University of California,Davis
STA242
Spring 2015

Assignment 5

Junxiao Bu
999452701
SSH: git@bitbucket.org:shingtime/sta242-project5.git

June 2, 2015

Overview

This assignment was focused on exploring different approaches to a simple problem complicated by the scale of the data. We want to calculate the quantiles and run regressions of two variables split between 24 csv files amounting to approximately 50 GB of uncompressed data. I was able to complete the task using three different techniques: Method 1-Using Python to read entire files on by one, Method 2-Using MySQL database, Method 3-Using R + Parallel Processing. In the following parts, firstly, the computational time for each method are presented. Then the deciles and regression results are presented. At last, each method is discussed in detail.

Result Analysis

Looking at the files, it turns out that the number of lines match for each numbered trip and fare file. It would be nice to merge these, but we should make sure before merging that the rows match. Each record seems to be able to be uniquely defined by the medallion, hack_license, and pickup_datetime, which are present in both the trip and fare files. I can run a simple awk command to make sure each these match for each row. The code is as the followings.

```
for i in {1..12}; do
   echo "$i"
   awk -F',' 'NR == FNR { a[$1,$2,$6]; next } ($1,$2,$4) in a { c++ }
    END { print c }' "trip_data_$i.csv" "trip_fare_$i.csv"
 done
```

Time Comparison

For consisitency, this report uses elapsed times as the benchmark to compare different method's time. Each of the three methods' time (in seconds) are shown in the following table.

Table 1: Time of Each Method

Method	User Time	System Time	Total Time(Elapsed Time)	Machine	RAM	CPUs
Method 1	1020.8	720.3	1741.1	MAC	8 GB	4
Method 2	9.028	65.268	3326.5	MAC	8 GB	4
Method 3	7936.659	449.724	2763.236	MAC	8 GB	4

The detailed analysis of each method's time are in the following parts.

Result Comparison

For the first question, all the three methods gave me the same result. The deciles of total_amout minus tolls_amount is in the following table. In the following parts, this variable is called **amount**.

From the table, the range of the variable seems to be very large. After a little more investigation, I found that there are some extreme values and implusible values for variable amount. The total number of observation is 173179759. In order to get

Table 2: Deciles of Total_amount - tolls_amount

Deciles of amount ($)	Value
0.1	6
0.2	7.5
0.3	8.5
0.4	9.75
0.5	11
0.6	13
0.7	15
0.8	18.5
0.9	26.12
1.0	685908.1

more pluasible result in the regression part, the values of amount was set in the range of 0 to 100.

Besides, I recalculated the **trip_time_seconds** variable using **dropoff_time - pickup_time**. I found there are some implusible values that the time is less than 0. So I also delete these implusible values in the regression part. after deleting some observations, there are 172564585 observations left.

For simple regression, all the three methods gave me the same results. The results are in the following table.

Table 3: Simple Regression Result

	Total_amount - Tolls_amount ($)
time_in_seconds	0.0046***
	(0.00002)
Intercept	10.9501***
	(0.0009)
R^2	0.200
Adj. R^2	0.200
N	172564583

For multiple regression, all the three methods gave me the same results. The results are in the following table.

Table 4: Multiple Regression Result

	Total_amount - Tolls_amount ($)
time_in_seconds	0.0046***
	(0.00002)
surcharge	-0.4117***
	(0.002)
Intercept	11.085***
	(0.0011)
R^2	0.200
Adj. R^2	0.200
N	172564583

In the simple linear regression, all the coefficients are significant. The repsonse variable and predictor has positive correlations. In the multiple regression, strangly, the variable **surcharge** coefficient is negative and significant. The R^2 doesn't change after adding one more predictor.

Method I: Python

This method uses the common modules in Python (Pandas,Numpy,sklearn). I used IPython environment to finish all the tasks. First, the method loops over all the extracted csv files. I used **pandas.read_csv** to read in columns are useful. For all the fare data, **total_amount** ,**tolls_amount** and **surcharge** are extracted. For all the trip data, **pickup_datetime** and **dropoff_datetime** are extracted. Using **pandas.concat** to combine data together. Then I used **numpy.timedelta64** to convert time into python's format and calculated the time in seconds.

The method 1 is the fastest method among all of my three methods.From the time part, the total time for this method is roughly 30 minutes. The most heavily time-consuming part is to read data into Python.

This method is relatively straightforward. The programming time is relatively short, since only the a few line of commands are used. Since I am not very familiar with Python, I made a lot of mistakes at the beginning,i.e,syntax errors. Like in R, a potential way to speed up the code is to implement some C or C++ code in Python,i.e.**Cython**. Another way to speed up more is to use multiple threads to read and process data. Due to the time limit, I am not able to implement these in the Python code.

Method II: R + MySQL

This method uses MySQL and R. First, I created a database in MySQL and loaded all the csv files into the 24 tables. Only several columns are loaded since not all of them are used in this assignment. Then I created an empty table to store the converted variables(converted time and total_amount - tolls_amount) from 24 tables. According to the criterion mentioned before,some observations are deleted in the regression part.

SQL queries are then used to generate deciles and calculated regression parameters. Unlike PostgreSQL, MySQL doesn't provide any functions to calculate quantiles and regression. So I figured out some ways to calculate them combing with R. For example, to compute the regression parameters, I calculated variance and covariance of the variables from the SQL table and used user-defined variables to store them. Then I used the formula of regression's parameter to calculate the results.

The computational time is roughly one hour, which is the slowest one among three methods.

This method is also not hard to understand. This is the first time I have tried to use the traditional database. I think the main benefit of SQL database language is that is allows me to immediately insert, update, delete, or retrieve data with simple commands. The most of the commands are almost common sense. For this assigment, the major difficulties are to learn how to set up my own database and load in data in appropriate format. Besides, some import commands like **join** and **union** are kind of hard to implement. If we want to implement many basic commands to manage tables, SQL is the best way. But different SQL database has slightly different syntax and functions. For statistical analysis, it is hard to only use SQL to finish tasks. For example, in this assignment, it is relatively useful to calculate the coefficients of simple regression but becomng much harder to calculate the coefficients of multiple regression only using SQL. The better way is to implement SQL and any other kind of statistical package together.

Method III: R + Parallel Processing

In this assignment, loading all the data into R is not a good idea, since it is slow and needs a lot of memory. In this method, the main idea is to save some variables to complete the tasks rather than save data in R. For deciles part, I saved each csv files's frequency table and combined them together at last. For regression part, I calculated each pair's of data(trip_data_i and trip_fare_i)'s summary statistics(mean_X, mean_Y, Var_X , Var_Y, COV_XY) and update the summary statistics when reading in a new set of summary statistics. The detailed information of combining all the summary statistics is getting from internet.[1] After combining 12 sets of summary statistics, the linear regression's coefficients can be calculated using variance, covariance and mean of corresponding variables.

Among these three methods, obviously, I am most familiar with R language. But for this assignment, if using brute force looping over all csv files are bottlenecked on disk IO. I need to combine R with parallel algorithm. This is the hardest part of method 3. Unlike the previous two methods, I spent a huge amount of time to implement my parallel algorithm to finish the task. In this method, I used each csv file as the "block" to split tasks to each working node. The result is not bad, although the computational time is longer than method 1. Becuase I don't have access to statistics department's server, I used my own MAC with 4 cores for this method. Obviously, this method would be faster when paralleling among more cores. In this sense, method 3 should be faster than method 1 in my case.

Comparison

For consistency, all these three methods were running on my personal machine. Comparing these three methods, method 1 using Python only is the fastest one. Method 2 using R and MySQL is the slowest method. One interesting thing of method 1's time is that user time is much longer than elapsed time.That is becausehe user time for parallel tasks is the sum of the user times for the worker nodes. So it can be a lot more than elapsed time.

Method is basically "brute force" way to finish this task.**pandas.read_csv** from pandas module is actually very fast faster than R. But it uses a lot of memory.The key problem with the existing code is that all of the existing parsing solutions in pandas first read the file data into pure Python data structures: a list of tuples or a list of lists. If I have a very large file, a list of 1 million or 10 million Python tuples has an extraordinary memory footprint significantly greater than the size of the file on disk .

Generally, **pandas.read_csv** reads in data quicker than R, but both methods are bottlenecked by memory issue. Like parallel processing in R, reconstructing python code using multiple threads should be able to make code's performance better.

In this assignment, method 2's speed is slowest. That might be due to multiple tables' operations.But It is evident that if you wanted to access the data multiple times for different pieces of information, a database would be the best choice. There is a lot of overhead in creating the database but once it's created it is handful to find information you need. But for statistical analysis, it is hard to only use SQL to finish tasks.The better way is to implement SQL and any other kind of statistical package together.

Reference

1. `http://www.emathzone.com/tutorials/basic-statistics/combined-variance.html`

2. classnotes

3. Piazza: 388, 380,343.

Appendix

Python Code

```python

import pandas as pd
import numpy as ny
import os
from dateutil import parser
import datetime
import csv
from sklearn import linear_model
import statsmodels.api
import statsmodels.formula.api as smf
from pandas.stats.api import ols

path =r'/Volumes/MyPassport/242data'
os.chdir(path) #change working directory
# get the paths for fare data and trip data
files_fare = !ls *fare*.csv
files_trip = !ls *data*.csv
## loop over all the data to get columns we want
trip_fare_all= pd.concat([pd.read_csv(f, sep = ",",error_bad_lines =
    False,usecols = [6,9,10]) for f in files_fare])
trip_fare_all.columns = ['surcharge','tolls_amount','total_amount']
all_trip= pd.concat([pd.read_csv(g, sep = ",",error_bad_lines = False,
    usecols = [5,6],parse_dates = [5,6],skiprows = [0],header=None)
    for g in files_trip])
all_trip.columns = ["pickup_datetime","dropoff_datetime"]

### calculate time_sec using dropoff_datetime - pickup_data_time

time_sec = (all_trip['dropoff_datetime'] - all_trip['pickup_datetime'
    ])/ny.timedelta64(1,'s')

#### calculate total_amount - tolls_amount
total_minus_tolls = trip_fare_all["total_amount"] - trip_fare_all["
    tolls_amount"]

## calculate quantiles for total - tolls;
total_minus_tolls.sort(ascending=True)
deci = ny.linspace(0.1,1,10)
total_minus_tolls.quantile(deci)

### combine all columns for the regression
surcharge = trip_fare_all["surcharge"]
result = pd.DataFrame(total_minus_tolls)
result['surcharge'] = surcharge.tolist()
result['time_sec'] = time_sec.tolist()
result.columns = ['amount','surcharge','time_sec']

### filter some implausible observations
filter_result = result[(result['time_sec'] > 0) & (result['amount'] >
    0) & (result['amount'] < 100)]


#### simple regression
model = smf.ols('amount ~ time_sec', data = filter_result)
fit = model.fit()
fit.summary2()

### multiple regression
model_mul = smf.ols('amount ~ time_sec + surcharge', data =
    filter_result)
fit_mul = model_mul.fit()
fit_mul.summary2()
```

R + MySQL

```
#### Method2: MySQL + R
dir = "/Users/Bruce/desktop/hw5"
setwd(dir)

start = proc.time()
library(DBI)
library(RMySQL)
con =dbConnect(MySQL(), user='root', password='1234', dbname='NYCTaxi'
    , host="localhost")
#dbClearResult(dbListResults(con)[[1]])

## create trip_table
table_trip_name = sapply(1:12,function(i){
  paste0("CREATE TABLE trip",i,"( pickup_datetime DATETIME,",
  "dropoff_datetime DATETIME)ENGINE = MYISAM;")
  })

## create fare_table
table_fare_name = sapply(1:12,function(i){
  paste0("CREATE TABLE fare",i,"( surcharge DOUBLE,",
        "tolls_amount DOUBLE, total_amount DOUBLE)ENGINE = MYISAM;")
})
## create all the trip tables
sapply(1:12,function(i)dbSendQuery(con,table_trip_name[i]))
## create all the fare tables
sapply(1:12,function(i)dbSendQuery(con,table_fare_name[i]))

## load trip data
#start = proc.time()
load_trip_table = sapply(1:12,function(i){
  paste0("LOAD DATA LOCAL INFILE",
         " '/Users/Bruce/desktop/hw5/trip_data_",i,".csv'",
         " INTO TABLE trip",i,
         " FIELDS TERMINATED BY ','",
         " LINES TERMINATED BY '\n'",
         " IGNORE 1 ROWS",
         " (@dummy,@dummy,@dummy,@dummy,@dummy,pickup_datetime,",
         " dropoff_datetime,@dummy,@dumy,@dummy,@dummy,@dummy,@dummy,
    @dummy)")
})
## load fare data
load_fare_table = sapply(1:12,function(i){
  paste0("LOAD DATA LOCAL INFILE",
         " '/Users/Bruce/desktop/hw5/trip_fare_",i,".csv'",
         " INTO TABLE fare",i,
         " FIELDS TERMINATED BY ','",
         " LINES TERMINATED BY '\n'",
         " IGNORE 1 ROWS",
         " (@dummy,@dummy,@dummy,@dummy,@dummy,@dummy,",
         "surcharge,@dummy,@dummy,tolls_amount,total_amount)")
})
#load in all trip tables
sapply(1:12,function(i)dbSendQuery(con,load_trip_table[i]))
#load in all fare tables
sapply(1:12,function(i)dbSendQuery(con,load_fare_table[i]))

##create columns to store difference between total_amount and tolls_
    amount
create_total_tolls = sapply(1:12,function(i){
  paste0("ALTER TABLE fare",i," ADD diff DOUBLE")
})

sapply(1:12,function(i) dbSendQuery(con,create_total_tolls[i]))

## add new columns of total_amount - tolls_amount
add_total_tolls = sapply(1:12,function(i){
```

```
  paste0("UPDATE fare",i," set diff = total_amount - tolls_amount")
})

sapply(1:12,function(i) dbSendQuery(con,add_total_tolls[i]))

##create columns of time(dropoff_time - pickup_time)
create_transfer_time = sapply(1:12,function(i){
  paste0("ALTER TABLE trip",i," ADD time_sec DOUBLE")
})
sapply(1:12,function(i) dbSendQuery(con,create_transfer_time[i]))

## add new columns of time_sec using dropoff.time - pickup.time
add_transfer_time = sapply(1:12,function(i){
  paste0("UPDATE trip",i," set time_sec = TIMESTAMPDIFF(SECOND,pickup_datetime,dropoff_datetime)")
})

sapply(1:12,function(i) dbSendQuery(con,add_transfer_time[i]))

### create table to store total_amount - tolls_amount from all files
dbSendQuery(con,"CREATE TABLE amount (total_tolls DOUBLE,id INT NOT NULL AUTO_INCREMENT PRIMARY KEY)")
## store all the total less tolls into a new table
add_amount = sapply(1:12,function(i){
  paste0("INSERT INTO amount(total_tolls) SELECT diff FROM fare",i)
})

for(i in 1:12){
  dbSendQuery(con,add_amount[i])
}

### create table to store time in seconds from all files
dbSendQuery(con,"CREATE TABLE time (time_sec_all DOUBLE,id INT NOT NULL AUTO_INCREMENT PRIMARY KEY)")

## store time in sec in table
add_time = sapply(1:12,function(i){
  paste0("INSERT INTO time(time_sec_all) SELECT time_sec FROM trip",i)

})

for(i in 1:12){
  dbSendQuery(con,add_time[i])
}
### create table to store surcharge from all files
dbSendQuery(con,"CREATE TABLE sur_charge (surcharge_all DOUBLE,id INT NOT NULL AUTO_INCREMENT PRIMARY KEY)")

## store surcharge
add_surcharge = sapply(1:12,function(i){
  paste0("INSERT INTO sur_charge(surcharge_all) SELECT surcharge FROM fare",i)
})

for(i in 1:12){
  dbSendQuery(con,add_surcharge[i])
}

######## calculate quantiles for total less tolls #############
deci = seq(0.1,1,length.out =10)
len_data = dbGetQuery(con,"select count(*) from amount")
len_data = len_data[1,1]
position = round(deci*len_data-1)
## create table to store sorted amount table
dbSendQuery(con,"CREATE TABLE sort_amount as SELECT total_tolls FROM amount ORDER BY total_tolls")

quantile_position = sapply(1:10,function(i){
```

```
  paste0("select total_tolls from sort_amount limit ",position[i],",1")
})
quantile = vector(length = 10,mode = "numeric")
for(i in 1:10){
  quantile[i] = dbGetQuery(con, quantile_position[i])
}
## inner join of THREE table -- FOR REGRESSION
dbSendQuery(con,"CREATE TABLE amount_time AS SELECT total_tolls,time.id, time_sec_all from time INNER JOIN amount WHERE amount.id = time.id")
dbSendQuery(con,"CREATE TABLE amount_time_surcharge AS SELECT total_tolls,time_sec_all,surcharge_all from amount_time INNER JOIN sur_charge WHERE amount_time.id = sur_charge.id")
# drop total_tolls larger than $100 or smaller than $0 and time_in_secs < 0
dbSendQuery(con,"CREATE TABLE filter_amount_time AS SELECT * FROM amount_time_surcharge WHERE total_tolls < 100 && total_tolls > 0 && time_sec_all>0")

## CALCULATE regression result
dbSendQuery(con,"SELECT
@sumXY := SUM(total_tolls*time_sec_all),
@sumXX := SUM(time_sec_all*time_sec_all),
@sumYY := SUM(total_tolls*total_tolls),
@n := count(*),
@meanX := AVG(time_sec_all),
@sumX :=SUM(time_sec_all),
@meanY := AVG(total_tolls),
@sumY :=SUM(total_tolls)
FROM filter_amount_time")


## calculate correaltion coefficient
dbClearResult(dbListResults(con)[[1]])
dbGetQuery(con,"SELECT  (@n*@sumXY - @sumX*@sumY)  / SQRT((@n*@sumXX - @sumX*@sumX) * (@n*@sumYY - @sumY*@sumY))")
## calculate slope
dbGetQuery(con,"SELECT
@b := (@n*@sumXY - @sumX*@sumY) / (@n*@sumXX - @sumX*@sumX) AS slope");
## calculate intercept
dbGetQuery(con,"SELECT
@a := (@meanY - @b*@meanX) AS intercept");

time = proc.time() - start
```

R + Parallel Processing

```

Directory ="/Users/Bruce/desktop/hw5"
setwd(Directory)
library(data.table)
library(parallel)
library(doParallel)
library(plyr)

registerDoParallel(cores=4)


################## summary part: get all the results ##########

### this part calls functions from part1,part2 and part3 to get deciles and regression result

# get paths
paths_trip = list.files(,pattern = "trip_data_[0-9]")
```

```r
paths_fare = list.files(,pattern = "*_fare_")
path_all = cbind(paths_trip,paths_fare)

start = proc.time()
##### get all 12 pairs of summary statistics and frequency tables
data_all = foreach(i=1:12) %dopar% get_one_pair_result(path_all[i,])

####calculate deciles
deciles_all = get_deciles(data_all)

##### calculate regression #######
regression_all = regression_result(data_all)

time = proc.time() - start
################## part1: process one pair's result ##############################
####### 1. read in the columns we want from one pair of data and transfer these columns(total - tolls, transfer time,etc.)

ReadOneData = function(filename){
  fare = fread(filename[2], sep = ",", header = TRUE, select = c(7,10,11))
  colnames(fare) = c("surcharge","tolls_amount","total_amount")
  trip = fread(filename[1], sep = ",", header = TRUE, select = c(6,7))
  colnames(trip) = c("pickup_datetime","dropoff_datetime")
  amount = fare$total_amount - fare$tolls_amount
  pickup = strptime(trip$pickup_datetime, "%Y-%m-%d %H:%M:%S")
  dropoff =strptime(trip$dropoff_datetime, "%Y-%m-%d %H:%M:%S")
  triptime = as.numeric(difftime(dropoff, pickup, unit = "secs"))
  data_out = data.frame(amount = amount,time = triptime, surcharge = fare$surcharge)
  ### filter data
  data_out = subset(data_out,data_out$amount > 0 & data_out$amount < 100 & data_out$time >0)
  data_out
}

###### 2. use one pair's data from ReadOneData to calculate one frequency table
get_one_frequency = function(data) {

  freq.table = as.data.frame(table(data$amount))
  names(freq.table) = c("amount", "freq")
  freq.table$amount = as.numeric(levels(freq.table$amount))
  freq.table = freq.table[order(freq.table[ ,1]), ]
  freq.table
}

###### 3. use one pair's data from ReadOneData to get summary statistics for each pair ########
get_summary_statistics = function(data){
  response = data$amount
  predictor = data$time
  N = length(predictor)
  mean.response = mean(response,na.rm = TRUE)
  mean.predictor = mean(predictor,na.rm = TRUE)
  Var.response = var(response,na.rm = TRUE)
  Var.predictor = var(predictor,na.rm = TRUE)
  cov = cov(predictor,response)
  c(Obs = N, MEAN.X = mean.predictor, MEAN.Y = mean.response, VAR.X = Var.predictor,VAR.Y =Var.response, COV = cov)

}

######## 4. combine the previous three step together ############
get_one_pair_result = function(filename){
  data.in = ReadOneData(filename)
  data.in.freq = get_one_frequency(data.in)
```

```r
  data.in.summary = get_summary_statistics(data.in)
  list(frequenct_table = data.in.freq,summary_table = data.in.summary)
}


####################################
####################################


######## part2: calculate deciles #################
#########################################

get_deciles = function(data_all){
  ## save all the frequency into a big list
  freq_list = lapply(1:12,function(i){
    data_all[[i]][[1]]
  })
  ## combine all the frequency table into one table(sorted)
  freq_combine = combine_freq(freq_list)
  deciles = deciles_calc(freq_combine,cut_point = 10)
  ## show deciles for all data
  deciles

}
####### combine all the frequency tables #########

combine_freq = function(freq_list) {
  freq.name = names(freq_list[[1]])[1]
  for (i in 1:length(freq_list)) {
    # rename the variable names before join the dataframes
    names(freq_list[[i]])[2] = as.character(i)
  }

  freq.all = join_all(dfs = freq_list , by = freq.name, type = "full")
  freq.all$freq = rowSums(freq.all[ , -1], na.rm = TRUE)
  freq.all = freq.all[ , c(freq.name, "freq")]

  freq.all = freq.all[order(freq.all[ ,1]), ]
  names(freq.all) = c("amount", "freq")
  freq.all
}

########### calculate deciles from the combined frequency table ############

deciles_calc =  function (freq_all,cut_point = 10) {
  # The default of breaks is 10, which gives deciles.
  cut = seq(0.1,1,length.out = cut_point)
  freq_all$cumsum = cumsum(freq_all$freq)
  ## how many observations
  N = sum(freq_all$freq)
  ### the ith decilies' position
  position = cut*N
  ### calculate deciles
  ## prespecify a vector to store deciles
  deciles_value = vector(length = 10 , mode = "double")

  deciles_value = sapply(position,function(i){
    head(x = freq_all$amount[freq_all$cumsum >= i], n = 1)
  })

  cbind(deciles = cut,value = deciles_value)
}

###############################
###########################


```

```r
############## part3: regression parts ####################
####################################################
### function to calculate regression result based on all 12 summary statistics vectors
regression_result = function(data_all){

  summary_list = lapply(1:12,function(i){
    data_all[[i]][[2]]
  })
  summary_all = get_full_statistics(summary_list)
  regression_all = get_regression(summary_all)
  regression_all
}

### function to update summary statistics based on previous one
get_full_statistics = function(summary_list){
  ## inital value of two tables' summary statistics
  summary_old_ini = summary_list[[1]]
  summary_new_ini = summary_list[[2]]
  ## initial value of combining two tables
  summary_combine_ini = update_summary_stat(summary_old_ini,summary_new_ini)
  for(i in 3:12){
  summary_new = summary_list[[i]]
  ## update summary statistics based on new set of summary statistics
  summary_combine = update_summary_stat(summary_combine_ini,summary_new)
  summary_combine_ini = summary_combine
  }
  summary_combine
}

### for simple linear regression

get_regression = function(summary_all){
  beta1 = summary_all["COV"]/summary_all["VAR.X"]
  beta0 = summary_all["MEAN.Y"] - beta1*summary_all["MEAN.X"]
  rsquare = (summary_all["COV"])^2/(summary_all["VAR.X"]*summary_all["VAR.Y"])

  summary = c(beta0,beta1,rsquare)
  names(summary) = c("intercept","slope","R-square")
  summary
}


##### function to combine two summary tables and get a new summary table for next step of calculation

update_summary_stat = function(summary_old,summary_new){

  new_N = summary_old["Obs"] + summary_new["Obs"]
  new_mean_x = weighted.mean(x = c(summary_old["MEAN.X"], summary_new["MEAN.X"]),
                                  w = c(summary_old["Obs"], summary_new["Obs"]))

  new_mean_y = weighted.mean(x = c(summary_old["MEAN.Y"], summary_new["MEAN.Y"]),
                                                w = c(summary_old["Obs"], summary_new["Obs"]))


  new_var_x = combine_var(summary_old,summary_new)[1]
  new_var_y = combine_var(summary_old,summary_new)[2]
  new_cov = combine_cov(summary_old,summary_new)

  summary = c(new_N,new_mean_x,new_mean_y,new_var_x,new_var_y,new_cov)
```

```
  names(summary) = c("Obs","MEAN.X","MEAN.Y","VAR.X","VAR.Y","COV")
  summary

}

########### sub functions to update variances and covariances ####

#### function to update covariance
combine_cov = function(summary_old,summary_new){

 N.total = summary_old["Obs"] + summary_new["Obs"]
  cov.total = summary_old["COV"]*((summary_old)["Obs"]-1)/(N.total-1) +
               summary_new["COV"]*((summary_new)["Obs"]-1)/(N.total-1) +
               (summary_old["MEAN.X"] - summary_new["MEAN.X"])*(summary_old["MEAN.Y"] - summary_new["MEAN.Y"])*(summary_old["Obs"]*summary_new["Obs"]/N.total/(N.total-1))
}

#### function to update variance
combine_var = function(summary_old,summary_new){

  N.total = summary_old["Obs"] + summary_new["Obs"]

  Mean.x.total = weighted.mean(x = c(summary_old["MEAN.X"], summary_new["MEAN.X"]),
                               w = c(summary_old["Obs"], summary_new["Obs"]))

  Mean.y.total = weighted.mean(x = c(summary_old["MEAN.Y"], summary_new["MEAN.Y"]),
                               w = c(summary_old["Obs"], summary_new["Obs"]))
  delta_x = summary_old["MEAN.X"] - summary_new["MEAN.X"]
  delta_y = summary_old["MEAN.Y"] - summary_new["MEAN.Y"]
  var.x.total = summary_old["VAR.X"]*(summary_old["Obs"]-1)/(N.total-1) +
               summary_new["VAR.X"]*(summary_new["Obs"]-1)/(N.total-1) +
               delta_x^2*summary_old["Obs"]*summary_new["Obs"]/N.total/(N.total-1)

  var.y.total = summary_old["VAR.Y"]*(summary_old["Obs"]-1)/(N.total-1) +
    summary_new["VAR.Y"]*(summary_new["Obs"]-1)/(N.total-1) +
    delta_y^2*summary_old["Obs"]*summary_new["Obs"]/N.total/(N.total-1)

  c(var.x.total,var.y.total)

}
###############################################
###############################################
```