Lecture 2, R fundamentals

Most slides courtesy of Vivian Lew and Roger Peng

R is a language comprised of expressions

- Important types of expressions are:
  - Constants (e.g., 2)
  - Arithmetic expressions (e.g., 2+2)
  - Function calls (e.g., mean(x) , exp(x))
  - Assignments (e.g., x <- c(1, 2, 3))

Recap: RStudio

- We use scripts to store R commands
- We can highlight and run a script by pressing Run or use Ctrl+Enter(Windows) , Command+Enter(Mac)
- The console echoes back and shows results
- History tab reveals the most recent submitted commands
- Help tab can be used to tell us about the R language

Everything in R is an object

- An object is storage space with an associated name.
- EVERYTHING in R is stored in an object.
- Anything that is to be used again: data, variables, functions, absolutely anything, are stored in computer memory as NAMED objects -Objects have attributes.

Example

```
test <- 2+2
test
```

```
[1] 4
```

```
> .Last.value
[1] 4
```

- The .Last.value command gets the value returned by the most recent R command.
- test and .Last.value are named objects

Everything in R is an object

x<-3

- We create R objects using a a "gets". Its symbol is less than sign followed by a dash <- and we would read the above as "object x gets 3"
- Yes, you can use an equal sign,i.e. **x = 3**, instead of a "gets"

```
> 5
> .Last.value
[1] 5
```

- When we typed 5, R created an object called .Last.value.
- We can call it back up because we know its name.
- But typically, we use a "gets" to name and create our own R objects.

How to name objects in R?

- R is case sensitive so test is different from Test.
- R object names MUST start with a period OR a letter (and nothing else), so .Last.value is a valid R object name.
- R object names cannot have spaces (use a period or underscore instead), so .Last value is NOT a valid name.
- Periods may be found anywhere in an R name, but the underscore cannot start an R name, so _Last.value is NOT a valid name.
- No quotes around object names, so ".Last.value" is NOT a valid name.

Functions do the work in R

- Functions are a special type of R object. They exist to do something. They typically take arguments and produce a result by executing a set of operations (which are usually other functions).
- Functions have the basic form: functionname(argument), so a function name, a set of parentheses and an argument. Here is a simple example

```
help(exp)
```

-But sometimes, functions don't need arguments

```
getwd()
```

Functions do the work in R

- Sometimes functions have complex arguments:
  - plot(x <- sort(rnorm(50)),type="s",main="A plot of x, stair steps")
  - R has a set of built-in functions available for our use.
  - However, any user can create new R functions or modify existing functions. We will cover this later.

Functions do the work in R- Example

- exp( ) computes the exponential function using Euler's number.
- Notice
  - the parentheses
  - can operate on a constant
  - can operate on other R objects
  - can be part of a larger arithmetic expression
  - can have other functions nested in them

Example

```
> exp(1) #operates on constant
[1] 2.718282

> exp(test) #operates on the R object named test
[1] 54.59815

> exp(1)^9 # functions can be part of arithmatic operations
[1] 8103.084

> exp(rnorm(5))#can also have other functions nested insideof them
[1] 1.4237904 1.4222555 1.5741567 0.5730742 3.3567779
```

Data types in R: Vectors

- A vector is the simplest R data type.
- There are no scalars in R, a single value is a vector of length 1 (that is, [1])
- R functions operate on vectors; data is basically stored as vectors or sets of vectors in R.
- There are two different kinds: atomic vectors and lists (generic vectors)
- Atomic vectors can be thought of as arrays of a single kind of value (integers, logical, characters, etc.)
- Generic vectors (AKA Lists) may contain any type of data.

Creating Vectors

-The **c()** function can be used to create vectors of objects, "c" means "concatenate"

```
x<- c(0.5, 0.6)       #numeric
x<- c(TRUE, FALSE)    #logical
x<- c("a", "b", "c") #character
x<- c(2L,4L)          #integer
```

-R by default stores numbers as double. Putting capital 'L' after an integer forces it to be stored as an integer.

```
typeof(c(2,4))
```

```
[1] "double"
```

```
typeof(c(2L,4L))
```

```
[1] "integer"
```

Creating Vectors (Con't)

- Using colon: generates a sequence from:to in steps of 1 or -1

```
1:10   #ascending
10:1   #descending
```

- Using seq(...):Typically wants 3 numeric values, a "from", a "to" and a "by".

```
> seq(1, 10, 2)   #ascending
[1] 1 3 5 7 9
> seq(10, 1, -2) #descending
[1] 10  8  6  4  2
```

Creating Vectors (Con't)

- Using vector(mode,length) function:e.g

```
vector("logical", length = 8)
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
vector("numeric", length = 8)
```

```
## [1] 0 0 0 0 0 0 0 0
```

- Using rep function.

```
rep(c(0,1),5)
```

```
 [1] 0 1 0 1 0 1 0 1 0 1
```

Mixing Objects

- What about the following?

```
> c(1.7, "a")  #character
[1] "1.7" "a"
> c(TRUE, 2)  # numeric
[1] 1 2
> c("a", TRUE) # character
[1] "a"    "TRUE"
```

- When different objects are mixed in a vector, coercion occurs so that every element in the vector is of the same class.
- The output vector type is determined by the highest type of the components of the vector. logical $<$ numeric $<$character.

Explicit Coercion

- Objects can be explicitly coerced from one class to another using the as.* functions, if available.

```
> x<- c(0,1,2)
> typeof(x)
[1] "double"
> as.integer(x)
[1] 0 1 2
> as.logical(x)
[1] FALSE  TRUE  TRUE
> as.character(x)
[1] "0" "1" "2"
```

Explicit Coercion

- Nonsensical coercion results in NAs.

```
> x <- c("a", "b", "c")
> as.numeric(x)
[1]NA NA NA
Warning message:
NAs introduced by coercion
> as.logical(x)
[1]NA NA NA
> as.complex(x)
[1]NA NA NA
Warning message:
NAs introduced by coercion
```

Indexing - the key to unlocking data in R

- Recall that a vector in R is a numbered sequence of elements (which typically are numeric, character, or logical).
- One of the most important operations in R is accessing individual elements or subsets through indexing operations.
- Subsets of the elements of a vector may be selected by appending to the name of the vector an index vector in square brackets.
- The format is vector1[vector2], with the result that we select those elements of vector1 whose indices are given in vector2. Vector2 is an index vector

Example

```
> rivers #built-in vector river
[1]  735  320  325  392  524  450 1459  135  465  600  330  336  280  315  870
 [16]  906  202  329  290 1000  600  505 1450  840 1243  890  350  407  286  280
 [31]  525  720  390  250  327  230  265  850  210  630  260  230  360  730  600
 [46]  306  390  420  291  710  340  217  281  352  259  250  470  680  570  350
 [61]  300  560  900  625  332 2348 1171 3710 2315 2533  780  280  410  460  260
 [76]  255  431  350  760  618  338  981 1306  500  696  605  250  411 1054  735
 [91]  233  435  490  310  460  383  375 1270  545  445 1885  380  300  380  377
[106]  425  276  210  800  420  350  360  538 1100 1205  314  237  610  360  540
[121] 1038  424  310  300  444  301  268  620  215  652  900  525  246  360  529
[136]  500  720  270  430  671 1770

> rivers[28]
[1] 407
> rivers[132:137]
[1] 525 246 360 529 500 720
> rivers[c(1,3,5,7)]
[1]  735  325  524 1459
rivers[seq(5,120,10)]
[1]  524  870 1243  327  600  259  332  260  696  460  377 1205
> rivers[rivers > 2000]
[1] 2348 3710 2315 2533
> which(rivers>2000) #This is different, understand the difference
[1] 66 68 69 70
```

Data types in R: Matrices

- Matrices are vectors with a dimension attribute. The dimension attribute is itself an integer vector of length 2 (nrow, ncol).i.e. the number of rows and the number of columns.
- A matrix, like an atomic vector, may only have one type of data (e.g., all numeric, all character, all logical)
- The rows of a matrix must all be the same size
- The columns of a matrix must all be the same size
- But the size of the rows are not necessarily equal to the size of the columns
- And like vectors, there are many ways to create matrices

Matrices (Con't)

- Matrices are filled *column-wise*, so entries can be thought of starting in the "upper left" corner and running down the columns.

```
> m <- matrix(1:6, nrow = 2, ncol = 3)
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

- If we want to fill a matrix *row-wise*, we need to set byrow = TRUE. The default is FALSE.

```
> m <- matrix(1:6, nrow = 2, ncol = 3, byrow = TRUE)
> m
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Matrices (Con't)

- Matrices can also be created directly from vectors by adding a dimension attribute.

```
> m<- 1:10
> m
 [1]  1  2  3  4  5  6  7  8  9 10
> dim(m) <- c(2, 5)
> m
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

Matrices (Con't)

- Matrices can be created by column-binding or row-binding with cbind() and rbind().

```
> x<-1:3
> y<-10:12
> cbind(x, y)
     x  y
[1,] 1 10
[2,] 2 11
[3,] 3 12
> rbind(x, y)
  [,1] [,2] [,3]
x    1    2    3
y   10   11   12
```

Examples

```
> a<- 1:15
> # matrix function needs either the number of rows or
> #the number of columns
> matrix(a,nrow=5)
     [,1] [,2] [,3]
[1,]    1    6   11
[2,]    2    7   12
[3,]    3    8   13
[4,]    4    9   14
[5,]    5   10   15
> matrix(a,ncol=5)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    4    7   10   13
[2,]    2    5    8   11   14
[3,]    3    6    9   12   15

> matrix(c("a","A","b","B"), nrow=2, byrow=TRUE)
     [,1] [,2]
[1,] "a"  "A"
[2,] "b"  "B"
> matrix(c(TRUE,TRUE,FALSE,TRUE), nrow=2)
     [,1]  [,2]
[1,] TRUE FALSE
[2,] TRUE  TRUE
```

Example: Creating a matrix and indexing

```
> BOD2<- as.matrix(BOD) #as.matrix forces BOD to become a matrix
> BOD2
     Time demand
[1,]    1    8.3
[2,]    2   10.3
[3,]    3   19.0
[4,]    4   16.0
[5,]    5   15.6
[6,]    7   19.8
> dim(BOD2)
[1] 6 2
> length(BOD2)
[1] 12
> nrow(BOD2)
[1] 6
> ncol(BOD2)
[1] 2
> BOD2[9] #If we wish to extract a single cell from matrix
[1] 19
> BOD2[3,2] #clearer
demand
    19
> # If we wish to extract a specific row
> BOD2[3,] # gives row 3, all columns
  Time demand
     3     19
> # If we wish to extract a specific column
> BOD2[,2]
[1]  8.3 10.3 19.0 16.0 15.6 19.8
> BOD2[4:6, 1:2] #rows 4,5,6 of columns 1 and 2 (here same as BOD2[4:6,])
     Time demand
[1,]    4   16.0
[2,]    5   15.6
[3,]    7   19.8
#Try BOD2[c(4,6), 1:2]
```

Data types in R: Lists

- Lists are a special type of vector (generic vector) that can contain elements of different classes.
- A list allows you to gather a variety of (possibly unrelated) objects under one name.

```
> mylist<- list(c(2,3,4), c(TRUE,FALSE), c("A","B","C"))
> mylist
[[1]]
[1] 2 3 4

[[2]]
[1]  TRUE FALSE

[[3]]
[1] "A" "B" "C"
```

Identify elements of a list using the [[]] convention.

```
> mylist[[1]] # first component of the list
[1] 2 3 4
> mylist[[3]] #3rd element of the list
[1] "A" "B" "C"
> mylist<- list(Jane = c(2,3,4), Dave = c(TRUE,FALSE), Mary = c("A","B","C"))
> mylist
$Jane
[1] 2 3 4

$Dave
[1]  TRUE FALSE

$Mary
[1] "A" "B" "C"

> mylist$Jane
[1] 2 3 4
```

Data types in R: Data Frames

- A data frame is a rectangular list.
- In other words, they are represented as a special type of list where every element of the list has to have the same length.
- A data frame is like a matrix with columns possibly of differing modes and attributes (e.g., numeric columns, character columns, logical columns).
- Data frames typically have "observations" in the rows and "variables" in columns in a rectangular matrix-like structure (equal length rows and equal length columns).

Data Frames - Example

```
> x <- data.frame(foo = 1:4, bar = c(T, T, F, F))
> x
  foo   bar
1   1  TRUE
2   2  TRUE
3   3 FALSE
4   4 FALSE
> nrow(x)
[1] 4
> ncol(x)
[1] 2
```

Data frames (Con't)

- Data frames have a row.names( ) attribute which labels the observations.
- Data frames have a names( ) attribute which labels the variables.
- Data frame rows and columns are accessed using matrix indexing rules OR list indexing rules.

Data frames -Example

```
> x
  foo   bar
1   1  TRUE
2   2  TRUE
3   3 FALSE
4   4 FALSE
> row.names(x)
[1] "1" "2" "3" "4"
> names(x)
[1] "foo" "bar"
> names(x) <- c("Dave" , "Jane")
> x
  Dave  Jane
1    1  TRUE
2    2  TRUE
3    3 FALSE
4    4 FALSE
> x[1,2]
[1] TRUE
> x[,2] # like a matrix extracts all rows, column 2
[1]  TRUE  TRUE FALSE FALSE
> x[[1]] #extracts a vector from a list
[1] 1 2 3 4
> x$Jane # extracts a named column
[1]  TRUE  TRUE FALSE FALSE
```

Swirl Package

- For some of the problem sets we'll be using the swirl software package for R in order to illustrate some key concepts. The swirl package turns the R console into an interactive learning environment.
  - Install swirl using: **install.packages("swirl")**
  - Load Swirl using: **library(swirl)**
  - Install the R Progroamming course using: **install_from_swirl("R Programming")**
  - Start swirl and complete the lessons using: **swirl()**