From ST597: Statistical Computing notes (2011)

This extracts 3 sections from the S597 notes which cover both SAS and R. As such there may be a few references to parts of the SAS piece of the notes, which is not here.

Contents

1 Introduction to R

As with the treatment of SAS, this module only touches on the basics of R; reading, manipulating and describing data and running some basic statistical analyses. No attempt is made here to describe R in all its complexity or its overall logic/functionality. We simply note that the basic structure of R is built on working with objects, that functions operate on objects and functions often create other objects (which can then be arguments to other functions). Additional features of R, including graphics, one of R's strong points, will be discussed later. Also, there are many user defined packages that can be obtained.

If interested in just reading and the describing/analyzing data, you can look at just subsections 2 and 3 below and then jump to subsection 6 on describing and analyzing data. *When we get to the statistical analyses, we primarily rerun things done previously in SAS; mostly just showing the R code and output for the analyses, without repeating the earlier comments concerning the analyses.*

NOTE: It looks like there are different versions of double quotes here but that is a formatting thing in Latex. They are all the same.

1.1 Some basic in running R

Entering commands

Once R is started there is a command window, the R console window, with a > prompt. You can either

1. Enter commands directly, or cut and paste a set of command lines into the window from elsewhere.
2. Run a program (set of commands) that has been created elsewhere as a text file. This is done by saying source("filemane") where filename has both the path and file name where the commands can be found.

3. Use the R editor. You can also use the R editor to write and submit your commands. This is a good way to debug and use previously created commands. When you close this window you can save what is in the editor to a file.

   To open the editor go to file and choose "new script". You can then type command lines in this editor. Or if you have a file where you already have some R commands chose "open script") from the menu. If the cursor is located in a line of text and you hit ctrl-r (control key plus r key) it will submit that line to R console and execute it. If you highlight a block of lines and hit ctrl-r it will run that block of commands.

4. Use Tinn-R or R-studio (won't discuss here).

Scrolling through commands: You can recall commands using the up and down arrow key.

Saving your output: Without doing anything the output from the commands you run will show up in the console window. You can either cut and paste things from there to other files or under the file menu you *save to file*; The latter allows you to save the contents of the console window as a text file.

Another option is to use the *sink* command. If you type sink("filename") everything that would go to the console goes to filename (where as elsewhere, filename has both a path and file name, as needed). When the sink is on the output will not show up in the console window but just be directed to the chosen file.

The "equal" sign and assignment. In R, the $<-$ sign (note there is no space between $<$ and $-$), the assignment operator, and is used like an equal sign. It assigns whatever is generated by the expression on the right to the object on the left. Often, it can be replaced by = (although this is not true in S-Plus which is essentially the commercial version of R) but it has become fairly standard practice to use the assignment operator in defining quantities.

Graphics output: With no routing of it, graphics output will show up in the graph window. It can be exported from there as various types of files (postscript, pdf, etc.). (More on graphics later)

Help: There is interactive help available, either through the help menu or typing help(xxx). For example help(read.table) will give you information on the read.table function.

What's in the workspace? Typing *ls()* will list all of the dataframes and user defined functions that are in the workspace. You can clear everything from the workspace with *rm(list=ls())* and selected objects using rm(list = ...)

Listing what makes up an object: An object in the workspace can be described by typing *str(objectname)*

Listing an object Just typing an object's name will list that object.

Variable names are **case sensitive**, unlike SAS.

Comments: A line that begins with # is treated as a comment (non-executable)

1.2 Reading text files and referring to variables

External text data are usually read through read.table, read.csv or read.dlm. There are other read functions for specific types of data (SAS files, Stata files, etc.) and the function scan is sometimes used, but we don't use these now. Notice that in any of these if you have header=F then there cannot be a first line with variable names in the text file (or it will try to read it as data). You must have header=T if there is such a line.

To just enter small amounts of data directly into vectors, see Section 1.4.1.

- **read.table**

The basics of the read.table (there are other options) are given by

dname <− read.table("filename",sep = " ",header=T, na.strings=" ")

All that is required in the read.table arguments is the filename.

dname is the name of the R object. In SAS we would refer to this as a dataset. In R it is referred to as a **dataframe**.

If the sep = is omitted, the data is assumed to be space delimited. Otherwise sep = can be used to denote the delimiter (for comma delimited you can also use read.csv and for tab delimited, read.dlm).

header = T means the first line contains variable names. If header = F (which is the default for read.table if header = is not included) this means there is no first line of variable names. In this case the variables are assigned names of V1, V2, etc. In this case you can rename the variables using the *within* function as demonstrated below.

na.strings = "xx" indicates that xx denotes a missing value. The default in read.table is that missing values are indicated by NA. With space delimited data (the default in read.table) then, similar to SAS, there has to be some character for a missing value; it cannot be blank. (However, if you have delimited data with a delimiter not a blank then here and in read.csv and read.dlm either NA or a blank will be read as a missing value.) Use na.strings = "." when the . is a missing values. If you have a . for missing values and do not have the na.strings option then the variable will be defined as a character variable with . as one of the values.

- *read.csv("filename")*

  This will read a comma separated file. It assumes that the first line contains variable names (that is, the default is header = T) which must also be separated by a , .

- *read.dlm("filename")*

  This defaults to reading the data as tab delimited. As with read.csv the default is header=T and variable names are also separated by tabs.

Referring to variables.

Once you have a dataframe, if you do nothing else then variables need to be referred to using both the dataframe name combined with the variable name with a $ sign before the variable name.

The **attach(data)** function will make the variables in the dataframe data available in the workspace as objects that can be referred to by the name of the variable only. **NOTE: IF THE WORKSPACE ALREADY HAS A VARIABLE WITH THE SAME NAME, IT WILL NOT BE REPLACED.**. Using detach(data) will remove the variables from the workspace as individual items.

Note that the dataframe is similar to a matrix, a data matrix if you like, and you can print and do certain operations on elements, rows or columns as illustrated below.

Examples

Things between **** are comments that have been edited into the output file.

Reading data with no names in the file.

```
> turt<-read.table("g:/s597/data/days.dat")
> turt
   V1 V2 V3 V4 V5 V6 V7 V8 V9 V10
1  93  Q  1  F 12 88  0  0  0 165
2  93  Q  3  M 40  4  0 56  0 157
3  93  Q  4  F 41 20  0 39  0 164
```

```
 .....

27 94 LS 87  F 89  0  9  0  2 159



> str(turt) #show the contents of data frame turt
'data.frame':   27 obs. of  10 variables:
 $ V1 : int  93 93 93 93 93 93 93 93 93 93 ...
 $ V2 : Factor w/ 2 levels "LS","Q": 2 2 2 2 2 2 2 2 2 2 ...
 $ V3 : int  1 3 4 5 6 7 8 9 10 14 ...
 $ V4 : Factor w/ 2 levels "F","M": 1 2 1 2 1 2 2 1 2 1 ...
 $ V5 : int  12 40 41 34 26 32 37 32 39 31 ...
 $ V6 : int  88 4 20 57 74 12 48 68 3 35 ...
 $ V7 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ V8 : int  0 56 39 9 0 56 15 0 58 34 ...
 $ V9 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ V10: int  165 157 164 164 163 164 162 160 161 157 ...

> V1
Error: object "V1" not found
> $V1
Error: unexpected '$' in "$"

**********************
THE ABOVE SHOWS THAT YOU CAN'T JUST REFER TO THE VARIABLE NAME.
YOU CAN IF YOU DO AN attach.  SO, TO LIST VARIABLE V1 IN TURT
YOU CAN DO EITHER OF THE FOLLOWING
**********************

> turt$V1
 [1] 93 93 93 93 93 93 93 93 93 93 94 94 94 94 94 94 94 94 94 94 94 94 94 94 94 94 94

> attach(turt)
> V1
 [1] 93 93 93 93 93 93 93 93 93 93 94 94 94 94 94 94 94 94 94 94 94 94 94 94 94 94 94

*** You can create year, another name for V1 as below or just year<-V1 if used attach***

> year<-turt$V1
year
 [1] 93 93 93 93 93 93 93 93 93 93 94 94 94 94 94 94 94 94 94 94 94 94 94 94 94 94 94
[26] 94 94
```

Reading data with names in the first row

```
> turtleh <- read.table("g:/s597/data/days_h.dat", header=T)
> turtleh
   Year site turtle sex vp em fo up to tdays
```

```
1    93    Q      1   F 12 88  0  0  0   165
2    93    Q      3   M 40  4  0 56  0   157

.... etc.

************************************
LISTING ELEMENTS, ROWS OR COLUMNS
************************************
> turtleh[1,10]
[1] 165

> turtleh[1,4]
[1] F
Levels: F M


> turtleh[1,]
  Year site turtle sex vp em fo up to tdays
1   93    Q      1   F 12 88  0  0  0   165

> turtleh[,3]
 [1]  1  3  4  5  6  7  8  9 10 14  2  3  4  5  9 10 14 16 17 74 75 77 82 83 84 86 87

> turtleh[,10]
 [1] 165 157 164 164 163 164 162 160 161 157 164 171 172 168 161 164 167 163 154 164  ....
```

Reading comma separated values using agpop data. The variables at end have been cut off for space reasons

```
> ag2<-read.csv("g:/s597/data/agpopnew.csv")
> ag2
                  COUNTY STATE ACRES92 ACRES87 ACRES82 FARMS92 FARMS87 FARMS82  etc.
1  ALEUTIAN ISLANDS AREA    AK  683533  726596  764514      26      27      28
2          ANCHORAGE AREA    AK   47146   59297  256709     217     245     223

22         COLBERT COUNTY    AL  138135  145104  161360     488     563     686
```

```
***********************************************
Below reads nut2.dat in which a . is used for a missing value.
Note that with no na.strings = ,  variables with any . 's  will be treated
as factor(character) variables and cannot be operated
on numerically.
***********************************************

> nutdat<-read.table("g:/s597/data/nut2.dat")
> nutdat
     V1 V2 V3 V4 V5 V6 V7  V8 V9 V10 V11
1     1  3  4 10 10  8 15  14 14  38  39

17   22  2  4  9  .  .  6   .  .  25   .

237 501  1  4  8  8  .  6   8  .  23   .


> str(nutdat)
'data.frame':   237 obs. of  11 variables:
 $ V1 : int  1 3 4 5 6 8 9 12 13 14 ...

 **** other variables omitted ***

 $ V11: Factor w/ 25 levels ".","16","18",..: 23 7 16 14 7 4 13 16 15 12 ...

***********************************************
Now designate . as missing values. The variables
are all numerical with missing values in the dataframe as NA
***********************************************

> nutdat2<-read.table("g:/s597/data/nut2.dat",na.strings=".")
> str(nutdat2)
'data.frame':   237 obs. of  11 variables:
 $ V1 : int  1 3 4 5 6 8 9 12 13 14 ...

 .....................

 $ V11: int  39 23 32 30 23 20 29 32 31 28 ...
> nutdat2
     V1 V2 V3 V4 V5 V6 V7  V8 V9 V10 V11
1     1  3  4 10 10  8 15  14 14  38  39
2     3  3  4  6  9  7 13  14 16  26  23

17   22  2  4  9 NA NA  6  NA NA  25  NA
```

```
*****************************************************************
Here agpopnew_m.csv has some missing values in as indicated by ,,
Below are the first three lines of agpopnew.csv.

COUNTY,STATE,ACRES92,ACRES87,ACRES82,FARMS92,FARMS87,FARMS82,LARGEF92, ....
ALEUTIAN ISLANDS AREA,AK,683533,726596,764514,,27,28,,16,20,6,4,1,W
ANCHORAGE AREA,AK,47146,,256709,217,245,223,9,10,11,,52,38,W

Not all variable shown below
*****************************************************************
> ag<-read.csv("g:/s597/data/agpopnew_m.csv")
> ag
                   COUNTY STATE ACRES92 ACRES87 ACRES82 FARMS92 FARMS87 FARMS82 LARGEF92  ...
1  ALEUTIAN ISLANDS AREA    AK  683533  726596  764514      NA      27      28       NA
2          ANCHORAGE AREA    AK   47146      NA  256709     217     245     223        9
3          FAIRBANKS AREA    AK  141338  154913  204568     168     175     170       25
```

1.2.1 Renaming variables.

This reads a file without names in the first line and then attaches new names (Year, site and var1) to the variables (originally V1,V2 and V3) and, via the rm, removes the old names.

```
> a<-read.table("g:/s597/data/file1")
> a<-within(a,{Year<-V1;site<-V2;var1<-V3;rm(V1,V2,V3)})
> a
  var1   site Year
1    5 BELCH1 1997
2    2 BELCH2 1997
3    3 BELCH3 1997
4   12 BELCH1 1998
5   11 BELCH2 1998
6   13 BELCH3 1998
7    8 BELCH6 1998
```

1.3 Working with the dataframe

1.3.1 Selecting cases

There are various ways to create a new dataframe containing only those "cases" (rows) meeting certain conditions.

The following will select values of the turtlef file (with variable names attached) for which year = 93. Note that names are case sensitive so it is Year that must be used. Note the == also.

```
>  turtleh <- read.table("g:/s597/data/days_h.dat", header=T)
> turt93<-turtleh[turtleh$Year==93,]
> turt93
```

```
   Year site turtle sex vp em fo up to tdays
1    93    Q      1   F 12 88  0  0  0   165
2    93    Q      3   M 40  4  0 56  0   157
        ......
10   93    Q     14   F 31 35  0 34  0   157
```

If you had used attach(turtleh) then you could just use

```
> turt932<-turtleh[year==93,]
```

Note the ,] at the end of the command. This means there is a selection of rows going on but not columns. See below.

Whether you attach(turteh) or not you can also use

```
> turtle93<-subset(turtleh,Year==93)
```

or selection with multiple criteria uses

```
> turtle93160<-subset(turtleh,Year==93 & tdays<160)
> turtle93160
   Year site turtle sex vp em fo up to tdays
2    93    Q      3   M 40  4  0 56  0   157
10   93    Q     14   F 31 35  0 34  0   157
```

You can also select cases numerically

```
> newt3<-turtleh[2:3,]
> newt3
  Year site turtle sex vp em fo up to tdays
2   93    Q      3   M 40  4  0 56  0   157
3   93    Q      4   F 41 20  0 39  0   164

> newt4<-turtleh[c(1,4,8,10),]
> newt4
   Year site turtle sex vp em fo up to tdays
1    93    Q      1   F 12 88  0  0  0   165
4    93    Q      5   M 34 57  0  9  0   164
8    93    Q      9   F 32 68  0  0  0   160
10   93    Q     14   F 31 35  0 34  0   157
```

Note that the c( ) above represents a collection of values.

There are also some functions where if you put [expression] after the function then it only uses those cases that satisfy the expression. Either of the following will list cases with Year equal to 93

```
> turtleh[turtleh$Year==93,]
or
> attach(turtleh)
```

```
> turtleh[Year==93,]

   Year site turtle sex vp em fo up to tdays
1    93    Q      1   F 12 88  0  0  0   165
2    93    Q      3   M 40  4  0 56  0   157

10   93    Q     14   F 31 35  0 34  0   157
```

1.3.2 Selecting variables

This can be done in various ways. Below we select the 2nd and 3rd columns (variables) in turtleh, two ways. You could also have put turtleh[,2:3] and turtleh[,c("site","turtle")]. This means no selection of rows but a selection of columns. **NOTE that the default when we refer to the dataframe like this is that if there is no , inside the brackets is that the entry refers to the selection of columns!**

```
> newt<-turtleh[2:3]
> newt
   site turtle
1     Q      1
2     Q      3
 ...
27   LS     87


> newt2<-turtleh[c("site","turtle")]
> newt2
   site turtle
1     Q      1
2     Q      3
...
27   LS     87
```

Finally you can **select both rows and columns**.

```
> newt4<-turtleh[1:5,c("sex","turtle")]
> newt4
  sex turtle
1   F      1
2   M      3
3   F      4
4   M      5
5   F      6

> newt5<-turtleh[Year==93,c("Year","sex","tdays")]
> newt5
   Year sex tdays
1    93   F   165
2    93   M   157
3    93   F   164
```

```
4    93   M   164
5    93   F   163
6    93   M   164
7    93   M   162
8    93   F   160
9    93   M   161
10   93   F   157
```

1.3.3 Combining dataframes

You can easily combine files using rbind, which binds rows. The general command is

```
new<-rbind(data1,data2,, )
```

Will create a dataframe named new which combines data1, data2, ... where these are previously created dataframes, all with the same variables. (Later rbind can also be used to combine matrices with equal numbers of columns).

1.3.4 Merging dataframes.

This can be done using *merge(data1,data2)*. There are some options available. You can only merge two files at a time. With no options, it will merge by matching on variables that are common to both data sets.

```
> a<-read.table("g:/s597/data/file1")
> b<-read.table("g:/s597/data/file2")
  *** RENAME VARIABLES ***
> a<-within(a,{Year<-V1;site<-V2;var1<-V3;rm(V1,V2,V3)})
> b<-within(b,{Year<-V1;site<-V2;var2<-V3;rm(V1,V2,V3)})

> a
  var1   site Year
1    5 BELCH1 1997
2    2 BELCH2 1997
3    3 BELCH3 1997
4   12 BELCH1 1998
5   11 BELCH2 1998
6   13 BELCH3 1998
7    8 BELCH6 1998

> b
  var2   site Year
1   20 BELCH1 1998
2   17 BELCH2 1998
3   14 BELCH3 1998
4   12 BELCH6 1998
5    7 BELCH1 1997
6    3 BELCH2 1997
7    4 BELCH3 1997
```

```
> c<-merge(a,b)
> c
    site Year var1 var2
1 BELCH1 1997    5    7
2 BELCH1 1998   12   20
3 BELCH2 1997    2    3
4 BELCH2 1998   11   17
5 BELCH3 1997    3    4
6 BELCH3 1998   13   14
7 BELCH6 1998    8   12
```

1.3.5 Sorting data

If you just use sort(var) where var is a variable in a dataframe, all it will do is sort that vector/variable and leave the rest as is. This is fine if doing something with just that vector by itself, but that is often not the case. To rearrange the whole dataframe, sorted on one or more variables the order function needs to be used. NOTE: I've used tsort for the new data frame, but you can reuse the original name. So, it could be *turtleh< −turtleh[order(tdays),]*. Notice that the first column of tsort (without a name) has the original order number. This actually the vector of indices that results by ordering on vdays; that is order(vdays) creates the vector (19,25, ... 22).

```
>  turtleh <- read.table("g:/s597/data/days_h.dat", header=T)
> attach(turtleh)
> tsort<-turtleh[order(tdays),]
> tsort
   Year site turtle sex vp em fo up to tdays
19   94    Q     17   F 15  0 72 13  0   154
25   94   LS     84   F 60  0 30  0 10   155
2    93    Q      3   M 40  4  0 56  0   157
 .... etc...
21   94   LS     75   M 73  0  5  6 16   173
22   94   LS     77   M 48 25  2  2 23   173

****To sort on years and then on tdays within years ****

> turtleh<-turtleh[order(Year,tdays),]
> turtleh
   Year site turtle sex vp em fo up to tdays
2    93    Q      3   M 40  4  0 56  0   157
10   93    Q     14   F 31 35  0 34  0   157
          .....
1    93    Q      1   F 12 88  0  0  0   165
19   94    Q     17   F 15  0 72 13  0   154
25   94   LS     84   F 60  0 30  0 10   155
          .....
22   94   LS     77   M 48 25  2  2 23   173
```

Helpful site for sorting: http://www.ats.ucla.edu/stat/r/faq/sort.htm.

1.3.6 Saving dataframes.

1. **Saving as a text file.**

   The contents of a dataframe, say called data, can be written to a text file using

   ```
   write.table(data, file = filename, sep = ‘‘ ‘‘)
   ```

   This creates a text file with spaces as delimiters, unless the sep = option is used to designate a delimiter, and with NA for missing values. The current variable names are given in the first line, in quotes, and there is a first column indicated observation number.

   ```
   >  nutdat2<-read.table("g:/s597/data/nut2.dat",na.strings=".")
   > write.table(nutdat2,file="g:/s597/data/nut2r.txt")
   ```

   creates the text file nut2r.txt located in the given path with first three lines given by

   ```
   "V1" "V2" "V3" "V4" "V5" "V6" "V7" "V8" "V9" "V10" "V11"
   "1" 1 3 4 10 10 8 15 14 14 38 39
   "2" 3 3 4 6 9 7 13 14 16 26 23
   ```

   There are ways, using the foreign library, to write SAS datafiles as well as datafiles for other packages.

2. **Saving as an R file for easy reloading**

   The command *save(data,file = “filename”)* will save the dataframe data to a file called filename (in the default working directory). In a later session you can recall this dataframe using *load(“filename”)*.

   For example if we have the dataframe ladata, *save(ladata, file =”ladata.R”)* saves the dataframe to the file ladata.R . The *load(“ladata.R”)* reloads it and you can then just refer to ladata.

1.4 Calculating, expressions, creating new variables, etc.

1.4.1 Calculating, operators and directly creating numerical vectors

You can use R as just a calculator. If you just type an expression in (with no assignment) it will just list the answer (or you could assign the expression to a variable and list the variable.

```
> 5+3
[1] 8
> sum<-5+3
> sum
[1] 8
  *** with this second form the variable sum can be used in subsequent calcuations***

> log(8)
[1] 2.079442
```

A column (variable) in a data frame is a vector. You can create a vector directly using the *c*() function (c is for combine). (Later we will see how to create and work with matrices).

Table 1: Some R operators

	function
addition	+
subtraction	-
division	
multiplication	*
square root	sqrt( )
x to the nth power	$x \hat{} n$
natural log	log( )
log base 10	log10()
exponential	exp ( )
absolute value	abs( )
equal to	==
logical and	&
logical or	|
not equal to	!=
less than or equal to	<=
greater than or equal to	>=
logical not	!

```
> x<-c(1,2,3,8,10)
> x
[1]  1  2  3  8 10
```

We can operate on vectors/variables to create new vectors/variables. Note that when working on variables from a dataframe the new variables are not automatically part of the dataframe.

```
> y<-x^2
> y
[1]   1   4   9  64 100
> xplusy<-x+y
> xplusy
[1]   2   6  12  72 110

> z<-x+(x^2)
***yields the same vector as xplusy  ***
```

1.4.2 Adding new variables; cbind, transform and data.frame

You can create a new dataframe with the new variables by using cbind or transform.

cbind(d1,d2 , .) will bind quantities by column (there can be more than two entries. There are options).

If one of the elements is a dataframe then the result will be a data frame. It will be a matrix if combining numerical vectors. To create a dataframe by binding vectors you can use *data.frame(cbind( ))*.

Using turtleh, which has been attached, below are two ways to create a new dataframe which also has the variable vpper = proportion of total days in vernal pools.

```
> vpper<-vp/tdays
```

```
> vpper
 [1] 0.07272727 0.25477707 0.25000000 0.20731707 0.15950920 0.19512195
  ...
[25] 0.38709677 0.30188679 0.55974843
> turt2<-cbind(turtleh,vpper)

******** The dataframe turt2 has the original turtleh plus vpper ***

> str(turt2)
'data.frame':   27 obs. of  11 variables:
 $ Year  : int  93 93 93 93 93 93 93 93 93 93 ...
 ....
 $ tdays : int  165 157 164 164 163 164 162 160 161 157 ...
 $ vpper : num  0.0727 0.2548 0.25 0.2073 0.1595 ...

**** Or you can do the following. The transform automatically
adds the new variables to the dataframe *****

> turt3<-transform(turtleh,vpper=vp/tdays)
> turt3
   Year site turtle sex vp em fo up to tdays      vpper
1    93    Q      1   F 12 88  0  0  0   165 0.07272727
 .......
27   94   LS     87   F 89  0  9  0  2   159 0.55974843
```

1.4.3 Creating a vector via sequencing using the seq function.

The following creates a vector x going from low to up with increments of inc.

```
x<- seq(low,up,inc)
```

The following creates x going from 1 to 10 in increments of .1, then creates a vector containing value of x-squared and plots x^2 versus x.

```
> x<-seq(1,10,.1)
> x2<-x*x
> plot(x,x2)
```

To make a dataframe named new containing the vector of x and x^2 values, use

```
> new<-data.frame(cbind(x,x2))
> str(new)
'data.frame':   91 obs. of  2 variables:
 $ x : num  1 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 ...
 $ x2: num  1 1.21 1.44 1.69 1.96 2.25 2.56 2.89 3.24 3.61 ...
```

The following results in dmat being a matrix (as opposed to a dataframe) with two columns.

```
> dmat<-cbind(x,x2)
> dmat
```

```
         x     x2
 [1,]  1.0   1.00
  ....
[91,] 10.0 100.00
```

1.4.4 Creating string/character variables from numeric variables

When reading a text file, any variable with just numbers in it is treated as a numerical variable. If you want it to be treated to a categorical variable in certain functions, it needs to be converted. You can do this using the factor function.

The following reads the nutrition data from a file that has the variables added to the first column with . as a missing value. Learning style has multiple values just indicating different styles. The numbers do not have particular meaning and for summarizing data and other purposes this should be treated as categorical. This shows a conversion of lrnsty using factor Note that the new lrnsty is a factor variable. For many functions these are just treated as categorical variables (including summary, which here just tells you how many are in each category).

```
> nutdat<-read.table("g:/s597/data/nut2_h.dat",header=T,na.strings=".")
> nutdat
     id method lrnsty k1 k2 k3 a1  a2 a3 b1 b2
1     1      3      4 10 10  8 15  14 14 38 39

> nutdat$lrnsty<-factor(nutdat$lrnsty)

*****THE USE OF nutdat$lrnsty means the variable is updated in the dataframe ********

> str(nutdat)
'data.frame':   237 obs. of  11 variables:
 $ id    : int  1 3 4 5 6 8 9 12 13 14 ...

 $ lrnsty: Factor w/ 4 levels "1","2","3","4": 4 4 2 1 1 2 4 4 4 4 ...
  ....
 $ b2    : int  39 23 32 30 23 20 29 32 31 28 ...

> summary(nutdat$lrnsty)
 1  2  3  4
67 75 32 63
```

There is a labels= option that lets you label the different values. For example

```
> nutdat$lrnsty<-factor(nutdat$lrnsty, labels = c(''A'',''B'',''C'',''D''))
```

To **categorize a continuous variable** you can use the cut function as illustrated below. And you can assign labels to the resulting categories. This reads the LA data (from a file with variable names in the first line) and then categorizes into five categories in $[20, 30)$, $[30, 40)$, $[40, 50)$, $[50, 60)$, $[60, 100)$, respectively (there are no observations less than 20). Note that the breakpoints yield intervals that are inclusive on the right. So, if we want the intervals as above and with age give in integers we need the cuts 19, 29, etc.

```
> ladata<-read.table("g:/s597/data/ladata_h.dat",header=T)
```

```
> ladata
     id age md50 sp50 dp50 ht50 wt50 sc50 soec cs md62 sp62 dp62 sc62 wt62 ihdx yrdth death
1     1  42    1  110   65   64  147  291    2  8    4  120   78  271  146    2    68     1
2     2  53    1  130   72   69  167  278    1  6    2  122   68  250  165    9    67     1

200 200  36    1  100   70   70  157  260    3  8    3  120   86  251  152    0     0     0

> agecat<-cut(age,breaks=c(19,29,39,49,59,100))
> agecat
  [1] (39,49]  (49,59]  (49,59]  (39,49]  (49,59]  (49,59]  (39,49]  (59,100]  ...
  ......
[193] (39,49]  (39,49]  (59,100] (49,59]  (19,29]  (19,29]  (39,49]  (29,39]

> agecat<-cut(age,breaks=c(19,29,39,49,59,100),labels=c("1","2","3","4","5"))
> agecat
  [1] 3 4 4 3 4 4 3 5 4 3 4 4 5 4 2 4 3 3 3 1 2 4 2 4 3 4 3 3 1 2 3 4 1 5 2 2 4 ...
 [76] 5 5 4 4 2 4 3 4 4 5 5 4 3 5 3 3 4 4 3 4 5 4 2 3 5 3 4 2 1 1 5 5 3 4 5 4 4 ...
[151] 4 3 4 2 1 3 3 3 4 3 2 3 3 1 3 3 2 2 3 2 5 3 2 2 3 2 1 1 1 4 4 3 3 3 5 2 4 ...
Levels: 1 2 3 4 5
```

agecat is not part of the dataframe. If you want it to be then you can use cbind as discussed earlier. Note that you can reuse the dataframe name.

```
> ladata<-cbind(ladata,agecat)
> ladata
     id age md50 sp50 dp50 ht50 wt50 sc50 soec cs md62 sp62 dp62 sc62 wt62 ihdx yrdth death agecat
1     1  42    1  110   65   64  147  291    2  8    4  120   78  271  146    2    68     1      3
2     2  53    1  130   72   69  167  278    1  6    2  122   68  250  165    9    67     1      4

200 200  36    1  100   70   70  157  260    3  8    3  120   86  251  152    0     0     0      2
```

1.5 Single samples: describing and analyzing

1.5.1 Missing values, sample size and the length function

In summarizing and analyzing data, we need to pay attention to missing values. Some of the functions below (e.g., mean, sd, var, median) default to returning an NA if the are run on a vector with missing values; the option na.rm=T will take care of this. The other functions will use just non-missing values, some, but not all, will tell you how many missing (or non-missing) values there are.

length(x) tells you how long x is but it counts missing values. If there are no missing values then this will give the sample size n.

If you want to explicitly compute the number of non-missing cases in a vector, you can use the following.

is.na(x) returns a vector containing elements that are TRUE if missing and FALSE if not. If we take the sum of this vector it gives the number of TRUE cases. So, the number of non-missing values in x can be obtained via

```
n<-length(x)-sum(is.na(x))
```

1.5.2 A quantitative variable

Table 2: Functions for summary statistics, plots and inference for a quantitative variable.

Function	Purpose
mean	mean (trim option will give a trimmed mean)
sd	standard deviation
var	variance
med	median
summary	mean plus quantiles
quantile	quantiles (option to specify which ones)
length	gets length of a vector
boxplot	boxplot
hist	histogram
t.test	one-sample t-test (with options)
wilcox.test	wilcoxon test of location
ks.tes	Kolmogorov-Smirnov goodness of fit test

The example below works with the LA data, running many of the analyses used earlier in working with SAS. There is no missing data here.

```
ladata<-read.table("g:/s597/data/ladata_h.dat",header=T)
attach(ladata)
> diffsc<-sc62-sc50

**SUMMARY STATISTICS **

> mean(wt50)
[1] 168.075
> sd(wt50)
[1] 26.63959
> median(wt50)
[1] 165
> var(wt50)
[1] 709.6677

*** YOU CAN GET THE SE FOR THE MEAN (since no missing values) ****
> serror<-sd(wt50)/sqrt(length(wt50))
> serror
[1] 1.883703

*** YOU CAN ORGANIZE THE RESULTS INTO AN OBEJCT. NOTE THAT
    COLUMN NAMES ARE GIVEN TO THE LEFT OF THE = SIGN IN USING
    CBIND

> meanwt50<-mean(wt50)
> sdwt50<-sd(wt50)
> varwt50<-var(wt50)
> medwt50<-median(wt50)
```

```
> wt50stats<-cbind(mean=meanwt50,st.dev=sdwt50,variance=varwt50,median=medwt50)
> wt50stats
        mean   st.dev variance median
[1,] 168.075 26.63959 709.6677    165

**** QUANTILES ****
> quantile(wt50)
  0%  25%  50%  75% 100%
 109  147  165  189  245

> quantile(wt50,c(.2,.4,.6,.8,1))
  20%   40%   60%   80%  100%
144.0 158.6 171.4 192.0 245.0

> summary(wt50)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  109.0   147.0   165.0   168.1   189.0   245.0
> sumwt50<-summary(wt50)
> sumwt50
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  109.0   147.0   165.0   168.1   189.0   245.0
> min=sumwt50[1]
> min
Min.  109
> boxplot(wt50)
> hist(wt50)
> qqnorm(wt50)
```

One-sample T-test. *t.test(x,mu=m, conf.level = p, alt= " ")*

Default is that null hypothesis is mu = 0, confidence level = .95 and the alternative hypothesis is not equal to m; alt = "g" gives an alternative of greater than m and "l" of less than m.

```
> t.test(diffsc)
        One Sample t-test
data:  diffsc
t = -1.0672, df = 199, p-value = 0.2872
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 -11.490938   3.420938
sample estimates:
mean of x     -4.035

> t.test(diffsc,conf.level=.90)
alternative hypothesis: true mean is not equal to 0

90 percent confidence interval:
 -10.283255   2.213255

> t.test(age,mu=45)
        One Sample t-test
```

```
data:  age
t = 1.3929, df = 199, p-value = 0.1652
alternative hypothesis: true mean is not equal to 45
95 percent confidence interval:
 44.55729 47.57271
sample estimates:
mean of x    46.065

> t.test(age,mu=45,alt="g")

t = 1.3929, df = 199, p-value = 0.0826
alternative hypothesis: true mean is greater than 45

> t.test(age,mu=45,alt="l")

t = 1.3929, df = 199, p-value = 0.9174
alternative hypothesis: true mean is less than 45
```

wilcox.test provides a nonparametric test of the null hypothesis that the median is equal to m. The options mu= , alt= and conf.level= work the same as in t-test.

```
> wilcox.test(diffsc)

        Wilcoxon signed rank test with continuity correction
data:  diffsc
V = 9553, p-value = 0.7129
alternative hypothesis: true location is not equal to 0
```

Testing for normality using the Kolmogorov-Smirnov Goodness of fit test.

```
> meandsc<-mean(diffsc)
> sddsc<-sd(diffsc)
> ks.test(diffsc,"pnorm",mean=meandsc,sd=sddsc)

        One-sample Kolmogorov-Smirnov test
data:  diffsc
D = 0.0776, p-value = 0.1794
alternative hypothesis: two-sided

Warning message:
In ks.test(diffsc, "pnorm", mean = meandsc, sd = sddsc) :
  cannot compute correct p-values with ties
```

The K-S test is having trouble here (compare the P-value to that in SAS, where the P-value is $< .01$). Other tests can be done making use of the nortest package.

An example with some missing values. Here we work with the nutrition data. For the variable k2 (nutrition knowledge at time 2) there 18 missing values, out of 237 cases. Note that to use the mean function you need to use na.rm=T. The t-test, summary and other functions will, however, directly exclude missing values. Summary tells you how many missing values there are. The t-test does not but you can tell from the degrees of freedom (218) that there are 219 non-missing cases.

```
> rm(list=ls()) #clear the workspace
> nutdat<-read.table("g:/s597/data/nut2_h.dat",header=T,na.strings=".")
> attach(nutdat)
> k2
  [1] 10  9  8 13  8  8  7  8  9 11 10 10 10 10 13  9 NA  2 11 NA 13 11 13 NA 10
   ...
[226] 11 10 11 NA 10  4  7 10 10 12 NA  8
> length(k2)  [1] 237

****Looking at effect of missing values and getting the number of non-missing ****
> mean(k2)
[1] NA
> mean(k2,na.rm=T)    [1] 9.36073

> is.na(k2)
  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
    ....
[229]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE

> sum(is.na(k2)) [1] 18
> n<-length(k2)-sum(is.na(k2))   #n = number of non-missing values
> n  [1] 219
> t.test(k2)
        One Sample t-test
data:  k2
t = 63.5908, df = 218, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 9.070608 9.650853
sample estimates: mean of x    9.36073

> summary(k2)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
  2.000   8.000  10.000   9.361  11.000  13.000  18.000
```

1.5.3 A categorical variable and inferences for a proportion.

Here we look at analyzing coronary status (cs) in the LA data, where cs=8 is normal and the other codes indicate some heart disease.

First, using all of the categories

```
> cs<-factor(cs)
> cs
  [1] 8 6 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 7 8 8 8 8 8 8 ...
 [76] 8 8 8 7 8 7 8 8 8 7 8 8 8 3 8 8 8 8 8 8 8 8 8 8 8 8 8 ...
[151] 8 8 8 8 8 8 8 8 8 8 8 8 8 3 8 7 8 8 8 8 8 8 8 8 8 8 8 ...
Levels: 0 3 4 5 6 7 8
> summary(cs)
  0   3   4   5   6   7   8
  1   6   1   1   5  15 171
```

```
> csprop<-summary(cs)/length(cs)   #Get proportion in each category
> csprop
    0     3     4     5     6     7     8
0.005 0.030 0.005 0.005 0.025 0.075 0.855
> barplot(summary(cs))
> barplot(csprop)
> pie(summary(cs))
```

Proportions: Now convert cs to disease status where disease = 0 if cs = 8 and = 1 otherwise. Note that disease is a factor variables, so cannot operate on it numerically. (We could also create a numerical vector of 0's and 1's and work with it as a quantitative variable as illustrated in the SAS section.) The following estimates the proportion with coronary heart disease and gets and exact and approximate confidence interval using binom.test and prop.test. These are used here to get confidence intervals. For testing the default null value is .5, which is not really of interest here. Other nulls, and the direction of the alternative can be chosen with options as in t.test and the confidence level can be changed.

```
> disease<-cut(cs,breaks=c(-1,7,10),labels=c("1","0"))
> disease
  [1] 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1
   ....
[176] 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0
Levels: 1 0

> dcounts<-summary(disease)
> dcounts
  1   0
 29 171
> dprop<-dcounts/length(disease)
> dprop
    1     0
0.145 0.855

> n<-length(disease)  #Sample size since no missing values
> count<-dcounts[1]   #number of successes in sample

> binom.test(count,n)   # could directly use binom.test(dcounts[1],length(disease))
        Exact binomial test
data:  count and n
number of successes = 29, number of trials = 200, p-value < 2.2e-16
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.09930862 0.20156150
sample estimates: probability of success                   0.145

> prop.test(count,n)
        1-sample proportions test with continuity correction
data:  count out of n, null probability 0.5
X-squared = 99.405, df = 1, p-value < 2.2e-16
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
 0.1007793 0.2032735
```

```
  *** Computing some statistics directly and approximate CI with no
      continuity correction***
> p<-dprop[1]
> sep<-sqrt(p*(1-p)/n)
> lower=p - 1.96*sep
> upper =p+1.96*sep
> cisum <-cbind(proportion=p,serror=sep,lower=lower,upper=upper)
> cisum
  proportion     serror      lower     upper
1      0.145 0.02489729 0.09620131 0.1937987
```

1.6 Multiple groups with a quantitative outcome

1.6.1 Describing data

Statistics for each group can be computed using the tapply function; *tapply(y,group,function)*. This runs function on the variable y, for each level of the factor variable group. You can also get some summary measures across groups when running a one-way analysis; see Section 1.6.3

Chick example; a quantitative Five chicks in each of four diets with weight gain recorded.

```
> chick<-read.table("g:/s597/data/chick.dat")
> chick
   V1  V2
1   1  55
   ...
20  4 154
> chick<-within(chick,{diet<-V1;gain<-V2;rm(V1,V2)}) #rename variables
> chick
   gain diet
1    55    1
 ....
20  154    4
> attach(chick)

> tapply(gain,diet,mean)
    1     2     3     4
 43.8  71.0  81.4 142.8

> tapply(gain,diet,sd)
       1        2        3        4
13.62718 31.02418 22.87575 34.90272

*********PUT THE SUMMARY STATISTICS IN A TABLE *************
> gmean<-tapply(gain,diet,mean)
> gsd<-tapply(gain,diet,sd)
> n<-tapply(gain,diet,length)
> groupsum<-cbind(mean=gmean,st.dev=gsd,samplesize=n)
> groupsum
   mean   st.dev samplesize
```

```
1  43.8 13.62718          5
2  71.0 31.02418          5
3  81.4 22.87575          5
4 142.8 34.90272          5


> tapply(gain,diet,summary)
$`1`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   21.0    42.0    49.0    43.8    52.0    55.0
$`2`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
     30      61      63      71      89     112
$`3`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   42.0    81.0    92.0    81.4    95.0    97.0
$`4`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   85.0   137.0   154.0   142.8   169.0   169.0

> boxplot(gain~diet) #produces side by side box plots
```

Histograms. Will look more at graphs showing histograms for each groups when do graphics in more detail. Note that one way to do separate histogram for each group is as follows (which does a histogram for group 1 and can be repeated for other groups.)

```
> gain1<-gain[diet==1]
> gain1
[1] 55 49 21 52 42
> hist(gain1)
```

1.6.2 Comparing two groups, t-test, etc.

The following illustrates how to compare two groups using the speed data and comparing oxygen consumption at speed 6 (vo6) between males and females.

```
> speed<-read.table("g:/s597/data/speed2_h.dat",header=T)
> speed
   id sex vo34  vo4 vo45  vo5 vo55  vo6
1   1   1 15.7 18.4 22.0 34.8 45.3 51.1
...
24 24   2 14.3 16.9 23.9 36.8 37.4 36.2
> attach(speed)
> tapply(vo6,sex,summary)
$`1`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  39.20   41.55   43.10   45.21   49.00   51.30
$`2`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  36.20   39.50   40.80   41.16   43.60   45.00
```

Testing for equal means and a confidence interval for the difference, allowing unequal means (the default in t.test).

```
> t.test(vo6~sex)

        Welch Two Sample t-test
data:  vo6 by sex
t = 2.6932, df = 21.906, p-value = 0.01331
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.932324 7.183232
sample estimates:
mean in group 1 mean in group 2
       45.21333        41.15556
```

Assuming the variances are equal.

```
> t.test(vo6~sex,var.equal=T)

        Two Sample t-test
data:  vo6 by sex
t = 2.4028, df = 22, p-value = 0.02515
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.5554262 7.5601293
```

Testing for equal variances under normality

```
> var.test(vo6~sex)

        F test to compare two variances

data:  vo6 by sex
F = 2.5514, num df = 14, denom df = 8, p-value = 0.1857
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.6178245 8.3821112
sample estimates:
ratio of variances
          2.551408
```

Nonparametric test that the distribution of vo6 is the same for males and females.

```
> wilcox.test(vo6~sex)

        Wilcoxon rank sum test with continuity correction

data:  vo6 by sex
W = 100, p-value = 0.05627
alternative hypothesis: true location shift is not equal to 0
```

```
Warning message:
In wilcox.test.default(x = c(51.1, 51.3, 49.1, 47.6, 48.9, 48.5,  :
  cannot compute exact p-value with ties
```

1.6.3 Comparing many groups, one-way ANOVA.

Using the chick data; see Section 1.6.1

A one-way analysis of variance; assumes equal variances.

```
> diet<-factor(diet)
> anova(lm(gain~diet))
Analysis of Variance Table
Response: gain
          Df  Sum Sq Mean Sq F value    Pr(>F)
diet       3 26235.0  8745.0  12.105 0.0002180 ***
Residuals 16 11558.8   722.4
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

The following provides a summary using the summary function applied to the object created by the lm function.

```
> summary(lm(gain~diet))

Call: lm(formula = gain ~ diet)
Residuals:
   Min     1Q Median     3Q    Max
 -57.8   -8.5    6.7   14.1   41.0

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)    43.80      12.02   3.644  0.00219 **
diet2          27.20      17.00   1.600  0.12914
diet3          37.60      17.00   2.212  0.04187 *
diet4          99.00      17.00   5.824 2.59e-05 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 26.88 on 16 degrees of freedom
Multiple R-squared: 0.6942,     Adjusted R-squared: 0.6368
F-statistic: 12.11 on 3 and 16 DF,  p-value: 0.0002180
```

The function *pairwise.t.test* will go in and compare each pair of groups (assuming equal variance and using the pooled estimate of the variance.)

```
> pairwise.t.test(gain,diet)
        Pairwise comparisons using t tests with pooled SD
data:  gain and diet
```

```
  1       2       3
2 0.25828 -       -
3 0.12561 0.54927 -
4 0.00016 0.00323 0.00936
```

Allowing the variances to be unequal: Use the *oneway.test* function.

```
> oneway.test(gain~diet)

        One-way analysis of means (not assuming equal variances)
data:  gain and diet
F = 11.3763, num df = 3.000, denom df = 8.297, p-value = 0.002632

> pairwise.t.test(gain~diet,pool.sd=F)
Error in typeof(x) : element 1 is empty;
   the part of the args list of '.Internal' being evaluated was:
   (x)
> pairwise.t.test(gain,diet,pool.sd=F)

        Pairwise comparisons using t tests with non-pooled SD
data:  gain and diet

  1     2     3
2 0.255 -     -
3 0.054 0.564 -
4 0.010 0.045 0.054
```

A nonparametric test (Kruskal-Wallis) that the distribution of gain is the same across all diets.

```
> kruskal.test(gain~diet)

        Kruskal-Wallis rank sum test
data:  gain by diet
Kruskal-Wallis chi-squared = 11.42, df = 3, p-value = 0.009659
```

1.7 Grouping and a categorical outcome: two-way tables.

Here the data can be summarized in a two-way table of cell counts using the *table* function. Proportions can be formed for the overall table, or for each of the margins using *margin.table*. The test for "independence" (see Section 12.2) can be carried out using chisquare.test and fisher.test.

We first do this for a general two-way table then for a 2 by 2 table, which involves comparing two proportions.

1.7.1 General two-way tables

LA example: Here we repeat the analysis from section 12.2 looking at the relationship of socioeconomic status and disease status where a new disease status (newcs) was created with values of 1 (some coronary disease, cs ≤ 3)), 2 (other disease; cs = 4, 5, 6 or 7) and 3 (normal; cs = 8). This also illustrates how you can easily recode a variable.

```
> ladata<-read.table("g:/s597/data/ladata_h.dat",header=T)
> attach(ladata)

  *** CREATE NEWCS****
> newcs<-rep(NA,length(cs))
> newcs[cs==8]=3
> newcs[cs<=7 & cs>=4]=2
> newcs[cs<=3]=1

> table(newcs)  # SHOW COUNTS FOR NEWCS VARIABLE
newcs
  1   2   3
  7  22 171

> sdtab<-table(soec,newcs) # GET TABLE OF FREQUENCIES
> sdtab
    newcs
soec  1  2  3
   1  1  6 15
   2  0  3 32
   3  2  8 93
   4  0  3 15
   5  4  2 16

> margin.table(sdtab,1) #GET MARGINAL TOTALS FOR FACTOR 1
soec
  1   2   3   4   5
 22  35 103  18  22

> margin.table(sdtab,2) #GET MARGINAL TOTALS FOR FACTOR 2
newcs
  1   2   3
  7  22 171

> prop.table(sdtab,1) # GET PROPORTIONS ACROSS EACH ROW
    newcs
soec          1          2          3
   1 0.04545455 0.27272727 0.68181818
   2 0.00000000 0.08571429 0.91428571
   3 0.01941748 0.07766990 0.90291262
   4 0.00000000 0.16666667 0.83333333
   5 0.18181818 0.09090909 0.72727273

> prop.table(sdtab,2) # GET PROPORTIONS ACROSS EACH COLUMN
    newcs
soec          1          2          3
   1 0.14285714 0.27272727 0.08771930
   2 0.00000000 0.13636364 0.18713450
   3 0.28571429 0.36363636 0.54385965
   4 0.00000000 0.13636364 0.08771930
```

```
   5 0.57142857 0.09090909 0.09356725

> overallp<-sdtab/length(newcs) # GET PROPORTIONS OVER WHOLE TABLE
> overallp
    newcs
soec     1     2     3
   1 0.005 0.030 0.075
   2 0.000 0.015 0.160
   3 0.010 0.040 0.465
   4 0.000 0.015 0.075
   5 0.020 0.010 0.080
```

The following do a chi-square and then Fisher's exact test for independence in the table

```
> chisq.test(sdtab)

        Pearson's Chi-squared test
data:  sdtab
X-squared = 24.8701, df = 8, p-value = 0.001635

Warning message:
In chisq.test(sdtab) : Chi-squared approximation may be incorrect


> fisher.test(sdtab)
        Fisher's Exact Test for Count Data
data:  sdtab
p-value = 0.00903
alternative hypothesis: two.sided
```

1.7.2 Two by two tables and comparing proportions.

Using the Kids data and creating the variable goals2, which equal Grades if grades is the main goal and equals Other, otherwise (see Section 9). The objective is to compare the proportion who have Grades as the most important goal between boys and girls.

```
> kids<-read.table("g:/s597/data/kids_h.dat",header=T)
> kids
    gender grade age  race  type school   goals grades sports looks money
1      boy     5  11 White Rural    Elm  Sports      1      2     4     3

200   girl     5  10 White Urban   Main  Grades      3      2     1     4
> attach(kids)

   *** Definte goals2 ***
> goals2<-rep(NA,length(gender))
> goals2[goals=="Popular"]="Other"
> goals2[goals=="Sports"]="Other"
> goals2[goals=="Grades"]="Grades"
```

```
> table(goals2,gender)
        gender
goals2   boy girl
  Grades  35   61
  Other   45   59

> success<-c(35,61)
> total<-c(80,120)
> prop.test(success,total)

        2-sample test for equality of proportions with continuity correction
data:  success out of total
X-squared = 0.702, df = 1, p-value = 0.4021
alternative hypothesis: two.sided
95 percent confidence interval:
 -0.22202572  0.08035905
sample estimates:
   prop 1    prop 2
0.4375000 0.5083333

*** could also use**
> GG2<-table(goals2,gender)
> total<-margin.table(GG2,2)
> total
gender
 boy girl
  80  120
> success<-GG2[1,]
> success
 boy girl
  35   61
> prop.test(success,total)
```

1.8 Comparing paired samples

Quantitative variable. You can either form the difference(change) and work with it or run a paired analysis. Here we rerun the analysis with the nutrition data comparing knowledge post (k3) to knowledge pre (k1). Note that change is defined as k3-k1, while if you run the paired analysis directly on k1 and k3 it works with k1-k3. This doesn't effect the test (for two sided alternatives) but does reverse the confidence interval.

```
> nutdat<-read.table("g:/s597/data/nut2_h.dat",header=T,na.strings=".")
> attach(nutdat)
> change<-k3-k1
> t.test(change)  # WORKING WITH CHANGE

        One Sample t-test
data:  change
t = 7.6055, df = 200, p-value = 1.086e-12
alternative hypothesis: true mean is not equal to 0
```

```
95 percent confidence interval:
 0.8439118 1.4346952
sample estimates:
mean of x
 1.139303

> t.test(k1,k3,paired=T)   #RUNNING PAIRED T-TEST ON k1 AND k3
        Paired t-test
data:  k1 and k3
t = -7.6055, df = 200, p-value = 1.086e-12
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.4346952 -0.8439118
sample estimates:
mean of the differences
              -1.139303

> wilcox.test(k1,k3,paired=T)
        Wilcoxon signed rank test with continuity correction
data:  k1 and k3
V = 2189, p-value = 5.749e-12
alternative hypothesis: true location shift is not equal to 0
```

Paired analysis for binary data. This uses McNemar's test as discussed in Section 10.2. We reanalyze the data on nest predation comparing predation rates for the two types of nests. 32 out of 40 of type 1 had predation and 15 out of 40 for type 2.

```
> predate<-read.table("g:/s597/data/mc.dat")
> predate
        V1 V2 V3
1   BELCH1  1  1
2   BELCH2  1  0

40   WARE9  1  0

> attach(predate)
> type1<-V2
> type2<-V3

> table(type1,type2)
     type2
type1  0  1
    0  4  4
    1 21 11

> mcnemar.test(type1,type2)
        McNemar's Chi-squared test with continuity correction
data:  type1 and type2
McNemar's chi-squared = 10.24, df = 1, p-value = 0.001374
```

You could also form a difference and get a confidence interval for the difference in proportions. Note you don't want to use the t-test (the observations are either -1, 0 or 1) but the confidence interval for the difference is approximately correct.

```
> diffp<-type1-type2
> t.test(diffp)
        One Sample t-test
data:  diffp
t = 3.9815, df = 39, p-value = 0.0002894
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 0.2090904 0.6409096
sample estimates: mean of x     0.425
```

1.9 Regression and correlation

Look at simple linear regression and pairwise Pearson correlation.

House price example. Regressing house price on square footage.

```
> house<-read.table("g:/s597/data/house.dat")
> house
      V1   V2 V3 V4 V5 V6 V7   V8
1   2050 2650 13  7  1  1  0 1639

117  739  970  4  4  0  0  1  541
> attach(house)
> price<-V1
> sqft<-V2

> lm(price~sqft)
Call:
lm(formula = price ~ sqft)

Coefficients:
(Intercept)         sqft
    47.8193       0.6137


> summary(lm(price~sqft))

Call:
lm(formula = price ~ sqft)

Residuals:
     Min       1Q   Median       3Q      Max
-1054.07   -99.06     6.68    69.42   753.66

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 47.81931   62.85482   0.761    0.448
```

```
sqft         0.61367    0.03625  16.931   <2e-16 ***
---
Signif. codes:  0 '*** 0.001 '** 0.01 '* 0.05 '. 0.1 '  1

Residual standard error: 204.5 on 115 degrees of freedom
Multiple R-squared: 0.7137,     Adjusted R-squared: 0.7112
F-statistic: 286.6 on 1 and 115 DF,  p-value: < 2.2e-16

****OR YOU COULD USE ***
> regout<-lm(price~sqft)
> summary(regout)

**** THE VECTOR RESIDS WILL HAVE THE RESIDUALS IN IT ****
> resids<-resid(regout)
> hist(resids)   #histogram of the residuals
```

To create a simple scatter plot, you can use plot. The abline(regout) function will overlay the least squares fit. Recall that regout was used to store the result of the lm() command. This plot can be enhanced in many ways.

```
> plot(sqft,price)
> abline(reg)
```

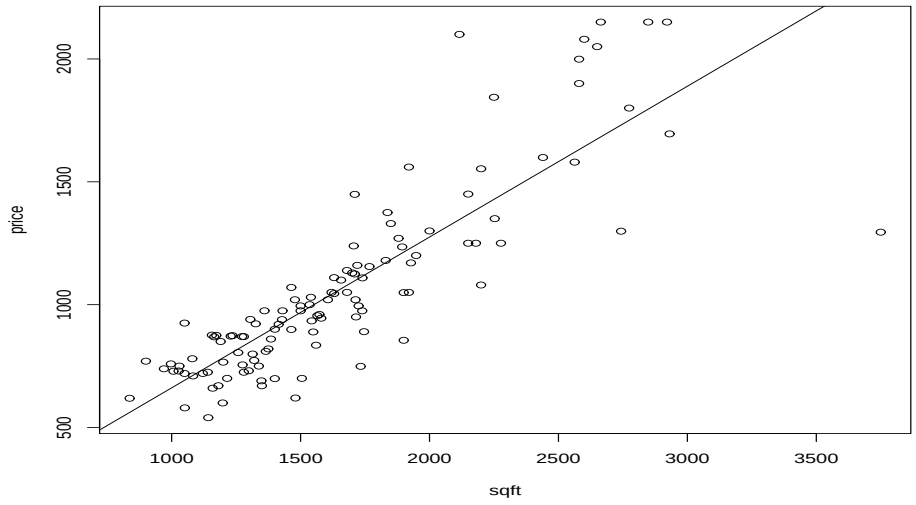

Figure 1: Scatterplot and regression line from R.

Correlation: Pearson correlations are found using the cor function. If there no missing values you can just use
cor(var1,var2).
But, if there missing values this will return an NA. To overcome this use
cor(var1,var2, use = "complete.obs"). To test for correlations you can use *cor.test( ).*

You can also replace var1,var2 with data (a defined dataframe) and it will get correlations among all pairs of numerical variables. Note that a categorical variable coded using numerical values will be read and treated as a numerical value; correlations using this variable don't make sense.

```
> cor(sqft,price)
[1] 0.8447951

> cor(house)
           V1         V2 V3          V4          V5           V6           V7 V8
V1  1.0000000 0.84479510 NA  0.42027250  0.16784024  0.555291961 -0.079292601 NA
V2  0.8447951 1.00000000 NA  0.39492498  0.14502997  0.520101642  0.040527966 NA
V3         NA         NA  1          NA          NA           NA           NA NA
V4  0.4202725 0.39492498 NA  1.00000000  0.19001561  0.241963969 -0.041546921 NA
V5  0.1678402 0.14502997 NA  0.19001561  1.00000000  0.043033148 -0.077336028 NA
V6  0.5552920 0.52010164 NA  0.24196397  0.04303315  1.000000000 -0.003993615 NA
V7 -0.0792926 0.04052797 NA -0.04154692 -0.07733603 -0.003993615  1.000000000 NA
V8         NA         NA NA          NA          NA           NA           NA  1
Warning message:
In cor(house) : NAs introduced by coercion

> cor(house,use="complete.obs")
           V1          V2          V3         V4          V5          V6          V7         V8
V1  1.0000000  0.88394183 -0.16666201  0.3663458  0.28916464  0.58211638 -0.18758563  0.8775270
V2  0.8839418  1.00000000 -0.03769359  0.3573967  0.36254721  0.49187084 -0.07850150  0.8752496
V3 -0.1666620 -0.03769359  1.00000000 -0.1834804  0.21642412  0.00851722  0.16272813 -0.2918422
V4  0.3663458  0.35739666 -0.18348040  1.0000000  0.30963494  0.31219490 -0.24912353  0.3039824
V5  0.2891646  0.36254721  0.21642412  0.3096349  1.00000000  0.15018688 -0.02371519  0.3024040
V6  0.5821164  0.49187084  0.00851722  0.3121949  0.15018688  1.00000000 -0.05368755  0.4370276
V7 -0.1875856 -0.07850150  0.16272813 -0.2491235 -0.02371519 -0.05368755  1.00000000 -0.1531738
V8  0.8775270  0.87524956 -0.29184225  0.3039824  0.30240397  0.43702756 -0.15317383  1.0000000

> cor.test(sqft,price)

        Pearson's product-moment correlation

data:  sqft and price
t = 16.9306, df = 115, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.7834034 0.8898607
sample estimates:
      cor
0.8447951
```

2 R: Part II

2.1 Graphics

The graphics in R are one of its strong points. This section describes the basic features of plotting in R. There is much, much more. A good reference is "R Graphics" by Murrell (Chapman & Hall).

In creating a figure the initial plot is created using a **plot** command. The plot command creates the axes and the frame for the plot and does and (almost always) does an initial plot. It has many options. Subsequent overlays to the plot, if desired, are done using the **points** command or the **lines** command. These have fewer options since they aren't involved with the axes set-up, titles, etc.

The plot command:

The simplest form of the plot command is *plot(x,y)* where x and y are vectors of equal length. What is in x is plotted on the x-axis and y on the y-axis. Only the x and y are required but there are many options that can be included. Below is a general form of the plot function with many of the commonly used options in it. Information on a number of the options can be found via help(par) and help(plot). Some of the text below is extracted directly from the help.

plot(x,y,type = , lty = ,pch = , main =" " , sub = " ", xlab = " " , ylab = " ", xlim = c(,), ylim = c(,), col = ,cex=, font=)

- type = specifies the type of plot, here are some of the options (points is the default)

  ```
  "p" for points,
  "l" for lines,
  "b" for both,
  "h" for 'histogram' like (or 'high-density') vertical lines,
  ```

- lty = a number, specifies the line type if type = "l". The default is 1.

  The line type. Line types can either be specified as an integer (0=blank, 1=solid (default), 2=dashed, 3=dotted, 4=dotdash, 5=longdash, 6=twodash) or as one of the character strings "blank", "solid", "dashed", "dotted", "dotdash", "longdash", or "twodash", where "blank" uses 'invisible lines (i.e., does not draw them).

- pch = specifies the symbol for a point, if included. The default is a circle.

  Either an integer specifying a symbol or a single character to be used as the default in plotting points. See points for possible values and their interpretation. Note that only integers and single-character strings can be set as a graphics parameter (and not NA nor NULL).

- main = gives the main plot title

- sub = gives a subtitle

- xlim and ylim specify the limits of the x and y axis; so xlim = c(4,10) gives a plot where the x-axis ranges from 4 to 10;

- col = sets a color; e.g., col= "red"

- cex = number controls the size of symbols. From the help

```
cex= A numerical value giving the amount by which plotting
text and symbols should be magnified relative to the default.
 Note that some graphics functions such as plot.default have
 an argument of this name which multiplies this graphical parameter,
 and some functions such as points accept a vector of
values which are recycled. Other uses will take just
 the first value if a vector of length greater than one is supplied.
```

There is also cex.axis, cex.lab,cex.main,cex.sub which lets you control individual parts.

- font = controls the font type. Below is from the help. It is a little obtuse, so experiment.

```
An integer which specifies which font to use for text. If possible, device drivers
arrange so that 1 corresponds to plain text (the default), 2 to bold face, 3 to
italic and 4 to bold italic. Also, font 5 is expected to be the symbol font, in
Adobe symbol encoding. On some devices font families can be selected by family
to choose different sets of 5 fonts.
```

The points statement.

Add points to the plot. This is of the form

points(x,y, pch=, cex = , col =)

where only the x and y are required. There are a few other options.

The lines statement.

Adds a line (or line combined with points) to the plot. This is of the form

lines(x,y, options)

The options can include type = , lty = , and some of the other options in the plot statement but, obviously not those concerned with the plot layout (e.g., main = , xlab = , etc.)

The abline statement.

abline(a,b) will plot a straight line with intercept 0 and slope b. (Sometimes the argument for the line might come from the output of a function; as in when doing simple linear regression we used abline(reg); see page 96 of part I of notes.)

Multiple plots on a page

The command

```
par(mfrow=c(a,b))
```

lays out a plotting page which will allows a*b plots arranged in a rows and b columns. This is put before the first plot statement. As a plot statement is encountered in puts the plot in the next available spot where the the plot are filled in a row at a time and within a row from left to right.

Plotting only some points.

In the plot, lines or points commands you can select out values by putting a condition after the variable. For example plot(x[expression], y[expression],) will plot just y versus x for observations satisfying the expression; this might for example be selecting based on values of another variables or the x and y themselves.

Unemployment example. The following does the unemployment plot (see page 15). Here it does it three different ways; two with points connected by a line and one with a line with no points. The par statement lays out the page to have three plots top to bottom.

```
edata<-read.table('g:/s597/data/employ.dat',header=F)
par(mfrow=c(3,1)) # lines up three plots vertically
Findex<-edata$V1
unemploy<-edata$V2
year<-edata$V3
newyear<-year+1950;
plot(newyear,unemploy,type="b",main="Unemployment over Time",
xlab="Year", ylab = "Unemployment")
#below uses a * rather than the default circle
plot(newyear,unemploy,type="b",pch="*",main="Unemployment over Time",
xlab="Year", ylab = "Unemployment")
#below uses no points, and chooses line type 3
plot(newyear,unemploy,type="l",lty=3,main="Unemployment over Time",
xlab="Year", ylab = "Unemployment")
```

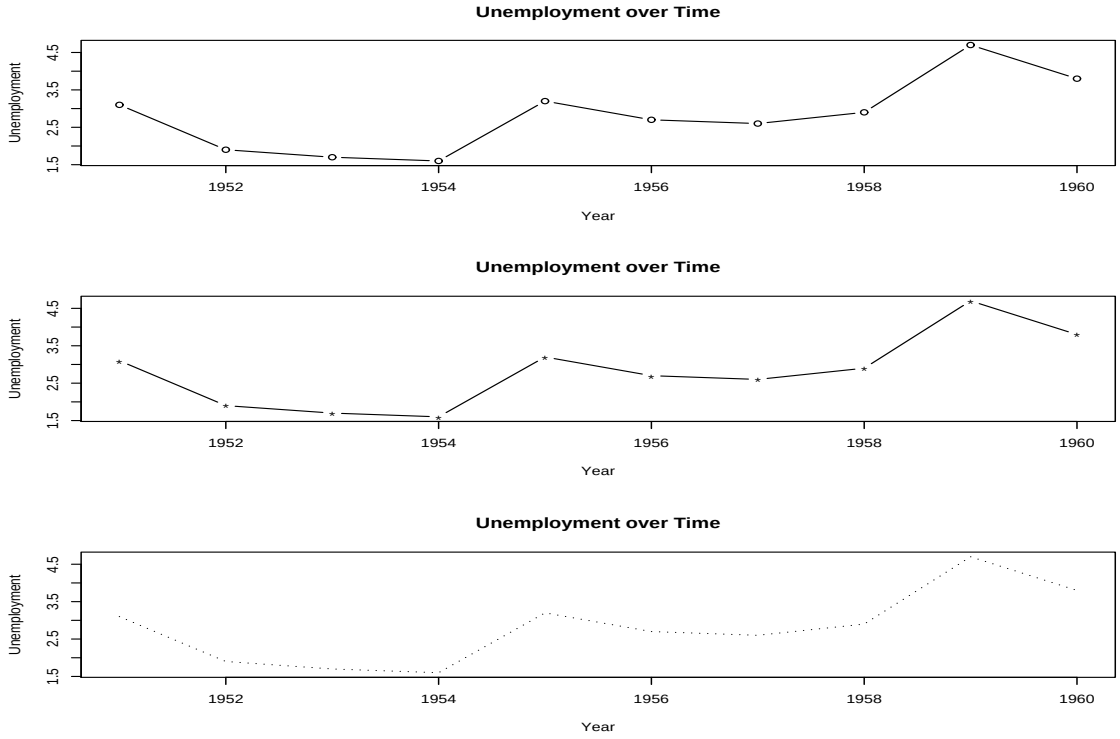

Figure 2: Unemployment plots using R

Esterase assay example. This does the plot on page 16. Note: We could have read an labeled variables

as above, but here I've illustrated the use of *matrix(scan* as an alternative to a read. The result is a matrix called data, rather than a dataframe. We create the variables, by name, by equating them columns of the matrix called data. It also shows overlaying using lines and points and the need to control the range of the y-axis. Finally it also demonstrates how you can read another data set and overlay a graph from it on the original plot. In this case it gets the original points that went into getting the fitted lines and prediction intervals.

```
data<-matrix(scan('g:/s597/data/pred.out'),ncol=7,byrow=T)
par(mfrow=c(1,1)) # single plot per page; not needed if in new session
                  # but resets if had a different layout earlier
x0<-data[,1]
yhat<-data[,2]
low <-data[,3]
up <- data[,4]
yhatw<-data[,5]
loww<-data[,6]
upw<-data[,7]
plot(x0,yhat,type="l",lty=1,xlab = "concentration", ylab = "count",
main = "Prediction intervals for Assay Data", ylim = c(-200,1400))
lines(x0,low,lty=1)
lines(x0,up, lty=1)
lines(x0,yhatw,lty=2)
lines(x0,loww,lty=2)
lines(x0,upw,lty=2)
edata<-read.table('g:/s597/data/ester.dat')
conc<-edata$V1   # true esterase concentration
count<-edata$V2  # radioactive binding count
points(conc,count,pch="*")
```

This plots verbal IQ versus brain size, for males and females, using the brain data

```
 brain<-read.table("g:/s597/data/Brain_h.dat",na.string=".",header=T)
 head(brain)
 attach(brain)
 par(mfrow=c(2,1))
 plot(mriCount[Gender=='Male'],VIQ[Gender=='Male'],
xlab="MRI COUNT", ylab = "Verbal IQ", main = "Verbal IQ versus
Brain Size: Males")
 plot(mriCount[Gender=='Female'],VIQ[Gender=='Female'],
xlab="MRI COUNT", ylab = "Verbal IQ", main = "Verbal IQ versus
Brain Size: Females")
```

2.1.1 Directing the graphics output

Graphics go to a graphics device. Without doing anything, when you create a plot it opens a graphics window as the active device. (This is equivalent to using the device function windows() in a windows environment). A plot in the graphics window can be saved using "save as", as a pdf, eps, JPEG, and some others.

You can also explicitly open another device/file and the graphics output will be sent to that device/file. For example the following would create a postscript file. Note some of the options used. In place of postscript, other options are pdf( ) and jpeg( ) as well as some others.

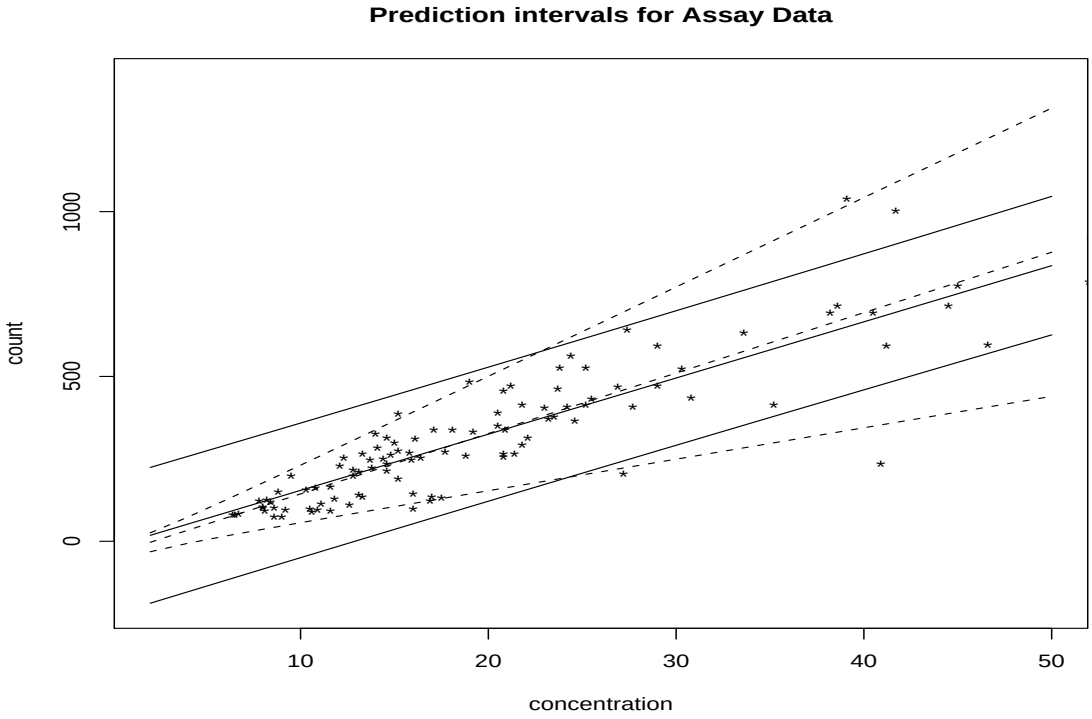

Figure 3: Esterase Assay intervals using unweighted and weighted least squares.

```
postscript("h:/mecourse/phplot.ps",horizontal=F,height=10,width=8,font=3)
```

use help(postscript) or help(jpeg) etc. to see the options. Notice that the default is horizontal = T which creates a landscape plot.

Once you have opened another device (e.g., via ps, pdf or jpeg) output will be routed there. If you open another device it becomes active.

dev.off() will shut off the active device and return to using the interactive graph window when you next plot something.

2.2 Miscellaneous

- **Sequencing** See Section 15.4.3 for definition and use of the seq command.

- **Creating a vector.** The **rep** command can be used to create a vector. *rep(values,n)* creates n repeats of values. If values is a single item, this is a vector of length n, but if values has p components, this is a vector length $n * p$. If the n is replaced by each=n, then it repeats each quantity in values n times.

```
> rep(4,10)
 [1] 4 4 4 4 4 4 4 4 4 4
> rep(NA,13)
 [1] NA NA NA NA NA NA NA NA NA NA NA NA NA
```

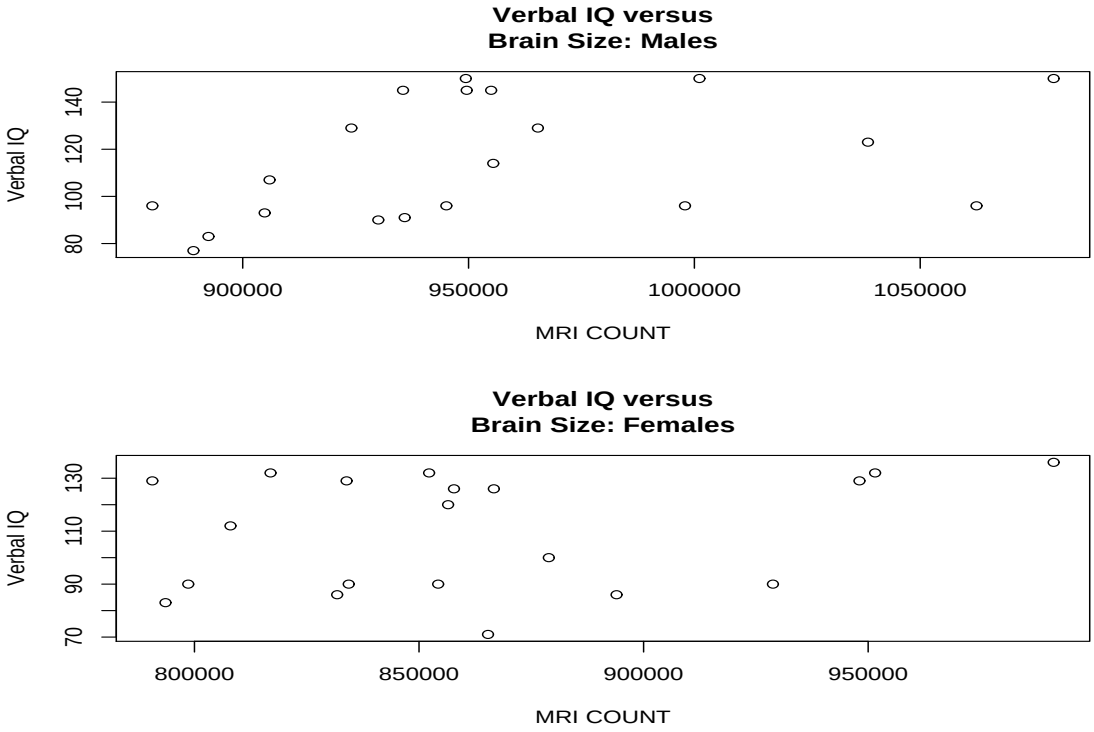

Figure 4: Verbal IQ versus brain size for each gender

```
> rep(1:p,10)
Error: object "p" not found
> rep(1:5,10)
 [1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3
[39] 4 5 1 2 3 4 5 1 2 3 4 5
> rep(1:5,each=10)
 [1] 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4
[39] 4 4 5 5 5 5 5 5 5 5 5 5
```

- **The cat statement.** The cat() statement lets you write script or variables to a file or to the console. This lets you add text to your output and it also is a way to output variables to a file.

  cat(....,file = , append = T)

  if file = is omitted, then the output goes to the console.

  Append = T is only an option if writing to a file, it appends to the file. If append = T is omitted then append=F is assumed which means the file is overwritten.

  The ... is what is output. It can be a collection of "items" separate by commas. An item in quotes is written as is except for some quantities using\.
  "$\backslash n$ "is a carriage return and "$\backslash t$" produces a tab.

- **multiple commands on a line:** You can put multiple commands on a line if all but the last end in a semicolon

- **If statements:** If statements are executed as below. Notice that the first piece is in parentheses, ( ), and the second in brackets, .

```
 if (expression1) {expression2}
```

or you can have an if then else

```
 if (expression1) then {expression2} else
{expression 3}
```

The expressions can be multiple lines.

2.3 looping

Do loops are done in R using the **for** command or the **while** command.

The following loops through values of varname and carries out expression2 using the for command.

```
 for (varname in seq) {expression2 involving varname}
```

where seq defines a range of possible values for varname.

The while command is of the form

```
while (condition) {statements}
```

Example: Here is an example involving the for and the if statement both. In homework 5, this would convert the missing values for CRIMTYPE (treated as numerical)

```
 for (k in 1:length(CRIMTYPE)){if (CRIMTYPE[k]==9){CRIMTYPE[k]=NA}}
```

Other applications appear in later examples.

2.4 Functions

The general form for defining a function is

```
fname<-function(argument1,argument2, ...)
{statements
return(objects)}
```

At the end of the expression the return(objects) specifies what values are returned to the main program from the function. The following function gets summary statistics for a numerical variable where the missing code is contained in mcode.

```
statc<-function(x,mcode){
for (k in 1:length(x)){if (x[k]==mcode){x[k]=NA}}
mean<-mean(x,na.rm=T)
```

```
sd <-sd(x,na.rm=T )
nmiss<-sum(is.na(x))
n<-length(x)- nmiss
med<-median(x, na.rm=T)
sum<-summary(x)
min<-sum[1]
max<-sum[6]
statt<-cbind(n=n,nmiss=nmiss,mean=mean,SD=sd,median=med,min=min,max=max)
return(statt)
}
```

If we knew there was no missing values you could use just stats< −function(x) and skip some of the rest.

You can save the text defining your function to a file, say the above is saved to g:/s597/statcom.R. You can then run this using the source command and execute it as in the following example; where AGE (from the SYC data) was already in the workspace from previous commands.

```
> source("g:/s597/stats.R")
> stats(AGE,99)
        n nmiss     mean       SD median min max
Min. 2621     0 16.80923 1.911258     17  11  24
```

NOTE: The **source command** executes what is in the file. (Also note that when using source it doesn't automatically list things. Suppose somewhere in the file it said just AGE. If we typed AGE in the console it would list age, but it doesn't do so when it is in a file being "sourced". Instead you need to use print(AGE)

2.5 Probability Functions with examples

There are four general functions, with first letters d, p, q and r that are used in working with probability distributions and generating samples from them. In each case the arguments involve the parameters of the distribution and are distribution specific.

- dName(x, arg1, ...) returns the PDF (density or mass function) evaluated at x of a random variable with distribution Name
- pName(x, arg1, ...) returns $P(X \leq x)$, the CDF (density or mass function) evaluated at x of a random variable with distribution Name.
- qName(p, arg1, ...) returns the quantile, the value q_p such that $P(X \leq q_p) = p$.
- rName(n, arg1, ...) generates n observations from the distribution.

Both the q and the p functions have options that come after the arguments that reverse the tail being worked with:

pName(x, arg1, ..., lower.tail=F) returns $P(X > x)$.

qName(p, arg1, ... , lower.tail = F) returns q_{1-p}; i.e. the value with probability p to the right of it.

There are also options that convert to log scales that we won't discuss here. See the help( ) results.

Some specific distributions, illustrated with the d function.

- Uniform: dunif(x,minv,maxv) or dunif(x, min = minv , max = maxv), minv = lower bound, maxv=upper bound

  If minv and maxv are omitted, then minv = 0 and maxv=1 (standard uniform)

- Exponential: dexp(x,ratev) or dexp(x,rate=ratev).

  rate is 1/mean, not the mean. If rate is omitted, then rate = 1 is assumed.

- Normal; dnorm(x, meanv, sdv) or dnorm(x, mean = meanv, sd = sdv)

- t-distribution dt(x, df, ncp)

  ncp = 0 if third argument omitted.

- Chi-square: dchisq(x, df, ncp)

  ncp = 0 if third argument omitted.

- F: df(x, df1, df2,ncp)

  ncp = 0 if fourth argument omitted.

```
# plotting the binomial
n<-10
pi<-.2
pval<-rep(NA,n)
pval
for (k in 1:n)
{pval[k] = dbinom(k,n,pi)}
kvec<-seq(1:n)
kvec
pval
plot(kvec,pval, type = "h")

#PLOTTING THE BINOMIALS AS A FUNCTION

bplot<-function(n,pi){
pval<-rep(NA,n)
pval
for (k in 1:n)
{pval[k] = dbinom(k,n,pi)}
kvec<-seq(1:n)
kvec
pval
plot(kvec,pval,type="h")}
bplot(10,.2)

# This does four plots to a page.
par(mfrow=c(2,2))
bplot<-function(pi){
for (n in c(5,10,20,50)){
```

```
pval<-rep(NA,n)
maint<-paste("n = ",n)
for (k in 1:n)
{pval[k] = dbinom(k,n,pi)}
kvec<-seq(1:n)
plot(kvec,pval,type="h",xlab = "k", ylab= "probability",main = maint)}}
bplot(.2)

# HERE IS A LONGER WAY WITH EACH GRAPH LABELED BY THE SAMPLE SIZE.  NOT NECESSARY.
par(mfrow=c(2,2))
pi<-.2
n<-5
pval<-rep(NA,n)
pval
for (k in 1:n)
{pval[k] = dbinom(k,n,pi)}
kvec<-seq(1:n)
plot(kvec,pval,type="h",xlab = "k", ylab= "probability", main =  "n = 5")
n<-10
pval<-rep(NA,n)
for (k in 1:n)
{pval[k] = dbinom(k,n,pi)}
kvec<-seq(1:n)
plot(kvec,pval,type="h",xlab = "k", ylab= "probability", main = "n = 10")
n<-20
pval<-rep(NA,n)
for (k in 1:n)
{pval[k] = dbinom(k,n,pi)}
kvec<-seq(1:n)
plot(kvec,pval,type="h",xlab = "k", ylab= "probability",main = "n = 20")
n<-50
pval<-rep(NA,n)
for (k in 1:n)
{pval[k] = dbinom(k,n,pi)}
kvec<-seq(1:n)
plot(kvec,pval,type="h",xlab = "k", ylab= "probability",main = "n = 50")
```

Hypergeometric Example on page 21. We will do this problem four different ways to show some of the features of R.

```
METHOD 1
# hypergeometric example on page 21.
# Here we write to the console using the cat
# function and a print.
 popsizea <- 698
 upper <-4
 prob <-rep(NA,upper)
 value <- rep(NA,upper)
 ssavalues<-seq(7,70,by=7)
 numberssa <-length(ssavalues)
 dvalues<-seq(1,101,by=5)
```

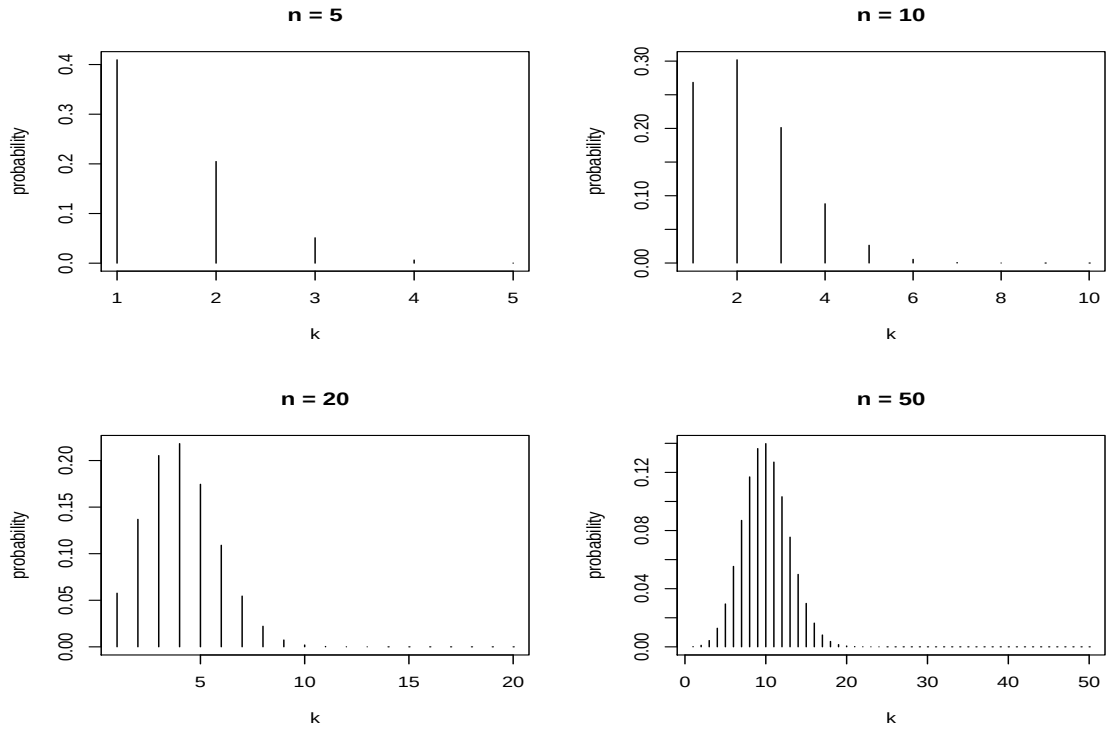

Figure 5: Binomial probability density functions with $\pi = .2$

```
  numberdef<-length(dvalues)
  for (j in 1:numberssa)
   {ssa= ssavalues[j]
   ssrate=ssa/popsizea
    for (def in seq(1,101, by = 5))
     {defrate=def/popsizea;
     nondef = popsizea-def;
     for (k in 0:upper)
     {prob[k+1] = dhyper(k,def,nondef,ssa)
      value[k]=k}
    cat("probabilities with N = 698,sample size= ", ssa,
   "number defective = ", def, "\n")
    prob0<-prob[1]; prob1<-prob[2]; prob2<-prob[3];prob3<-prob[4]
    values<-cbind(prob0,prob1,prob2,prob3)
    print(values)}}

probabilities with N = 698,sample size=  7 number defective =  1
         prob0      prob1 prob2 prob3
[1,] 0.9899713 0.01002865     0     0
probabilities with N = 698,sample size=  7 number defective =  6
         prob0      prob1       prob2        prob3
[1,] 0.9411107 0.05761902 0.001258057 1.219048e-05
 ...
```

METHOD 2

```
 #  Now we'll create a "file" with a separate line
 #  for each combination of sample size and number defective
 #  Here we do it showing also how you can write to a file
 #  using the cat command

  myfile<-"g:/s597/houtput"
  popsizea <- 698
  upper <-4
  prob <- rep(NA,upper)
  value <- rep(NA,upper)
  ssavalues<-seq(7,70,by=7)
  numberssa <-length(ssavalues)
  dvalues<-seq(1,101,by=5)
  numberdef<-length(dvalues)
  cat("ssa","\t", "def","\t", "p0","\t","p1","\t","p2","\t","p3\n", file=myfile)
  for (j in 1:numberssa)
   {ssa= ssavalues[j]
   ssrate=ssa/popsizea
    for (m in 1:numberdef)
     {def = dvalues[m]
     defrate=def/popsizea
     nondef = popsizea-def
     for (k in 0:upper)
     {prob[k+1] = dhyper(k,def,nondef,ssa)
      value[k]=k}
    prob0<-prob[1]; prob1<-prob[2]; prob2<-prob[3];prob3<-prob[4]
        cat(ssa, "\t", def, "\t",  prob0, "\t", prob1, "\t", prob2,
            "\t", prob3,"\n",file=myfile,append=T)
    }}
   hdata<-read.delim("g:/s597/houtput")
   head(hdata)
```

```
  ssa def        p0         p1          p2           p3
1   7   1 0.9899713 0.01002865 0.000000000 0.000000e+00
2   7   6 0.9411107 0.05761902 0.001258057 1.219048e-05
3   7  11 0.8943318 0.10112120 0.004448147 9.768991e-05
4   7  16 0.8495603 0.14075550 0.009355982 3.219856e-04
5   7  21 0.8067238 0.17673380 0.015779810 7.424871e-04
6   7  26 0.7657521 0.20925960 0.023529940 1.408978e-03
```

METHOD 3

```
 #  Here we do it showing how you can create a matrix and write to
 #  the matrix.

  popsizea <- 698
  upper <-4
```

```
 prob <- rep(NA,upper)
 value <- rep(NA,upper)
 ssavalues<-seq(7,70,by=7)
 numberssa <-length(ssavalues)
 dvalues<-seq(1,101,by=5)
 numberdef<-length(dvalues)
 total = numberssa*numberdef
 total
 data<-matrix(NA,total,6)  # create a total x 6 matrix with NA's for entries
  index=0
  for (j in 1:numberssa)
  {ssa= ssavalues[j]
  ssrate=ssa/popsizea
   for (m in 1:numberdef)
    {def = dvalues[m]
    defrate=def/popsizea
    nondef = popsizea-def
    for (k in 0:upper)
    {prob[k+1] = dhyper(k,def,nondef,ssa)
     value[k]=k}
    index=index+1
    data[index,1]<-ssa
    data[index,2]<-def
    data[index,3]<-prob[1]
    data[index,4]<-prob[2]
    data[index,5]<-prob[3]
    data[index,6]<-prob[4]
   }}
 data

      [,1] [,2]         [,3]         [,4]        [,5]         [,6]
 [1,]    7    1 9.899713e-01 0.0100286533 0.000000000 0.000000e+00
 [2,]    7    6 9.411107e-01 0.0576190211 0.001258057 1.219048e-05
 [3,]    7   11 8.943318e-01 0.1011212164 0.004448147 9.768991e-05

 ...

[208,]   70   91 3.240206e-05 0.0003836452 0.002210053 8.256321e-03
[209,]   70   96 1.752117e-05 0.0002209048 0.001355834 5.399683e-03
[210,]   70  101 9.422897e-06 0.0001261740 0.000822874 3.484018e-03



 METHOD 4
#  Here we do it showing how you can create vectors
#  write to them and then bind them to a dataframe
 popsizea <- 698
 upper <-4
 prob <- rep(NA,upper)
  ssavalues<-seq(7,70,by=7)
 numberssa <-length(ssavalues)
```

```
  dvalues<-seq(1,101,by=5)
  numberdef<-length(dvalues)
  total = numberssa*numberdef
ssav<-rep(NA,total)
defv<-rep(NA,total)
p0v<-rep(NA,total)
p1v<-rep(NA,total)
p2v<-rep(NA,total)
p3v<-rep(NA,total)
     index=0
   for (j in 1:numberssa)
   {ssa= ssavalues[j]
   ssrate=ssa/popsizea
    for (m in 1:numberdef)
     {def = dvalues[m]
     defrate=def/popsizea
     nondef = popsizea-def
     for (k in 0:upper)
      {prob[k+1] = dhyper(k,def,nondef,ssa)
        }
     index=index+1
     ssav[index]<-ssa
     defv[index]<-def
     p0v[index]<-prob[1]
     p1v[index]<-prob[2]
     p2v[index]<-prob[3]
     p3v[index]<-prob[4]
    }}
  hdata<-cbind(ssav,defv,p0v,p1v,p2v,p3v)
  head(hdata)

     ssav defv       p0v        p1v         p2v          p3v
[1,]    7    1 0.9899713 0.01002865 0.000000000 0.000000e+00
[2,]    7    6 0.9411107 0.05761902 0.001258057 1.219048e-05
[3,]    7   11 0.8943318 0.10112122 0.004448147 9.768991e-05
[4,]    7   16 0.8495603 0.14075555 0.009355982 3.219856e-04
[5,]    7   21 0.8067238 0.17673382 0.015779805 7.424871e-04
[6,]    7   26 0.7657521 0.20925958 0.023529938 1.408978e-03
```

Power example: This gets the power function and plots it for a one sided test for the mean. See page 31. Shows the use of the **cat** command to write out header information. Note the need for a , to separate something in quotes from a variable. These , 's are not printed as you'll see.

```
powerone<-function(mu0,sigma,n,alpha){
cat("Power example with sample size ", n," null = ", mu0,
" sigma = ", sigma, " alpha = ", alpha, "\n")
muval<- seq(25,35,by=.5)
#muval
nmu <-length(muval)
power<-rep(NA,nmu)
for(k in 1:nmu)
```

```
{mu = muval[k]
df= n-1
nc = sqrt(n)*(mu - mu0)/sigma
tval= qt(1-alpha,df)
power[k] = 1 - pt(tval,df,nc)}
plot (muval,power,type = "l",ylab="power",xlab="Null value",
main= "power function")
values<-data.frame(muval,power)
return(values)}
powerone(30,2,5,.05)

Power example with sample size  5  null =  30  sigma =  2  alpha =  0.05
   muval        power
1   25.0 1.554168e-11
2   25.5 4.549612e-10
3   26.0 1.016218e-08
4   26.5 1.720899e-07
5   27.0 2.219886e-06
6   27.5 2.193684e-05
7   28.0 1.671735e-04

19  34.0 9.748306e-01
20  34.5 9.914523e-01
21  35.0 9.975115e-01
```

Sample size determination. This gets the sample size needed for the one-sample t-test over different target values at a specified alternative, μ. See page 33.

```
ssizeone<-function(mu0,sigma,mu,alpha)
{targets <- seq(.5,.98, by=.02)
 ntarget<- length(targets)
 nv<-rep(NA,ntarget)
 pv<-rep(NA,ntarget)
 m=0
 for (j in 1:ntarget)
 {target=targets[j]
  target
  n<-2
  power<-0
  while(power < target)
   {df=n-1
   nc = sqrt(n)*(mu - mu0)/sigma
   tval= qt(1-alpha,df)
   power = 1 - pt(tval,df,nc)
   n= n+1}
 m<-m+1
 nv[m]=n-1;
 pv[m]=power;
 } #end j/target loop
 data<-cbind(targets,nv,pv)
 plot(targets,nv,xlab="target",ylab="sample size",
```

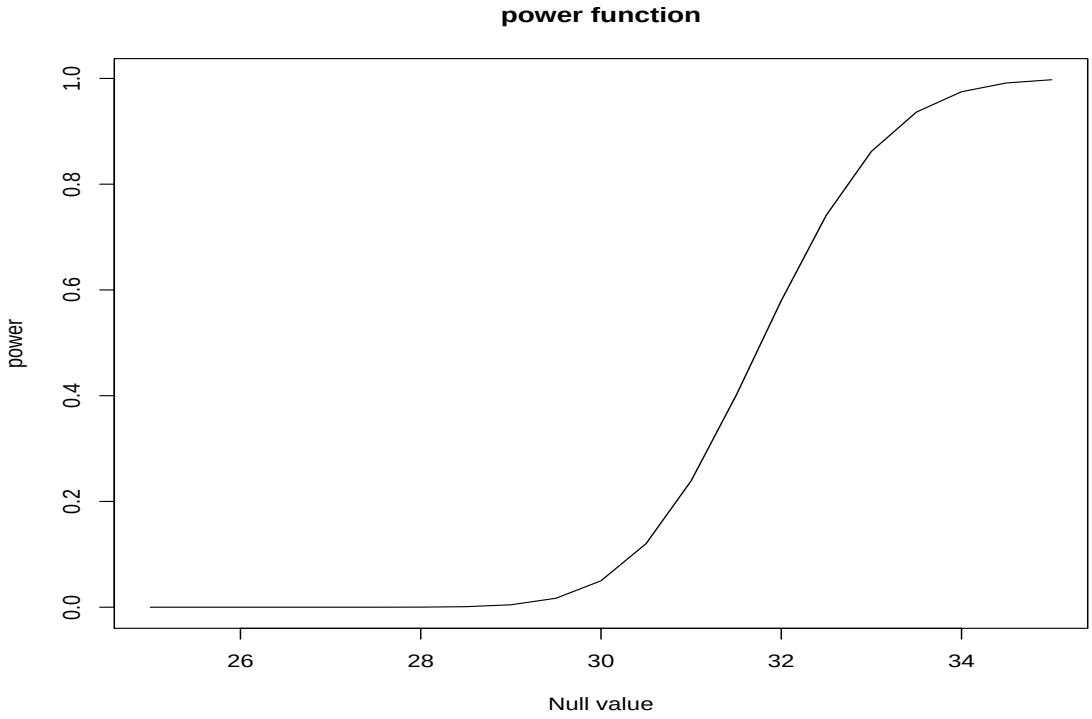

Figure 6: One-sided power using R

```
type = "b", main= " H0:mu <= 30, alt = 31, sigma=2, alpha=.05")
return(data)
} # end function.
ssizeone(30,2,31,.05)

     targets nv        pv
 [1,]    0.50 13 0.5220115
 [2,]    0.52 13 0.5220115
 [3,]    0.54 14 0.5507256
 ....
[24,]    0.96 48 0.9615312
[25,]    0.98 57 0.9814151
```

Simulating and plotting the mean mean from an exponential

```
simc<-function(mu)
{par(mfrow=c(3,2))
 nsim<-1000
 means<-rep(0,nsim)
 for (n in c(1,5,10,30,50,100))
  {for (j in 1:nsim)
     {values<-rexp(n,1/mu)
     means[j]<- sum(values)/n}
```

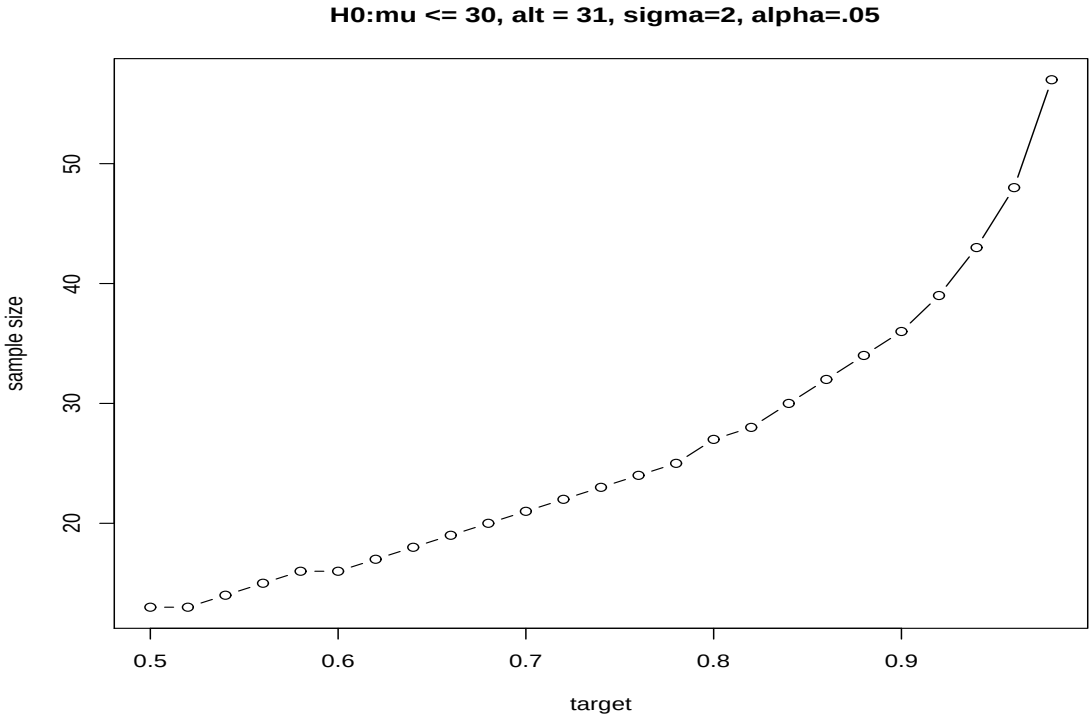

Figure 7: Sample size needed to have power = target at $\mu = 31$ for test of $H_0 : \mu \leq 30$, versus $H_A : \mu > 30$ with $\alpha = .05$.

```
 hist(means,main="mean",freq=FALSE)
 lines(density(means))}}
simc(4)
```

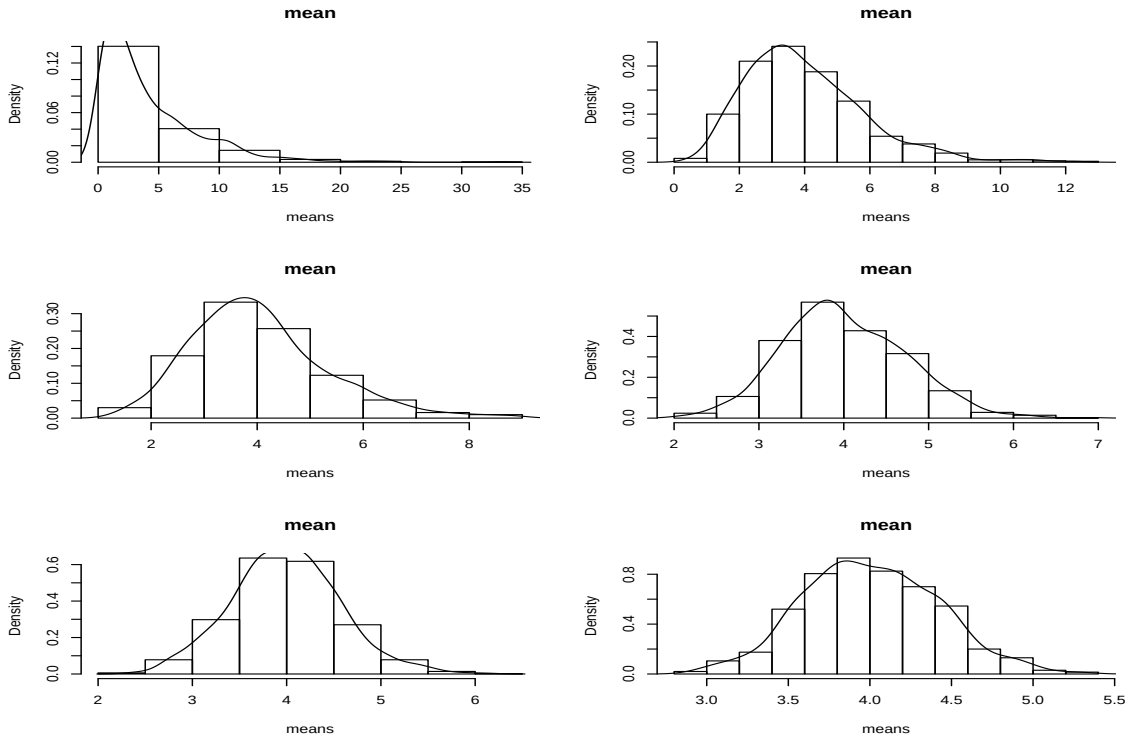

Figure 8: Distribution of sample mean with samples from the exponential; n = 1,5,10,30,50,100

2.6 Some More graphics

2.6.1 Using legends

This comes from my book ("Measurement Error: Models, Methods and applications"). It is from an example from Montgomery and Peck (Regression Analysis: 1992) for which a quadratic model was used to model the tensile strength in Kraft paper as a function of the hardwood concentration in the batch of pulp used. This program plots five different quadratic fits, one just a regular fit to the data (called naive), the other four are fits based on various methods that correct for the fact that the hardwood concentration can't be observed exactly but is estimated with some uncertainty. (This measurement error in the predictors causes bias in the fitted coefficients).

In this code the position of the legend is give by the first two arguments in the legend statement, which positions the upper left of the box with the legend at x = 0 and y = 60.

```
#postscript("g:/mecourse/paper.ps",horizontal=F,height=6,width=4.5)
# Plot of five different quadratic functions with a legend.
x<-seq(0,20,.1)
fitn<- 1.11 + 8.99*x -.44*(x**2)
fitc<- -11.03 + 13.36*x -.73*(x**2)
fitrc<- -2.95 + 10.11*x -.52*(x**2)
fitrci<- -3.99 + 11.03*x -.60*(x**2)
fits<- -1.42 + 9.77*x -.499*(x**2)
plot(x,fitn,xlab="Hardwood Concentration",ylab = "Tensile strength",
```

```
type = "l", lty=1, cex=0.8,ylim=c(0,60))
lines(x,fitc,type = "l", lty=2, cex=0.8)
lines(x,fitrc,type = "l", lty=3, cex=0.8)
lines(x,fitrci,type = "l", lty=4, cex=0.8)
lines(x,fits,type = "l", lty=5, cex=0.8)
legend(0,60,c("Naive","MOM", "RC", "RC-I", "SIMEX"),lty=1:5,cex=0.8)
```

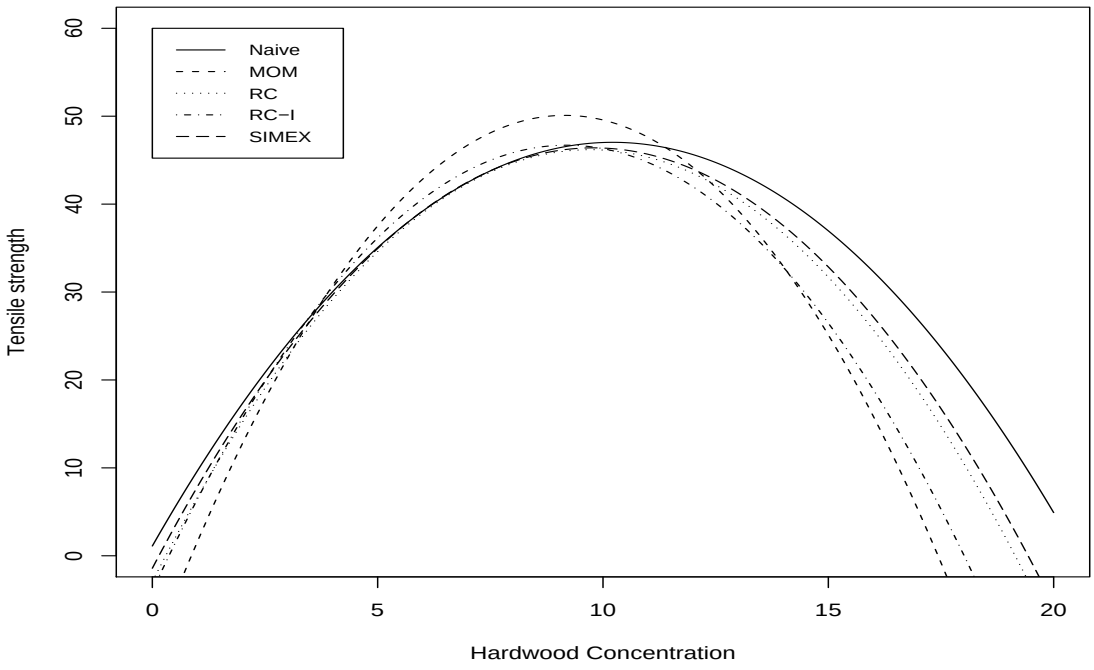

Figure 9: Naive and other fits (accounting for measurement error) of strength versus hardwood concentration

2.6.2 Writing text in margins or in graphs

You can write in the margins of the plot using mtext and in the graph itself using text.

```
#showing how to write in margins with mtext and that
#default is side = 3 (top)
par(mfrow=c(3,2))
plot(1:10, (-4:5)^2, main="Parabola Points", xlab="xlab")
mtext("10 of them ",side=1)
plot(1:10, (-4:5)^2, main="Parabola Points", xlab="xlab")
mtext("10 of them",side=2)
plot(1:10, (-4:5)^2, main="Parabola Points", xlab="xlab")
mtext("10 of them",side=3)
plot(1:10, (-4:5)^2, main="Parabola Points", xlab="xlab")
mtext("10 of them",side=4)
```

```
plot(1:10, (-4:5)^2, main="Parabola Points", xlab="xlab")
mtext("10 of them")


plot(1:10, (-4:5)^2, main="Parabola Points", xlab="xlab")
text(5,10,"THIS SHOWS HOW YOU CAN WRITE IN TEXT")
# The  text is centered at x = 5 and y = 10
```

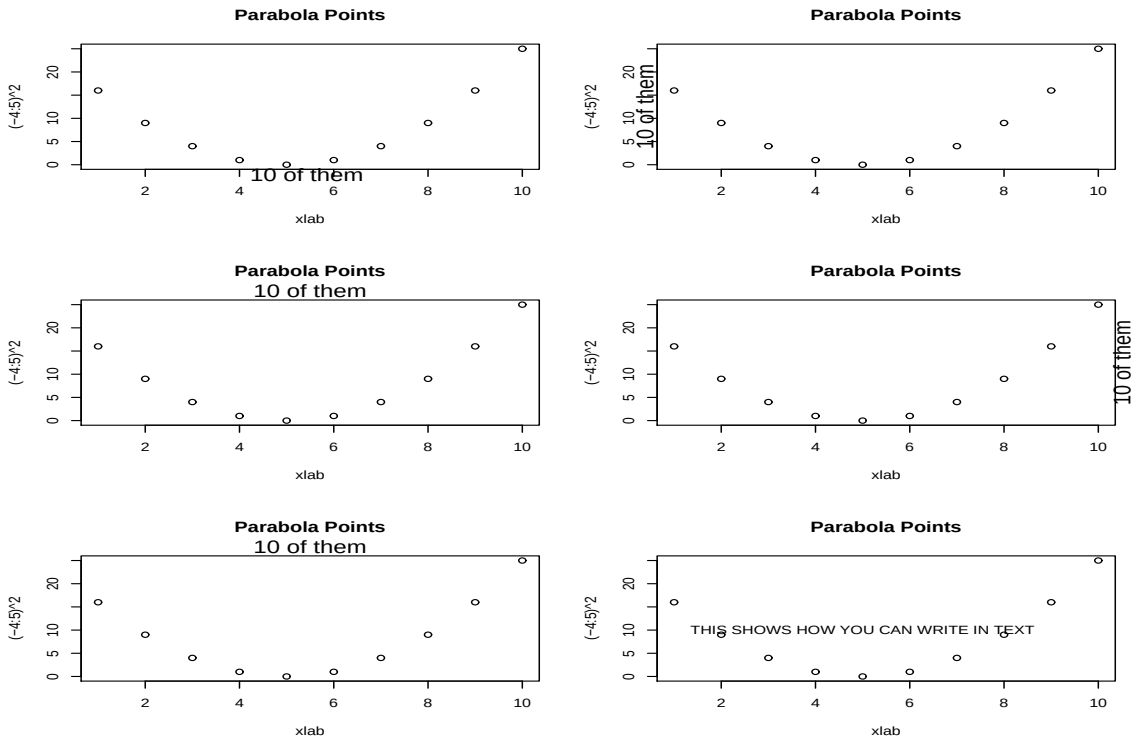

Figure 10: Unemployment plots using R

3 R: Part 3. Working with matrices

While the way SAS IML operates can be different than how we worked in the data step (which uses the so-called base language), there is no such distinction in R. As noted when we first introduced R, it basically applies functions to objects and those objects can be matrices. So, there is no real fundamental difference in handling matrices in R, compared to what we've done earlier. In fact some of our earlier examples already wrote to vectors using an index. In addition, how we referred to elements (or rows or columns) in a dataframe carries over in working with a matrix.

A matrix is an two-dimensional array with r rows and c columns. If c is equal to 1 then this is a "column" vector, or in R just referred to as a vector.

As seen earlier, if we bind together vectors of numbers using **rbind** or **cbind** then the result is a matrix. We also saw (see p. 61 part II) that we can read data right into a matrix using *matrix(scan(...* and we used the **rep** and **seq** commands to create vectors.

Creating matrices via direct assignment:

This can be done with the matrix function: the basics syntax is

```
matrix(data, nrow, ncol, byrow = F(default) or T)
```

data is a vector of values (with total elements equal to nrow*ncol; or a single value which will be assigned to all positions.
nrow is the desired number of rows.
ncol is the desired number of columns.
byrow: If FALSE (the default) the matrix is filled by columns, otherwise the matrix is filled by rows.

Here is R script for creating matrices and showing basic matrix operations. See http://www.statmethods.net/advstats/ma for more details. There are numerous other webpages that can be found with details on matrix operations.

```
# a is symmettric so the fact that it reads by column
 # doesn't matter
 a<-matrix(c(1,5,8,5,3,6,8,6,4),3,3)
 a
 #showing the need for byrow = T to read a row at a time
 b<-matrix(c(5,2,4,1,3,2,-5,6,7),3,3)
 b
 b<-matrix(c(5,2,4,1,3,2,-5,6,7),3,3,byrow=T)
 b
 M<-matrix(NA,5,6)
 M
 M<-matrix(1,5,6)
 M

# create an identity matrix
C <-diag(5)
C

 # BASIC MATRIX OPERATIONS
 sumab<-a + b     # summation
 sumab
 diffab<-a-b      # difference
 diffab
 prodab<- a %*% b # product
 prodab
 trana <-t(a)  # transpose
 trana
 ainv<-solve(a)   # inverse
 ainv
 y<-eigen(a)      # y$val has eigenvalues, y$vec has eigenvectors
 y$val
 y$vec
 deta<-det(a)    # determinant
 deta
 ranka<-rank(a)  #ranks the elements in a
 ranka
```

```
> # a is symmettric so the fact that it reads by column
>  # doesn't matter
>  a<-matrix(c(1,5,8,5,3,6,8,6,4),3,3)
>  a
     [,1] [,2] [,3]
[1,]    1    5    8
[2,]    5    3    6
[3,]    8    6    4
>  #showing the need for byrow = T to read a row at a time
>  b<-matrix(c(5,2,4,1,3,2,-5,6,7),3,3)
>  b
     [,1] [,2] [,3]
[1,]    5    1   -5
[2,]    2    3    6
[3,]    4    2    7
>  b<-matrix(c(5,2,4,1,3,2,-5,6,7),3,3,byrow=T)
>  b
     [,1] [,2] [,3]
[1,]    5    2    4
[2,]    1    3    2
[3,]   -5    6    7
>  M<-matrix(NA,5,6)
>  M
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]   NA   NA   NA   NA   NA   NA
[2,]   NA   NA   NA   NA   NA   NA
[3,]   NA   NA   NA   NA   NA   NA
[4,]   NA   NA   NA   NA   NA   NA
[5,]   NA   NA   NA   NA   NA   NA
>  M<-matrix(1,5,6)
>  M
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    1    1    1    1    1
[2,]    1    1    1    1    1    1
[3,]    1    1    1    1    1    1
[4,]    1    1    1    1    1    1
[5,]    1    1    1    1    1    1
>
> # create an identity matrix
> C <-diag(5)
> C
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    0    0    0
[2,]    0    1    0    0    0
[3,]    0    0    1    0    0
[4,]    0    0    0    1    0
[5,]    0    0    0    0    1
>
>  # BASIC MATRIX OPERATIONS
>  sumab<-a + b     # summation
```

```
>  sumab
     [,1] [,2] [,3]
[1,]    6    7   12
[2,]    6    6    8
[3,]    3   12   11
>  diffab<-a-b      # difference
>  diffab
     [,1] [,2] [,3]
[1,]   -4    3    4
[2,]    4    0    4
[3,]   13    0   -3
>  prodab<- a %*% b # product
>  prodab
     [,1] [,2] [,3]
[1,]  -30   65   70
[2,]   -2   55   68
[3,]   26   58   72
>  trana <-t(a)  # transpose
>  trana
     [,1] [,2] [,3]
[1,]    1    5    8
[2,]    5    3    6
[3,]    8    6    4
>  ainv<-solve(a)   # inverse
>  ainv
            [,1]       [,2]        [,3]
[1,] -0.14634146  0.1707317  0.03658537
[2,]  0.17073171 -0.3658537  0.20731707
[3,]  0.03658537  0.2073171 -0.13414634
>  y<-eigen(a)      # y$val has eigenvalues, y$vec has eigenvectors
>  y$val
[1] 15.513954 -1.874493 -5.639461
>  y$vec
           [,1]       [,2]        [,3]
[1,] -0.5420786  0.3354056  0.77048941
[2,] -0.5294635 -0.8483266 -0.00321535
[3,] -0.6525482  0.4096890 -0.63744469
>  deta<-det(a)
>  deta
[1] 164
>  ranka<-rank(a) #ranks the elements in a
>  ranka
[1] 1.0 4.5 8.5 4.5 2.0 6.5 8.5 6.5 3.0
```

3.1 Least squares in R

This does linear regression with two predictors, first using glm and then illustrating matrix calculations.

```
data<-read.table('g:/s597/data/smsa.dat')
rain<-data$V5
mortal<-data$V6
so2pot<-data$V16
con <-rep(1,length(mortal))
regmodel<-glm(mortal ~ rain+  so2pot)
summary(regmodel)
# doing least squares explicitly
y<-mortal
x<-cbind(con,rain,so2pot)
dimx<-dim(x)
p=dimx[2]
n=dimx[1]
n; p
xpxinv<-solve(t(x)%*%x)  #inverse of X'X
betahat<- xpxinv%*%t(x)%*%y  #estimated coefficients
residual <- y - x%*%betahat
sse <- t(residual)%*%residual; #sum of squared residuals
mse <- sse/(n-p)               # estimate of variance
covb<- mse[1]*solve(t(x)%*%x)     #estimate of variance covariance of betahat
sevec <- rep(0,p);
for (j in 1 :3)
{sevec[j] = sqrt(covb[j,j])}
info<-cbind(y,x,residual)
info
betahat
mse
 covb
estimates<-cbind(betahat,sevec)
estimates
```

```
> data<-read.table('g:/s597/data/smsa.dat')
> rain<-data$V5
> mortal<-data$V6
> so2pot<-data$V16
> con <-rep(1,length(mortal))
> regmodel<-glm(mortal ~ rain+  so2pot)
> summary(regmodel)
Call:
glm(formula = mortal ~ rain + so2pot)
Deviance Residuals:
     Min        1Q    Median        3Q       Max
-111.747   -29.239    -3.163    27.045   156.066
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 811.77692   23.13109  35.095  < 2e-16 ***
rain          2.68223    0.54710   4.903 8.22e-06 ***
```

```
so2pot        0.47648    0.09939   4.794 1.21e-05 ***
---
(Dispersion parameter for gaussian family taken to be 2307.898)
> # doing least squares explicitly
> y<-mortal
> x<-cbind(con,rain,so2pot)
> dimx<-dim(x)
> p=dimx[2]
> n=dimx[1]
> n; p
[1] 60
[1] 3
> xpxinv<-solve(t(x)%*%x)  #inverse of X'X
> betahat<- xpxinv%*%t(x)%*%y  #estimated coefficients
> residual <- y - x%*%betahat
> sse <- t(residual)%*%residual; #sum of squared residuals
> mse <- sse/(n-p)               # estimate of variance
> covb<- mse[1]*solve(t(x)%*%x)     #estimate of variance covariance
>                              # matrix of coefficients
> sevec <- rep(0,p);
> for (j in 1 :3)
+ {sevec[j] = sqrt(covb[j,j])}
> info<-cbind(y,x,residual)
> info
            y con rain so2pot
 [1,]  921.87   1   36     59  -14.579593
 [2,]  997.87   1   35     39   73.632241
 [3,]  962.35   1   44     33   16.831034
 ...........
[58,]  895.70   1   65      8  -94.233833
[59,]  911.82   1   62     49  -89.602822
[60,]  954.44   1   38     39   22.155545
> betahat
              [,1]
con    811.7769190
rain     2.6822319
so2pot   0.4764801
>  mse
         [,1]
[1,] 2307.898
>  covb
               con          rain       so2pot
con    535.0473586 -11.841137326 -0.782642347
rain   -11.8411373   0.299317271  0.006553182
so2pot  -0.7826423   0.006553182  0.009878042
> estimates<-cbind(betahat,sevec)
> estimates
                         sevec
con    811.7769190 23.13109073
rain     2.6822319  0.54709896
so2pot   0.4764801  0.09938834
```

3.2 Some additional comments on using functions in R

Printing:

As noted elsewhere when working within a function just listing an object will not result in it being printed to the console (as would be done if working interactively;i.e. not in a function). Within a function you can print to the console using print( ) or the cat command.

Return. *return(object)* will end the function and print what is in the object to the console. There doesn't have to be a return in a function (see earlier examples). You CANNOT return multiple quantities by using return(obj1,obj2,..). This does not work. Those individual quantities have to be combined into one object. To illustrate, below is the function we used to find power values for a one-sided test for a mean; see page 72 for the output.

```
powerone<-function(mu0,sigma,n,alpha){
cat(‘‘Power example with sample size ‘‘, n,’’ null = ‘‘, mu0,
‘‘ sigma = ‘‘, sigma, ‘‘ alpha = ‘‘, alpha, ‘‘\n’’)
muval<- seq(25,35,by=.5)
muval
nmu <-length(muval)
power<-rep(NA,nmu)
for(k in 1:nmu)
{mu = muval[k]
df= n-1
nc = sqrt(n)*(mu - mu0)/sigma
tval= qt(1-alpha,df)
power[k] = 1 - pt(tval,df,nc)}
plot (muval,power,type = ‘‘l’’,ylab=’’power’’,xlab=’’Null value’’,
main= ‘‘power function’’)
values<-data.frame(muval,power)
return(values)}
powerone(30,2,5,.05)
```

Below we see what we get if use a list (with no names) and then a list with names to return muval, power and some other quantities.

```
values<-list(mu0,sigma,n,alpha,muval,power)
return(values)
Power example with sample size  5  null =  30  sigma =  2  alpha =  0.05
[[1]]
[1] 30

[[2]]
[1] 2

[[3]]
[1] 5

[[4]]
[1] 0.05

[[5]]
```

```
 [1] 25.0 25.5 26.0 26.5 27.0 27.5 28.0 28.5 29.0 29.5 30.0 30.5 31.0 31.5 32.0
[16] 32.5 33.0 33.5 34.0 34.5 35.0

[[6]]
 [1] 1.554168e-11 4.549612e-10 1.016218e-08 1.720899e-07 2.219886e-06
 [6] 2.193684e-05 1.671735e-04 9.901027e-04 4.598786e-03 1.692908e-02
[11] 5.000000e-02 1.201795e-01 2.389952e-01 4.008470e-01 5.797374e-01
[16] 7.414620e-01 8.619466e-01 9.364164e-01 9.748306e-01 9.914523e-01
[21] 9.975115e-01

values<-list(null = mu0,sigma = sigma,n = n, alpha = alpha,
muval = mu,power = power)
return(values)

Power example with sample size  5  null =  30  sigma =  2  alpha =  0.05
$null
[1] 30

$sigma
[1] 2

$n
[1] 5

$alpha
[1] 0.05

$muval
[1] 35

$power
 [1] 1.554168e-11 4.549612e-10 1.016218e-08 1.720899e-07 2.219886e-06
 [6] 2.193684e-05 1.671735e-04 9.901027e-04 4.598786e-03 1.692908e-02
[11] 5.000000e-02 1.201795e-01 2.389952e-01 4.008470e-01 5.797374e-01
[16] 7.414620e-01 8.619466e-01 9.364164e-01 9.748306e-01 9.914523e-01
[21] 9.975115e-01
```

Using the object outside of the function.

Once you run a function, quantities computed in the function are not available, even those that are part of the object specified in the return() statement. If you want access to quantities after you have run the function, then you can save the result of the function. For example if you used

```
presults<-powerone(30,2,5,.05)
```

then presults will have whatever is in the return (values in the example above). Note that you could use the name values again. That is, you could use

```
values<-powerone(30,2,5,.05)
```

and this will return what is values in the function to the object values outside the function. As in general, you cannot refer to the things inside values individually. It if is a dataframe then you can use attach, or

if it is a dataframe or a list you can refer to items using *presults$name*. So, in the first case above where values was just a dataframe that came from binding muval and power you could used *presults$muval* or *presults$power* to refer to the individual vectors. In the case of using the list with names, if you used you can refer to *presults$sigma*, etc.