pollutantmean<-function(directory,pollutant,id=1:332){

filenames<-dir(directory,full.names=T)

n<-length(id)

m<-rep(0,n)

ct<-rep(0,n)

for(i in id){

data<-read.csv(filenames[i],header=TRUE)

ct[match(i,id)]<-sum(!is.na(data[pollutant]))

m[match(i,id)]<-mean(as.matrix(data[pollutant]),na.rm=TRUE)

}

result<-(m[ct>0] %\*% ct[ct>0])[1,1]/sum(ct[ct>0])

result

}

> source("function1.R")

> pollutantmean("specdata","sulfate",1:10)

[1] 4.064128

> pollutantmean("specdata", "nitrate", 70:72)

[1] 1.706047

> pollutantmean("specdata", "sulfate", 34)

[1] 1.477143

> pollutantmean("specdata", "nitrate")

[1] 1.702932

complete <- function(directory, id = 1:332) {

files\_list <- dir(directory, full.names=T)[id]

data\_list<-lapply(files\_list,read.csv)

nobs<-sapply(lapply(data\_list,complete.cases),sum)

result<-as.data.frame(cbind(id,nobs))

result

}

> source("function1.R")

> cc <- complete("specdata", c(6, 10, 20, 34, 100, 200, 310))

> print(cc$nobs)

[1] 228 148 124 165 104 460 232

> cc <- complete("specdata", 54)

> print(cc$nobs)

[1] 219

> set.seed(42)

> cc <- complete("specdata", 332:1)

> use <- sample(332, 10)

> print(cc[use, "nobs"])

[1] 87 96 576 76 237 4 121 117 361 932

corr <- function(directory, threshold = 0) {

source("function1.R")

IN<-complete(directory,id = 1:332)

id<-as.vector(IN[IN$nobs>threshold,"id"])

result<-c()

for(i in id){

files\_list <- dir(directory, full.names=T)[i]

data<-read.csv(files\_list,header=TRUE)

dataNoNA<-na.omit(data)

corrresult<-cor(dataNoNA$sulfate,dataNoNA$nitrate)

result<-c(result,corrresult)

}

result

}

> source("function1.R")

> cr <- corr("specdata")

> cr <- sort(cr)

> set.seed(868)

> out <- round(cr[sample(length(cr), 5)], 4)

> print(out)

[1] -0.0331 0.5509 0.2621 0.1624 0.1433

> cr <- corr("specdata", 129)

> cr <- sort(cr)

> n <- length(cr)

> set.seed(197)

> out <- c(n, round(cr[sample(n, 5)], 4))

> print(out)

[1] 243.0000 0.1384 0.2996 -0.0648 -0.1063 -0.1405

> cr <- corr("specdata", 2000)

> n <- length(cr)

> cr <- corr("specdata", 1000)

> cr <- sort(cr)

> print(c(n, round(cr, 4)))

[1] 0.0000 -0.0190 0.0419 0.1901

makeCacheMatrix <- function(x = matrix()) {

I <- NULL

set <- function(y) {

x <<- y

I <<- NULL

}

get <- function() x

setInverse <- function(solve) {

I <<- solve

## store the matrix that resulted in getting inverse

## so we can compare next time for equality if we can reuse old inverse or ##recalculate

stored <<- x

}

getInverse <- function() I

getStored <- function() stored

list(set = set, get = get,

setInverse = setInverse,

getInverse = getInverse,

getStored = getStored)

}

cacheSolve <- function(x=matrix(), …) {

## Return a matrix that is the inverse of ‘x’

I <- x$getInverse()

if(!is.null(I)){

##check to see if nrow and ncol for new matrix is same as the one stored whose ##inverse is cached

if((nrow(x$getStored()) == nrow(x$get())) && (ncol(x$getStored()) == ncol(x$get())))

{

##check to see if the new matrix is identical to stored one

##after they are found to have same nrow and ncol

if(identical(x$getStored(),x$get())){

message(“getting cached data”)

return(I)

}

}

}

data <- x$get()

I <- solve(data)

x$setInverse(I)

I

}

<https://blog.csdn.net/MieAn100/article/details/45113599>

<https://blog.csdn.net/kidpea_lau/article/details/82758238>

<https://www.dataanalytictips.com/r-programming-caching-inverse-of-matrix-to-avoid-recalculation/>

<https://www.cnblogs.com/yifeili/p/5437384.html>