Rbit002

**Notes on manipulating data in a data frame.**

In my last bit about R I provided examples of reading data into a data frame. Here I will provide examples on manipulating data in a data frame. For these examples I will switch to the ‘snook.tdf’ data set which is much smaller than the Mattawoman creek continuous monitoring data. Using the smaller data set will enable us to more easily see the results of manipulations.

My notes here are complementary to the material in Chapter 2 of “*An Introduction to R*”. In the *Introduction*, the authors focus more on manipulating vectors of data. I encourage you to read the *Introduction* because it addresses the fundamentals of R. However, because most of us are working with large data sets, I think it will be more useful to focus on using data frames. It is important to keep in mind that a data frame is just a collection of vectors where each vector is a column of the data frame. Thus understanding vector manipulations can be key to understanding what is happening with your data frame. However, if your goal is to use R just for data analysis and not data management, that is you plan to do data management in some other environment and just feed data to R, then learning to do fundamental programming in R is less important. My goal is to give you the tools to get data into R, do a few manipulations, and produce an analysis.

As I mentioned in my introductory session, I start my R programs with a template that looks like this:

#=====================================================================

# file:

# function:

#

# programmer: Elgin S. Perry, Ph. D.

#

# date:

#

# address: 2000 Kings Landing Rd.

# Huntingtown, Md. 20639

#

# voice phone: (410)535-2949

# email: EPERRY@chesapeake.net

#=====================================================================

#install.packages()

#library(lattice) #Used for contour plots [contourplot()]

#library(nlme) #used for gam Mixed model [gamm()]

#library(MASS) #used for glm Mixed model [glmmPQL()]

#library(mgcv) #Wood's gam package

#library(chron) #date functions

#library(doBy) # Allows "BY processing similar to SAS

#library(FitAR) #AR package from McLeod and Zhang

#library(Hmisc) #stat function by Frank Harrell

#library(cluster) #cluster analysis routines

options(stringsAsFactors = FALSE)

source("C:/Projects/Rtp/dfsum.r")

source("C:/Projects/Rtp/RTF.r")

# be sure to change \ to /

ProjRoot <- 'C:/Projects/ '

setwd(ProjRoot);

RTFout <- paste(ProjRoot,"RTFexample.rtf",sep='')

datafile <- paste(ProjRoot,"dummy.txt",sep='');

a <- count.fields(datafile, sep = "\t", quote = "\"'", skip = 1,

 blank.lines.skip = TRUE, comment.char = "#")

range(a)

#rbind(1:length(a),a)

dum <- read.table(datafile, header=TRUE, sep=",", na.strings="NA", dec=".", strip.white=TRUE,stringsAsFactors = FALSE)

dfsum(dum)

In the beginning, the lines beginning with “#”, are comment lines which are ignored by the R interpreter. I fill in the file name, the purpose of the program, and the date. In the next section are statements for loading libraries or add on packages that are also commented. I uncomment these libraries as they are needed for any programming project. Finally there are some programming lines that are commonly used in any project that I use as a starting place. These lines include:

1. Using the options() function to turn off R’s default of converting strings to factors (we will discuss factors later).
2. Using source() to load some files of functions that I have written and like to keep handy.
3. Using setwd() to set my working directory for this project
4. Using paste() to create a string variable for a file where I will output results in RTF format
5. Using paste() again to create a string for the input datafile then checking the fields in that file with count.fields()
6. Using read.table() with my favorite options to put the data into a data frame
7. Using dfsum() to get a dataframe summary. Note that “dfsum() is one of my user written functions (loaded by the source() function above) and is not available in base R.

As you develop you R programming style, you might consider working on a similar template.

**Reading the snook data**

For the snook data, the template gets revised to this:

#=====================================================================

# file: c:\Projects\CBP\Rcourse\snook.r

# function: length weight regression for snook

#

# programmer: Elgin S. Perry, Ph. D.

#

# date: 5/11/2014

#

# address: 2000 Kings Landing Rd.

# Huntingtown, Md. 20639

#

# voice phone: (410)535-2949

# email: EPERRY@chesapeake.net

#=====================================================================

#install.packages()

#library(lattice) #Used for contour plots [contourplot()]

#library(nlme) #used for gam Mixed model [gamm()]

#library(MASS) #used for glm Mixed model [glmmPQL()]

#library(mgcv) #Wood's gam package

#library(chron) #date functions

#library(doBy) # Allows "BY processing similar to SAS

#library(FitAR) #AR package from McLeod and Zhang

#library(Hmisc) #stat function by Frank Harrell

#library(cluster) #cluster analysis routines

options(stringsAsFactors = FALSE)

#source("C:/Projects/Rtp/dfsum.r")

#source("C:/Projects/Rtp/RTF.r")

# be sure to change \ to /

ProjRoot <- 'c:/Projects/CBP/Rcourse/'

setwd(ProjRoot);

RTFout <- paste(ProjRoot,"RTFexample.rtf",sep='')

datafile <- paste(ProjRoot,"snook.tdf",sep='');

a <- count.fields(datafile, sep = "\t", quote = "\"'", skip = 1,

 blank.lines.skip = TRUE, comment.char = "#")

range(a)

#rbind(1:length(a),a)

snook <- read.table(datafile, header=TRUE, sep="\t", na.strings="NA", dec=".", strip.white=TRUE,stringsAsFactors = FALSE)

#dfsum(snook)

snook

You should be able to take this code, substitute the proper path to point to the snook.tdf in your environment, and get this to run. Note that for simplicity, I have commented references to my user defined functions. We will get to those soon. Your results should look like:



**Review of Referencing Data in a Data Frame**

Before we start doing data manipulations, let’s review how to reference elements of the data frame. Typically each row of a data frame is an observation and each column of the data frame is a variable that is measured for each observation. Here observations are defined by length (25 values) crossed with water body (2 values) crossed with season (2 values). There are potentially 100 observations but not all lengths are observed in all seasons and water bodies and thus the data frame falls a few short with 97 rows. There are three common methods of selecting elements of the data frame: the vector method, the numeric index method, and the name method. Here are examples of the three methods to show the vector of mean weights:

Vector method

snook$wgt.mean

Numeric index method

snook[,4]

name method

snook[,”wgt.mean”]

note: MSword substitutes some special character for the “ character so that this last method will not work if you cut and paste from word. It does work if you create it with an ascii editor or just type it into the R-console.

**Data Manipulations**

**Changing a single observation**

In the snook data, I identified a datum that appears to be miss-typed. Here I change that value to NA which is the R missing value code.

snook[snook$length==40 & snook$water.body=='Atlantic'&snook$season=="May-Oct",'wgt.mean'] <- NA

This statement uses a mixture of the vector and name method for defining the element of the data frame to be replace by NA. The column (2nd position inside of the brackets) is defined by the name 'wgt.mean'. The row is defined by a sequence of Boolean vectors connected by the “and operator”, (&), which identifies the row where all of the Boolean statements are true. To illustrate this, the vector

snook$length==40



Has 4 observations for which it is true. Taking this first condition of length=40 and “anding” it with a second condition of water.body = ‘Atlandtic’ results in the vector

snook$length==40 & snook$water.body=='Atlantic'



which has 2 observations for which it is true. Continuing to add a third condition for season = “May-Oct”, results in the vector

snook$length==40 & snook$water.body=='Atlantic'&snook$season=="May-Oct"



Has only 1 observation for which it is true. To check that this is the correct observation, list it with the line

snook[snook$length==40 & snook$water.body=='Atlantic'&snook$season=="May-Oct",]



Here we see that the mean is not even bracketed by the min and the max and thus it is clearly an error. Thus we finally execute the command

snook[snook$length==40 & snook$water.body=='Atlantic'&snook$season=="May-Oct",'wgt.mean'] <- NA

which sets the value to missing. We can check this by listing lines 80-90 as follows:

snook[80:90,]



We see that line 81 has wgt.mean set to NA. We also see that some other lines have NA for min and max that were missing values read from the data file. We could have used simpler code to set this value to missing by employing the numeric index method

snook[81,4] <- NA

However, if in some future revision of the code, you do some revision of the data that changes the line numbers such as deleting observations, then the indices may change and this may result is setting the wrong value to missing.

**Creating a New Variable or Data Frame Column**

Here is an example to creating a new column in the data frame as a function of two other columns. This computes the range of weights for each observation and stores the result as a new column in the data frame.

snook$wgt.range <- snook$wgt.max – snook$wgt.min

or

snook[,'wgt.range'] <- snook[,'wgt.max'] - snook[,'wgt.min']

or

snook[,'wgt.range'] <- snook[,6] - snook[,5]



We can see a new column in the data frame with the heading wgt.range.

Here is a list of operators that you can use:

Operator Description

+ addition

- subtraction

\* multiplication

/ division

^ or \*\* exponentiation

x %% y modulus (x mod y) 5%%2 is 1

x %/% y integer division 5%/%2 is 2

%\*% matrix multiplication

t() matrix transpose

Relational operators

x < y less than

x > y greater than

x <= y less than or equal

x >= y greater than or equal

x == y exactly equal

x != y not equal

identical()

Logical Operators

< less than

<= less than or equal to

> greater than

>= greater than or equal to

== exactly equal to

!= not equal to

!x Not x

x | y x OR y

x & y x AND y

isTRUE(x) test if X is TRUE

x & y and

x && y and

x | y or

x || y or

xor(x, y) element wise or

isTRUE()

**Creating a Subset of Data**

So for example, to create a new data frame that contains only data for the Gulf Coast fish use the line

snook.g <- snook[snook$water.body=='Gulf',]



Note that data transformations can be embedded in function calls. For example

plot(log(wgt.mean)~length,data=snook.g)



This should get you started on manipulating data. A few special topics are omitted here such as working with factors and date-time variables. This I will cover in the future.