing is processing data from an cross sectional prospective.

Besides the idea of reading data in blocks, I learn how to use Unix shell tools to pre-process data and open connections via pipe() to stream the data into R. Basic commands such as grep, cut, uniq, sort and the redirection operators in the shell are very handy in pre-processing data. In R, I learn how to open connections include pipe, file, url, bzfile, gzfile, xzfile.

Computing the same information from relational database is part of the assignment. I create tables and write queries in SQLite. It turns out that the speed is amazingly fast. Database makes life easier with command such as GROUP BY (similar to tapply).

Character encoding is an issue while dealing with the csv file for 2001 and 2002. It is important to read the data with correct encoding which pairs each character from a given repertoire particular bit patterns.

URL for the repositiory on GitHub:
https://github.com/ykangxie/airport.git

Appendix I.

```
##############################################
# Download and Unzip Files
##############################################

#/*get fileURl*/
#txt = readLines("http://eeyore.ucdavis.edu/stat242/data")
#ll = grep("csv.bz2", txt)   #line.number containing "csv.bz2"
#files = gsub(".*([0-9]{4}.(csv).bz2).*", "\\1", txt[ll]) #file names
#fileUrl<-sprintf("http://eeyore.ucdavis.edu/stat242/data/%s", files) #fileUrl

#/*download bz2 files*/
#destFile<-sprintf("./Desktop/airport/%s",files)
#mapply(function(url,output){download.file(url,destfile=output,method="curl")
},fileUrl,destFile)

#/*unzip bz2 files*/
#cmd<-sprintf("cd ./Desktop/airport; bunzip2 -d %s", files)
#sapply(cmd,system)
```

Appendix II. Tools Exploration

```
files = gsub(".*([0-9]{4}.csv).*", "\\1", list.files()[grep(".*([0-9]{4}.csv).*",
list.files())]])
year = gsub(".*([0-9]{4}).csv.*", "\\1", files[grep(".*([0-9]{4}.csv).*", files)]])

#########################################################
# Tools Exploration- system.time(counts) for 2008.csv
#########################################################
setwd("./Desktop/airport")

dept=c("LAX","OAK","SFO","SMF")
year="2008"
```

#(I) Shell Scripting

#1.1 /grep & wc -l/ [multiple passes]

```
system.time(SHcounts.wc<-system("for airport in LAX OAK SFO SMF; do cut -f 17
-d , 2008.csv | grep $airport | wc -l; done",intern=TRUE))
#user  system elapsed
#67.933   0.833  55.612
```

#1.2 /egrep & (sort + uniq -c)/ [one pass]

```
system.time(SHcounts.uniq<-system("egrep '([0-9]|NA),(LAX|OAK|SFO|SMF),[A-
Z]' 2008.csv | cut -f 17 -d , | sort | uniq -c",intern=TRUE))
#user  system elapsed
#62.077   0.242  61.230
```

#(II) R Streaming (csv files)

```
rstreaming<-function(B,csvfile="2008.csv"){
 B=as.integer(B)
 con=file(csvfile,"r")
 Rcounts<-structure(c(data=rep(0,4)),names=c("LAX","OAK","SFO","SMF"))
 while(TRUE){
  txt=readLines(con,n=B)
  if (length(txt)==0)
    break
  temp=sapply(strsplit(txt,","),"[[",17)
  update<-as.numeric(table(temp)[names(Rcounts)])
  update[is.na(update)]<-0
  Rcounts<-Rcounts+update
 }
 Rcounts
}

system.time(rstreaming(400L,"2008.csv")) #268.294   1.229 269.525
system.time(rstreaming(12000L,"2008.csv")) #255.774   0.816 256.571
```

#(III) R Streaming from Unix via pipe()

```
con<-pipe("egrep '([0-9]|NA),(LAX|OAK|SFO|SMF),[A-Z]' 2008.csv","r")
rcount<-function(B,con,year){
 B=as.integer(B)
 Rcounts<-structure(c(data=rep(0,4)),names=c("LAX","OAK","SFO","SMF"))
 while(TRUE){
  txt=readLines(con,n=B)
  if (length(txt)==0)
   break
  temp=sapply(strsplit(txt,","),"[[",17)
  update<-as.numeric(table(temp)[names(Rcounts)])
  update[is.na(update)]<-0
  Rcounts<-Rcounts+update
 }
 Rcounts
}
system.time(RSHcount<-rcount(400L,con,"2008"))
#user  system elapsed
#19.476   0.061  57.478
```

Appendix III. Function rstats()

```r
#rstats
rstats<-function(B,con){
  B=as.integer(B)
  #set up counters
  Rcounts<-structure(c(data=rep(0,4)),names=c("LAX","OAK","SFO","SMF"))
  Rsum<-structure(c(data=rep(0,4)),names=c("LAX","OAK","SFO","SMF"))
  Rsumvar<-structure(c(data=rep(0,4)),names=c("LAX","OAK","SFO","SMF"))
  #reading blocks
  while(TRUE){
    txt=readLines(con,n=B)
    if (length(txt)==0)
      break
    #ArrDelay & Airport
    ArrDelay=as.numeric(sapply(strsplit(txt,","),"[[",15))
    Origin=as.factor(sapply(strsplit(txt,","),"[[",17))
    #dropNA
    dropNA<-!is.na(ArrDelay)
    ArrDelay<-ArrDelay[dropNA]
    Origin<-Origin[dropNA]
    #1.1 Rcounts
    update<-as.numeric(table(Origin)[names(Rcounts)])
    #update<-tapply(ArrDelay,Origin,length)[names(Rcounts)] #table() is faster than tapply(length)
    update[is.na(update)]<-0
    Rcounts<-Rcounts+as.numeric(update)
    #1.2 Rsum
    update<-tapply(ArrDelay,Origin,sum)[names(Rsum)]
    update[is.na(update)]<-0
    Rsum<-Rsum+as.numeric(update)
    #1.3 Rvar
    update<-tapply(ArrDelay,Origin,function(x){var(x)*length(x)})[names(Rsumvar)]
    update[is.na(update)]<-0
    Rsumvar<-Rsumvar+as.numeric(update)
  }
  cbind(Rcounts,Rsum,Rsumvar)
}
```

Appendix IV. Codes for All-Years Statistics

```
################################################
# All Years: 1987-2012.csv
################################################
source('rstats.R')
Sys.setlocale(locale="C")
```

 # i. Each Year (array)

```
year = gsub(".*([0-9]{4}).csv.*", "\\1", files[grep(".*([0-9]{4}.csv).*", files)])
cmds<-sprintf("egrep '([0-9]|NA),(LAX|OAK|SFO|SMF),[A-Z]' %s.csv",year)
system.time(RS.ALL<-lapply(cmds,function(cmd){con<-pipe(cmd,"r")
                    tmp<-rstats(400L,con)
                    close(con)
                    tmp}))
#user   system  elapsed  #1824.276   10.011 1285.353
names(RS.ALL)<-year
RS.ALL.ARRAY<-array(unlist(RS.ALL), dim = c(nrow(RS.ALL[[1]]),
ncol(RS.ALL[[1]]), length(RS.ALL))) #reconstruction: (4 x 3 x 22)
rownames(RS.ALL.ARRAY)<-rownames(RS.ALL[[1]])
colnames(RS.ALL.ARRAY)<-colnames(RS.ALL[[1]])
dimnames(RS.ALL.ARRAY)[[3]]<-year
#counts
apply(RS.ALL.ARRAY,c(3),function(x){x[,1]})
#means
apply(RS.ALL.ARRAY,c(3),function(x){x[,2]/x[,1]})
#standard deviation
apply(RS.ALL.ARRAY,c(3),function(x){sqrt(x[,3])/x[,1]})
```

 # ii. All Year (matrix)

```
TOTAL<-apply(RS.ALL.ARRAY,c(1,2),sum)
TOTAL[,1]#counts
TOTAL[,2]/TOTAL[,1]  #mean
sqrt(TOTAL[,3]/TOTAL[,1]) #std.dev
TOTAL.STATS<-
structure(data.frame(cbind(TOTAL[,1],TOTAL[,2]/TOTAL[,1],sqrt(TOTAL[,3]/T
OTAL[,1]))),names=c("counts", "means", "std.dev"))
```

 # iii. All Years (awk-validation)

```
# notes: the awk validation is a quick validation which has a larger n because of
the inclusion of NA ArrDelay observations

system.time(cmean2<-system("egrep '([0-9]|NA),LAX,[A-Z]' [12]*.csv | cut -f 15 -
d , | awk 'BEGIN {s=0; c=0}; {s=s+$1;c=c+1}; END {print c,s,s/c}'",intern=TRUE))

# counts, sum & means for (LAX [12]*.csv)
#[1] "3947365 5.90915"
```

Appendix V. Statistical Information for Each Year

```
> RS.ALL.ARRAY
, , 1987

   Rcounts   Rsum  Rsumvar
LAX   44718 698222 46629543
OAK    8256  72969  3989204
SFO   34387 576020 33625962
SMF    6797  58956  4028147

, , 1988

   Rcounts   Rsum  Rsumvar
LAX  168315 494824 72032151
OAK   28069 119662  8219810
SFO  130309 987464 66265631
SMF   27202  77202  8481110

, , 1989

   Rcounts    Rsum  Rsumvar
LAX  161229 1026141 82462028
OAK   27849  170309  9736698
SFO  124400  909651 75711416
SMF   25118  136822 11395138

, , 1990

   Rcounts    Rsum  Rsumvar
LAX  168100 1017515 77791138
OAK   36207  125261 10410748
SFO  128921  820036 61602123
SMF   24919  107973  8888563

, , 1991

   Rcounts    Rsum  Rsumvar
LAX  155002 1059089 80226036
OAK   39793  159760 11079901
SFO  122423 1098101 66316359
SMF   26805  117460  8886301

, , 1992

   Rcounts   Rsum  Rsumvar
LAX  153756 668930 61829138
OAK   36497  55264  6543787
SFO  121240 550297 44178952
SMF   29709  69666  8026645

, , 1993

   Rcounts   Rsum  Rsumvar
LAX  151020 445402 58514989
OAK   39316  82182  7731066
SFO  116523 411976 49284859
SMF   29460  89367  8990148

, , 1994

   Rcounts   Rsum  Rsumvar
LAX  151919 706131 56519438
OAK   44138 182071  9094462
SFO  117053 528507 48494829
SMF   28717 123118  8317834

, , 1995

   Rcounts    Rsum   Rsumvar
LAX  175894 1674907 117273324
OAK   64524  364986  22515629
SFO  126283 1128107  86908794
SMF   35628  176701  15460096

, , 1996

   Rcounts    Rsum   Rsumvar
LAX  179193 1840182 144844249
OAK   59669  346971  22738325
SFO  132074 1639385 138390756
SMF   36424  230301  22428645

, , 1997

   Rcounts    Rsum   Rsumvar
LAX  183447 1496839 131055064
OAK   57718  302076  19093393
SFO  135199 1371023 106307297
SMF   36767  229068  18147255

, , 1998

   Rcounts    Rsum   Rsumvar
LAX  178421 1145846 153113710
OAK   55057  364247  27315707
```

SFO 133557 1748564 181440746
SMF 36073 252998 21311474

, , 1999

	Rcounts	Rsum	Rsumvar
LAX	185812	1612380	156361588
OAK	54725	366694	22596092
SFO	131444	1206069	153878008
SMF	36308	261399	20734114

, , 2000

	Rcounts	Rsum	Rsumvar
LAX	203598	2430501	237508835
OAK	56106	616442	40836851
SFO	127532	1857824	200120593
SMF	38301	421899	31972521

, , 2001

	Rcounts	Rsum	Rsumvar
LAX	124679	649956	82336267
OAK	47206	319414	22143188
SFO	26523	80109	22390373
SMF	26216	148242	13099860

, , 2002

	Rcounts	Rsum	Rsumvar
LAX	170178	44932	102203954
OAK	59085	278342	27347972
SFO	86901	40047	61185023
SMF	37570	140758	20426443

, , 2003

	Rcounts	Rsum	Rsumvar
LAX	221122	169021	126010691
OAK	66662	99080	29496751
SFO	120866	93931	70251106
SMF	46338	114649	23423413

, , 2004

	Rcounts	Rsum	Rsumvar
LAX	229731	980717	154611681
OAK	70049	320588	38459632
SFO	127990	526323	88707721
SMF	48110	328413	37680364

, , 2005

	Rcounts	Rsum	Rsumvar
LAX	228122	1081568	153685230
OAK	69652	377980	38562684
SFO	127055	813418	118866372
SMF	50161	314357	35258171

, , 2006

	Rcounts	Rsum	Rsumvar
LAX	230381	1341989	195359125
OAK	73761	383264	49994165
SFO	129245	1234712	143417625
SMF	53218	265041	41288074

, , 2007

	Rcounts	Rsum	Rsumvar
LAX	233940	1779392	221358053
OAK	73871	419482	43786892
SFO	135609	1582832	172304007
SMF	57423	364578	51483842

, , 2008

	Rcounts	Rsum	Rsumvar
LAX	181308	961077	174730165
OAK	52818	108424	27175568
SFO	116029	1228686	165629128
SMF	44907	149550	36196022

Appendix VI. Statistical Results for Each Year

```
> #counts
> apply(RS.ALL.ARRAY,c(3),function(x){x[,1]})
      1987   1988   1989   1990   1991   1992   1993   1994   1995   1996   1997   1998   1999
LAX 44718 168315 161229 168100 155002 153756 151020 151919 175894 179193 183447 178421 185812
OAK  8256  28069  27849  36207  39793  36497  39316  44138  64524  59669  57718  55057  54725
SFO 34387 130309 124400 128921 122423 121240 116523 117053 126283 132074 135199 133557 131444
SMF  6797  27202  25118  24919  26805  29709  29460  28717  35628  36424  36767  36073  36308
      2000   2001   2002   2003   2004   2005   2006   2007   2008
LAX 203598 124679 170178 221122 229731 228122 230381 233940 181308
OAK  56106  47206  59085  66662  70049  69652  73761  73871  52818
SFO 127532  26523  86901 120866 127990 127055 129245 135609 116029
SMF  38301  26216  37570  46338  48110  50161  53218  57423  44907
> #means
> apply(RS.ALL.ARRAY,c(3),function(x){x[,2]/x[,1]})
         1987     1988     1989     1990     1991     1992     1993     1994     1995      1996
LAX 15.613891 2.939869 6.364494 6.053034 6.832744 4.350594 2.949291 4.648076 9.522252 10.269274
OAK  8.838299 4.263137 6.115444 3.459580 4.014776 1.514207 2.090294 4.125040 5.656593  5.814929
SFO 16.751098 7.577865 7.312307 6.360764 8.969728 4.538906 3.535577 4.515109 8.933166 12.412625
SMF  8.673827 2.838100 5.447169 4.332959 4.382018 2.344946 3.033503 4.287286 4.959610  6.322782
         1997      1998     1999     2000     2001      2002      2003     2004     2005
LAX  8.159517  6.422148 8.677480 11.93774 5.213035 0.2640294 0.7643789 4.268980 4.741182
OAK  5.233653  6.615816 6.700667 10.98710 6.766386 4.7108742 1.4863040 4.576625 5.426693
SFO 10.140778 13.092268 9.175535 14.56751 3.020360 0.4608347 0.7771499 4.112220 6.402094
SMF  6.230261  7.013500 7.199488 11.01535 5.654638 3.7465531 2.4741896 6.826294 6.266960
        2006      2007      2008
LAX 5.825085  7.606190  5.300798
OAK 5.196025  5.678575  2.052785
SFO 9.553267 11.672028 10.589473
SMF 4.980289  6.348989  3.330216
> #standard deviation
> apply(RS.ALL.ARRAY,c(3),function(x){sqrt(x[,3])/x[,1]})
         1987       1988       1989       1990       1991       1992       1993       1994
LAX 0.1527032 0.05042436 0.05632275 0.05246834 0.05778570 0.05114045 0.05065229 0.04948651
OAK 0.2419209 0.10214190 0.11204597 0.08911451 0.08364910 0.07009022 0.07072132 0.06832440
SFO 0.1686332 0.06246975 0.06994558 0.06087993 0.06651925 0.05482287 0.06024834 0.05949290
SMF 0.2952809 0.10705957 0.13439241 0.11964242 0.11121022 0.09536280 0.10177724 0.10043060
          1995       1996       1997       1998       1999       2000       2001       2002
LAX 0.06156709 0.06716292 0.06240455 0.06935233 0.06729631 0.07569486 0.07277836 0.05940602
OAK 0.07353952 0.07991540 0.07570600 0.09492786 0.08686221 0.11389819 0.09968341 0.08850857
SFO 0.07382220 0.08907098 0.07626198 0.10085577 0.09437295 0.11092431 0.17840541 0.09001146
SMF 0.11036075 0.13002111 0.11586367 0.12797481 0.12541238 0.14763126 0.13805969 0.12029711
          2003       2004       2005       2006       2007       2008
LAX 0.05076586 0.05412545 0.05434366 0.06066947 0.06359796 0.07290662
OAK 0.08147207 0.08853207 0.08915589 0.09585899 0.08957732 0.09869777
SFO 0.06934616 0.07358761 0.08580997 0.09265898 0.09679639 0.11091794
SMF 0.10444504 0.12759163 0.11837601 0.12074072 0.12495384 0.13397272
```

Appendix VII. SQL Part

```
#---------------------------
#SQLite
#----------------------------
library("RSQLite")
dr=dbDriver("SQLite")
con=dbConnect(dr,dbname =
"airlineTable.db")
system.time(rr<-
dbSendQuery(con,"SELECT count(*)
FROM delays WHERE Origin IN
('LAX', 'OAK', 'SFO', 'SMF') GROUP
BY Origin;"))
system.time(fetch(rr,100))
#counts
fetch(rr,100)
# user  system elapsed
# 0.557   0.016   0.576
library("RSQLite.extfuns")
con=dbConnect(dr,dbname =
"airlineTable.db")
init_extensions(con)
#average
aa<-dbSendQuery(con,"SELECT
avg(ArrDelay) FROM delays WHERE
Origin IN ('LAX', 'OAK', 'SFO', 'SMF')
GROUP BY Origin;")
fetch(aa,4)
#std.dev
std<-dbSendQuery(con,"SELECT
stdev(ArrDelay) FROM delays
WHERE Origin IN ('LAX', 'OAK',
'SFO', 'SMF') GROUP BY Origin;")
fetch(std,4)
```