**DS 710**

**Homework 10**

**R assignment**

In this problem, you will work with the “cleaned” version of the US News and World Report data on colleges and univers# import the cleaned data from previous assingment, week 9

1. Read the data into R and attach it.  Use *length* and *which* to determine how many schools have a per-student instructional expenditure higher than their out-of-state tuition.  Then use control flow to answer the same question.  (Check that the two methods give the same answer!)

# import the cleaned data from previous assingment, week 9

colleges\_data **<-** read.csv**(**"C:/Users/pedbv9699/Documents/GitHub/ds710assignment9/cleaned\_US.csv"**)**

attach**(**colleges\_data**)**

# using length and which

length**(**which**(**Instructional.expenditure.per.student **>** Out.of.state.tuition**))**

**[**1**]** 457

# control flow

find.cost **<-** **function(**x**){**

school\_entry **<-** length**(**colleges\_data**)**

**for(**i **in** school\_entry**){**

col **<-** which**(**Instructional.expenditure.per.student **>** Out.of.state.tuition**)**

return**(**length**(**col**))**

**}**

**}**

find.cost**(**col**)**

[1] 457

b. Use *system.time* to compare the running times of the two methods you wrote in part a.  Iterate each method enough times that you can see a difference in the running times.  Report the user time + system time for each method.  Which is more efficient?

# function 1 time

func\_1 **<-** system.time**(**

**for(**t **in** 1**:** 100000**){**

length**(**which**(**Instructional.expenditure.per.student **>** Out.of.state.tuition**))**

**}**

**)**

func\_1

user system elapsed

0.7 0.0 0.7

# function 2

func\_2 **<-** system.time**(**

**for(**x **in** 1**:**100000**){**

find.cost**(**x**)**

**}**

**)**

func\_2

user system elapsed

1.25 0.00 1.25

Func\_1 vs func\_2 comparison shows how much quicker func\_2 is with a elapsed time of .07 and func\_1 almost 50% slower with a time 1.25 seconds. Shows using functions that are predefined with in R are suggested using over loops . Like the Length function helps boost performace instead of sticking to write loop yourself.

1. Consider three different methods of finding the mean of each numeric column of the data:
2. Using apply() and the built-in function mean()

dataFrame\_us = colleges\_data[sapply(colleges\_data,is.numeric)]

func\_1 **<-** **function(**df**){**

firstMeantype **=** apply**(**df, 2, mean, na.rm **=** T**)**

return(firstMeantype)

**}**

# just to see how it looks

method1\_df **=** colleges\_data**[**sapply**(**colleges\_data,is.numeric**)]**

**>** firstMeantype **=** apply**(**method1\_df, 2, mean, na.rm **=** T**)**

**>** firstMeantype

ID.Number Public.private

3126.374808 1.639017

Avg.Math.SAT Avg.Verbal.SAT

506.837838 461.223938

Avg.combined.SAT Avg.ACT

967.978177 22.120448

First.quartile...Math.SAT Third.quartile...Math.SAT

462.235751 583.148964

First.quartile...Verbal.SAT Third.quartile...Verbal.SAT

418.487047 530.452073

First.quartile...ACT Third.quartile...ACT

19.819005 25.113122

Num.applications.received Num.applicants.accepted

2752.097523 1870.683191

Num.students.enrolled Pct.new.students.from.top.10..of.HS.class

778.880493 25.671978

Pct.new.students.from.top.25..of.HS.class Num.full.time.undergraduates

52.350000 3692.665127

Num.part.time.undergraduates In.state.tuition

1081.526772 7897.274371

Out.of.state.tuition Room.and.Board.costs

9276.905616 4162.106852

Room.costs Board.costs

2514.681957 2060.983831

Additional.fees Estimated.book.costs

392.012646 549.972887

Estimated.personal.spending Pct.of.faculty.with.PhDs

1389.291704 68.645669

Pct.of.faculty.with.terminal.degree Student.faculty.ratio

75.231132 14.858769

Pct.alumni.who.donate Instructional.expenditure.per.student

20.912963 8987.890736

Graduation.rate InterQ\_Math\_SAT

60.405316 120.913212

InterQ\_Verbal\_SAT

111.965026

1. Using apply() and a function you write, called mymean(), which takes the sum of all of the non-missing values and divides by the number of non-missing values

myMean **<-** **function(**df**){**

non\_missing\_val **=** sum**(**sapply**(**df, sum, na.rm **=** **TRUE))**

sum\_of\_non\_missing\_val **=** sum**(**apply**(!**is.na**(**df**)**, 2 , sum**))**

return**(**non\_missing\_val**/**sum\_of\_non\_missing\_val**)**

**}**

**>** myMean**(**method1\_df**)**

**[**1**]** 1805.38

1. Using a for() loop to iterate over the numeric columns, and a for() loop inside it to iterate over the values within that column

func\_3 **<-** **function(**df**){**

**for(**i **in** colnames**(**df**)){**

sum **=** 0

length **=** 0

**for** **(**j **in** df**[[**i**]]){**

**if(**is.numeric**(**j**)** **&** **!**is.na**(**j**)){**

sum **=** sum **+** j

length **=** length **+**1

**}**

**}**

**}**

print**(**sum**/**length**)**

**}**

Which do you expect to be most efficient?  Explain your answer in 1-3 sentences.

I expect the data frame built-in functions be a lot more efficient than the for loops. In particular I expect first mean function to be faster than all the others. Mainly because the use of Apply ans Sapply will be a lot more efficient than anything we write with loops.

d. Use *microbenchmark* to compare the median running time of the methods in part c.  Write 1-3 sentences describing which method is most efficient, and by what percentage.

**>** microbenchmark**(**func\_1**(**dataFrame\_us**))**

Unit**:** milliseconds

expr min lq mean median uq max neval

func\_1**(**dataFrame\_us**)** 4.5477 5.3217 7.114696 5.58135 5.9413 45.8209 100

**>** microbenchmark**(**myMean**(**dataFrame\_us**))**

Unit**:** milliseconds

expr min lq mean median uq max neval

myMean**(**dataFrame\_us**)** 1.7612 1.79205 2.463805 1.8447 2.5062 38.5202 100

**>** microbenchmark**(**func\_3**(**dataFrame\_us**))**

Unit**:** milliseconds

expr min lq mean median uq max neval

func\_3**(**dataFrame\_us**)** 65.9436 67.1173 68.61687 67.7216 68.41945 103.3767 100

The foor loop method obviously takes the longest, the loop sets and destroys references which takes extra time and computing. And with Apply using standard functions such as MEAN, Length makes easy calculation without the need for allocating space or memory. It is all predefined.

Submit a .doc, .docx, or .pdf file to GitHub, containing your R code, R output, and written interpretations and explanations.