

# ENGR 113: Lab 1

## Introduction to R

### Lab Objectives

To demonstrate basic operations in R.

### Instructions

As you read through, you will see things in typewriter font enclosed in curly braces. You are meant to actually type in those things (but not the braces themselves), and hit enter, so that you are following along and seeing how it all works.

### Operating System Choices

Please note that these labs are written assuming you are using R from a unix machine in a CATS lab. As R is open-source, you can get either the source code or pre-compiled binaries for a variety of platforms online (<http://lib.stat.cmu.edu/R/CRAN/>) and you can use them on whatever machine you feel most comfortable with. However, technical support will only be provided for CATS unix versions. The official home page for R is <http://www.r-project.org/>, and you can find additional information there for all versions.

### Starting R

First open up a shell terminal window (using an OpenWindows environment, you can right-click on the background, go to the tools menu, and choose shell tool; using CDE, you can right click, go to tools, and choose terminal, or from the file manager, go to the file menu and choose open terminal). In the shell window, enter `{R}`. R will start up and you should get something remotely resembling:

```
R : Copyright 2002, The R Development Core Team
Version 1.6.0 (2002-10-01)
```

```
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
```

```
R is a collaborative project with many contributors.
Type 'contributors()' for more information.
```

```
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for a HTML browser interface to help.
Type 'q()' to quit R.
```

```
>
```

Note that the `>` is the prompt within R. This is where you enter commands. If you ever get the `+` prompt, this means that you didn't finish the command on the previous line, and R wants you to complete it (for example, you may have left off a closing parenthesis).

## Quitting R

To quit R, the command is `q()`

## Getting Help

R has on-line help available. First start netscape. Second, go back to the R window and enter `help.start()`

## R and S

R was developed as an open-source alternative to the commercial package S-Plus (which was also originally free). As I have used S-Plus at other places, if I forget and refer to "S", you should just assume I mean "R".

## Simple Arithmetic

R will do all standard arithmetic operations, and it knows the correct order of operations. For example, enter `{1+2*3}` and you should get the correct answer of 7.

## Assigning Objects

You may want to create a variable or assign the result of a function or operation to another object that you can keep around for future use. To do this, you use the assignment operator `<-` (which is a less-than symbol immediately followed by a hyphen). You can also use the notational shortcut of an underscore (`_`) instead of the "arrow".

1. For example, suppose you want `x` to have the value 6. You would enter `{x <- 6}`. It is usually helpful to use descriptive names, such as `{total.revenue <- 18.42}`.
2. To copy an object, we can just assign it to another name: `{expenses <- x}`
3. You can assign the result of an operation to an object:  
`{net.profit <- total.revenue - expenses}`
4. To see the result, just enter the name of the object: `{net.profit}`

## Note on naming of objects

Because the underscore has the special meaning of assignment, *do not use an underscore in any object or function names*. You may use a period within a name. Also, the letters `c`, `q`, and `t` are all built-in commands, so R will be unhappy if you try to name an object with one of these letters (but it will let you do it, and then it may cause all sorts of problems later on).

## Commands

The structure of functions in R is that you enter the function name followed by parentheses, putting arguments (if any) in the parentheses. If the function doesn't require any arguments (such as `q()` to quit), you still need to include the parentheses so that R knows that it is supposed to run the function. If you type in the name of a function without the parentheses, it will give you the code for the function. Also note that all entries in R are case-sensitive.

## Removing Objects

As you create more and more objects, they will start to clutter up your workspace. To see what objects you have, use `ls()`. Objects that you no longer need should be discarded. Objects are removed with the function `rm()`, putting the name of the object in the parentheses, e.g., `{rm(x)}`. You can use `{ls()}` to verify that `x` has been deleted.

## Getting the Data

Most of the datasets we will use in this class are on the CD-ROM included in the back of the textbook. We will use the ASCII (plain text) versions of the data files. To access those, change to the CD-ROM directories. Then go to its ascii files subdirectory `content/datafi~1/asciid~1`. From there you can copy whichever data files you need over to your working directory. Please note that R will only find the datafiles that are in the same subdirectory as you are when you start R from the unix prompt. Note that all of the ascii-formatted files end in `.txt`

Alternately, I will try to put all of the relevant data files on the course web page at <http://www.soe.ucsc.edu/classes/engr113/Winter03/Data/data.html> You can download the files from there using your web browser of choice.

## Reading Data Into R

The first example of a dataset is the UTILITY data (see problem 2.12 on p. 57 of text; note that this dataset is also used on homework #1). The data are the electricity cost of a two-bedroom unit for July 2000 for a sample of 50 apartments.

If the dataset contains only one variable, as it does here, then the easiest way to read it into R is with the `scan()` function. Scan can take several arguments, but the key one is the name of the datafile. Please note that this datafile must be in the same directory as you started R in. To read in the utility data and store it in an object named "utility", enter `{utility <- scan("utility.txt")}` (if you named the file something other than `utility.txt` when you copied or downloaded it, put that name in there instead). Now enter `{utility}` to verify that it read in the data. You should get something like the following:

```
> utility
 [1]  96 171 202 178 147 102 153 197 127  82 157 185  90 116 172 111 148 213 130
[20] 165 141 149 206 175 123 128 144 168 109 167  95 163 150 154 130 143 187 166
[39] 139 149 108 119 183 151 114 135 191 137 129 158
```

Depending on the width of your window, the display might look a little different. Note that R has stored the data in a *vector* that is named `utility`. There are 50 observations, so there are 50 elements in this vector. The `[1]`, `[20]`, `[39]` on the left side are labels of which elements are

which in the vector (note: if your screen width is different, you may have different labels on the left), so for example, the 20th observation has value 165. You can access individual elements of a vector by using square brackets after the name of the vector, for example, to see what the 18th observation is, enter `{utility[18]}` . You can also specify a range by using a colon, e.g., enter `{utility[1:5]}` to get the first five values.

## Using Vectors

Many functions take a vector as their primary argument. For example, to find the average, or arithmetic mean, of the utility charges, enter `{mean(utility)}`. You can also perform arithmetic operations using vectors. Standard operations are done *element-wise*, for example, `{10+utility[1:5]}` will return `[1] 106 181 212 188 157`, and `{2*utility[1:5]}` returns `[1] 192 342 404 356 294`. If you actually want to do vector/matrix multiplication, you need to use `%*%`. (I'm happy to explain this in further detail if you are interested, but it is not important for now.)

To make your own vector, use the `c()` function (think *c* for combine or column or concatenate). For example, to get a vector with the values 1, 2, 3, and 4, enter `{x <- c(1,2,3,4)}`. This could also be done with the sequence notation, i.e., `{x <- c(1:5)}` (note that we are over-writing the previous value of `x` here, and this cannot be undone once you hit enter).

## Reading A Table Into R

Some data will contain more than one variable, and will be presented as a table, usually with observations along the rows and variables in columns. A dataset you will need for homework #1 is the TIGERS dataset of attendance at Detroit baseball home games in the 1997 season. This dataset has three columns (variables): game number (in order), temperature that day, and attendance at the game. In the homework problem, we will be interested in seeing if the temperature affects the attendance. In this lab, we will just practice with tables. To read in a table, we use the `read.table()` function. So to read in the file `tigers.txt`, enter `{tigers <- read.table("tigers.txt")}` . Now look at the data by just entering `{tigers}` . You should see four columns, with the first two being the same. R doesn't realize that the first variable is just an index, and R has added an index of its own. There are worse things in life than redundancy.

## Dataframes

What R has actually done is stored the data in an object called a *data.frame*, which is just a fancy matrix. If you scroll back up to the top of the data, you will see that R has labeled the columns as `V1`, `V2`, `V3`. You can also get the names using the `names()` function, i.e., `{names(tigers)}`. R doesn't know what else to call them, so it uses `V` for variable and then starts counting. You can stick with these defaults, or you can provide your own names. To change the names of the columns, you simply assign new names to the names object, for example, `{names(tigers) <- c("game","temp","attend")}`. Note that character strings (names, as opposed to numbers) must be enclosed in double quotes. You can also specify the variable names at the same time you read the table in by specifying the `col.names` argument, i.e., `tigers <- read.table("tigers.txt",col.names=c("game","temp","attend"))`

One advantage of a `data.frame` is that you can use the variables as separate vectors. For example, enter `{tigers$temp}` to see all of the daily temperatures. You can get the average temperature with `{mean(tigers$temp)}`. It can be a pain to keep typing in the name of the `data.frame`, so if you are only using one `data.frame`, you can tell R this with the `attach()` function. Enter `{attach(tigers)}` and then you can refer to its columns just by their own names, e.g., you can use `{mean(temp)}`.

## Graphics

Before you can make any plots, you need to open a graphical device (window). On most systems, this will be `X11`, although on some it could be `Motif`. On CATS, open the window in R by entering `{x11()}`. Now you are ready to plot.

Let's look at the temperature data. You can get a histogram with `{hist(temp)}`. What does the histogram tell you about daily temperatures? Does this match what you would expect in Detroit during baseball season (April through September)?

Since the data are in time order, we can plot the observations in the order they appear in the dataset to see how temperature changes over time. The `plot()` function will plot the observations in the order they appear if you just give it one argument, and will make a scatterplot (x vs. y) if you give it two arguments. Here, we'll just look at temperature by itself. Enter `{plot(temp)}`. You should see a strong pattern in the plot. Does this match what you would expect in Detroit during baseball season?

## Finishing Up

When you are done, it is good practice to remove any objects you have created that you won't need again later (use `ls()` to see what you have, and `rm()` to remove anything unnecessary, like `net.profit`). To quit from R, enter `{q()}`. This will also automatically close any graphics windows. When quitting, it will ask you if you want to **save workspace image?** `[y/n/c]` . If you want any of your created objects to be there the next time you run R, you need to enter `y` here, otherwise it won't save anything you created during this session. Of course, if you don't want to save what you did, then choosing `n` is an easy way to delete whatever temporary things you made without having to `rm` them individually.