Homework 4: Shell and Database

Kejia Hu

1. Using the shell

To write a shell command that will give a line with the name of each of these four airports and corresponding count. Origin airport is among LAX, OAK, SFO, SMF. For all data after 2000,

```
awk -F '$17=="LAX"{LAX++} $17=="OAK"{OAK++}$17=="SFO"{SFO++} $17=="SMF"{SMF++} END{print "LAX =", LAX, "OAK = ", OAK, "SFO =", SFO, "SMF = ", SMF }' 20*.csv
```

2. Using R and Shell

To call the shell command from R, we use the structure

```
system(cmd, intern=True)
```

Since we need to compute the mean and standard deviation of the arrival delay times for each of the four airports for all flights either, we call the command from system.

We use the following command to count the total appearance

```
cmd=”awk -F '$17~/LAX/{LAX++} $17~/OAK/{OAK++}$17~/SFO/{SFO++} $17~/SMF/{SMF++} END{print LAX, OAK, SFO, SMF }' 20*.csv”
system(cmd, intern=True)
```

To calculate the mean, variance, the structure is the same but just replace the content within the bracelet. {LAX++}→ {LAXD=$15; SLAXD+= x*x; totalLAX += $15; LAX++}

This calculates the number of flights departure from LAX, the sum of their arrival delay and the sum of square of their arrival delay.

To compute the mean, we simply need to compute totalLAX/LAX

To compute the standard deviation, we simply need to compute {SLAXD-(totalLAX)^2}/LAX And then take its square root.

The simple command using LAX as an illustration is:

```
gawk -F, '$17=="LAX"{ total ++} END{print total}' 2001.csv
gawk -F, '$17=="LAX"{ ave +=$15; total ++} END{print ave/total}' 2001.csv
gawk -F, '$17=="LAX"{ sq=$15; wasabi +=sq*sq; ave +=sq; total ++} END{print (wasabi-ave*ave/total)/total}' 2001.csv
```

3. Using R and database

The bonus points sound attractive, so I explore how to use MYSQL and ODBC with R. Before we use R to access a MySQL database, we need the following component: MySQL Community Server, MySQL Connector/ODBC and R with package RODBC. MySQL and Connector/ODBC can be found online and the installation is quite straight forward. After installing it, we need to add the information about the databases which I wish to access with R. The setting procedure is Control Panel→Administrative Tools→Data Sources(ODBC)→Add…→ MySQL ODBC 5.2 Driver with allowance to set up user and password.
To read a data file into MySQL, we need to start the Command Window. The code to create the MySQL data table is attached. The R code to access the table using RODBC is also attached.

```
channel <- odbcConnect("mytest", uid = "root", pwd = "******")
query <- "SELECT ……"
queryResult <- sqlQuery(channel, query)
str(queryResult)
```

The main selections we did are

```
SELECT AVG(ArrDelay) FROM airline WHERE Origin ='LAX'
SELECT COUNT(ArrDelay) FROM airline WHERE Origin ='LAX'
SELECT STDEV(ArrDelay) FROM airline WHERE Origin ='LAX'
```

A much easier way to use the SQLite is using the following command and they have equivalent effect.

```
read.csv.sql("2001.csv",sql = "select count(ArrDelay) from file where Origin='LAX'")
read.csv.sql("2001.csv",sql = "select avg(ArrDelay) from file where Origin='LAX'")
read.csv.sql("2001.csv",sql = "select stdev(ArrDelay) from file where Origin='LAX'")
```

4. Compare the difference between using shell and database

From the computation result and the speed for two different approaches, we see that database is different from the shell significantly. The computation difference I think is mainly comes from different treatment to the missing values. Database will ignore missing values and calculate the mean and standard deviation via the shell command doesn't consider that. That's why the statistics is always smaller in the output of shell then database. Here we consider the estimates from database are more reliable.

Approach	LAX			OAK		
	Count	Mean	SD	Count	Mean	SD
Shell	224984	5.42911	28.0002	62763	7.08174	23.7692
Database	37257	8.0009	32.9641	9551	12.20951	30.09148

Approach	SFO	SMF

	Count	Mean	SD	Count	Mean	SD
Shell	121735	5.74017	30.5338	40199	6.01236	25.0583
Database	19839	8.57941	36.15043	6149	10.45259	30.32461

Comparing the average speed of two approaches, shell overall is slower than database approach for counting total and calculating mean and standard deviation. In general, Database approach is preferred.

Approach	Count	Mean	SD
Shell	11.2	11.8	11.4
Database	2.3	2.5	2.3

4. Block Size effect

Block size processing means for each time we only deal with a chunk of data. The selecting parameter is B, the size of the chunk which usually is equivalent to the number of rows to process.

There are two extremes：

1. If we set B=1, we basically treat data line by line and this is not very sufficient.
2. If we set B=number of rows in the full data, we only deal with data in one chunk, the entire original data.

Before we treat the data, I don't know about the optimal choice of B, but I suspect the B is neither too large nor too small. If B is too small, the time is mainly used in reading the data and even though each line doesn't cost much time but the time multiplied by the number of rows in the large data may still be considerable large. If B is too small, it occupies a large amount of the memory and thus lowers the computation speed. Thus neither of the extreme cases should be optimal. We will use the data set to be our judge and let it tells us about the selection.

For database queries, we can do

```
sqlFetch(channel, "airlines", max = B, rows_at_time = 1)
sqlFetchMore(channel, max = B)
```

For shell connection, we can do

```
ff <- file("2001.csv", open="rt")      #To open a connection
While (True) {
        x <- readLines(ff, n=B)         #read data block by block
         if(length(x)==0)stop("end") #until they reach an end
       result[block]=system(cmd,x ) #process the reading data
       block= block+1
      }
close(ff)                                  #To close the connection
```

The table illustrates how the block affects the speed of calculation. It supports our

expectation for extreme large blocks or extreme small blocks. And we consider the optimal block size is in the middle. Since the optimal processing blocks is related to the size of the dataset, we consider in our sampled data B=100,000 is sufficiently good, with B=1,000,000 follows.

Approach	B=10	B=500	B=1,000	B=10,000	B=100,000	B=1,000,000	B=10,000,000
Shell	25.7	15.4	13.2	11.6	11.2	11.4	11.8
Database	5.2	4.3	3.2	2.5	2.3	2.4	2.4

6. Other things learnt:

Using sprintf to pass the variable to command line:

```
x = "LAX"
system(sprintf("gawk –F '$17~/%s/{total+}END{print "LAX=", total}' 2004.csv",    x))
```

load and decompress data

```
lwp-download    http://eeyore.ucdavis.edu/stat242/data/1987.csv.bz2 1987.csv.bz2
bzip2 -d 1987.csv.bz2
```

Read multiple data at one time

```
library(XML)
doc = htmlParse("http://eeyore.ucdavis.edu/stat242/data")
sprintf("http://eeyore.ucdavis.edu/stat242/data/%s", unlist(getNodeSet(doc, "//a/@href[contains(., 'csv.bz2')]")))
```

Code:

The used code is already highlighted in the documentation so we don't bother to paste them again. Here we just attached the code to creat the MySQL data table.

```
/* Create table in the mytest database */
use mytest;

create table delays (
   Year int,
   Month int,
   DayofMonth int,
   DayOfWeek int,
   DepTime    int,
   CRSDepTime int,
   ArrTime int,
   CRSArrTime int,
   UniqueCarrier varchar(5),
   FlightNum int,
   TailNum varchar(8),
   ActualElapsedTime int,
   CRSElapsedTime int,
   AirTime int,
   ArrDelay int,
   DepDelay int,
   Origin varchar(3),
   Dest varchar(3),
   Distance int,
   TaxiIn int,
   TaxiOut int,
   Cancelled int,
   CancellationCode varchar(1),
   Diverted varchar(1),
   CarrierDelay int,
   WeatherDelay int,
   NASDelay int,
   SecurityDelay int,
   LateAircraftDelay int
);

### on windows
load data infile " C:/Users/Carrie/Documents/2004.csv"
            into table airlines
            fields terminated by ','
             optionally enclosed by ""
            lines terminated by '\r\n'
```

```
        ignore 1 lines
        (Year, Month, DayofMonth, DayOfWeek, DepTime, CRSDepTime,
ArrTime, CRSArrTime, UniqueCarrier, FlightNum, TailNum,
ActualElapsedTime, CRSElapsedTime, AirTime, ArrDelay, DepDelay, Origin,
Dest, Distance, TaxiIn, TaxiOut, Cancelled, CancellationCode, Diverted,
CarrierDelay, WeatherDelay, NASDelay, SecurityDelay, LateAircraftDelay)
;
```