

Basic R Commands:

Comments—

If you want to insert a comment (i.e., non-executable commands) anywhere in a “program”, you need to preface the line of code with a # sign. Everything in the line after the # sign will be considered a comment. Only starting a new line will break the comment “mode”. This means that, if you have a multi-line comment to make, you must have a # sign on every line.

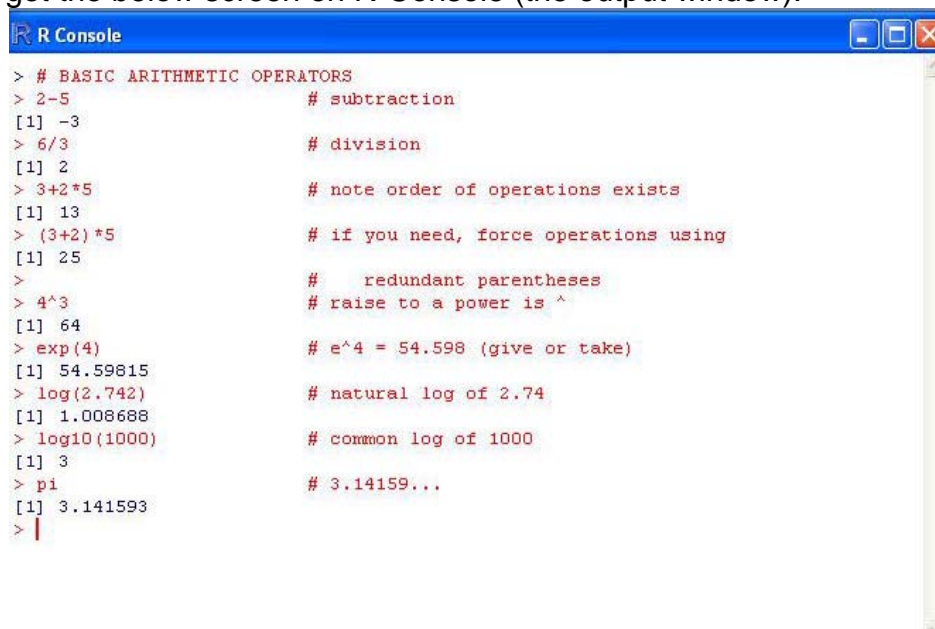
=====

Basic Arithmetic with R—

Below is a script called *arithmetic.R* . Start R, minimize R-Commander, then go to FILE NEW SCRIPT, opening up a blank script window. Copy the lines below into the script window.

```
# BASIC ARITHMETIC OPERATORS
2-5           # subtraction
6/3          # division
3+2*5        # note order of operations exists
(3+2)*5      # if you need, force operations using
              #   redundant parentheses
4^3          # raise to a power is ^
exp(4)       # e^4 = 54.598 (give or take)
log(2.742)   # natural log of 2.74
log10(1000)  # common log of 1000
pi           # 3.14159...
```

Now click once in the R-Console window, then right click and CLEAR WINDOW, then click once back in the script window. Now click on EDIT RUN ALL. You should get the below screen on R-Console (the output window).



```
R Console
> # BASIC ARITHMETIC OPERATORS
> 2-5           # subtraction
[1] -3
> 6/3          # division
[1] 2
> 3+2*5        # note order of operations exists
[1] 13
> (3+2)*5      # if you need, force operations using
[1] 25
>              #   redundant parentheses
> 4^3          # raise to a power is ^
[1] 64
> exp(4)       # e^4 = 54.598 (give or take)
[1] 54.59815
> log(2.742)   # natural log of 2.74
[1] 1.008688
> log10(1000)  # common log of 1000
[1] 3
> pi           # 3.14159...
[1] 3.141593
> |
```

The red letters are script commands or comments. The blue letters are output (in this case computations). Notice that before every output number is the [1] designation. This indicates that the computer program calls every number a vector or a matrix. In the cases above, every single number is considered a 1 x 1 matrix, and therefore takes on the index position [1].

Finally, click once in the script window again. Now click on FILE SAVE to save this script. Save it on your desktop (or on your flash stick) as **arithmetic.R**. Don't forget to type the .R (capital R) in the name.

I propose that you use this routine for a while, until you get more comfortable with R. That is, start R, open a new script window, type your program in the script window, then clear the R-Console window and RUN ALL of the script window. If the program runs flawlessly, then save it as a .R program, to be stored in your "R library" of scripts.

=====

Variables—

You can name variables, vectors, matrices, functions, and subroutines. You are not restricted by the names you use, as long as they are not reserved for existing functions (like the function **sum()**) in the R package you are using. Remember that R is case sensitive, so the variable x is different from the variable X. Copy the below into a new script window and RUN ALL in the R-Console window.

```
# VARIABLE NAMES
x <- 5
x                                     # 1 way to print out contents of a variable
var1 <- 7/2
print(var1)                           # another way to print contents

val i d. vari abl e. name <- 18.6
                                     # you can even have long variable names
                                     # if you are of that kind of wierdness

val i d. vari abl e. name
```

Now run the script below and see what it does.

```
# VARIABLES AND ARITHMETIC
a <- 3    # This stores the value 3 into variable a
a

# valid R variable names:  a
#                          N
#                          n203
#                          var. pop1
# very. long. name. comtai ni ng. many. characters. 12345

# R IS CASE SENSITIVE, SO B IS DIFFERENT FROM b, etc.

# some common arithmetic
sqrt(a)    # square root of a (which is 3 at this moment)
a^4        # a raised to the fourth power
```

```

log(a)      # natural log of a
log10(a)    # common log of a
exp(a)      # e to the power a
tan(a)      # tangent of a
b <- pi     # pi e, the number approx 3.14159
b

```

=====
Vectors and matrices—

Run the script below, noticing what it does.

```

#          VECTORS

x <- c(2, 3, 5, 1, 4, 4)      # create a row vector with those 6 numbers
                              #   and then call the vector x
x

sum(x)                       # sums the elements of vector x
mean(x)                      # finds the mean of vector x
sd(x)                        # finds standard deviation of vector x
median(x)                    # finds the median of x
sqrt(x)                      # finds square root of every element of x
x^2                          # finds the square of every element of x

seq(1, 10)                   # makes a vector 10 long, from 1 to 10, by
l's                          #   1's
seq(1, 10, 2)                # makes a vector from 1 to 10,
                              #   starting with 1, then skipping 1
                              #   ending up with only odds from 1 to 10

seq(1:10)                    # same as seq(1, 10)
seq(1, 10, by=2)             # same as seq(1, 10, 2)

y <- c(1:7)                  # create a row vector with elements 1
                              #   through 6
y
z <- 1:7                    # same as y--use when you increment by 1
only y
z
w <- c(1:12, 0, -6)         # use the c( when you increment by 1
                              #   followed by more numbers not in
                              #   sequence
w

```

Now run the script below, noting what it does

```

# SOME OPERATIONS WITH NUMERICAL VECTORS AND LOGICAL VECTORS

x <- c(-5, 0, 7, -6, 14, 27)  # data vector x
x[1]                         # prints first element of x
x[length(x)]                 # prints last element of x
x[i]                         # the i th entry if 1<=i<=n, NA if i>n,
all but                       #   the i th if -n<=i<=-1, an error if i
< -n, and                     #   an empty vector if i=0
x[c(2, 3)]                   # the second and 3rd entries

```

```

x[-c(2, 3)]           # all but the second and third entries
x[i] <- 5             # assign a value of 5 to the first entry;
                    # also can use x[i] = 5

x[c(1, 4)] <- c(2, 3) # assign values to the first and fourth
entries              # entries

x[indices] <- y      # assign to those indices indicated by
the values          # of INDICES: if y is not long enough,
values              # are recycled; if y is too long, just its
initial            # values are used and a warning is issued

x < 3                # vector with length n of TRUE or FALSE
                    # depending if x[i] < 3

which(x<3)           # which indices correspond to the TRUE values
                    # of x<3

x[x<3]               # the x values when x<3 is TRUE -- same
                    # as x[which(x<3)]

```

Graphing—

Run the script below, noting that a new window opens up, the graphics window, shown below.

```

# INTRO TO GRAPHING

x <- c(2, 4, 4, 6, 6, 5, 5, 7, 3, 7, 3, 8, 9, 7, 9, 6, 4, 3, 4, 4, 6, 2, 2, 1, 2, 4, 6, 6, 8)
# input a vector x with data

y <- c(1:29)
# input vector y of consecutive numbers
# 1 through 29

hist(x)
# make a histogram of x

plot(x, y)
# make a scatter plot of y vs x

plot(x, type="b")
# make a time plot of x, connecting dots

y <- rbinom(20, 12, .4)
hist(y)
# make a histogram of a sampling of 20 from
# the binomial distribution B(12, .4)

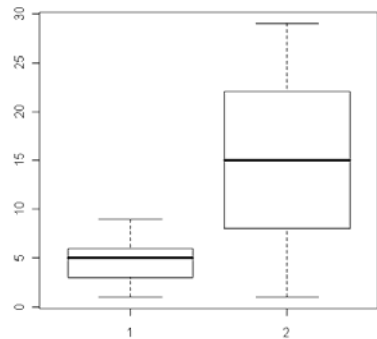
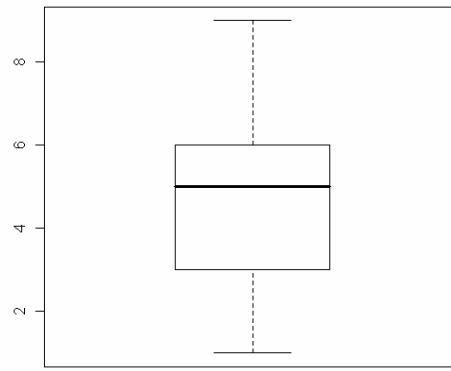
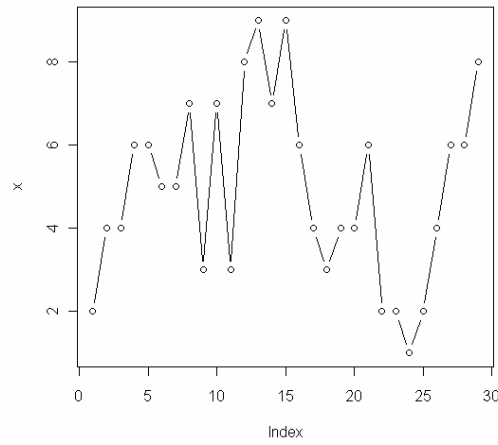
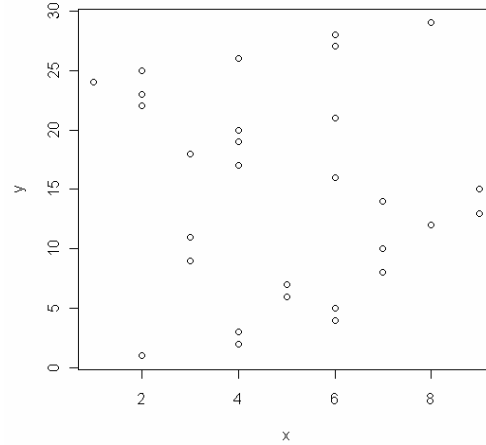
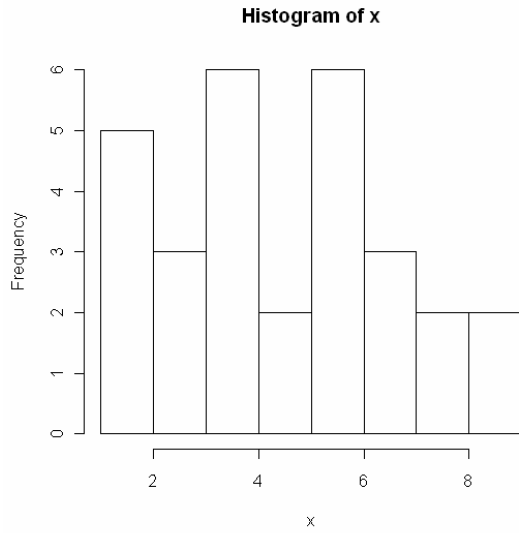
a
# make another binomial histogram, but using
# sample of 200
y <- rbinom(200, 12, .4)
hist(y)

boxplot(x)
# make a boxplot of x data

boxplot(x, y)
# make side by side boxplots of x and y data

```

Note that I can copy and paste the graphic to WORD by right clicking in the graphics window of R, then click on COPY AS BITMAP and then finally PASTE into WORD



You notice that these graphics “work”, but are not as stunning as SPSS. There are some things you can adjust and alter to get

them looking a little nicer, but we will not address those “refinements” in this intro session.

Logical Operators—

R does operations on constants, vectors and matrices logically as well as arithmetically. Run the script below to see.

Now run the next script.

```
# LOGICAL OPERATORS
x <- 1:5 # x is a row vector = [1, 2, 3, 4, 5]
test <- (x <4) # test is a row vector (logical <)
# = [TRUE TRUE TRUE FALSE FALSE]
test <- (x==3) # test is a row vector (logical =)
# = [FALSE FALSE TRUE FALSE FALSE]
test <- (x>1 & x<4) # test is a row vector (logical AND)
# = [FALSE TRUE TRUE FALSE FALSE]
test <- (x>4 | x==2) # test is a row vector (logical OR)
# = [FALSE TRUE FALSE FALSE TRUE]
test<- (x != 4) # test is a row vector (logical NOT =)
# = [TRUE TRUE TRUE FALSE TRUE]
test <- (x %in% c(2, 4)) # test is a row vector
# testing whether the entry is
# a 2 or a 4
# = [FALSE TRUE FALSE TRUE FALSE]
```

Some Distributions—

Run the script below.

```
# THE NORMAL DISTRIBUTION
pnorm(1.43) # finds P(Z < 1.43)
pnorm(129, 100, 16) # finds P(X < 129) where x comes from
# normal with mean 100 and std dev 16
qnorm(.994) # finds the .994 quantile of the standard
normal
qnorm(.95, 100, 16) # finds the .95 quantile of N(100, 16)
distribution
```

Now run the script below to learn a little about the uniform distribution and other misc. “tips to know”. (We also showed a simple loop, called a “for” loop)

```
# THINGS TO KNOW
x <- runif(30, 0, 1) # creates a vector 30 long
x # of random values from U(0, 1)
y <- c() # before you can use a vector in a
loop
for (i in 1:10) { # you must "save space" for it by
y[i] <- rbinom(1, 3, .2) # typing the y <- c() command
```

```

}
y
s <- c(1:5)           # for R you can use <- or =
s                     # for a variable assignment
t = c(2:6)
t

```

The Help Command—

There are many ways to invoke the help menu commands. One way is to click on the window on the top right side of the R-Console window.

Another way is to use the question mark after the red prompt `>` in the R-Console window. For example, if you know the syntax of the command you are looking for (like `mean`), you would type

```
>? mean
```

and the help information will show up on screen.

Another way is to use the help command. For example, to again find information on the command `mean` you would type after the R-Console prompt `>help("mean")`

If you don't quite know what the official name of the command you are looking for, you can still get R to assist you in searching for the command. For example, if I wanted to find out the command for standard deviation in R, but don't know what it is (it is `sd` by the way), I could type the following in R-Console

```
>help.search("standard deviation")
```

And you will get a window with some possibilities which may be what you are trying to find. R uses "fuzzy logic" to give you these possibilities, since you cannot tell R the exact command you need.

The fuzzy logic screen that shows up for this search is shown below.

The screenshot shows the R GUI with the R Console window open. The console displays the following text:

```

Loading required package: rnlmk
Loading R Information
Loading
Help files with alias or concept or title matching 'standard deviation'
using fuzzy matching:
Rcmdr:
> >std
Error: pooledSD(nlme)      Extract Pooled Standard Deviation
> ?std
Error: numSummary(Rcmdr)  Mean, Standard Deviation, and Quantiles for
> ? st
Error: sd(stats)         Standard Deviation
> help
Error: z.test(TeachingDemos) Z test for known population standard deviation
No doc
you co
> 'hel
+ |
Type 'help(FOO, package = PKG)' to inspect entry 'FOO(PKG) TITLE'.
[1] "h
> help
> help
> help
Error:
> help
Error:
> help
> |

```