Rbit004

6/20/2014

Elgin Perry

eperry@chesapeake.net

**Factors**

In this Rbit, we look at factors which are a special data structure in R. A *factor* is a vector which is used to define groups. Mostly I use them in dataframes to define groups of the other data in the data frame. A factor vector associates both an ordinal number and a character string with each group. The ordinal number defines the order of the groups and the character string gives the group a name. So for example, if you execute a boxplot where the groups are defined by a factor in the dataframe, the ordinal number controls the order in which the groups are plotted and character string will be used as a label for each box.

By default, the read.table() function is R forces any columns containing character strings to be a factor. I have previously advised against letting R do this. Further down, I will give examples of how this can lead to confusion. First let’s look at creating a factor using the factor() function which is what I recommend. For this we use the snook data. In this script it is read as usual and then factor is applied to water.body. The dataframe column ‘length’ is then plotted as a function of the new factor column ‘water.body.f’ using the boxplot() function.

options(stringsAsFactors = FALSE)

# be sure to change \ to /

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

setwd(ProjRoot);

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

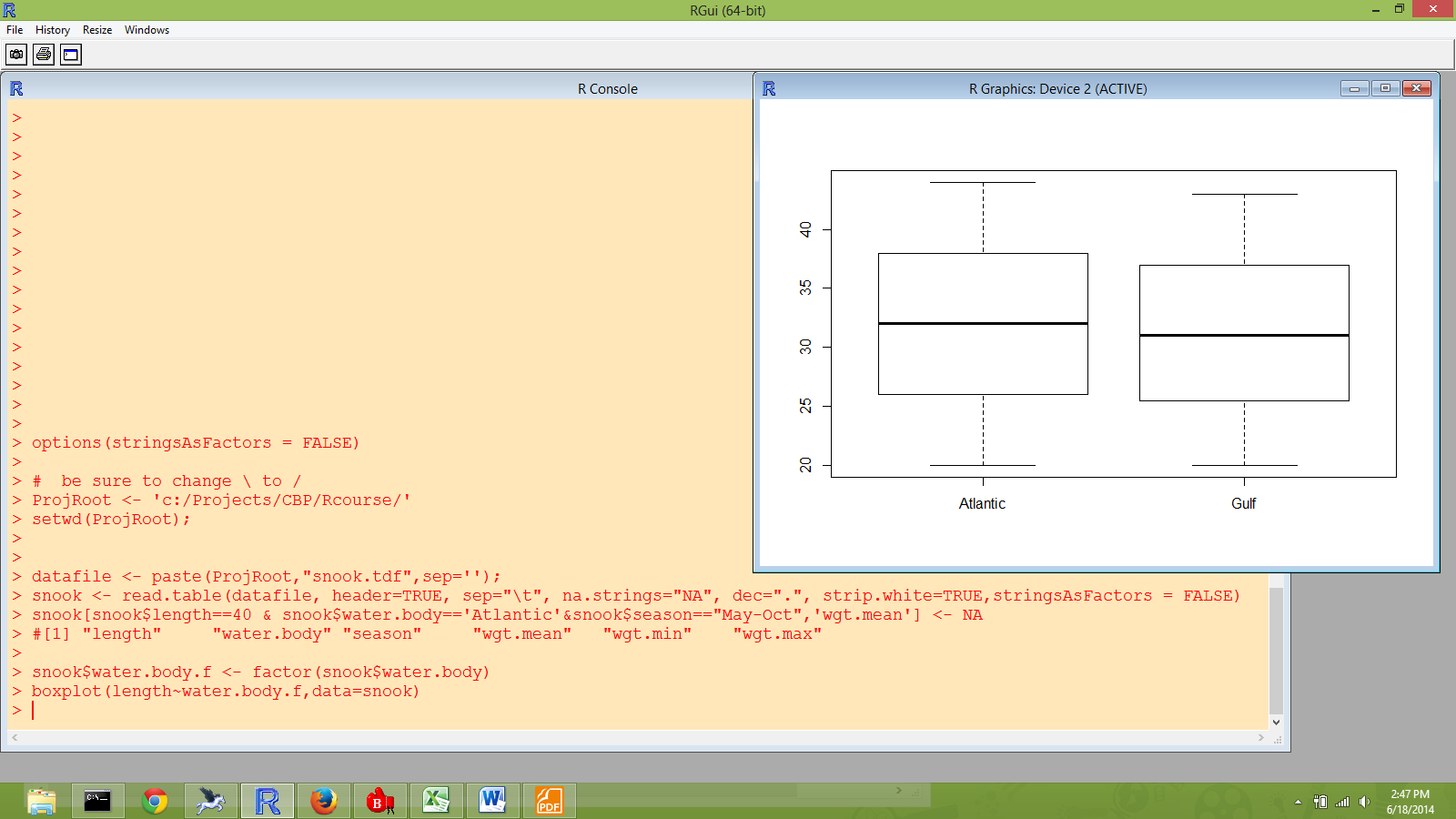
snook <- read.table(datafile, header=TRUE, sep="\t", na.strings="NA", dec=".", strip.white=TRUE,stringsAsFactors = FALSE)

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

#[1] "length" "water.body" "season" "wgt.mean" "wgt.min" "wgt.max"

snook$water.body.f <- factor(snook$water.body)

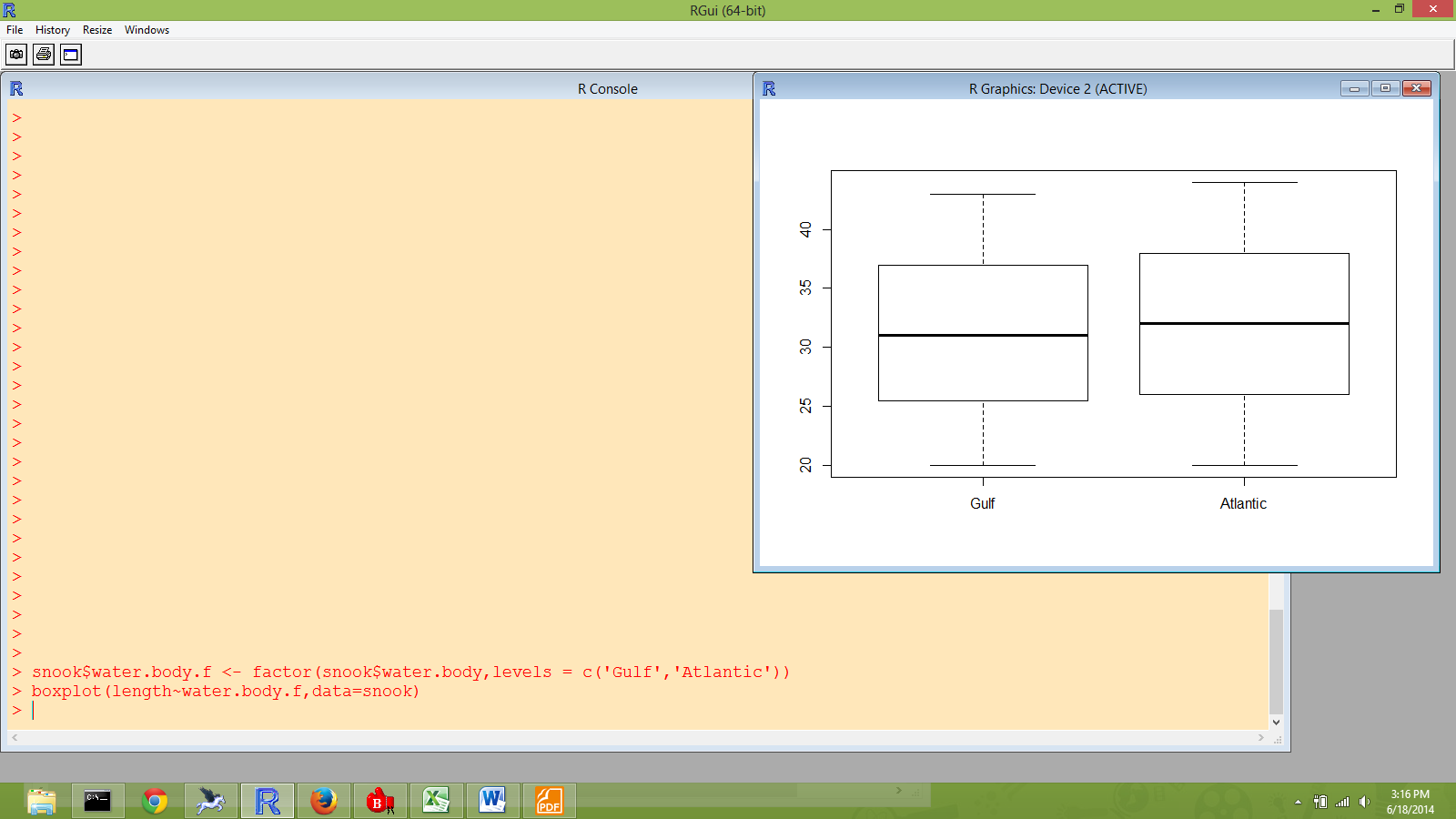
boxplot(length~water.body.f,data=snook)



Note that the order of the groups is alphabetical. You can control the order of the groups by adding a levels argument to factor(). For example:

snook$water.body.f <- factor(snook$water.body,levels = c('Gulf','Atlantic'))

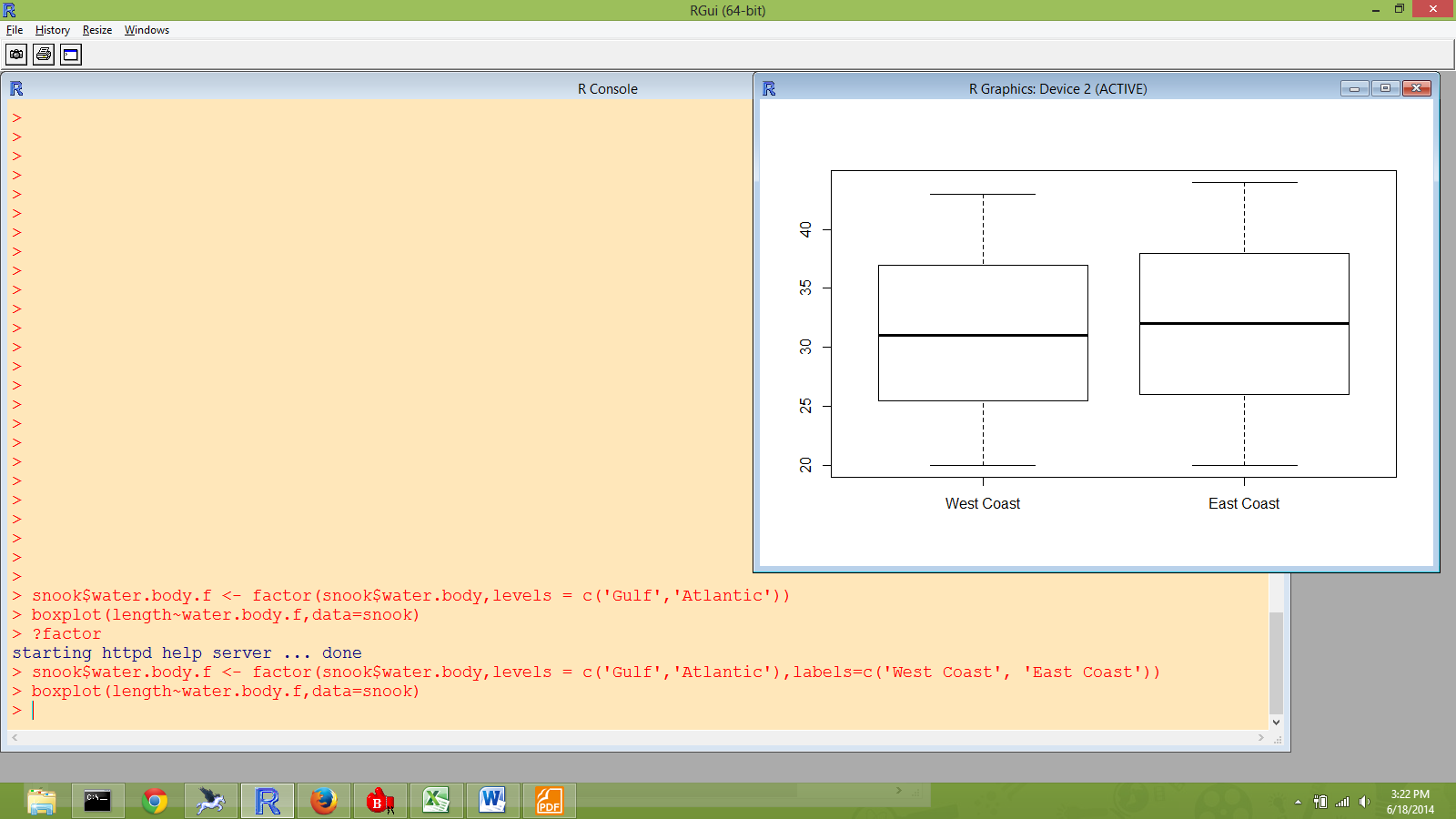
boxplot(length~water.body.f,data=snook)



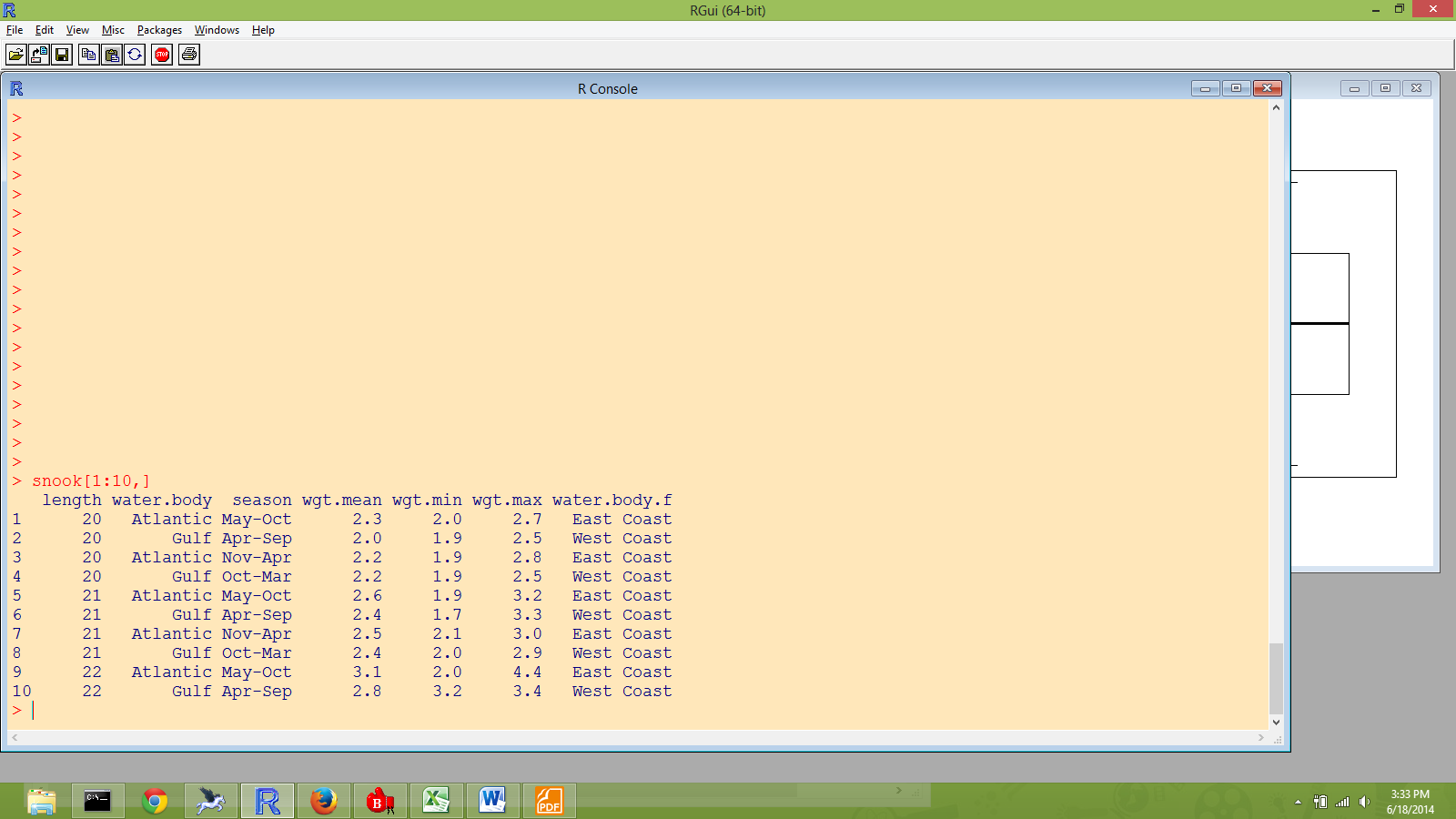
The elements given in the levels argument must equal exactly the values that appear in the data. Remember, R is case sensitive. You can also associate labels with the levels with a labels argument.

snook$water.body.f <- factor(snook$water.body,levels = c('Gulf','Atlantic'),labels=c('West Coast', 'East Coast'))

boxplot(length~water.body.f,data=snook)



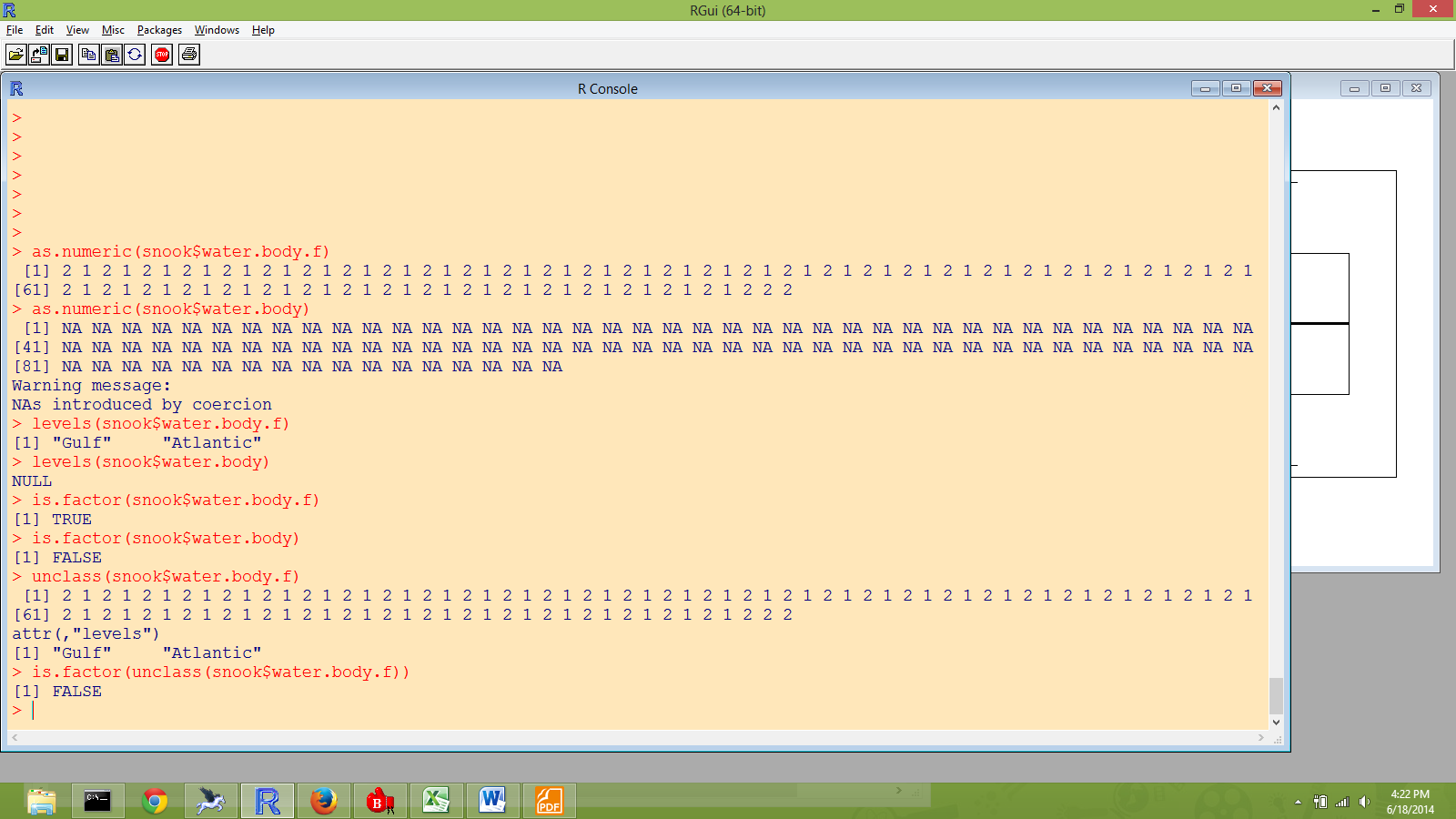
As far as I can tell, the use of labels completely supplants the original character strings. The data for water.body.f are now “East Coast” and “West Coast”.



I am switching it back to using the original data as labels:

snook$water.body.f <- factor(snook$water.body,levels = c('Gulf','Atlantic'))

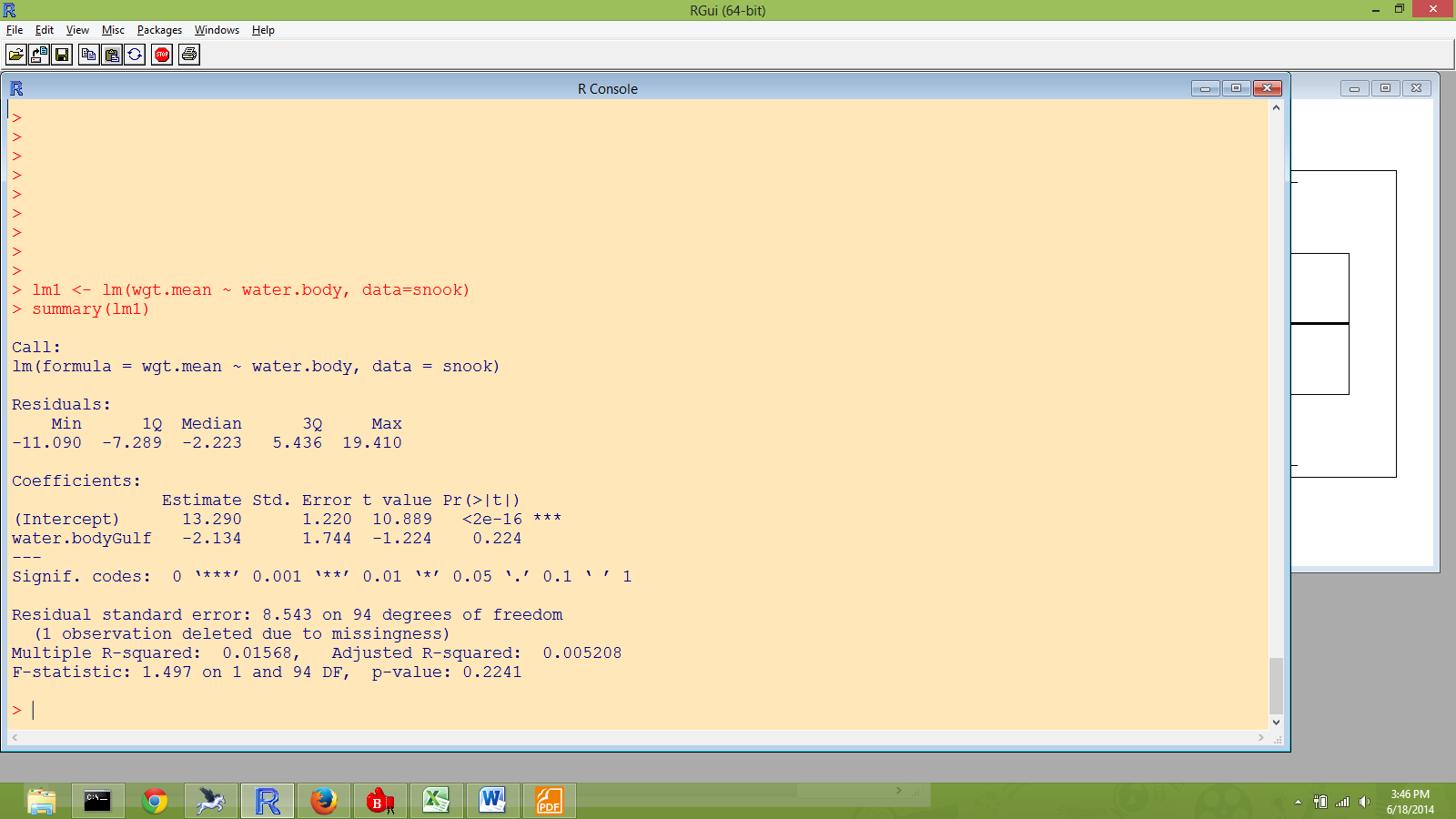
The next screen print shows some functions that can be used with factors as compared to characters. See if this all makes sense to you.



The order of factors is also important when using linear models. Here is a linear model with water.body (not the factor) as an explanatory variable.

lm1 <- lm(wgt.mean ~ water.body, data=snook)

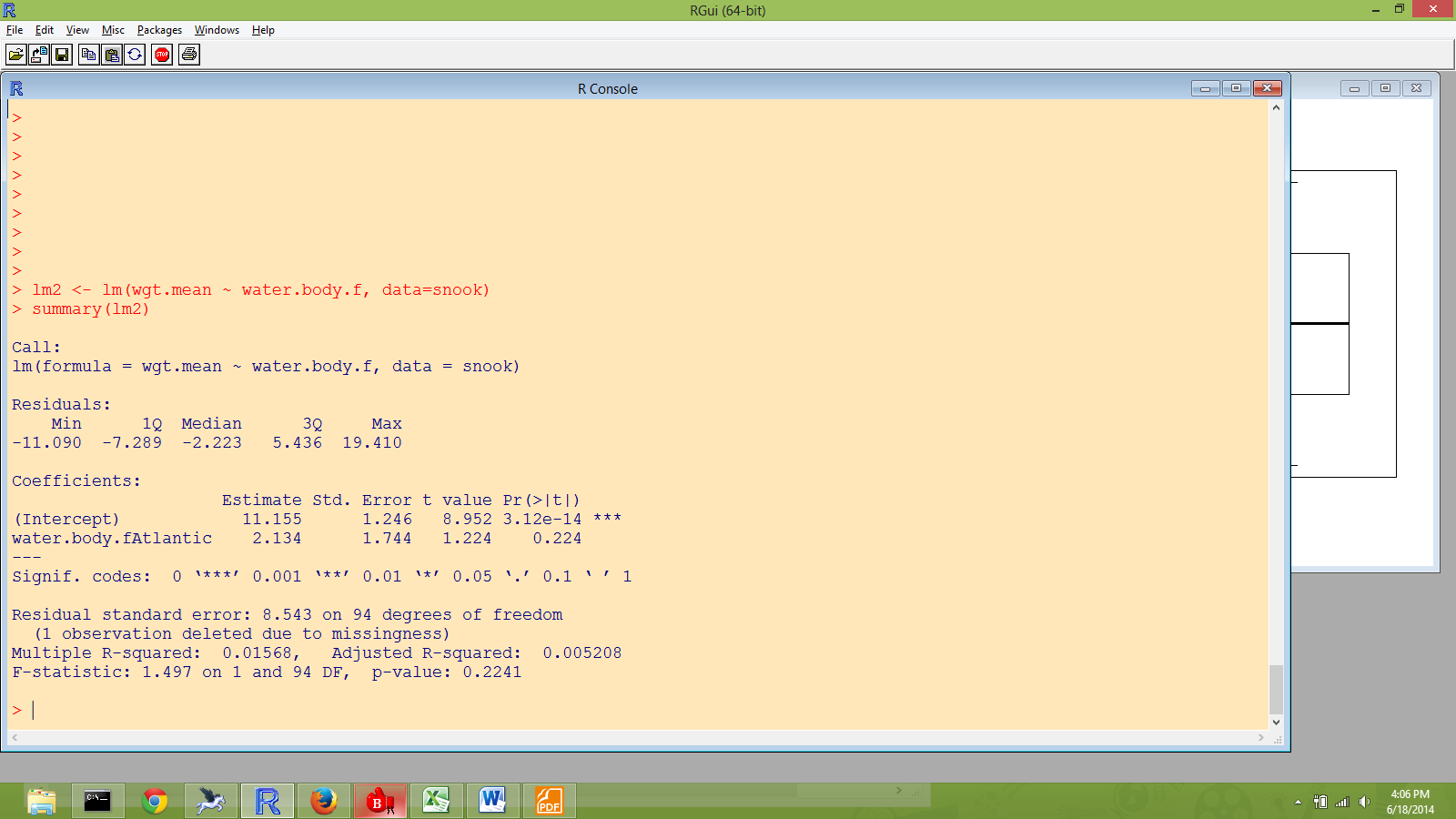
summary(lm1)



The lm() function has forced the column ‘water.body’ to be a factor and has used its default of setting the order to alphabetical, in this case levels=c(‘Atlantic’, ‘Gulf’). Furthermore, in order to make this linear model with a grouping variable identifiable, lm() has dropped out (set to zero) the parameter for the first member of the factor (in this case ‘Atlantic’). In the output, there is only a parameter estimate for water.body=Gulf and that estimate is actually the difference between the mean weight for Gulf and that for Atlantic. The statistics for this difference tell you if Atlantic is significantly different from Gulf (in this case p > 0.05). If you are not familiar with stat package methods for fitting linear models, this probably sounds very confusing. If it is confusing, lets save it for a session on linear models. The important thing is that you can control which group has its parameter set to zero. This is helpful if you have a control group and want all other treatments compared to the control. Here I run the same model with the column ‘water.body.f’ and the results are reversed:

lm2 <- lm(wgt.mean ~ water.body.f, data=snook)

summary(lm2)



Here we see that the parameter estimate is given for Atlantic which is actually the difference between Atlantic and Gulf. Note that the sign of the parameter estimate is reversed from above.

Earlier I promised a confusing example. For this I will simulate some data. That way you will get something a little extra in your study of factors.

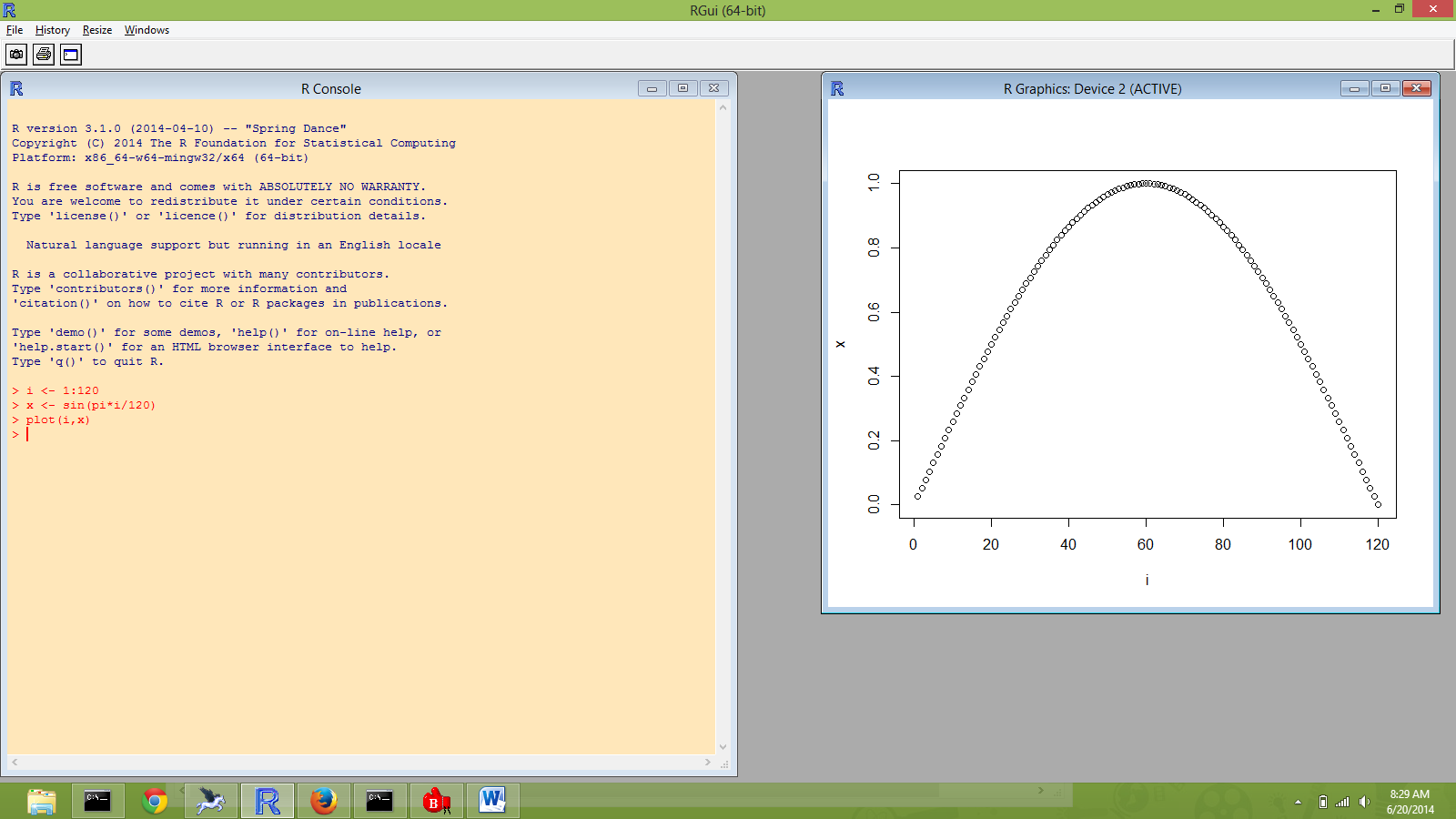
To simulate data, I first create the data structure. First create a vector of 120 sequential integers to number the observations.

i <- 1:120

Now I create a vector to represent a seasonal pattern over the 120 observations base on a sin() function. After creating this, check it out with plot().

x <- sin(pi\*i/120)

plot(i,x)

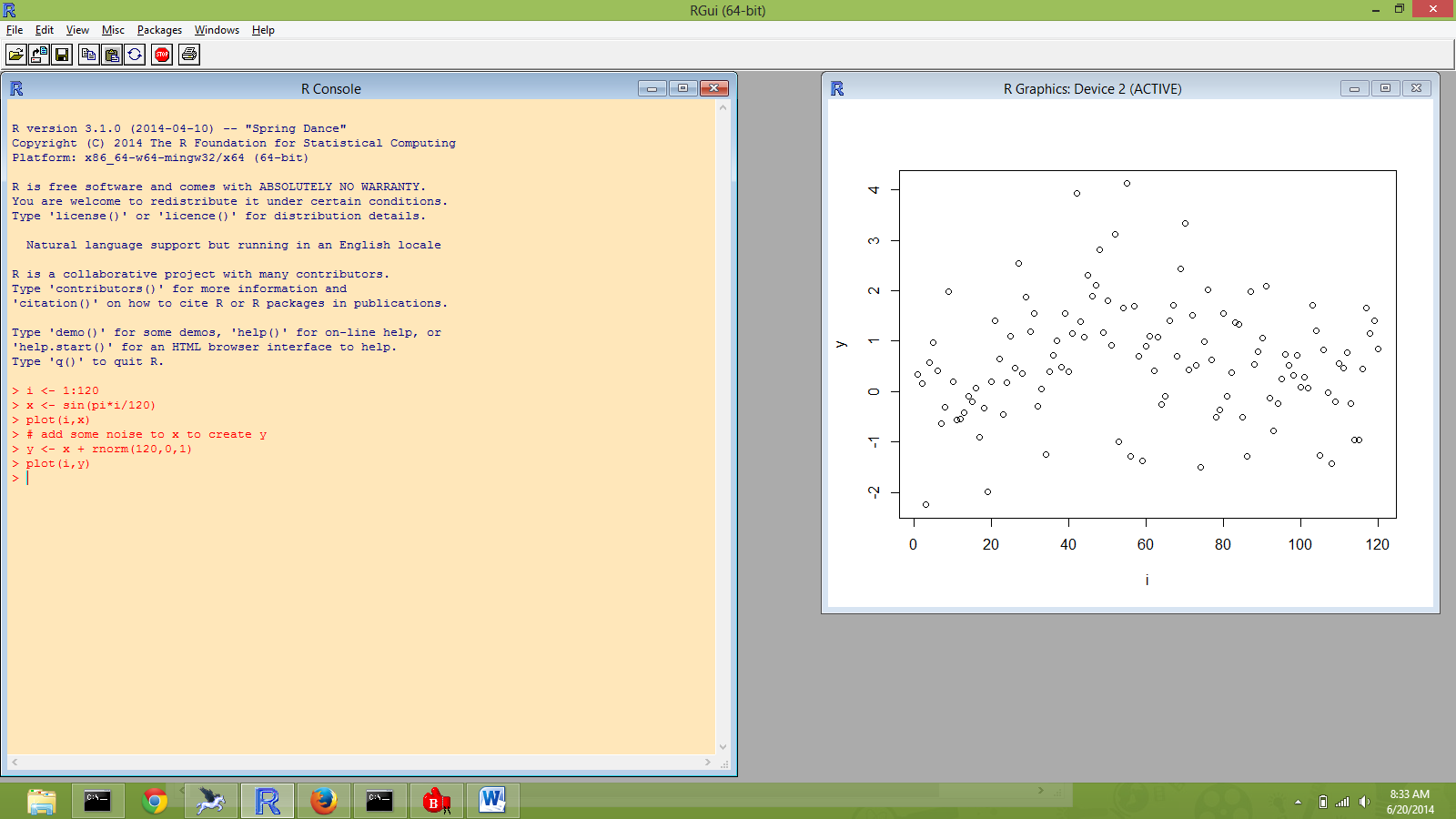


Now we add noise to the data using the normal random number generator rnorm().

# add some noise to x to create y

y <- x + rnorm(120,0,1)

plot(i,y)

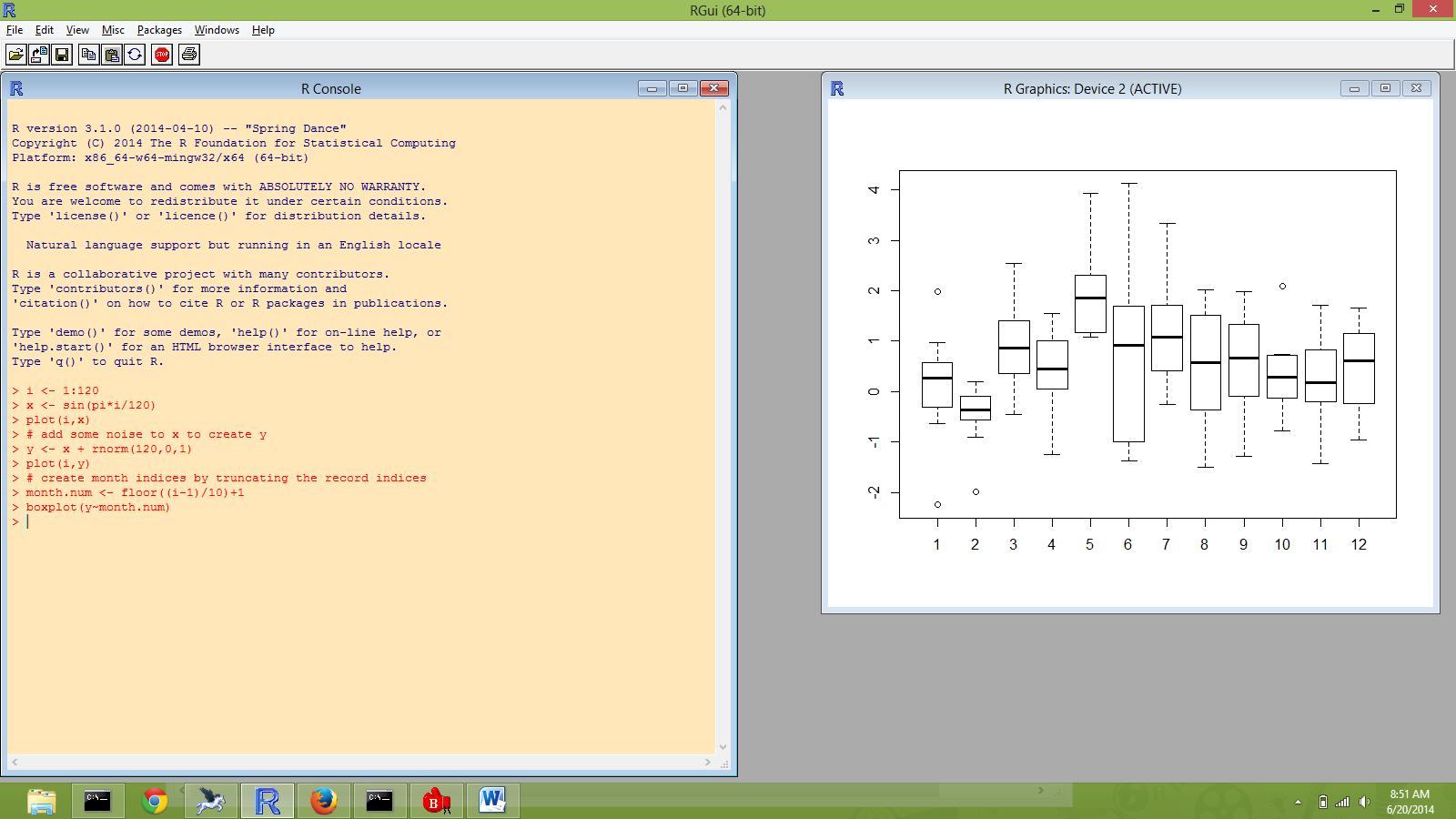


Now group the data by creating a numeric month variable from the index variable.

# create month indices by truncating the record indices

month.num <- floor((i-1)/10)+1

boxplot(y~month.num)



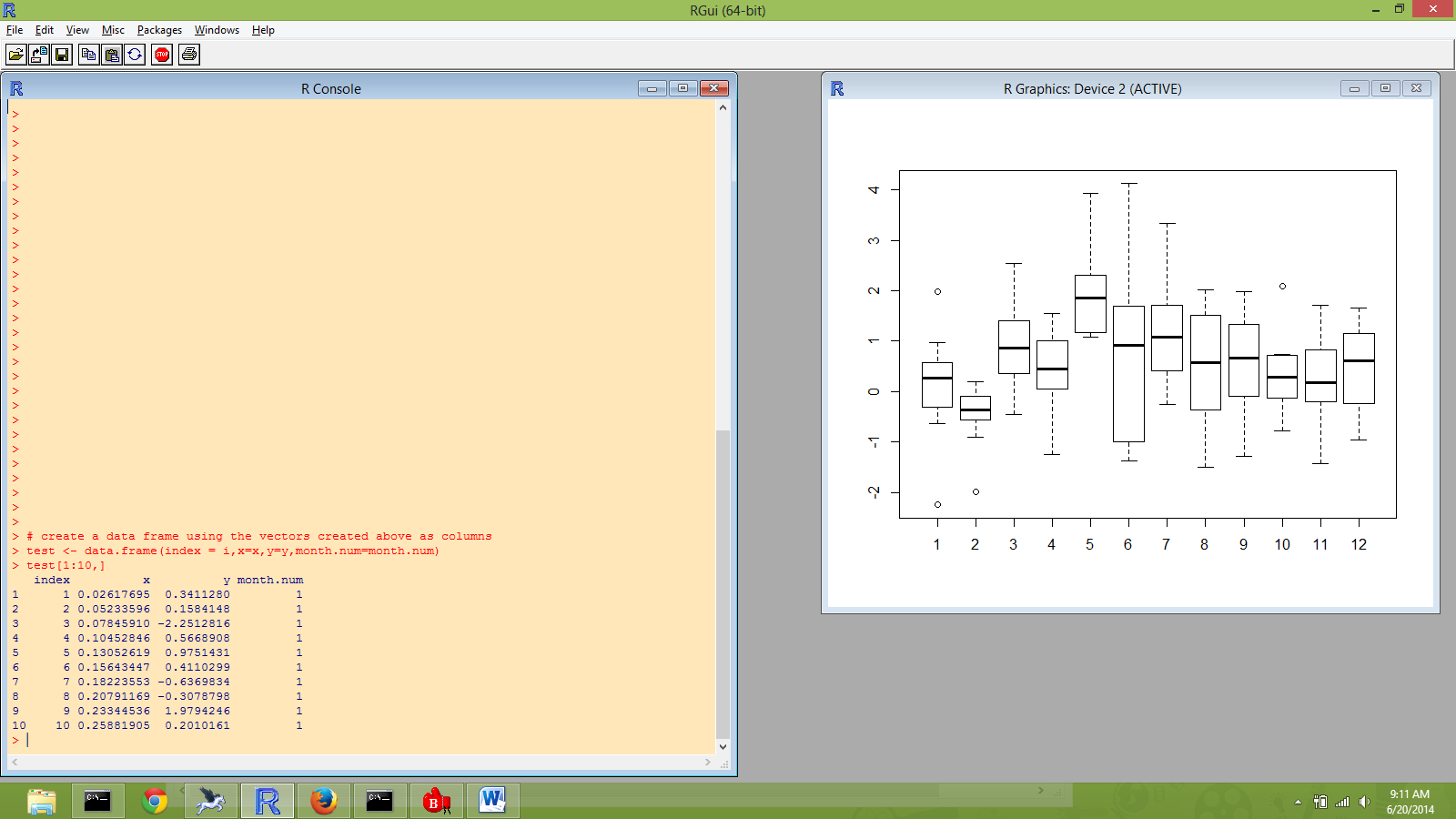
The seasonal pattern is obscured by noise, but still discernable. Keep this plot in mind because we will be comparing subsequent plots to this one.

Up to this point, I have been storing the data in vectors and relying on the fact that these is an element by element match between the x’s and y’s to keep the observations paired up. Many people do much of their data analysis using this method of data management. However, I feel that the data frame offers some advantages. For example, one variable may have a missing value causing the vector representing that variable to be a different length. At this point, I put the vectors I have created into a data frame.

# create a data frame using the vectors created above as columns

test <- data.frame(index = i,x=x,y=y,month.num=month.num)

test[1:10,]



Note the in the arguments of data.frame(), the term before the ‘=’ is the name of the data frame column, and the term after the ‘=’ is the data being assign to the data frame column. The contents of the vector i was put into the data frame column test$index and the contents of the vector x was put into test$x. Both x and test$x exist as separate objects. If you make a change in x, it will not change test$x unless you reassign the contents of x to test$x (e.g. test$x <- x).

Now I create some vectors of month names (vector length = 12) and then assign these to the records in the data frame based on month.num.

# create a vector of month names

months <- c('January','February','March','April','May','June','July','August','September','October','November','December')

# create a vector of month name abreviations

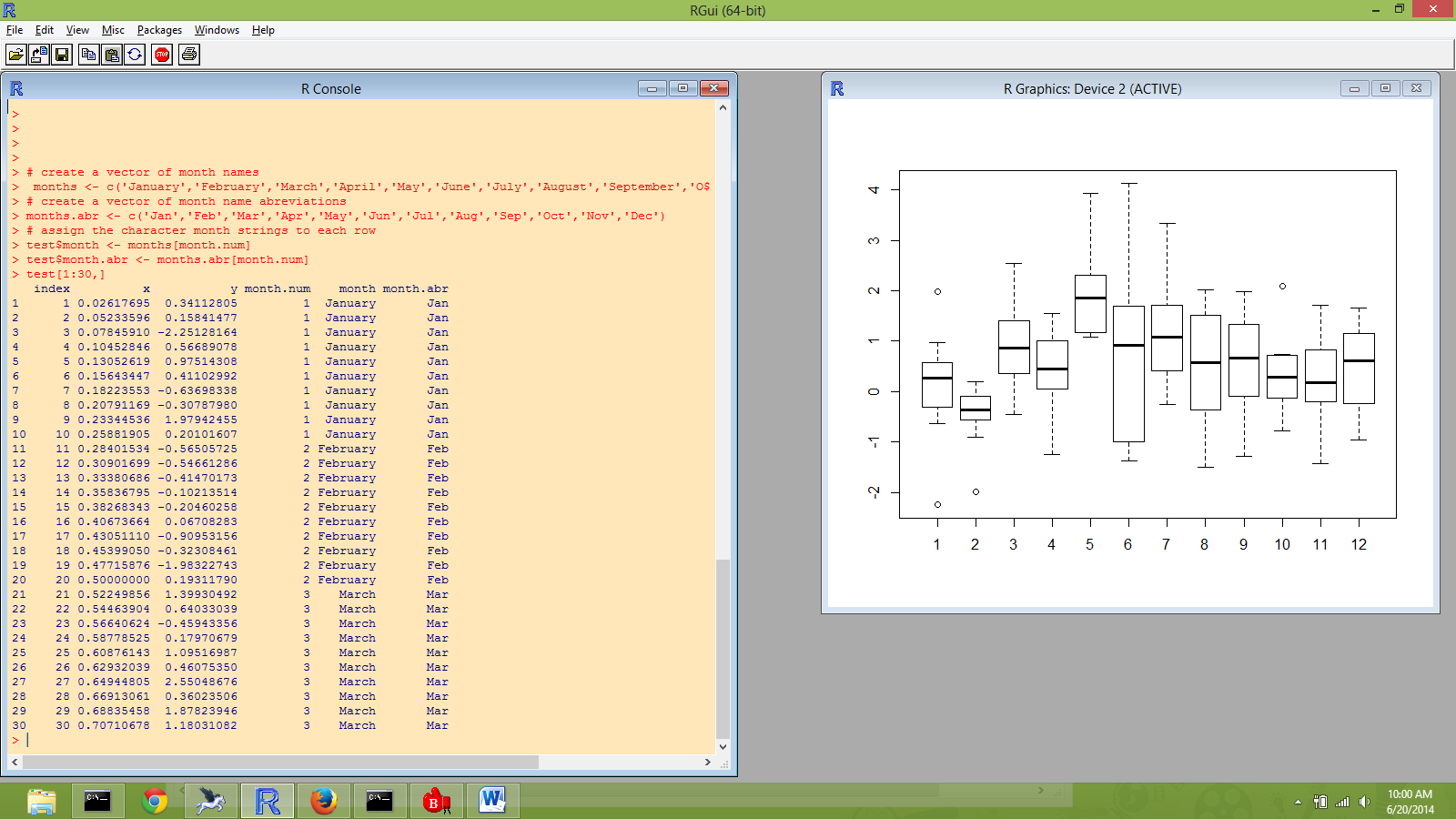
months.abr <- c('Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec')

# assign the character month strings to each row

test$month <- months[month.num]

test$month.abr <- months.abr[month.num]

test[1:30,]



That concludes the data simulation and we move forward with an example of a confusing factor. Here I convert the test$month column to a factor test$month.f letting R use the default assignment of levels which is alphabetical order.

# use factor function with default levels

test$month.f <- factor(test$month)

Now compare box plots using the original numeric month variable and using the factor month variable.

par(mfrow=c(2,1)) # sets up for multiple plots per page

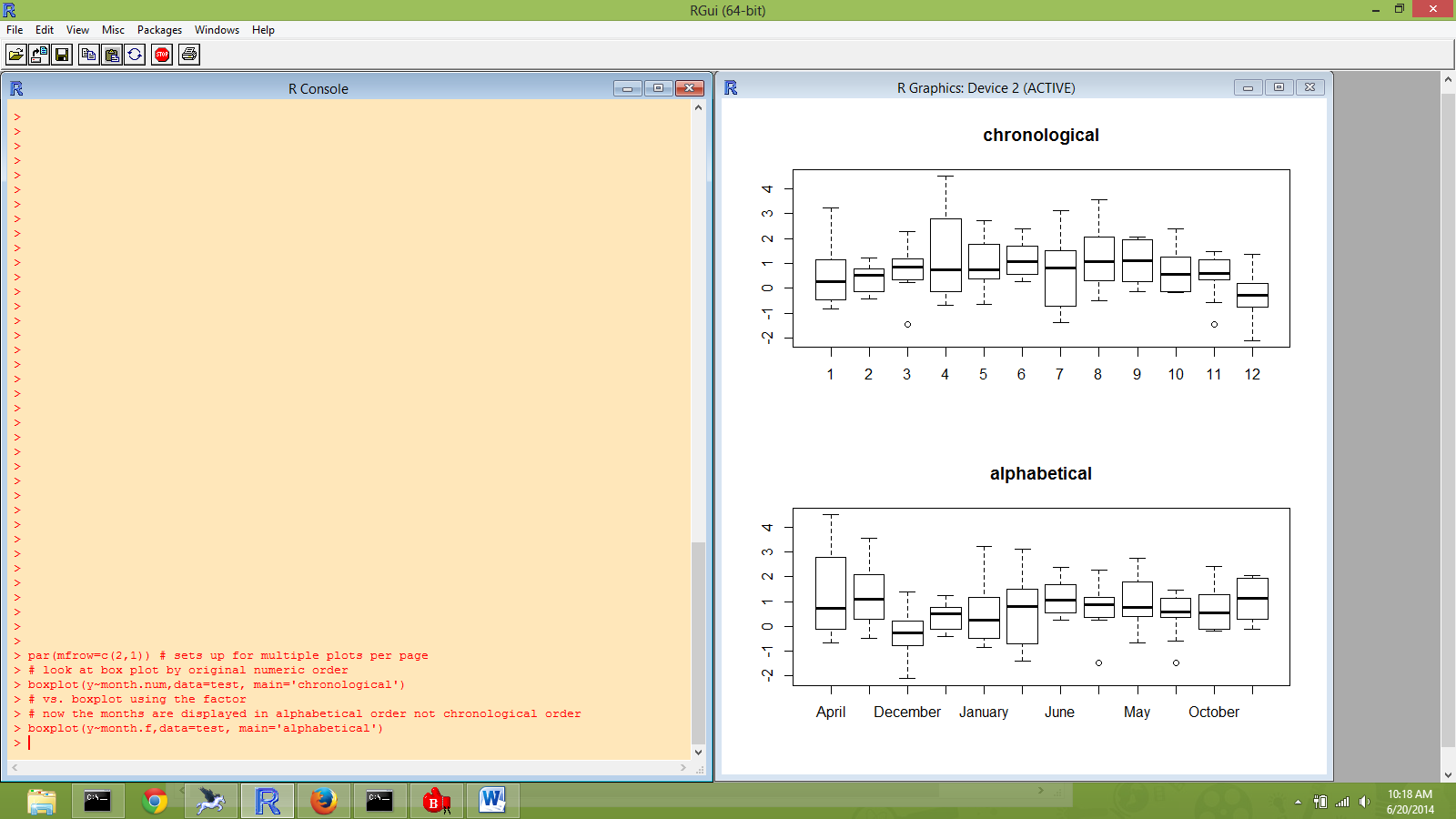
# look at box plot by original numeric order

boxplot(y~month.num,data=test)

# vs. boxplot using the factor

# now the months are displayed in alphabetical order not chronological order

boxplot(y~month.f,data=test)

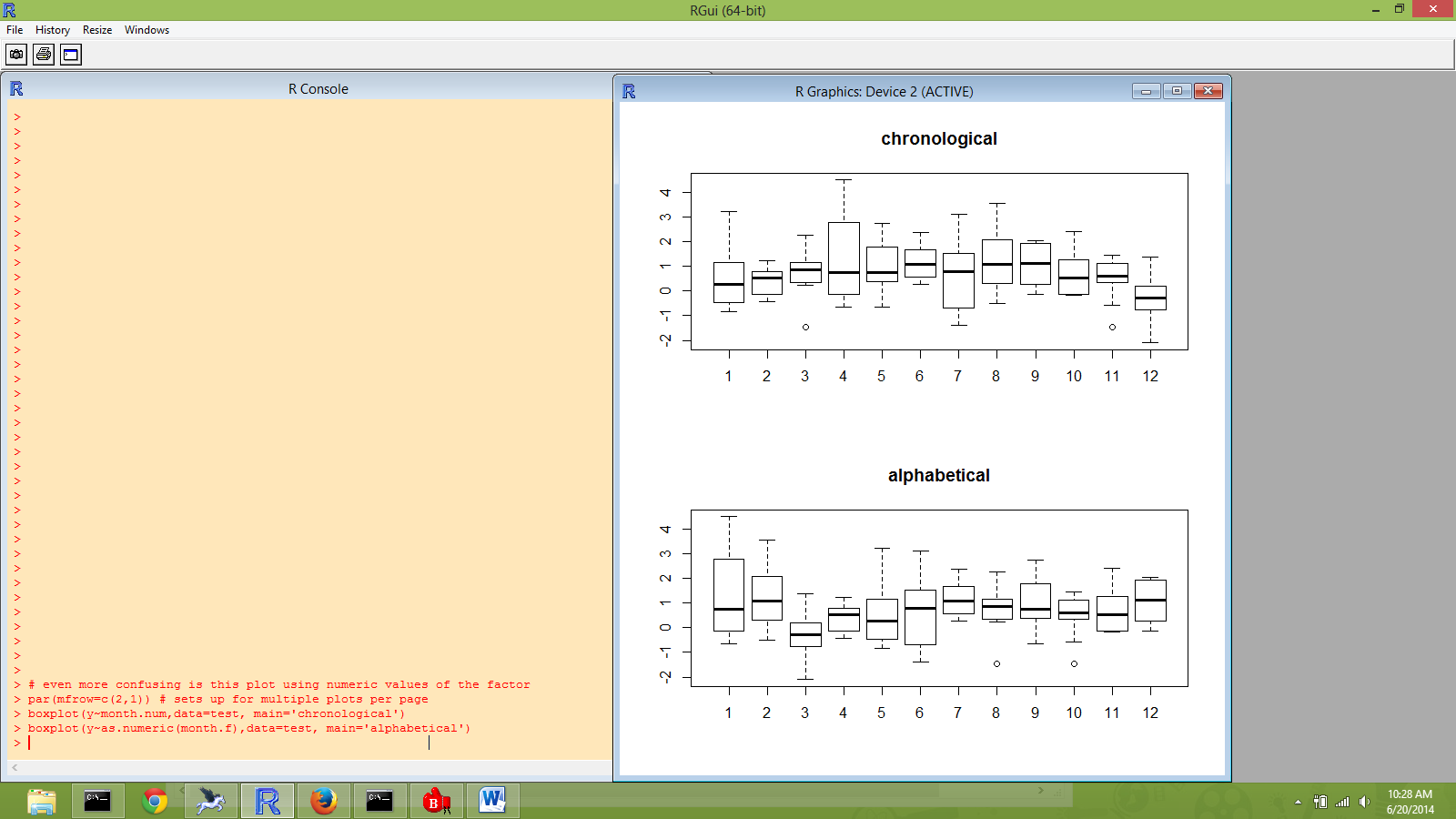


The data are plotted in chronological order in the top panel, but alphabetical order in the bottom panel where R uses its default levels. It is potentially more confusing to observe that there is not room on the x-axis for all of the month names and to decide to convert them to month numbers using the as.numeric() function.

# even more confusing is this plot using numeric values of the factor

boxplot(y~month.num,data=test, main='chronological')

boxplot(y~as.numeric(month.f),data=test, main='alphabetical')



This happen for me once when I let R read month names from a file and convert then to a factor automatically. I was left wondering why the data did not exhibit more of a seasonal pattern. I decided at that point that it is better to not let R create factors automatically, but rather to create factors with factor() and make a conscious decision regarding the ordering of the levels. In this case the factor() function would be used as follows:

#either of these two statements will assign levels to the months in chronolgical order

test$month.f <- factor(test$month.abr,levels=c('Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec'))

#or

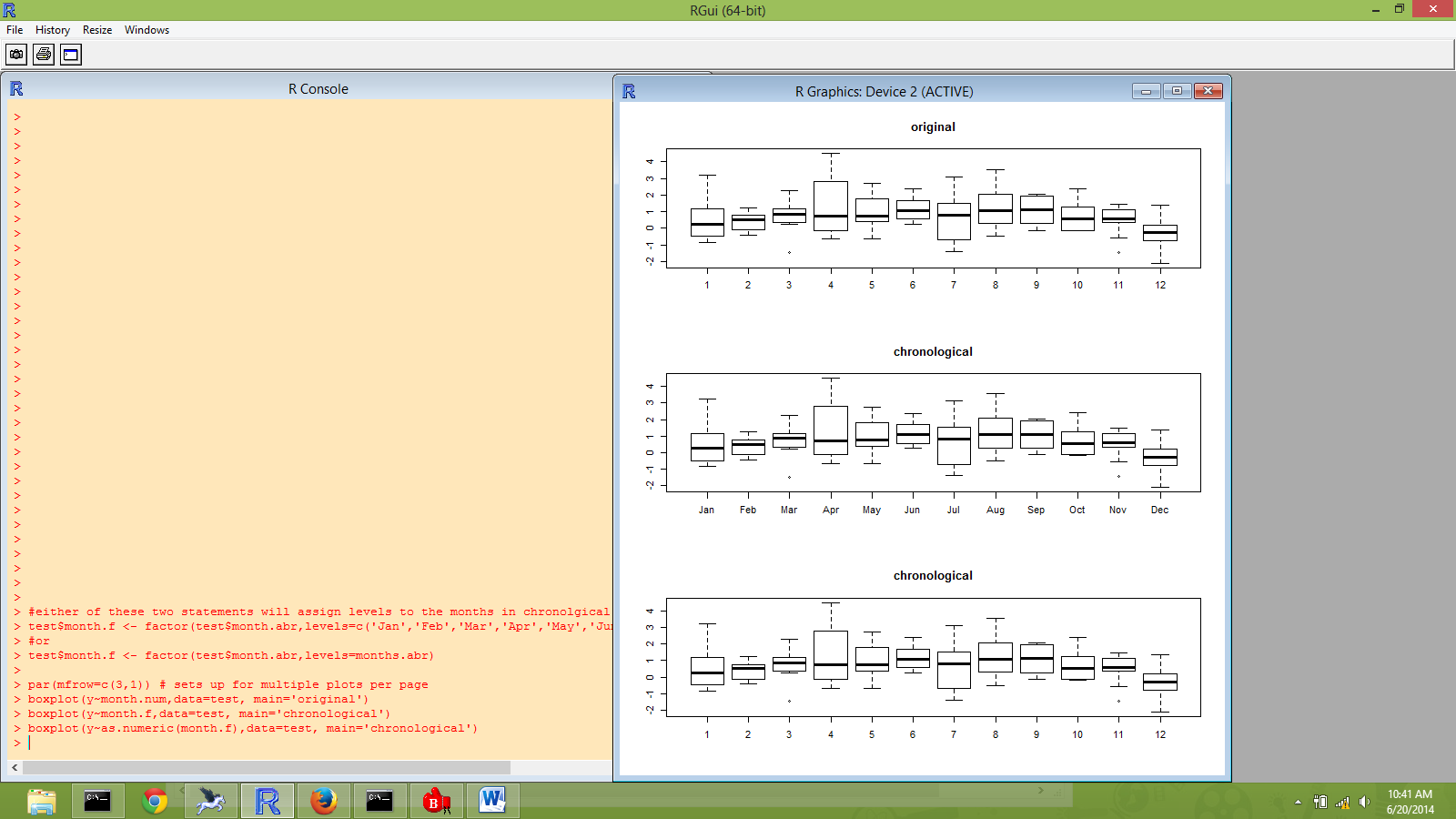
test$month.f <- factor(test$month.abr,levels=months.abr)

par(mfrow=c(3,1)) # sets up for multiple plots per page

boxplot(y~month.num,data=test, main='original')

boxplot(y~month.f,data=test, main='chronological')

boxplot(y~as.numeric(month.f),data=test, main='chronological')



End of Rbit004.