R - Getting Started CONTINUED

TRUE and FALSE (logical) data

Vectors can be assigned logical measurements, directly or as the result of a logical operation. Here's an example of direct assignment. You have seen some of this before. Here is a reminder.

```
z <- c(TRUE, TRUE, FALSE)  # put 3 logical values to a vector z
```

Logical operations can identify and select those vector elements for which a condition is TRUE. The logical operations are symbolized as follows

```
 ==   (equal to)
 !=   (not equal to)
 <    (less than)
 <=   (less than or equal to)
```

and so on.

The logical operators “&” and “|” refer to AND and OR. For example, put the following numbers into a vector “z”,

```
z <- c(-10, -5, -1, 0, 3, 92)
```

What do the following operations yield?

```
z < 0 & abs(z) > 5

z < 0 | abs(z) > 5

z[z < 0 | abs(z) > 5]
```

Combine vectors to make a data frame

Multiple vectors representing different measurements (variables) made on the same individuals can be placed into columns of a data frame. A data frame is a spreadsheet-like object for a data set of potentially many variables. We will see more about data frames later. Here we show how to make a data frame by combining vectors of the same length. The vectors need not be of the same data type.

First, obtain your vectors. For example,

```
custno <- c(1:7)

loyalty_level <- c(100,100,90,90,40,40,50)

location<-
c("madrid","madrid","london","london","dublin","paris","paris")
```

Then combine them into a data frame named `mydata`.

```
mydata  <-  data.frame(cust  =  custno,  loyalty_level  =  loyalty_level,
location = location, stringsAsFactors = FALSE)
```

The argument `stringsAsFactors = FALSE` is optional but recommended to preserve character data (otherwise character variables are converted to factors).

Getting Data Into R

Data.Frames

One of the most useful features of R is the data.frame. On the face of it a data.frame is just like an Excel spreadsheet in that it has columns and rows. In statistical terms, each column is a variable and each row is an observation.

In terms of how R organises data.frames, **each column is actually a vector**, each of which is the **same length**. That is very important because it lets each column hold a different type of data i.e. numeric data or character data. This also implies that within a column each element must be of the same type, just like with vectors.

There are many ways of to construct a data.frame, the simplest being to use the data.frame function. Let us create a basic data.frame using some of vectors x, y, z

```
#creating a data.frame from vectors creating a 10 X 3 data.frame
x <- 10:1
y<- -4:5
z <- c ("football", "golf", "tennis", "soccer", "hurling", "rugby",
"athletics", "fencing", "swimming", "basketball")

mydf <- data.frame(x,y,z)
mydf


# set the column names
mydf <- data.frame(First=x, Second=y, Sport=z)
mydf
```

```
# finding information about the data.frame
nrow(mydf)
ncol(mydf)
dim(mydf)

# set the row names and reset the rownames
rownames(mydf) <- c("one", "two", "three", "four", "five", "six", "seven",
"eight", "nine", "ten")
rownames(mydf)
rownames(mydf) <- NULL
rownames(mydf)
```

Reading Data from Files

There are a number of ways of reading data into R with CSV files one of the most common

Reading CSVs

One way to read data is to use read.table. The result of using `read.table` is a data.frame. The first argument is full path of the file to be loaded. The file can be sitting on disk or even on the web. Here is a simple CSV posted on the following web site. Let us read it into R using read.table.

```
#Read a CSV from the web and look at its contents
theUrl  <- "http://WWW.jaredlander.com/data/Tomato%20First.csv"
#  note the arguments set i.e. the file has captions and it is comma
separated
tomatoes <-read.table(file = theUrl, header=TRUE, sep =",")
head(tomatoes)

# Let us use read.table and read.csv. Note read.csv calls read.table with
some arguments preset. You will need to change the filepath to suit
(download .csv file from moodle)
cars<-read.table(file = "c:/rdatasets/cars.txt", header=TRUE, sep =",")
head(cars)
carsCSV<-read.csv(file = "c:/rdatasets/cars.txt", header=TRUE)
head(carsCSV)
```

Frequency tables

These commands generate tables of frequencies or summary statistics. In the examples below, “x” is a single numeric variable. “A” is a categorical variable (factor or character variable) identifying different groups; “B” is a second such variable.

Frequency table and a Pie Chart

Frequency table for a categorical variable A. First let us load a file that hold weather data with a response variable of Play (yes, no) representing whether a sport was played based on the weather conditions namely outlook, temperature, humidity, and windy. There are two datasets representing the same data where one is nominal (i.e. categorical) while the other has some variables that are numeric.

```
#Input dataset nominal weather into a dataframe

weathernom <-read.table(file ="c:/rdatasets/weathernominal.txt",
stringsAsFactors=FALSE, sep =",", header=TRUE)

#Show the first 10 records and ALL the columns
head(weathernom)
weathernom[1:10,]
#Show the first 12 records and first 3 columns
weathernom[1:12,1:3]

#Create a table of counts for each discrete value of outlook
A <- weathernom$outlook
ZZ <- table(A)
ZZ

#Let’s show the data as a piechart – not the best visualisation to use!
Good for getting a rough idea of proportions
pie(ZZ, labels=names(ZZ), edges=200, col=c("yellow","red","navy"), radius=
0.9)
```

Frequency (Contingency) table

The following command generates a frequency table for two categorical variables, A and B. The command can be extended to three or more variables.

```
B <- weathernom$play   # play is a Boolean indicating if a game was played

table(A, B)
```

To include the row and column sums in the contingency table, use the following commands.

```
mytable <- table(A, B)

addmargins(mytable, margin = c(1,2), FUN = sum, quiet = TRUE)
```

Tables of descriptive statistics

The command "tapply" creates tables of results. The function argument can be any statistical function that can be applied to a single numeric variable (e.g., mean, standard deviation, median, etc).

When creating tables for display purposes, such as in a report, you can round the results to a fixed number of decimal places. For example, to round a table of means "z" to two decimal places, use the **round** function

```
#Input dataset numeric weather into a dataframe

weathernum <-read.table(file ="c:/weathernumeric.txt",
stringsAsFactors=FALSE, sep =",", header=TRUE)

#Show the first 10 records and all the columns
weathernum[1:10,]

#Calculate the mean of 2 numeric attributes and place in a table

tempMean <- round(mean(weathernum$temperature),2)
humTemp <- round(mean(weathernum$humidity),2)

z <- table(tempMean, humTemp, dnn = c("Temp","Humidity"))
z
```

Table display for one variable

For example, here is how to generate a one-way table of group means. weathernum$play is the factor or character variable identifying the groups. Let us calculate the humidity means for groups **play** “yes” and “no”

```
tapply(weathernum$humidity,  INDEX=weathernum$play, FUN=mean, na.rm=TRUE)
```

The na.rm option removes missing values before computing (otherwise the mean returns "NA" if there are missing values). na.rm is not a tapply option -- rather it is an option for the function mean. In general you can pass optional arguments to FUN by including them immediately afterward.

The following shortcut works if the arguments are listed strictly in the order shown.

```
tapply(weathernum$humidity, weathernum$play, mean, na.rm=TRUE)
```

For You to do! The command for a one-way table of group standard deviations is similar. Write an R statement to do so.

Table display for two variables

The following example produces a two-way table of group medians. weathernum$humidity is a numeric variable, whereas weathernum$play and weathernum$outlook are categorical variables (factors or character variables).

```
# Shows the median humidity values for 2 categorical variables outlook and play. For example for instances where the outlook was rainy and the sport was played (yes) the median humidity value was 80.0

tapply(weathernum$humidity, INDEX
=list(weathernum$play,weathernum$outlook), median)

#different display of the same data

tapply(weathernum$humidity, INDEX =list(weathernum$outlook
,weathernum$play), median)
```

How to handle Missing Data

```
#Create subset of "weather" taking the first 6 records and variables 1,2,3 & 4. Remember we have loaded the weathernum.txt into weathernum

weather.subset <- weathernum[1:6,c(1,2,3,4)]
weather.subset

#Let us simulate some missing data

weather.subset[4,1] <- weather.subset[2,1] <- weather.subset[4,4]<-NA
weather.subset[2,2] <- weather.subset[5,3] <-weather.subset[6,2] <- NA
weather.subset

#Replace a missing value with field mean e.g.

weather.subset[5,3]<- mean(na.omit(weather.subset$humidity))
weather.subset
```

```
#Replace the missing value with field mode

our.table <- table(weather.subset$outlook)
our.mode <- names(our.table)[our.table == max(our.table)]
weather.subset[2,1] <- our.mode
weather.subset


#Replace missing values with a value generated at random from the observed distribution. Results will vary

#Random sample for Nominal Data
obs.windy <- sample(na.omit(weather.subset$windy),1)
weather.subset[4,4] <- obs.windy

# Random sample for Numeric Data
obs.temp <- sample(na.omit(weather.subset$temperature),1)
weather.subset[6,2] <- obs.temp
```

HANDS-ON R ANALYSIS

Using the carsTEST dataset, answer the following questions

1. How many records are there?
2. Explore whether there are missing values for any of the variables and update based on the following requirements:
   - If there are values missing for **mpg**, replace with a constant 0
   - If there are missing values for **brand** replace with mode
   - If there are missing values for **cylinders** replace with mean to the nearest integer.
   - Any other variables with missing values first find the min and max of the variable. You are also to be replace missing values with an “individual” random value from the observed distribution.
3. Provide summary statistics and standard deviation for the following variables weightlbs, cubic inches and mpg.
4. If we assume that outliers are more than 3 standard deviations from the mean (we know there are better methods to detect outliers!), how many outliers do we have for the mpg variable?
5. What mpg values are deemed outliers? What rows contain the mpg outliers?
6. Save the changes you have made to a designated directory and call it carsUPDATED.csv

STUDY EXERCISES -Conceptual

1. Describe the possible negative effects of proceeding directly to mine data that has not been preprocessed.

2. What is an outlier? Why do we need to treat outliers carefully?

3. Explain why it is not recommended, as a strategy for dealing with missing data, to simply omit the records or fields with missing values from the analysis.

4. Make up a classification scheme which is inherently flawed and would lead to misclassification. For example, classes of items bought in a grocery store. What is wrong with this classification?

Breakfast	**Count**
Cold Cereals	72
Sugar Snacks	1
Cheerios	2
Hot Cereals	28
Oatlets	3

5. Calculate the mean, median, and mode for the sample of stock price data.

6. Use the following stock price data (in dollars)

10	**7**	**20**	**12**	**75**	**15**	**9**	**18**	**4**	**12**	**8**	**14**

7. Compute the standard deviation of the stock price. Interpret what this number means.

The formula for **Sample Standard Deviation**:

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2}$$