Introduction to R

Intro to R

- In this lab, we will introduce some simple R commands. The best way to
- learn a new language is to try out the commands. R can be downloaded from
- http://cran.r-project.org/

Basic Commands

- R uses *functions* to perform operations. Questions we might ask:
- To run a function called funcname,
- we type funcname(input1, input2), where the inputs (or *arguments*) input1 and input2 tell R how to run the function.
- A function can have any number of inputs.
- to create a vector of numbers, we use the function c() (for *concatenate*).

Basic Commands

- The following command instructs R to join together the numbers

1, 3, 2, and 5, and to save them as a *vector* named x. When we type x, it gives us back the vector. Note that the > is the prompt.

```
> x <- c(1,3,2,5)
> x
[1] 1 3 2 5
```

- We can also save things using = rather than <-:

```
> x = c(1,6,2)
> x
[1] 1 6 2
> y = c(1,4,3)
```

Basic Commands

- Hitting the *up* arrow multiple times will display the previous commands, which can then be edited. In addition, typing ?funcname will always cause R to open a new help file window with additional information about the function funcname.
- We can tell R to add two sets of numbers together. It will then add the first number from x to the first number from y, and so on. However, x and y should be the same length. We can check their length using the length() function.

```
> length (x)
[1] 3
> length (y)
[1] 3
> x+y
[1]  2  10   5
```

Basic Commands

- The ls() function allows us to look at a list of all of the objects, such as data and functions, that we have saved so far.
- The rm() function can be used to delete any that we don't want.

```
> ls()
[1] "x" "y"
> rm(x,y)
> ls()
character (0)
```

- It's also possible to remove all objects at once:

```
> rm(list=ls())
```

Basic Commands

- The matrix() function can be used to create a matrix of numbers. Before we use the matrix() function, we can learn more about it:

```
> ?matrix
```

- The help file reveals that the matrix() function takes a number of inputs, but for now we focus on the first three: the data (the entries in the matrix), the number of rows, and the number of columns. First, we create a simple matrix.

```
> x=matrix (data=c(1,2,3,4) , nrow=2, ncol =2)
> x
     [,1] [,2]
[1,]  1    3
[2,]  2    4
```

- Note that we could just as well omit typing data=, nrow=, and ncol= in the matrix() command above: that is, we could just type

```
> x=matrix (c(1,2,3,4) ,2,2)
```

Basic Commands

Alternatively, the byrow=TRUE option can be used to populate the matrix in order of the rows.

```
> matrix (c(1,2,3,4) ,2,2,byrow =TRUE)
     [,1] [,2]
[1,]    1    2
[2,]    3    4
```

- Notice that in the above command we did not assign the matrix to a value such as x. In this case the matrix is printed to the screen but is not saved for future calculations.

Basic Commands

- The sqrt() function returns the square root of each sqrt() element of a vector or matrix. The command x^2 raises each element of x to the power 2; any powers are possible, including fractional or negative powers.

```
> sqrt(x)
     [,1]  [,2]
[1,] 1.00  1.73
[2,] 1.41  2.00
> x^2
     [,1] [,2]
[1,]    1    9
[2,]    4   16
```

Basic Commands

- The rnorm() function generates a vector of random normal variables, with first argument n the sample size. Each time we call this function, we will get a different answer. Here we create two correlated sets of numbers, x and y, and use the cor() function to compute the correlation between them.

```
> x=rnorm (50)
> y=x+rnorm (50, mean=50, sd=.1)
> cor(x,y)
[1]  0.995
```

- By default, rnorm() creates standard normal random variables with a mean of 0 and a standard deviation of 1. However, the mean and standard deviation can be altered using the mean and sd arguments

Basic Commands

- we can use the set.seed() function to to reproduce a set of random numbers. The set.seed() function takes an (arbitrary) integer argument.

```
> set.seed (1303)
> rnorm (50)
[1] -1.1440  1.3421  2.1854  0.5364  0.0632  0.5022  -0.0004
. . .
```

- We use set.seed() throughout the labs whenever we perform calculations involving random quantities.

Basic Commands

- The mean() and var() functions can be used to compute the mean and variance of a vector of numbers.
- Applying sqrt() to the output of var() will give the standard deviation. Or we can simply use the sd() function.

```
> set.seed (3)
> y=rnorm (100)
> mean(y)
[1]  0.0110
> Var(y)
[1]  0.7329
> sqrt(var(y))
[1]  0.8561
> sd(y)
[1]  0.8561
```

Graphics

- The plot() function is the primary way to plot data in R.
- There are many additional options that can be passed in to the plot()
- passing in the argument xlab will result in a label on the *x*-axis
- To find out more information about the plot() function,
- type ?plot.

```
> x=rnorm (100)
> y=rnorm (100)
> plot(x,y)
> plot(x,y,xlab=" this is the x-axis",ylab=" this is the y-axis",
main=" Plot of X vs Y")
```

Graphics

We will often want to save the output of an R plot.

To create a pdf, we use the pdf() function, and to create a jpeg, we use the jpeg() function.

```
> pdf (" Figure .pdf ")
> plot(x,y,col =" green ")
> dev.off ()
null device
```

- The function dev.off() indicates to R that we are done creating the plot.
- Alternatively, we can simply copy the plot window and paste it into an appropriate file type, such as a Word document.

Graphics

- The function seq() can be used to create a sequence of numbers. For instance, seq(a,b) makes a vector of integers between a and b.
- There are many other options: for instance, seq(0,1,length=10) makes a sequence of 10 numbers that are equally spaced between 0 and 1.
- Typing 3:11 is a shorthand for seq(3,11) for integer arguments.

```
> x=seq (1 ,10)
> x
[1] 1 2 3 4 5 6 7 8 9 10
> x=1:10
> x
[1] 1 2 3 4 5 6 7 8 9 10
> x=seq(-pi ,pi ,length =50)
```

Graphics

- The contour() function produces a *contour plot* in order to represent three-dimensional data; it is like a topographical map. It takes three arguments:
- 1. A vector of the x values (the first dimension),
- 2. A vector of the y values (the second dimension), and
- 3. A matrix whose elements correspond to the z value (the third dimension)
- for each pair of (x,y) coordinates.

```
> y=x
> f=outer(x,y,function (x,y)cos(y)/(1+x^2))
> contour (x,y,f)
> contour (x,y,f,nlevels =45, add=T)
fa=(f-t(f))/2
> contour (x,y,fa,nlevels =15)
```

Graphics

- The image() function works the same way as contour(), except that it produces a color-coded plot whose colors depend on the z value. This is known as a *heatmap*, and is sometimes used to plot temperature in weather forecasts. Alternatively, persp() can be used to produce a three-dimensional plot. The arguments theta and phi control the angles at which the plot is viewed.

```
> image(x,y,fa)
> persp(x,y,fa)
> persp(x,y,fa ,theta =30)
> persp(x,y,fa ,theta =30, phi =20)
> persp(x,y,fa ,theta =30, phi =70)
> persp(x,y,fa ,theta =30, phi =40)
```

Indexing Data

- We often wish to examine part of a set of data. Suppose that our data is stored in the matrix A.

```
> A=matrix (1:16 ,4 ,4)
> A
     [,1]  [,2]  [,3]  [,4]
[1,]  1     5     9     13
[2,]  2     6     10    14
[3,]  3     7     11    15
[4,]  4     8     12    16
```

- Then, typing

```
> A[2,3]
[1] 10
```

will select the element corresponding to the second row and the third column.

- > A[

Indexing Data

- The first number after the open-bracket symbol [always refers to the row, and the second number always refers to the column. We can also select multiple rows and columns at a time, by providing vectors as the indices.

```
> A[c(1,3) ,c(2,4) ]
     [,1]  [,2]
[1,]    5    13
[2,]    7    15
> A[1:3 ,2:4]
     [,1] [,2] [,3]
[1,]    5    9   13
[2,]    6   10   14
[3,]    7   11   15
> A[1:2 ,]
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
> A[ ,1:2]
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
```

Indexing Data

- The use of a negative sign - in the index tells R to keep all rows or columns except those indicated in the index.

```
> A[-c(1,3) ,]
     [,1] [,2] [,3] [,4]
[1,]    2    6   10   14
[2,]    4    8   12   16
> A[-c(1,3) ,-c(1,3,4)]
[1] 6 8
```

- The dim() function outputs the number of rows followed by the number of columns of a given matrix.

```
> dim(A)
[1] 4 4
```

Loading Data

- For most analyses, the first step involves importing a data set into R. The read.table() function is one of the primary ways to do this.
- We use the function write.table() to export data.
- Before attempting to load a data set, we must make sure that R knows to search for the data in the proper directory. (OS dependent)
- Once the data has been loaded, the fix() function can be used to view it in a spreadsheet like window.

```
> Auto=read.table ("Auto.data ")
>  fix(Auto)
```

- This particular data set has not been loaded correctly, because R has assumed that the variable names are part of the data and so has included them in the first row. The data set also includes a number of missing observations, indicated by a question mark ?.

Loading Data

- Excel is a common-format data storage program. An easy way to load such data into R is to save it as a csv (comma separated value) file and then use the read.csv() function to load it in.

```
> Auto=read.csv (" Auto.csv", header =T,na.strings ="?")
> fix(Auto)
> dim(Auto)
[1] 397 9
> Auto [1:4 ,]
```

- The dim() function tells us that the data has 397 observations, or rows, and nine variables, or columns. There are various ways to deal with the missing data. In this case, only five of the rows contain missing observations, and so we choose to use the na.omit() function to simply remove these rows.

```
> Auto=na.omit(Auto)
> dim(Auto)
[1] 392 9
```

Loading Data

- Once the data are loaded correctly, we can use names() to check the variable names.

```
> names(Auto)
[1] "mpg " "cylinders " " displacement" "horsepower "
[5] "weight " " acceleration" "year" "origin "
[9] "name“
```

Additional Functions

- We can use the plot() function to produce *scatterplots* of the quantitative variables, but R does not know to look in the Auto data set for those variables and will return an error.

```
> plot(cylinders , mpg)
Error in plot(cylinders , mpg) : object 'cylinders ' not found
```

- To refer to a variable, we must type the data set and the variable name joined with a $ symbol. Alternatively, we can use the attach() function in order to tell R to make the variables in this data frame available by name.

```
> plot(Auto$cylinders , Auto$mpg )
> attach (Auto)
> plot(cylinders , mpg)
```

- The cylinders variable is stored as a numeric vector, so R has treated it as quantitative. However, since there are only a small number of possible values for cylinders, one may prefer to treat it as a qualitative variable.
- The as.factor() function converts quantitative variables into qualitative variables.

```
> cylinders =as.factor (cylinders )
```

Additional Functions

- If the variable plotted on the *x*-axis is categorial, then *boxplots* will automatically be produced by the plot() function. As usual, a number of options can be specified in order to customize the plots.

```
> plot(cylinders , mpg)
> plot(cylinders , mpg , col ="red ")
> plot(cylinders , mpg , col ="red", varwidth =T)
> plot(cylinders , mpg , col ="red", varwidth =T,horizontal =T)
> plot(cylinders , mpg , col ="red", varwidth =T, xlab=" cylinders ",
ylab ="MPG ")
```

- The hist() function can be used to plot a *histogram*. Note that col=2
- histogram has the same effect as col="red".

```
> hist(mpg)
> hist(mpg ,col =2)
> hist(mpg ,col =2, breaks =15)
```

Additional Functions

- The pairs() function creates a *scatterplot matrix* i.e. a scatterplot for every pair of variables for any given data set. We can also produce scatterplots matrix for just a subset of the variables.

```
> pairs(Auto)
> pairs( ~ mpg + displacement + horsepower + weight + acceleration , Auto)
```

- In conjunction with the plot() function, identify() provides a useful interactive method for identifying the value for a particular variable for points on a plot.
- We pass in three arguments to identify(): the *x*-axis variable, the *y*-axis variable, and the variable whose values we would like to see printed for each point. Then clicking on a given point in the plot will cause R to print the value of the variable of interest.
- The numbers printed under the identify() function correspond to the rows for the selected points.

```
> plot(horsepower ,mpg)
> identify (horsepower ,mpg ,name)
```

Additional Functions

- The summary() function produces a numerical summary of each variable in a particular data set.

> summary (Auto)

```
> summary(Auto)
      mpg          cylinders      displacement
 Min.   : 9.00   Min.   :3.000   Min.   : 68.0
 1st Qu.:17.00   1st Qu.:4.000   1st Qu.:105.0
 Median :22.75   Median :4.000   Median :151.0
 Mean   :23.45   Mean   :5.472   Mean   :194.4
 3rd Qu.:29.00   3rd Qu.:8.000   3rd Qu.:275.8
 Max.   :46.60   Max.   :8.000   Max.   :455.0

   horsepower       weight      acceleration
 Min.   : 46.0   Min.   :1613   Min.   : 8.00
 1st Qu.: 75.0   1st Qu.:2225   1st Qu.:13.78
 Median : 93.5   Median :2804   Median :15.50
 Mean   :104.5   Mean   :2978   Mean   :15.54
 3rd Qu.:126.0   3rd Qu.:3615   3rd Qu.:17.02
 Max.   :230.0   Max.   :5140   Max.   :24.80

      year           origin                      name
 Min.   :70.00   Min.   :1.000   amc matador       :  5
 1st Qu.:73.00   1st Qu.:1.000   ford pinto        :  5
 Median :76.00   Median :1.000   toyota corolla    :  5
 Mean   :75.98   Mean   :1.577   amc gremlin       :  4
 3rd Qu.:79.00   3rd Qu.:2.000   amc hornet        :  4
 Max.   :82.00   Max.   :3.000   chevrolet chevette:  4
                                 (Other)           :365
```

Additional Functions

- For qualitative variables such as name, R will list the number of observations that fall in each category. We can also produce a summary of just a single variable.

```
> summary (mpg)
Min.   1st Qu.   Median   Mean    3rd Qu.   Max .
9.00   17.00     22.75    23.45    29.00    46.60
```

- Once we have finished using R, we type q() in order to shut it down, or quit.
- When exiting R, we have the option to save the current *workspace* so that all objects (such as data sets) that we have created in this R session will be available next time.
- Before exiting R, we may want to save a record of all of the commands that we typed in the most recent session; this can be accomplished using the savehistory() function. Next time we enter R, we can load that history using the loadhistory() function.