STAT243 Problem Set 6

Name: Chih Hui Wang SID: 26955255

October 20, 2015

1. My strategy to build up the table is that I will first read in the data and use the function **dbWriteTable** with **append** = **TRUE**. By doing so, we can keep appending data for each year into the database. Here, I used **fread** function in the package **data.table**. It can combine with the bash code to read in data like **fread('bunzip2 -c filename')** and it performs faster than **read.csv**. To deal with the **NA** value, I replace all the **NA** with 9999 and I also do a sanity check and there is no any 9999 in the DepDelay for each year.

```
library(data.table)
library(RSQLite)
```

```
#Create database
db <- dbConnect(SQLite(), dbname="Flight.sqlite")

#Length to check that the function read the data correctly
l <- 0

#Record whether there is obersvation is 9999
nine4 <- 0

#Start to create table in database
for(i in 1987:2008){
  #Construct the command to read data
  in_file <- paste("bunzip2 -c ", i, ".csv.bz2", sep="")

  #Read data
  data <- fread(in_file, data.table=FALSE)

  #Check 9999
  nine4 <- nine4 + sum(data$DepDelay[!is.na(data$DepDelay)] == 9999)

  #Replace NA with unmeaningful number for later filter
  data[which(is.na(data$DepDelay)), "DepDelay"] <- 9999

  #Check length
  l <- l + dim(data)[1]

  #Write data into the Flight table
  dbWriteTable(conn=db, name="Flight", value=data, append=TRUE)

  #Remove data
  rm(data)
}
```

```
#Total observation
l
## [1] 123534969

#No observation's DepDelay is 9999
nine4
## [1] 0

#Query: Count the total number of rows
query <- "select count(*) from Flight"
dbGetQuery(db, query)

#Match the previous row numbers
##   count(*)
## 1 123534969
```

The database file is approximately 8.8Gb, which is smaller than the original CSV file.

```
ls -lh
## -rw-r--r--  1 ubuntu ubuntu 8.8G Oct 21 23:27 Flight.sqlite
```

2. (a) SQLite: For the problem, I recreate a database. I remove the row for DepDelay with NA and change all the coding of Month (Jan-Dec), DayofWeek(Mon-Sun) and CRSDepTime(0-24) as well.

```
#Change the column coding in Month and DayOfWeek
myweek <- function(x){
  week <- c("Monday", "Tuesday", "Wednesday", "Thursday",
            "Friday", "Saturday", "Sunday")
  week[x]
}
mymonth <- function(x){
  month <- c("January", "February", "March", "April", "May",
             "June", "July", "August", "September", "October",
             "November", "December")
  month[x]
}

#Start to create table in database
for(i in 1987:2008){
  #Construct the command to read data
  in_file <- paste("bunzip2 -c ", i, ".csv.bz2", sep="")

  #Read data
  data <- fread(in_file, data.table=FALSE)

  #Change Month, DayOfWeek, CRSDepTime
  data$Month <- mymonth(data$Month)
  data$DayOfWeek <- myweek(data$DayOfWeek)
  data$CRSDepTime <- data$CRSDepTime %/% 100

  #Write data into the Flight table
  dbWriteTable(conn=db, name="FlightnoNA", value=data[!is.na(data$DepDelay), ], append=TRUE)
```

```
  #Remove data
  rm(data)
}

#Query: Count the total number of rows
query <- "select count(*) from FlightnoNA"
dbGetQuery(db, query)

##   count(*)
##1 121232833
```

For Spark: I first write the **removeNA** to remove the row with NA. Also, I remove the column name too. Then, use **filter** to subset the data and count the total number of line in the data. It is the same with the result from R.

```
from operator import add
import numpy as np
import time

lines = sc.textFile('/data/airline')
# Remove NA as well as the line for column name
def removeNA(line):
vals = line.split(',')
return(vals[0] != 'Year' and vals[15] != 'NA')

# Repartition the data
lines = lines.filter(removeNA).repartition(192).cache()

# Count how many line in the data to compare with the result by R
# The Result is the same
numLines = lines.count()

## took 218.625398 s
## 121232833
```

(b) SQLite: For the query part, I use **group by** those column we interested in and **sum** with **case when** to compute the number of flight with DepDelay bigger than 30, 60, 180 minutes. It took around 12 minutes to finish.

```
query_late <-
"select UniqueCarrier, Origin, Dest, Month, DayOfWeek, CRSDepTime, count(*) as Count,
        sum(case when DepDelay > 30 then 1 else 0 end)*1.0/count(*) as Prop30,
        sum(case when DepDelay > 60 then 1 else 0 end)*1.0/count(*) as Prop60,
        sum(case when DepDelay > 180 then 1 else 0 end)*1.0/count(*) as Prop180
 from FlightnoNA
 group by UniqueCarrier, Origin, Dest, Month, DayOfWeek, CRSDepTime"

system.time(result <- dbGetQuery(db, query_late))

##    user  system elapsed
## 639.408  42.716 729.523

head(result, n=10)
##    UniqueCarrier Origin Dest Month DayOfWeek CRSDepTime Count    Prop30    Prop60 Prop180
```

```
## 1              9E    ABE  DTW April    Friday          6     8 0.0000000 0.0000000       0
## 2              9E    ABE  DTW April    Friday         12     8 0.0000000 0.0000000       0
## 3              9E    ABE  DTW April    Friday         16     8 0.0000000 0.0000000       0
## 4              9E    ABE  DTW April    Monday          6     9 0.0000000 0.0000000       0
## 5              9E    ABE  DTW April    Monday         12     9 0.0000000 0.0000000       0
## 6              9E    ABE  DTW April    Monday         16     9 0.0000000 0.0000000       0
## 7              9E    ABE  DTW April  Saturday         12     8 0.0000000 0.0000000       0
## 8              9E    ABE  DTW April  Saturday         16     5 0.0000000 0.0000000       0
## 9              9E    ABE  DTW April    Sunday         12     9 0.1111111 0.1111111       0
## 10             9E    ABE  DTW April    Sunday         16     9 0.2222222 0.0000000       0
```

Spark: I write a **map** funcition, **count_late_flight**. I change the CRSDeptime into hour bin, let it join with other column and use the combination as a key. For the value, I return a list with 4 value, count whether the DepDelay is more than 30, 60, 180 minutes repectively and 1. For the reduce part, I write a function, **sum_v**, to add up each value in the list individually. By using **reduceByKey**, we can get the counts for flights more than 30, 60, 180 late and total count.

```
# Map funcition Flight late
def count_late_flight(line):
vals = line.split(',')
CRSDep = int(vals[5]) // 100
# Key is Uniquecarrier-Origin-Dest-Month-DayOfWeek-CRSDepTime
keyVals = '-'.join([vals[8],vals[16],vals[17],vals[1],vals[3], str(CRSDep)])
x1 = 0
x2 = 0
x3 = 0
if int(vals[15]) > 30:
x1 = 1
if int(vals[15]) > 60:
x2 = 1
if int(vals[15]) > 180:
x3 = 1
return(keyVals, [x1, x2, x3, 1])

# Reduce funcition (add up the element in the list)
sum_v = lambda x, y: [x[0] + y[0], x[1] + y[1], x[2] + y[2], x[3] + y[3]]

# Time evaluation
start_time = timeit.default_timer()
Flightlate = lines.map(count_late_flight).reduceByKey(sum_v).collect()
elapsed = timeit.default_timer() - start_time

elapsed
## 146.9846

Flightlate[0:10]
## [(u'TW-STL-ICT-8-6-9', [0, 0, 0, 34]), (u'CO-IAH-MAF-3-2-14', [2, 1, 0, 31]),
## (u'WN-RDU-PHX-1-7-11', [0, 0, 0, 3]), (u'EA-ATL-MLB-12-2-17', [0, 0, 0, 9]),
## (u'XE-MSY-CLE-4-6-13', [1, 0, 0, 9]), (u'PI-RIC-SDF-6-4-13', [2, 1, 1, 3]),
## (u'NW-MKE-DTW-2-2-8', [0, 0, 0, 4]), (u'UA-ROC-ORD-3-7-16', [6, 4, 1, 37]),
## (u'DL-JAN-MLU-12-3-22', [2, 1, 0, 4]), (u'US-LAS-LAX-9-1-23', [0, 0, 0, 9])]
```

(c) Spark: This beginning part is the same as previous problem. To compute the proportion, I write a function **proportion** and use **mapValues** to calculate the proportion. Then, I ue another

map step to run my **strprocess** to make the ouput into comma seperated string. Finally, save it and see the result from bash.

```
# Compute the proportion
proportion = lambda x: [round(float(x[0])/float(x[3]), 4), round(float(x[1])/float(x[3]), 4), round(f

# String process, comma-delimited
def strprocess (x):
return(",".join([x[0], str(x[1]).replace("[", "").replace("]", "")]))

# Since it is too long, I break it into pieces in the following comments
# mytry = lines.map(count_late_flight).reduceByKey(sum_v).
# mapValues(proportion).map(strprocess).repartition(1).
# saveAsTextFile('/data/FlightLateCount')

mytry = lines.map(count_late_flight).reduceByKey(sum_v).mapValues(proportion).map(strprocess).reparti


# The following is bash code
hadoop fs -ls /data

## Found 2 items
## drwxr-xr-x   - root supergroup          0 2015-10-31 01:32 /data/FlightLateCount
## drwxr-xr-x   - root supergroup          0 2015-10-30 23:30 /data/airline

hadoop fs -ls /data/FlightLateCount
## -rw-r--r--   3 root supergroup          0 2015-10-31 01:32 /data/FlightLateCount/_SUCCESS
## -rw-r--r--   3 root supergroup  281745552 2015-10-31 01:32 /data/FlightLateCount/part-00000

hadoop fs -cat /data/FlightLateCount/part-00000 | head
## WN-BWI-FLL-5-2-9,0.0, 0.0, 0.0, 5
## WN-MCO-PIT-5-6-20,0.0, 0.0, 0.0, 2
## AA-DFW-AUS-2-3-6,0.0208, 0.0208, 0.0, 48
## DL-PHX-DFW-11-7-8,0.0222, 0.0222, 0.0, 45
## NW-MSP-PIT-10-2-15,0.0714, 0.0714, 0.0, 14
## YV-IAD-BUF-7-6-16,0.25, 0.25, 0.0, 4
## MQ-MIA-CLT-10-6-15,0.0, 0.0, 0.0, 4
## CO-DAB-EWR-3-4-7,0.0909, 0.0, 0.0, 11
## XE-RDU-SAT-9-3-9,0.0, 0.0, 0.0, 3
## WN-PBI-ISP-11-6-18,0.2857, 0.1429, 0.0, 7
```

(d) SQLite: To create index, I use the SQL syntax "create index table on column" to construct the key on the field UniqueCarrier, Origin, Dest, Month, DayOfWeek, CRSDepTime. Then, I can use the index field to do query. It turns out that the index will speed up the query and it takes around 9minutes finish the query.

```
create_index <-
"create index flightindex
 on FlightnoNA (UniqueCarrier, Origin, Dest, Month, DayOfWeek, CRSDepTime)"

dbSendQuery(db, create_index)
## <SQLiteResult>

#Time with index
```

```
system.time(result <- dbGetQuery(db, query_late))

##    user  system elapsed
## 234.052  29.604 537.780
```

(e) To list out the proportion of late flight for those groupings with at least 150 flights, I use **filter** to remove those group with less than 150 flights. Then, **arrange**d by proportion of late flight to get the result.

Interestingly, for the top 5 flights more than 30 minutes late, the groups are all from Southwest Airlines (WN). Also, the flights' destination and origin airport are William P. Hobby Airport(HOU) and Dallas Love Field(DAL). Finally, they were all on Friday night. For the top 5 flights more than 60 minutes late, there are 4 from United Airlines and most of them are flight from SFO to LAX and LAX to SFO. For the top 5 flights more than 180 minutes late, all of them are from American Airlines.

```
library(dplyr)
#----------Flights more than 30 mins late----------
result %>%
  select(UniqueCarrier, UniqueCarrier, Origin, Dest, Month,
         DayOfWeek, CRSDepTime, Count, Prop30) %>%
  filter(Count > 150) %>%
  arrange(desc(Prop30)) %>%
  head(n=5)


##   UniqueCarrier Origin Dest    Month DayOfWeek CRSDepTime Count    Prop30
## 1            WN    DAL  HOU     June    Friday         20   160 0.4125000
## 2            WN    HOU  DAL February    Friday         19   151 0.4039735
## 3            WN    DAL  HOU     June    Friday         21   152 0.3750000
## 4            WN    HOU  DAL     June    Friday         19   163 0.3680982
## 5            WN    DAL  HOU    March    Friday         20   158 0.3670886

#----------Flights more than 60 mins late----------
result %>%
  select(UniqueCarrier, UniqueCarrier, Origin, Dest, Month,
         DayOfWeek, CRSDepTime, Count, Prop60) %>%
  filter(Count > 150) %>%
  arrange(desc(Prop60)) %>%
  head(n=5)


##    UniqueCarrier Origin Dest    Month DayOfWeek CRSDepTime Count    Prop60
##  1            UA    LAX  SFO December    Friday         11   162 0.2222222
##  2            UA    LAX  SFO  October    Friday         16   151 0.1986755
##  3            UA    LAX  SFO December    Friday         18   160 0.1937500
##  4            AA    ORD  LAX  January  Thursday          0   181 0.1878453
##  5            UA    SFO  LAX December    Friday         18   182 0.1868132

#----------Flights more than 180 mins late----------
result %>%
  select(UniqueCarrier, UniqueCarrier, Origin, Dest, Month,
         DayOfWeek, CRSDepTime, Count, Prop180) %>%
  filter(Count > 150) %>%
  arrange(desc(Prop180)) %>%
  head(n=5)
```

```
##    UniqueCarrier Origin Dest    Month DayOfWeek CRSDepTime Count    Prop180
##  1            AA    BOS  ORD December   Tuesday           0   197 0.04568528
##  2            AA    ORD  LGA December Wednesday           0   177 0.03954802
##  3            AA    ORD  DFW  January  Thursday           0   304 0.03947368
##  4            AA    LGA  ORD December Wednesday           0   185 0.03783784
##  5            AA    ORD  LGA  January  Thursday           0   187 0.03743316
```

3. From previous problem, we know that it takes about 9 minute to finish query if we don't do any parallel computing. I use **foreach** to parallel the query. I use 4 cores. My taskfun is to seperate the query into 4 parts by the Month. The first core will do the query for Jan, Feb and Mar and the second core will do the query for Apr, May, Jun so on and sorth. Then, use **rbind** to combine all the query result together. To prevent some repeated work, I write a function **query_parallel** which I can pass the months in and it will output the query for processing those months. For parallel computing, it took 6 minute to finish the query. I also do a check to see whether the result of two method are the same.

```
#----------Parallel----------
library(doParallel)
library(foreach)
library(iterators)

#set up
nCores <- 4
registerDoParallel(nCores)

query_parallel <- function(month){
 paste(
  "select UniqueCarrier, Origin, Dest, Month, DayOfWeek, CRSDepTime, count(*) as Count,
          sum(case when DepDelay > 30 then 1 else 0 end)*1.0/count(*) as Prop30,
          sum(case when DepDelay > 60 then 1 else 0 end)*1.0/count(*) as Prop60,
          sum(case when DepDelay > 180 then 1 else 0 end)*1.0/count(*) as Prop180
   from FlightnoNA
   where Month in (", month, ")
   group by UniqueCarrier, Origin, Dest, Month, DayOfWeek, CRSDepTime", sep="")
}

#Task function
taskFun <- function(i){
  if(i == 1){
    #Set up Connection
    con1 <- dbConnect(SQLite(), dbname="Flight.sqlite")

    #Query
    query_1 <- query_parallel("'January', 'February', 'March'")
    #Do Query
    dbGetQuery(con1, query_1)

  }else if(i == 2){
    con2 <- dbConnect(SQLite(), dbname="Flight.sqlite")
    query_2 <- query_parallel("'April', 'May', 'June'")
    dbGetQuery(con2, query_2)
  }else if(i == 3){
```

```
    con3 <- dbConnect(SQLite(), dbname="Flight.sqlite")
    query_3 <- query_parallel("'July', 'August', 'September'")
    dbGetQuery(con3, query_3)
  }else{
   con4 <- dbConnect(SQLite(), dbname="Flight.sqlite")
   query_4 <- query_parallel("'October', 'November', 'December'")
    dbGetQuery(con4, query_4)
  }
}

system.time(
out <- foreach(i = 1:4, .combine = rbind) %dopar% {
  cat('Starting ', i, 'th job at ', format(Sys.time(), "%a %b %d %X"), '.\n', sep = '')
  outSub <- taskFun(i)
  cat('Finishing ', i, 'th job at ', format(Sys.time(), "%a %b %d %X"), '.\n', sep = '')
  outSub # this will become part of the out object
}
)

## Starting 1th job at Sat Oct 31 11:38:46 PM.
## Starting 2th job at Sat Oct 31 11:38:46 PM.
## Starting 3th job at Sat Oct 31 11:38:46 PM.
## Starting 4th job at Sat Oct 31 11:38:46 PM.

##    user  system elapsed
## 393.488  42.884 331.329

## Finishing 1th job at Sat Oct 31 11:43:59 PM.
## Finishing 2th job at Sat Oct 31 11:43:59 PM.
## Finishing 3th job at Sat Oct 31 11:44:07 PM.
## Finishing 4th job at Sat Oct 31 11:44:11 PM.

#Check query result
all.equal(sort(result$Prop30), sort(out$Prop30))
[1] TRUE
all.equal(sort(result$Prop60), sort(out$Prop60))
[1] TRUE
all.equal(sort(result$Prop180), sort(out$Prop180))
[1] TRUE
```

4. Because the **fread** function can process the shell command, I directly find out the column we needed in R and use **paste** function to concatenate the bash code for removing column with previous command. For example, when it read the data for 1987, the in_file will be "bunzip2 -c 1987.csv.bz2 | cut -d',' -f2,4,6,9,16,17,18", which the command after pipe is to eliminate the column we do not need.

The time for constructing the table without removing any column is around 16 minutes while the time with removing some column is around 10 minutes. For the question whether it is a worthwhile preprocessing step, if only considering the speed, I will think the answer is yes. However, for overall purpose, I think it is not worthwhile to remove 22 columns just for speeding up the creating the database, because the time did not dramatically decrease. Also, there may be other analysis we want to do on other columns.

```
#Remove unwant column
head <- read.csv("1987.csv.bz2", nrows=1, header=FALSE, stringsAsFactors=FALSE)
col_need <- c("UniqueCarrier", "Origin", "Dest", "Month",
```

```
              "DayOfWeek", "CRSDepTime", "DepDelay")

#Find out the position of columns we need
position <- paste(which(head %in% col_need), collapse=",")

#Calculate Time
system.time(
for(i in 1987:2008){
  #Construct the command to read data
  in_file <- paste("bunzip2 -c ", i, ".csv.bz2",
                   " | cut -d',' -f", position,  sep="")

  #Read data
  data <- fread(in_file, data.table=FALSE)

  #Replace NA with unmeaningful number for later filter
  data[which(is.na(data$DepDelay)), "DepDelay"] <- 9999


  #Write data into the Flight table
  dbWriteTable(conn=db, name="Flight", value=data, append=TRUE)

  #Remove data
  rm(data)
}
)
#Time for removing some column not needed
##    user  system elapsed
## 655.936  43.804 605.008

#The following output is the time for creating the table without removing any columns
##    user   system  elapsed
## 928.932   57.132 1000.483
```