# Data

The zip file containing the data can be downloaded here:

specdata.zip [2.4MB]

The zip file contains 332 comma-separated-value (CSV) files containing pollution monitoring data for fine particulate matter (PM) air pollution at 332 locations in the United States. Each file contains data from a single monitor and the ID number for each monitor is contained in the file name. For example, data for monitor 200 is contained in the file "200.csv". Each file contains three variables:

Date: the date of the observation in YYYY-MM-DD format (year-month-day)

sulfate: the level of sulfate PM in the air on that date (measured in micrograms per cubic meter)

nitrate: the level of nitrate PM in the air on that date (measured in micrograms per cubic meter)

For this programming assignment you will need to unzip this file and create the directory 'specdata'. Once you have unzipped the zip file, do not make any modifications to the files in the 'specdata' directory. In each file you'll notice that there are many days where either sulfate or nitrate (or both) are missing (coded as NA). This is common with air pollution monitoring data in the United States.

# Part 1

Write a function named 'pollutantmean' that calculates the mean of a pollutant (sulfate or nitrate) across a specified list of monitors. The function 'pollutantmean' takes three arguments: 'directory', 'pollutant', and 'id'. Given a vector monitor ID numbers, 'pollutantmean' reads that monitors' particulate matter data from the directory specified in the 'directory' argument and returns the mean of the pollutant across all of the monitors, ignoring any missing values coded as NA. A prototype of the function is as follows

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

## 'directory' is a character vector of length 1 indicating

## the location of the CSV files

## 'pollutant' is a character vector of length 1 indicating

## the name of the pollutant for which we will calculate the

## mean; either "sulfate" or "nitrate".

## 'id' is an integer vector indicating the monitor ID numbers

## to be used

## Return the mean of the pollutant across all monitors list

## in the 'id' vector (ignoring NA values)

## NOTE: Do not round the result!

}

You can see some example output from this function. The function that you write should be able to match this output. Please save your code to a file named pollutantmean.R.

# Part 2

Write a function that reads a directory full of files and reports the number of completely observed cases in each data file. The function should return a data frame where the first column is the name of the file and the second column is the number of complete cases. A prototype of this function follows

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

## 'directory' is a character vector of length 1 indicating

## the location of the CSV files

## 'id' is an integer vector indicating the monitor ID numbers

## to be used

## Return a data frame of the form:

## id nobs

## 1 117

## 2 1041

## ...

## where 'id' is the monitor ID number and 'nobs' is the

## number of complete cases

}

You can see some example output from this function. The function that you write should be able to match this output. Please save your code to a file named complete.R. To run the submit script for this part, make sure your working directory has the file complete.R in it.

# Part 3

Write a function that takes a directory of data files and a threshold for complete cases and calculates the correlation between sulfate and nitrate for monitor locations where the number of completely observed cases (on all variables) is greater than the threshold. The function should return a vector of correlations for the monitors that meet the threshold requirement. If no monitors meet the threshold requirement, then the function should return a numeric vector of length 0. A prototype of this function follows

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

## 'directory' is a character vector of length 1 indicating

## the location of the CSV files

## 'threshold' is a numeric vector of length 1 indicating the

## number of completely observed observations (on all

## variables) required to compute the correlation between

## nitrate and sulfate; the default is 0

## Return a numeric vector of correlations

## NOTE: Do not round the result!

}

For this function you will need to use the 'cor' function in R which calculates the correlation between two vectors. Please read the help page for this function via '?cor' and make sure that you know how to use it.

You can see some example output from this function. The function that you write should be able to match this output. Please save your code to a file named corr.R. To run the submit script for this part, make sure your working directory has the file corr.R in it.

Grading and Submission

This assignment will be graded using unit tests executed via the submit script you run on your computer. To obtain the submit script, run the following code in R:

source("http://d396qusza40orc.cloudfront.net/rprog%2Fscripts%2Fsubmitscript1.R")

Or you can download the script to your working directory (NOTE: you may need to rename the file to be "submitscript1.R". Then source the file locally via

source("submitscript1.R")

The first time you run the submit script it will prompt you for your Submission login and Submission password. These can be found at the top of the Programmi ng Assignments page. To execute the submit script, type

submit()

at the console prompt (after source-ing the file). NOTE that the submit script requires that you be connected to the Internet in order to work properly. When you execute the submit script in R, you will see the following menu (after typing in your submission login email and password):

[1] 'pollutantmean' part 1

[2] 'pollutantmean' part 2

[3] 'pollutantmean' part 3

[4] 'pollutantmean' part 4

[5] 'complete' part 1

[6] 'complete' part 2

[7] 'complete' part 3

[8] 'corr' part 1

[9] 'corr' part 2

[10] 'corr' part 3

Which part are you submitting [1-10]?

We will compare the output of your functions to the correct output. For each test passed you receive the specified number of points on the Assignments List web page.

You are finished when you have successfully submitted everything using submit() and you see scores on the assignment page. You can ignore the Submit buttons to the right of each score. They are only to be used when firewall or proxy settings prevent users from successfully using the submit() script. The submit() script will describe how to create files for uploading if there are problems, but under normal circumstances, there is NO NEED to use the Submit buttons on the assignment page.

**Introduction**

This second programming assignment will require you to write an R function is able to cache potentially time-consuming computations. For example, taking the mean of a numeric vector is typically a fast operation. However, for a very long vector, it may take too long to compute the mean, especially if it has to be computed repeatedly (e.g. in a loop). If the contents of a vector are not changing, it may make sense to cache the value of the mean so that when we need it again, it can be looked up in the cache rather than recomputed. In this Programming Assignment will take advantage of the scoping rules of the R language and how they can be manipulated to preserve state inside of an R object.

**Example: Caching the Mean of a Vector**

In this example we introduce the <<- operator which can be used to assign a value to an object in an environment that is different from the current environment. Below are two functions that are used to create a special object that stores a numeric vector and cache's its mean.

The first function, makeVector creates a special "vector", which is really a list containing a function to

1. set the value of the vector
2. get the value of the vector
3. set the value of the mean
4. get the value of the mean

makeVector <- function(x = numeric()) {

m <- NULL

set <- function(y) {

x <<- y

m <<- NULL

}

get <- function() x

setmean <- function(mean) m <<- mean

getmean <- function() m

list(set = set, get = get,

setmean = setmean,

getmean = getmean)

}

The following function calculates the mean of the special "vector" created with the above function. However, it first checks to see if the mean has already been calculated. If so, it gets the mean from the cache and skips the computation. Otherwise, it calculates the mean of the data and sets the value of the mean in the cache via the setmean function.

cachemean <- function(x, ...) {

m <- x$getmean()

if(!is.null(m)) {

message("getting cached data")

return(m)

}

data <- x$get()

m <- mean(data, ...)

x$setmean(m)

m

}

**Assignment: Caching the Inverse of a Matrix**

Matrix inversion is usually a costly computation and there may be some benefit to caching the inverse of a matrix rather than compute it repeatedly (there are also alternatives to matrix inversion that we will not discuss here). Your assignment is to write a pair of functions that cache the inverse of a matrix.

Write the following functions:

1. makeCacheMatrix: This function creates a special "matrix" object that can cache its inverse.
2. cacheSolve: This function computes the inverse of the special "matrix" returned by makeCacheMatrix above. If the inverse has already been calculated (and the matrix has not changed), then the cachesolve should retrieve the inverse from the cache.

Computing the inverse of a square matrix can be done with the solve function in R. For example, if X is a square invertible matrix, then solve(X) returns its inverse.

For this assignment, assume that the matrix supplied is always invertible.

In order to complete this assignment, you must do the following:

1. Fork the GitHub repository containing the stub R files at <https://github.com/rdpeng/ProgrammingAssignment2> to create a copy under your own account.
2. Clone your forked GitHub repository to your computer so that you can edit the files locally on your own machine.
3. Edit the R file contained in the git repository and place your solution in that file (please do not rename the file).
4. Commit your completed R file into YOUR git repository and push your git branch to the GitHub repository under your account.
5. Submit to Coursera the URL to your GitHub repository that contains the completed R code for the assignment.

In addition to submitting the URL for your GitHub repository, you will need to submit the **40 character SHA-1 hash** (as string of numbers from 0-9 and letters from a-f) that identifies the repository commit that contains the version of the files you want to submit. You can do this in GitHub by doing the following

1. Going to your GitHub repository web page for this assignment
2. Click on the “?? commits” link where ?? is the number of commits you have in the repository. For example, if you made a total of 10 commits to this repository, the link should say “10 commits”.
3. You will see a list of commits that you have made to this repository. The most recent commit is at the very top. If this represents the version of the files you want to submit, then just click the “copy to clipboard” button on the right hand side that should appear when you hover over the SHA-1 hash. Paste this SHA-1 hash into the course web site when you submit your assignment. If you don't want to use the most recent commit, then go down and find the commit you want and copy the SHA-1 hash.

A valid submission will look something like (this is just an **example**!)

<https://github.com/rdpeng/ProgrammingAssignment2>

7c376cc5447f11537f8740af8e07d6facc3d9645

**Grading**

This assignment will be graded via peer assessment. During the evaluation phase, you must evaluate and grade the submissions of at least 4 of your classmates. If you do not complete at least 4 evaluations, your own assignment grade will be reduced by 20%.