RESEARCH SCHOOL OF FINANCE, ACTUARIAL STUDIES AND APPLIED STATISTICS

Solutions to R Worksheet 2.

Q1

```
>
> # Q2
>
> 3+3
[1] 6
>
> 3-3
[1] 0
>
> 3*3
[1] 9
>
> 3/3
[1] 1
>
> 3^3
[1] 27
>
> 3 < 5
[1] T
>
> # The value of this comparison is the logical constant T for true,
> # as 3 is indeed less than 5!
>
> 3 > 5
[1] F
>
> # False (F), 3 is not greater than 5
>
> 3 > 5 || 3 < 5
[1] T
>
> # It is true (T) that 3 is either greater than 5 or less than 5.
>
> sqrt(4)
[1] 2
>
>
> # Q3
>
> c <- 5
> c
[1] 5
>
> c^3
[1] 125
>
> c*3
[1] 15
>
> objects()
[1] "c"
>
> search()
```

```
>
> square <- function(x)
+ {x^2}
>
> # Note the "+" sign in the first column of the last line.  This is
> # the continuation symbol - it means that R expects you to
> # complete typing the command you started on the previous line.
> # If you ever get this symbol by mistake and you don't want to, or
> # don't know how to complete the command you've started, then use
> # control-C (^c) to abort the command.  You should then be returned
> # to the ">" prompt.
>
> objects()
[1] ".Last.value" "c"           "square"
>
> # As suggested, I now have the objects "c" and "square" in my area,
> # as well as another object ".Last.value" which contains the last
> # bit of function output in the session, in this case the results
> # of my last call on the objects() function.
>
> .Last.value
[1] ".Last.value" "c"           "square"
>
> square
function(x)
{
    x^2
}
>
> square(c)
[1] 25
>
> square()
Error in square: Argument "x" is missing, with no default: square()
Dumped
>
> # Note that as the last command resulted in an error, my area now
> # contains another object:
>
> ls()
[1] ".Last.value" "c"           "last.dump"   "square"
>
> # This object contains more detailed information on the last error:
>
> last.dump
[[1]]:
[1] "No Frame Available"

$"square()":
[1] "No Frame Available"

attr(, "message"):
[1] "Argument \"x\" is missing, with no default"
>
> # Q4
>
> # Scalars:
>
> c <- 4
> c
```

```
[1] 4
>
> # Vectors:
>
> seq(1, 10)
 [1]  1  2  3  4  5  6  7  8  9 10
>
> rep(1, 10)
 [1] 1 1 1 1 1 1 1 1 1 1
>
> x <- c(1, 0, 0)
> x
[1] 1 0 0
> y <- c(1, 2, 0)
> y
[1] 1 2 0
> z <- c(1, 2, 5)
> z
[1] 1 2 5
>
> # Matrices:
>
> cbind(x, y, z)
     x y z
[1,] 1 1 1
[2,] 0 2 2
[3,] 0 0 5
>
> rbind(x, y, z)
  [,1] [,2] [,3]
x    1    0    0
y    1    2    0
z    1    2    5
>
> mymat <- rbind(x, y, z)
> mymat
  [,1] [,2] [,3]
x    1    0    0
y    1    2    0
z    1    2    5
>
> t(mymat)
     x y z
[1,] 1 1 1
[2,] 0 2 2
[3,] 0 0 5
>
> # Functions that work on vectors and matrices:
>
> apply(mymat, 2, mean)
[1] 1.000000 1.333333 1.666667
>
> apply(mymat, 1, mean)
        x y        z
 0.3333333 1 2.666667
>
> t(apply(mymat, 1, sort))
  [,1] [,2] [,3]
x    0    0    1
y    0    1    2
```

```
z    1    2    5
>
> apply(mymat, 2, sort)
  [,1] [,2] [,3]
x    1    0    0
y    1    2    0
z    1    2    5
>
> # Exactly the same as the original, as mymat was already sorted
> # by columns.
>
> sweep(mymat, 2, apply(mymat, 2, mean))
  [,1]       [,2]      [,3]
x    0 -1.3333333 -1.666667
y    0  0.6666667 -1.666667
z    0  0.6666667  3.333333
>
> y + z
[1] 2 4 5
>
> y - z
[1]  0  0 -5
>
> y * z
[1] 1 4 0
>
> mymat %*% solve(mymat)
  [,1] [,2]          [,3]
x    1    0 -4.532467e-17
y    0    1  2.266233e-17
z    0    0  1.000000e+00
>
> # The help files will tell you that the solve() function will
> # invert a matrix, so multiplying mymat by its inverse should
> # give the identity matrix and the result above is very close.
>
> outer(y, z)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    2    4    6
[3,]    0    0    0
> y %o% z
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    2    4    6
[3,]    0    0    0
>
> # Both functions give the outer product (as opposed to the
> # scalar product) of the two vectors.
>
> z[3] <- 3
> z
[1] 1 2 3
>
> mymat[3, 3] <- 3
> mymat
  [,1] [,2] [,3]
x    1    0    0
y    1    2    0
z    1    2    3
```

```
>
>
> # Q5
>
> women.mat[ ,1]
 [1] 169.6 166.8 157.1 181.1 158.4 165.6 166.7 156.5 168.1 165.3
> women.mat[ ,2]
 [1] 71.2 58.2 56.0 64.5 53.0 52.4 56.8 49.2 55.6 77.8
>
> women.lst[[1]]
 [1] 169.6 166.8 157.1 181.1 158.4 165.6 166.7 156.5 168.1 165.3
> women.lst[[2]]
 [1] 71.2 58.2 56.0 64.5 53.0 52.4 56.8 49.2 55.6 77.8
>
> women.lst$height
 [1] 169.6 166.8 157.1 181.1 158.4 165.6 166.7 156.5 168.1 165.3
> women.lst$weight
 [1] 71.2 58.2 56.0 64.5 53.0 52.4 56.8 49.2 55.6 77.8
>
> women.df[ ,1]
 [1] 169.6 166.8 157.1 181.1 158.4 165.6 166.7 156.5 168.1 165.3
> women.df[ ,2]
 [1] 71.2 58.2 56.0 64.5 53.0 52.4 56.8 49.2 55.6 77.8
>
> women.df$height
 [1] 169.6 166.8 157.1 181.1 158.4 165.6 166.7 156.5 168.1 165.3
> women.df$weight
 [1] 71.2 58.2 56.0 64.5 53.0 52.4 56.8 49.2 55.6 77.8
>
> attach(women.df)
> height
     1     2     3     4     5     6     7     8     9    10
 169.6 166.8 157.1 181.1 158.4 165.6 166.7 156.5 168.1 165.3
> weight
    1    2  3    4  5    6    7    8    9   10
 71.2 58.2 56 64.5 53 52.4 56.8 49.2 55.6 77.8
>
>
> # Q6
>
> stem(height)

N = 10   Median = 166.15
Quartiles = 158.4, 168.1

Decimal point is 1 place to the right of the colon

   15 : 678
   16 : 56778
   17 : 0
   18 : 1

> stem(weight)

N = 10   Median = 56.4
Quartiles = 53, 64.5

Decimal point is 1 place to the right of the colon

   4 : 9
```

```
   5 : 236678
   6 : 4
   7 : 18

>
> # The remaining commands are graphics commands, which you should
> # try out for yourself.
>
> hist(height)
> hist(weight)
> boxplot(height, weight)
> plot(height, weight)
> plot(height, weight, type="l")
> plot(women.mat)
> plot(women.lst)
Error in plot.xy("plot"): Cannot find x and y in list
Dumped
> plot(women.df)
>
> # Not all the above commands produce a sensible plot!
>
> plot(height, weight)
> abline(lm(weight ~ height))
>
> plot(co2)
> lines(smooth(co2), lty=2)
> q()
```