Applied Statistics using S-PLUS: Short course

Sung Eun Kim, PhD
Department of Mathematics and Statistics
California State University, Long Beach
Long Beach, CA 90840

sung.kim@csulb.edu

Revised 2010

Introduction

What is S-PLUS?

> S is a language and an interactive programming environment for data analysis and graphics. The S language is a very high-level language for specifying computations. The language is part of an interactive environment: S encourages you to compute, look at the data, and program interactively, with quick feedback to enable you to learn and understand. Refer to Becker et al., 1988 for more details.

The best way to learn to use the S-PLUS language is to just using it! It is possible to use S-PLUS with the same techniques used in other programming language such as FORTRAN (though this may not be the most efficient way to use S-PLUS).

Newer version of S-PLUS for Windows (user friendly interface) was designed for easy ("click-and-see"), intuitive analysis and visualization of data. This allows you to import and export data from many sources including spreadsheets like Excel and Lotus, analytical software such as SAS and SPSS, and databases. The software is quite expensive but cheaper student versions are always available. In this note, we will discuss using S-PLUS in Windows that is available in Math department (we have the newest version, V6.0).

Before you start

You are in several projects or studies and you may want to have separate S+ workspaces. It is very convenient to have different S+ shortcut incons on your desktop so that you can directly get on the project without further configuration. For this make another S+ shortcut icon on the desktop and right-click then property. On the *Target* window add the folder name you are working on (usually where data are at) as follows;

```
"C:\Program Files\Insightful\splus6\cmd\SPLUS.exe" S_PROJ=C:\Research\OzoneStudy
```

Than change the icon name (this might be done beforehand).

Once it has been done, double-click on the icon and do your work in S+ system. Your work will be saved on the folder you specified above and next time when you open the workspace everything you have done late time will be there, unless you delete them on purpose.

Basics

Getting Started

To begin S-PLUS under Windows, double click the S-plus icon on your desktop. The layout might have two windows (Object Explorer and Commands Windows). Object Explorer does the same work as the Windows Explorer and the Commands Window is where you type your commands. Many commands can be executed via user interface menus, but yet some functions run only in the Commands Window. To receive help while in S-PLUS on various commands, use

```
> help("command")
```

Note that S-PLUS is case-sensitive; that is, "s" and "S" are different in S-PLUS.

Prompt

The S-PLUS Commands Window prompt is:

```
        >
A
        +
```

prompt means that S-PLUS is waiting for you to finish your command: for example, you may have forgotten to close a function with a right parenthesis. Type

```
> 3*(2+4
```

and see what happens.

Saving and Script File

You can type your commands here one-by-one or you may execute a series of S-Plus commands from a file (with an extension *.ssc*) using *source* command:

```
> source("filename")
```

or choosing *Run* button for running commands in a Script Window. Confusing?? Well, here's how to do. First, open a Script Window by choosing *New*, *Script File*, and then *OK*. Now you have the Script Window. In upper pane type

```
x<- 1:10    # this will create a vector x with the element 1,2,3,…,10
mean(x)     # this will return the mean value of the vector x
stdev(x)    # this will return the standard deviation of the vector x
```

Now click on the Run button (the right triangular arrow). You will see the calculated result in the lower pane. If you want to save the codes into a file, choose *File* from main menu then *Save As*.
More examples for using a Script Window will be discussed in class. If you type commands one-by-one in the Commands Window, you may want to record the resulted output and/or the commands you just typed into a file. For this we use *sink* command:

```
> sink("filename")
>…….          # all the commands you want to
>…….          # save of the outputs here
> sink( )
```

The *record* function in place of *sink* will also save the commands into the specified file. The outputs and graphs will also appear in the Object Explorer (Reports and Graphs folders), so that you can display on the screen and save into a file (by choosing *File-Save As*).

In-line Data Creation and Some Fun

Now, let's have some fun.

S-PLUS is primarily meant to be an interactive data analysis system. When you create data vectors and functions in an S-PLUS session, they stay in your *.Data* directory. For example, if you make the assignment

```
> x <- 1:10       # you may also type > x_1:10
[1] 1 2 3 4 5 6 7 8 9 10
```

to create a vector x with length 10 and the elements 1 through 10, and then leave S-PLUS, next time you start S-PLUS you could type

```
> x
[1] 1 2 3 4 5 6 7 8 9 10
```

and x will still be there!

You may either make a data frame objective (in vector or matrix) by typing within S-PLUS environment or reading from an external data file. Unless the data is very small, the second case is more common. Let's make a data vector x (you can name the vector as you like) of size 9 with the element 1,2,…,9.

```
> x<-c(1,2,3,4,5,6,7,8,9)
> x
[1] 1 2 3 4 5 6 7 8 9
```

Then x will return the data you typed. Let's now make a matrix xx (again, any name) of dimension 5x3

```
> xx<-matrix(nrow=5,ncol=3)
```

5 by 3 matrix format is assigned to xx. Since we didn't type in any data, if you type xx you will see

```
> xx
     [,1] [,2] [,3]
[1,]   NA   NA   NA
[2,]   NA   NA   NA
[3,]   NA   NA   NA
[4,]   NA   NA   NA
[5,]   NA   NA   NA
```

Let's type the data for each column. Note that xx[,n] means the n^{th} column of the matrix xx. Similarly, xx[k,] means the k^{th} row.

```
> xx[,1]<-c(1,2,3,4,5)
> xx[,2]<-c(6,7,8,9,10)
> xx[,3]<-c(11,12,13,14,15)
> xx
     [,1] [,2] [,3]
[1,]    1    6   11
[2,]    2    7   12
[3,]    3    8   13
[4,]    4    9   14
[5,]    5   10   15
```

The same result can be obtained by:

```
> xx<-1:15
> xx<-matrix(xx, nrow=5, ncol=3, byrow=F)
> xx
```

Reading Data from a Text File (Example)

More commonly, we import data from an external data file. Note in S-PLUS Windows version we can import data in other formats as mentioned in Introduction. The following ASCII file called *Sta131.txt* contains name of 28 students, two midterm scores, final, and total score of a graduate statistics course that I taught before.

```
Name                    HW MT.I MT.II Final  Total
Al-Mefleh, N         93.75   51    38    27    41.6
Bennett, T           91.75   80    52    33    53.7
Buyuktas, D          98.25   79    84    49    68.6
Cao, Zh              69.75   93    82    55    71.1
Chen, J             99.375   89    91    83    87.8
Choi, J              98.75   90    78    41    66.1
  ..                  ..      ..    ..    ..     ..
```

. . . . . .

The *read.table* command provides a method to bring this data into the current data frame as a *data frame objective*. An object has intrinsic which give its structure. The data will be read into the data frame objective *grade.data*. You may also easily import the data file by choosing *File*, *Import Data*, *From File*, then choosing ASCII Text File format. We will discuss more in class. I believe that, these days, many of data files have Formatted ASCII forms and so we will discuss these formats in details.

```
> grade.data<- read.table("C:/…./Sta131.txt",header=T)
> grade.data
```

```
          Name         HW MT.I MT.II Final Total
 1  Al-Mefleh, N 93.750   51    38    27  41.6
 2    Bennett, T 91.750   80    52    33  53.7
 3   Buyuktas, D 98.250   79    84    49  68.6
 4        Cao, Z 69.750   93    82    55  71.1
 5       Chen, J 99.375   89    91    83  87.8
 6       Choi, J 98.750   90    78    41  66.1
 7    Corberl, T 32.500   53    46    14  31.8
 8      Dubin, J 99.750   98    74    86  87.4
 9     Habbas, Y 74.500   88    62    39  58.8
10       Hope, T 81.125   67    61    57  62.6
11       Kang, S 97.375   77    77    67  74.5
12     Knight, K 96.250   39    36    12  31.9
13        Lam, T 96.750   33    67    17  39.8
14         Li, B 98.125  100    74    82  85.9
15        Lin, D 99.250   90    98    95  95.0
16       Love, B 99.750   96    98    88  93.2
17       Peck, L 98.625   97    92    94  94.7
18       Pham, H 97.000   86    97    53  74.7
19      Quian, P 98.625   95    87    55  75.6
20      Singh, R 99.750  100    98    90  95.0
21      Smith, J 78.000   74    68    50  62.3
22         Su, X 97.250   95   100    99  98.2
23  Sundstrom, K 99.250   57    61    80  72.5
24        Wai, N 98.875   87   100    99  96.5
25      White, J 99.250   70    70    57  67.1
26       Wong, G 95.375   88    65    68  74.6
27      Zhang, L 93.500   64    72    66  69.7
28 McConville, L 91.125   93   100    63  80.9
```

More detailed discussions on Importing and Exporting data are in the following section.
We can examine some of the properties of *grade.data*. It is a list object, made up of other objects. Objects can be *atomic* (with *mode* "logical", "numerical", "complex", or "character") or *non-atomic* (comprising other objects). The object grade.data is a *recursive*; that is, it is a list made up of several component objects. Let see several ways of accessing the parts of grade.data.

```
> mode(grade.data)
[1] "list"
> length(grade.data)
[1] 6
> grade.data[5]
   Final
 1    27
 2    33
  …    …
27    66
28    63

> grade.data[1,]
          Name    HW MT.I MT.II Final     Total
1  Al-Mefleh,N 93.75   51    38    27      41.6
```

```
> grade.data[,"Final"]
 [1] 27 33 49 55 83 41 14 86 39 57 67 12 17 82 95 88 94
[18] 53 55 90 50 99 80 99 57 68 66 63

> grade.data[1,1]
[1] Al-Mefleh,N

> grade.data[1,5]
[1] 27

> grade.data[grade.data[,"Final"]>80,]
        Name  HW... MT.I MT.II Final Total....
 X5  Chen, J 99.375   89    91    83      87.8
 X8 Dubin, J 99.750   98    74    86      87.4
X14    Li, B 98.125  100    74    82      85.9
X15   Lin, D 99.250   90    98    95      95.0
X16  Love, B 99.750   96    98    88      93.2
X17  Peck, L 98.625   97    92    94      94.7
X20 Singh, R 99.750  100    98    90      95.0
X22    Su, X 97.250   95   100    99      98.2
X24   Wai, N 98.875   87   100    99      96.5
> summary(grade.data[,"Final"])
 Min. 1st Qu. Median  Mean 3rd Qu. Max.
   12      47     60 61.39   83.75   99
```

Simple Linear Regression

Suppose that we are interested in relating the first midterm (MT.I) to the final score (Final). We can use the *lm* function to do the linear regression analysis. What returns from *lm* is a recursive object from which information about the regression can be extracted. Those information will be included in the object *grade.fit* (you name it).

The summary function can be used to print out the contents of the object. In the formula, the response variable will be placed on the left of a ~ operator and explanatory variable on the right. In case of multiple regression, the + sign will be placed between the explanatory variables. The constant term is in the model by default. na.action is a function to filter missing data. na.omit will delete observations that contain one or more missing values.

```
> grade.fit<-lm(formula = Final ~ MT.I, data = grade.data, na.action
+       = na.omit)
> summary(grade.fit)

Call: lm(formula = Final ~ MT.I, data = grade.data, na.a
ction
       = na.omit)
Residuals:
    Min     1Q Median    3Q   Max
 -30.58 -12.51 0.8994 11.07 40.67

Coefficients:
              Value Std. Error  t value Pr(>|t|)
(Intercept) -16.3021  15.9951    -1.0192   0.3175
       MT.I   0.9760   0.1958     4.9858   0.0000

Residual standard error: 19.09 on 26 degrees of freedom
Multiple R-Squared: 0.4888
F-statistic: 24.86 on 1 and 26 degrees of freedom, the p
-value is 0.00003488
```

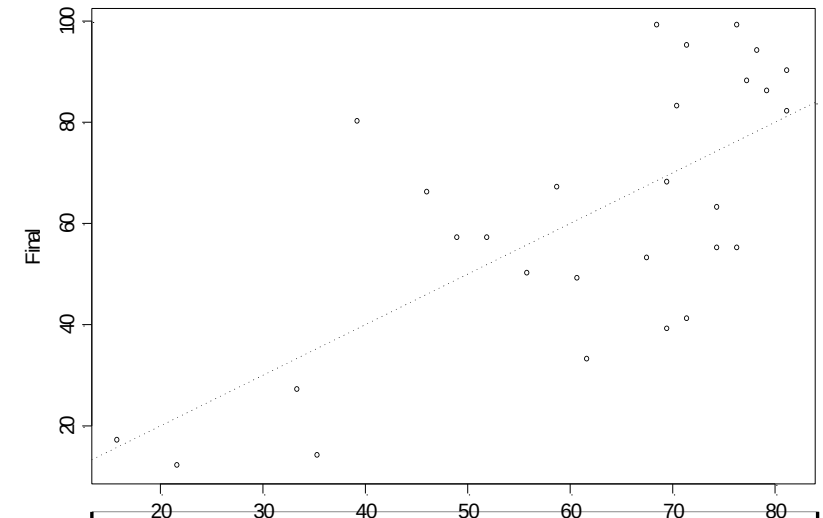
```
Correlation of Coefficients:
     (Intercept)
MT.I -0.9742
```

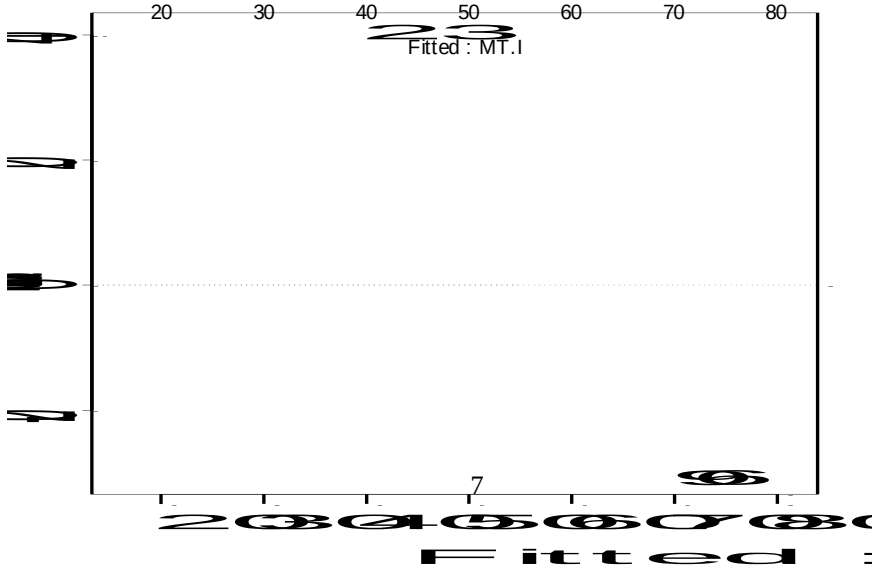
Following is to obtain Analysis of Variance table. The input to *anova* are the objects resulting from model-fitting function *lm*.

```
> anova(grade.fit)

Analysis of Variance Table

Response: Final

Terms added sequentially (first to last)
          Df Sum of Sq  Mean Sq  F Value         Pr(F)
     MT.I  1  9055.450 9055.450 24.85862 0.00003488384
Residuals 26  9471.229  364.278
```

Plotting

```
> xval<-grade.data[,"MT.I"]
> yval<-grade.data[,"Final"]
> grade.res<-residuals(grade.fit)
> grade.pre<-fitted.values(grade.fit)
> plot(xval,yval,xlab="MT.I",ylab="Final")
> plot(xval,grade.res,xlab="MT.I",ylab="Residuals")
```





It is important to note that you may make a command file (a file with list of commands separated by line) and execute the file in the S-PLUS environment. For exam, for the above regression work suppose you made a file (named, say, "grade.ssc") with contents

```
grade.data<- read.table("Sta131.dat",header=T)
grade.fit<-lm(formula = Final ~ MT.I, data = grade.data, na.action
+       = na.omit)
summary(grade.fit)

anova(grade.fit)

xval<-grade.data[,"MT.I"]
yval<-grade.data[,"Final"]
grade.res<-residuals(grade.fit)
grade.pre<-fitted.values(grade.fit)

plot(xval,yval,xlab="MT.I",ylab="Final")
plot(xval,grade.res,xlab="MT.I",ylab="Residuals")
```

Then,

```
>source("grade.ssc")
```

will execute the external command file (grade.ssc). To save the results, simply type `sink("filename")` and `sink()` at the beginning and the end. For example,

```
> sink("grade.out")
> source("grade.s")
> sink( )
```

Of course, it also can be done via Script Windows as discussed earlier.

References

Becker, R.A., Chambers, J.M., and Wilks, A.R. (1988). *The New S Language*. Wadsworth and Books.

Everitt, B.S. (1994*). A Handbook of Statistical Analysis using S-PLUS*. Champman and Hall.

Krause, A and Melvin Olson (2000) *The Basics of S and S-PLUS*, Second ed., Springer

DATA STRUCTURE

VECTOR

```
> x_c(4,3,5,7,12)   #"c" stands for "concatenate" command
> x
[1]  4  3  5  7 12
> mean(x)
[1] 6.2
> stdev(x)
[1] 3.563706
> y_c(12,13,11,10)
> x_c(x,y)
> x
[1]  4  3  5  7 12 12 13 11 10
```

There are two useful commands for creating vectors (sequence, replication):

```
> a_seq(0,15,2)            # seq(lower, upper, increment)
> a
[1]  0  2  4  6  8 10 12 14

> b_rep(0,5)         # rep(pattern, number of times)
> b
[1] 0 0 0 0 0
> rep(7,5)
[1] 7 7 7 7 7
> rep(c(1,2,3),4)
[1] 1 2 3 1 2 3 1 2 3 1 2 3
> rep(c(4,5,6),c(1,2,3))
[1] 4 5 5 6 6 6
> rep(c(4,5,6),length=7)
[1] 4 5 6 4 5 6 4
```

MATRIX

Some basics for matrix have been discussed from the last handout. Here, we will discuss how to modify a matrix and some matrix calculations. We can easily add extra columns and rows by using: *cbind(matrix1,matrix2)* and *rbind(matrix1,matrix2)*. Try the followings;

```
> x_matrix(1:15,nrow=3,ncol=5,byrow=T)
> x
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
[3,]   11   12   13   14   15
> x_cbind(x,c(6,11,16))
> x
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    2    3    4    5    6
[2,]    6    7    8    9   10   11
[3,]   11   12   13   14   15   16
> x_rbind(x,17:22)
> x
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    2    3    4    5    6
[2,]    6    7    8    9   10   11
[3,]   11   12   13   14   15   16
[4,]   17   18   19   20   21   22
```

Of course, this also can be done easily using the data spreadsheet in S-Plus. Most of discussion using window interface will be discussed in class. If you have a numeric ASCII data file, using the *scan* function you can create a matrix. For example,

```
> x_matrix(scan("filename"), ncol=5,byrow=T)
```

will create a matrix x out of the file with five columns.
For a cross product, x′x, of the matrix x we use either

```
> crossprod(x)
```

or

```
> t(x) %*% x      #t(x) means the transpose of x
```

Note that a function "*>t(x) * x"* will return a element-by-element multiplication. If the matrix is a nonsingular square matrix, a function *solve* will return the inverse matrix. For the singular value decomposition *svd* function can be used.

For some more discussion in matrix modification, let's revisit our STA131.dat data. Suppose that before assigning the final grade Sung decided to modify the distribution of proportion to be 20% HW, 25% MT.I, 25% MT.II, and 30% Final. Then, he wanted to recalculate the Total score using these new proportions (I don't know what proportions I used originally, but that might be different than above).

```
> attach(grade.data)    #this function will allow you to use the simple variable names (without the
                        #messy grade.data$Final or messier grade.data["Final"]

> grade.data$Total_HW*.2+MT.I*.25+MT.II*.25+Final
> detach( )
```

Above code will replace the Total score. You may want to record the new score as another name and keep the old one in the same data object. Try this out

```
> attach(grade.data)
> Total.New_HW*.2+MT.I*.25+MT.II*.25+Final
> grade.data_cbind(grade.data, Total.New)
> detach( )
```

Now, you will have another column named *Total.New* in *grade.data.* You may want to change some of values in the data. For example, suppose that I want to change the Final score of the student named Kang, S to 78 (was 67).

```
> grade.data[11,"Final"]_78
```

Note that this change will not automatically recalculate the Total.New score of the student (Excel does it!).

Now I'd like to calculate the mean and the standard deviation of each of the scores . For this we use the *apply* function.

```
> apply(grade.data[,2:6], 2, mean)    #the number 2 in the middle means apply the function mean to
                                      #columns. You may use 1 to refer to the rows
> apply(grade.data[,2:6], 2, stdev)
```

You may also "> colMeans(grade.data[,2:6])" and "> colStdevs(grade.data[,2:6])". The functions *colVars, colSums, rowMeans, rowStdevs, rowVars,* and *rowSums* are also available.

ARRAY

Array is the extension of the matrix in dimension. That is, if the dimension of data is 2, we call it "matrix" and more than 2 then "array". Assume that in an opinion poll we obtained the following results from the two states, California and Ohio.

California

	Yes	**No**	**Don't know**
Male	100	70	10
Female	50	60	20

Ohio

	Yes	**No**	**Don't know**
Male	120	50	20
Female	40	30	30

Try this out

```
poll_array(c(100,50,70,60,10,20,120,40,50,30,20,30),c(2,3,2))
gender_c("Male","Female")
opinion_c("Yes","No","don't know")
state_c("California","Ohio")
dimnames(poll)_list(gender,opinion,state)
poll
```

```
> poll

, , California
        Yes       no   Don't know
  Male  100       70        10
Female   50       60        20

, , Ohio
        Yes       no   Don't know
  Male  120       50        20
Female   40       30        30

> poll[,,"California"]

        Yes       no  Don't know
  Male  100       70        10
Female   50       60        20

> poll["Male",,]

             California      Ohio
     Yes            100       120
      no             70        50
Don't know           10        20
```

Note that each of poll[,,"California"], poll["Male",,], and etc. is treated as matrices. Hence, all functions we used for matrix can be used. For example,

```
> apply(poll["Male",,],1,sum)

 Yes     no      Don't know
 220     120             30

> apply(poll["Female",,],1,sum)

 Yes     no      Don't know
  90     90              50
```

DATA FRAME

Data Frame is a generalized version of the matrix in the sense that the data frames allow you to mix data of different types (numeric, character, etc) into a single data object. The data frame is the most common data structure in S-Plus.
Data frame can be created via several ways. To create a data frame from an external file, *read.table* command can be used as we discussed before. You may also import the data file directly from the File menu.

```
>your.frame.name_ read.table("filename",header=T) #use header=F if your data
                                                  #doesn't contain a header row.
```

You may want to bind S-Plus data objects of various kinds into a data frame. In the case use *data.frame* command.

```
> your.frame.name_data.frame(object1,object2,….)
```

Consider the following simple example.

```
> rdm_rnorm(10)     #this wiil generate random sample of size 10 from N(0,1)
> assgn <- rep(NA, 10)
> assgn[rdm > 0] <- "A"
> assgn[rdm <= 0] <- "B"
> assgn1 <- cbind(rdm, assgn)
> assgn1
                      rdm assgn
 [1,] "0.049086461868886"  "A"
 [2,] "0.918544752343534"  "A"
 [3,] "-0.126559993056653" "B"
 [4,] "0.33915786263237"   "A"
 [5,] "1.43901265660863"   "A"
 [6,] "0.0279969566945071" "A"
 [7,] "-1.47322223095435"  "B"
 [8,] "-0.668673135299956" "B"
 [9,] "-0.345348406178856" "B"
[10,] "0.0884835844457507" "A"

> max(assgn1[, 1])
Problem in max(assgn1[, 1]): Numeric summary undefined for mode "character"
```

This will create a matrix using *cbind* command. By binding numeric with character variable, the variable *rdm* is no longer numeric. Hence, some numeric calculations, like *max* and *min*, are no longer possible. It is interesting to note that some numeric calculations, like *mean* and *stdev*, are still working.

```
> mean(assgn1[, 1])
[1] 0.02484785
```

Now, let create a data frame, which will keep the original mode of the variables.

```
> assgn2 <- data.frame(rdm, assgn)
> assgn2
          rdm assgn
 1 -0.7615447     B
 2 -0.4632235     B
 3  1.8197463     A
 4 -0.1927243     B
 5 -0.1923333     B
 6 -0.1962358     B
```

```
 7 -1.5823391     B
 8 -0.6647673     B
 9 -0.4132076     B
10  2.0184172     A
> max(assgn2$rdm)
[1] 2.018417
```

Any functions we used for matrix, like apply, also works for data frames. To combine data frames, use data.frame, cbind, rbind. You can merge two or more data frames using *merge* function.

> merge(dataframe1,dataframe2,…)

It can be explained by the following example. Consider that we have two data frames: the first one is general information about students in a class (provided from the university) and the second is their test scores and the final grade (prepared by the professor). We want to merge the two data from different sources.

```
> example.list
     Name  Class College
1    Chen Senior      AS
2    Choi Master      AS
3 Jackson Master     Eng
4 Johnson Master      AS
5      Lu Senior     Edu
6 Mueller Junior     Eng
7    Park Master     Edu
8    Xiao Senior      AS

> example.grade
     Name MT Final Grade
1 Jackson 85    79     A
2 Mueller 70    98     A
3      Lu 92    91     A
4    Choi 78    68     B
5 Johnson 69    66     B
6    Park 62    80     B
7    Xiao 75    78     B
8    Chen 54    66     C

> merge(example.list, example.grade, by=1) #by=1 means merging by the first col.
     Name  Class College MT Final Grade
1    Chen Senior      AS 54    66     C
2    Choi Master      AS 78    68     B
3 Jackson Master     Eng 85    79     A
4 Johnson Master      AS 69    66     B
5      Lu Senior     Edu 92    91     A
6 Mueller Junior     Eng 70    98     A
7    Park Master     Edu 62    80     B
8    Xiao Senior      AS 75    78     B
```

More discussion on Data Frames will be given later.

LIST

List is a collection of vectors or matrices of arbitrary structures. List is useful when you want to collect related data with different structures into a big list object. Note that while both data frame and matrix require rectangular, list allows different structures. Consider the following example:

```
> a_rnorm(5)  #random sample of size 5 from N(0,1)
> b_a>0
> ab_data.frame(a,b)
```

```
> c_rt(10,29)       #random sample of size 10 from t(df=20)
> d_rchisq(15,29)   #random sample of size 10 from chisquare(df=20)
> x_list(ab,c,d)
> x
[[1]]:
             a   b
1 " 0.5284931" "T"
2 "-0.5783929" "F"
3 "-0.7104860" "F"
4 " 0.1749112" "T"
5 " 0.8443500" "T"

[[2]]:
 [1]  0.1661215 -0.2214252  0.8668092 -0.1193410 -0.6454605  0.5767275
 [7]  0.7018636  0.8049501  1.1132858  0.4405738

[[3]]:
 [1] 22.74985 32.81265 47.81313 23.51771 31.96326 17.26032 31.40059
 [8] 39.21596 27.57610 23.56536 32.28030 24.04670 28.70105 37.20586
[15] 35.17972
```

To call the data frame *ab* use either x[1] or x[[1]]. Either *x[[1]]$a* or *x[[1]][[1]]* will return the vector *a* (random sample from N(0,1)).

```
> x[[1]]$a
[1]  0.5284931 -0.5783929 -0.7104860  0.1749112  0.8443500
> x[[1]][[1]]
[1]  0.5284931 -0.5783929 -0.7104860  0.1749112  0.8443500
> mean(x[[1]]$a)
[1] 0.05177509
```

WRITING FUNCTIONS

This note discusses the basic techniques for writing functions in S+ (like macro in SAS). S+ provides a number of built-in functions (over 3800) and writing your own functions can extend it. The basic syntax of a function declaration is as follows:

Your-function-name <- function (*arguments*)
{ *function body*
return (*output arguments*) }

To call the function, type
> your-function-name (arguments).

As a simple example consider writing a function to calculate the two-sided p-value of a t statistic (text p3).

```
t.test.p_function (x, mu=0, alpha=0.05)
{ n_length(x)
  t_sqrt(n) * (mean(x) - mu) /stdev(x)
  p_2 * (1-pt(abs(t), n-1))
  if (p<alpha) result_"Reject Ho" else result_"Do Not Reject Ho"
  return(t,p, result) }
```

Here, *x* is data, *mu* is the mean under the null hypothesis, and *alpha* is the level of significance. To call this function for a t-test for Ho: $\mu=1$ at $\alpha=.01$,

```
t.test.p(data, 1, 0.01)
```

If you don't specify the last two arguments, S+ will use the default values of *mu=0* and *alpha=0.05*. Let's use the function to test if the mean Final score of STA131 class is 67 (the mean from last year) or not.

```
> z_grade.dat$Final
> t.test.p(z,67)

$t:
[1] -1.11745

$p:
[1] 0.2736473

$result:
[1] "Do Not Reject Ho"
```

The calculated t statistic and the corresponding p-value and the test result are printed above. Since we didn't specify the level of significance, the test is performed with the default value of 0.05. Since the p-value of the test is not less than 0.05, we do not reject the null hypothesis, Ho: μ=67, at α=.05. In deed, S+ has a built-in function for a t-test, for both one and two sample.

```
> t.test(z, mu=67)

One-sample t-Test

data:  z
t = -1.1174, df = 27, p-value = 0.2736
alternative hypothesis: true mean is not equal to 67
95 percent confidence interval:
51.29976 71.62881
sample estimates:
mean of x
61.46429
```

Functions also can be used to define your own function name of a built-in function. For example, suppose you use the logarithm function with a base 2 very often. The built-in function expression is *logb(x, 2).* You want to simplify the expression.

```
> log2 <- function(x) { return (logb(x, 2)) }
> log2(16)
[1] 4
```

To get the on-line help for the argument names and default values for S-function, type

> args(*function-name*).

For example,

```
> args(t.test)
function(x, y = NULL, alternative = "two.sided", mu = 0, paired = F,
var.equal = T, conf.level = 0.95)
```

If you want to perform one-sided (left) two-sample t-test for non-paired and unequal variances, type

```
> t.test(x,y, alternative="less", paired=F, var.equal=F)
```

Elementary Functions: See handout

***is* and *as* functions**

Before applying functions to our data, we may want to check if our data has the particular type that the function argument required. Doing so, you can protect against unexpected error message.

```
> is(x,"numeric")
```

Consider the following function, which returns mean for number variable.

```
Just.fun_function(x)
   { if(!is(x,"numeric")) stop ("Oops! The variable must be numeric.")
   else return(mean(x)) {
```

is(x,"numeric") is a logical function that will return T if *x* is numeric. The complementary expression is *!is(x,"numeric")* that will return T if *x* is not numeric. Now the variable *gender* is a factor with elements "Male" and "Female".

```
> Just.fun(gender)

Problem in Just.fun(gender): Oops! The variable must be numeric.
```

You can use the function to test for "array", "character", "complex", "data.frame", "double", "factor", "integer", "list", "logical", "matrix", "numeric", "single", and "vector".

as function is used to coerce the variable to have the specified data type. For example,

```
> as (x, "character")
```

will coerce the variable x to have a character data type. Consider the data with 12-hour pollutant volume at a certain day with code "9999" for a missing value.

```
> pollut
[1]    8    3    4    5    2    6 9999    6    7    5    4    3
> pollut[pollut==9999]_"Missing"
> pollut
[1] "8"       "3"       "4"       "5"       "2"       "6"
[7] "Missing" "6"       "7"       "5"       "4"       "3"
```

Of course, you could use the NA for the missing value (replace "Missing" with NA) keeping the variable numeric. We just assigned the character "Missing" to explain the use of as function. Since the *pollut* variable converts to character we use the *as* function to covert the variable back to numeric.

```
> pollut_as(pollut,"numeric")
Warning messages:
1 missing values generated coercing from character to numeric

> pollut
[1]  8  3  4  5  2  6 NA  6  7  5  4  3
```

Now, we want to calculate the mean omitting the missing values.

```
> mean(pollut[!is.na(pollut)])
[1] 4.818182
```

We have used the *is.na* function to determine which values are missing and to exclude the missing values from the mean calculation.

ITERATION

It is extremely important to note that S-Plus is vector or matrix oriented (Fortran and C are not!), and hence, we can mostly avoid complicated loops in iterative computations. For example we can simply use *x %*% x* for uncorrected sum of squares for variable *x* instead of using the *for* loop. We will briefly discuss *for*, *while*, and *repeat*. These loops can be used independently or in the body of a function.

The *for* loop

> for(i in *invalues*) {commands}

Consider the calculation of a uncorrected sum of square of a variable x

```
> x_c(2,1,3,4,2,3,1,5)
> y_0
> for (i in 1:length(x)) {y_x[i]^2+y; print(y)}
[1] 4
[1] 5
[1] 14
[1] 30
[1] 34
[1] 43
[1] 44
[1] 69
> y
[1] 69
```

The *while* loop

The *while* loop is used when the total number of iteration is unknown.

> while(*condition*) { commands }

For an example of the use of the while loop, we consider the maximum likelihood estimation for the parameter λ of the zero-truncated Poisson distribution as in the text p95. The probability distribution and the iterative estimation via Newton's method is in the test.

```
yp_rpois(100, 1)     #random sample of size 100 from Poisson(1)
y_yp[yp>0]           #truncate the zeros
ybar_mean(y)         #mean of the zero truncated Poisson r.v.
lam_ybar             #initial value of lambda
it_0                 #iteration count
del_1                #tolerance
while(abs(del) > .000001 && (it_it+1)<10)
   {del_(lam-ybar*(1-exp(-lam)))/(1-ybar*exp(-lam))
   lam_lam-del
   cat("The estimated lambda after", it, "th iteration is", lam, "\n")
   }
```

```
The estimated lambda after 1 th iteration is 1.148994930761
The estimated lambda after 2 th iteration is 1.06264898980154
The estimated lambda after 3 th iteration is 1.0581687548983
The estimated lambda after 4 th iteration is 1.05815588823481
The estimated lambda after 5 th iteration is 1.05815588812837
```

In the above example, the condition of the *while* loop states that the iteration continues until the tolerance limit of 0.00001 and the number of iteration of 10. To print the variable's values we use *cat* function, which is analogy to the cat function from the C language.

The *repeat* loop

> repeat {commands

stop criterion }

The repeat function repeats the commands until the stop criterion meets. We may rewrite the above code using *repeat* loop as follows;

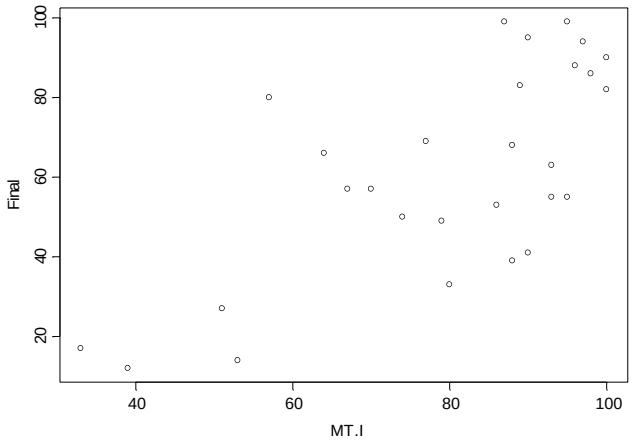
```
yp_rpois(100, 1)
y_yp[yp>0]
ybar_mean(y)
lam_ybar
it_1
del_1
repeat
    {del_(lam-ybar*(1-exp(-lam)))/(1-ybar*exp(-lam))
   lam_lam-del
   cat("The estimated lambda after", it, "th iteration is", lam, "\n")
   if(abs(del) < .00001 || (it_it+1)>10) break
   }

The estimated lambda after 1 th iteration is 1.148994930761
The estimated lambda after 2 th iteration is 1.06264898980154
The estimated lambda after 3 th iteration is 1.0581687548983
The estimated lambda after 4 th iteration is 1.05815588823481
The estimated lambda after 5 th iteration is 1.05815588812837
```

GRAPHICS

Getting Started: *plot( ), hist( )* functions, and more.

Scatter plot
Let's revisit our grade.data data.

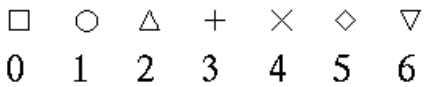
```
> attach(grade.data)
> plot(MT.I,Final)
```



We can change the character for the point by using *pch* parameter. For example,

> plot(MT.I, Final, pch="*")

will replace the points in the plot with *. S+ has a number of numeric assignments for special symbols. Of these, commonly used seven are;



> plot (MT.I, Final, pch=2)

can be used to have triangular points, etc. Numbers 7 to 14 are composite symbols formed by overprinting these symbols, and 15 to 19 are solid-filled versions of 0, 1, 2, 5, and 6.

We can also change the type of the plot. For example,

> plot (x, y, type="l")

will have line plot connecting the data. The default is type="p", the point plot. Other available types are "b", "h", "o", "s", and "n". I will leave you to explore these types. For complete list of useful options for plotting functions type > *help (par)*. Some of the most commonly used are:

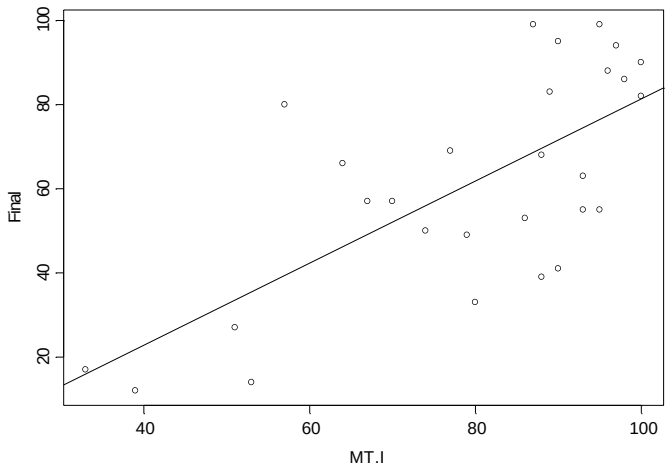
axes=T	T for with axes, F for without axes
main="*Title*"	Title string
sub="*Subtitle*"	Subtitle string
xlab="*x axis label*"	x axis label,
ylab= "*y axis label*"	y axis label,
xlim= c(*xmin, xmax*)	range for x axis,
ylim= c(*ymin, ymax*)	range for y axis.

pch="*"	point character (see above)
type="l"	plot type (see above)
lty= 1	1 for default solid line, 2 for dashed line
lwd=1	1 for default line width, 2 for twice thick, etc

We can also simply change the layout of the plot via interface (right click on the plot).

The function *abline( )* will superimpose a line on the scatter plot. For example, the least square line for regressing Final on MT.I has the intercept and slope estimates of –16.187 and 0.9754, respectively.

```
> abline(-16.187,0.9754)          # abline (intercept, slope)
```



Fitting and plotting the least square line can be done easily by using *lm* (linear model) function. The following code will return the same plot as above.

```
> fit_lm(Final~MT.I)
> abline(fit)
```

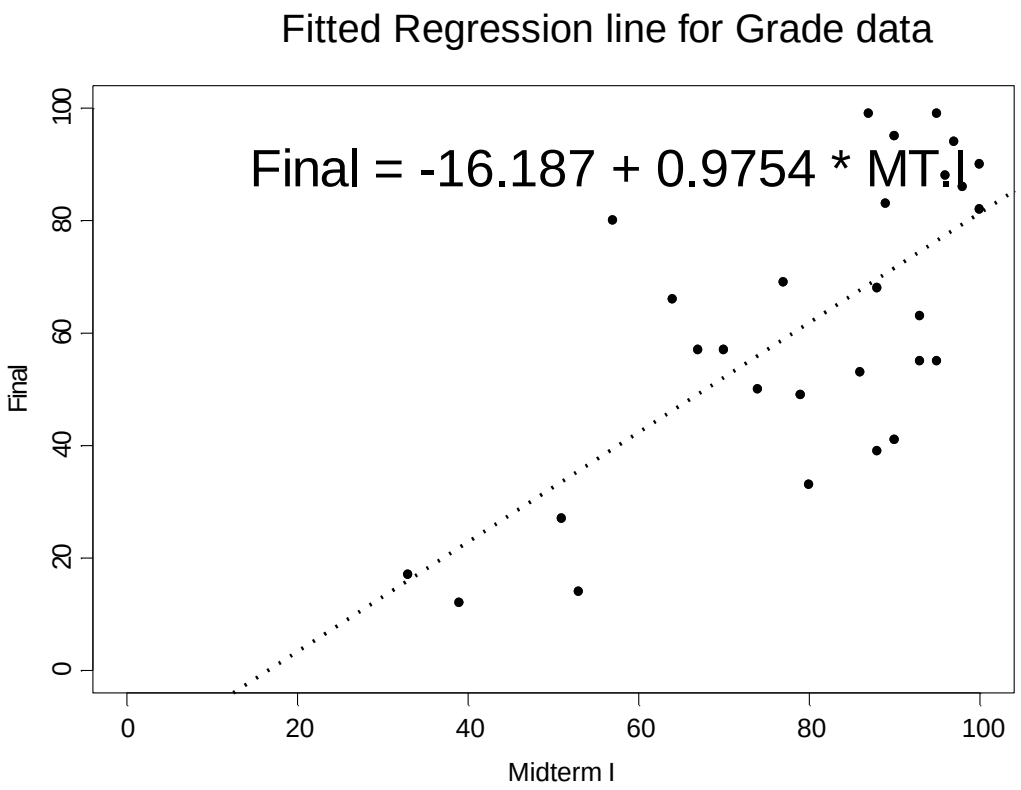
More detailed discussion on the *lm( )* function will follow.

Let put them together.

```
> plot(MT.I,Final, main="Fitted Regression line for Grade data",
+ xlab="Midterm I", ylab="Final", xlim=c(0,100), ylim=c(0,100), pch=16)
> abline(fit, lty=2,lwd=2)
> text(50, 90, "Final = -16.187 + 0.9754 * MT.I", cex=2)
```

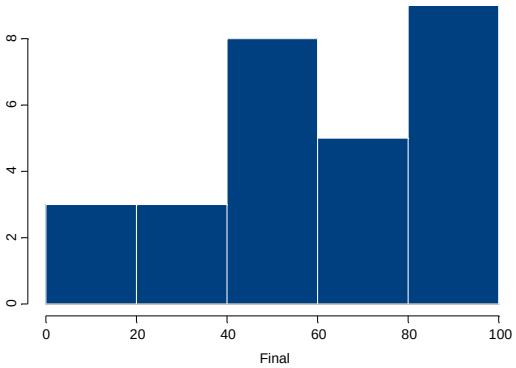
The text function will add a text at the specified location; *text(xlocation, ylocation," text")*. The option *cex=2* draws characters twice as big. You can rotate the character using the option *crt=counterclockwise degree.* The text adjustment option is adj= 0 (0 for left justified, 1 for right justified and 0.5 for centered). If you forgot to add title option in the plot function, you can add the title in the existing plot.

```
> title ("title","subtitle")
```



Histogram
The hist( ) function plots a histogram.

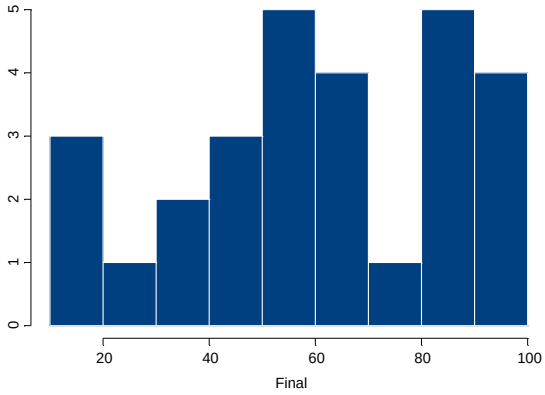
```
> hist(Final)
```



To change the number of class we use nclass option. For example,

```
>hist (Final, nclass=10)
```

will return



Check >help(hist) for more options.

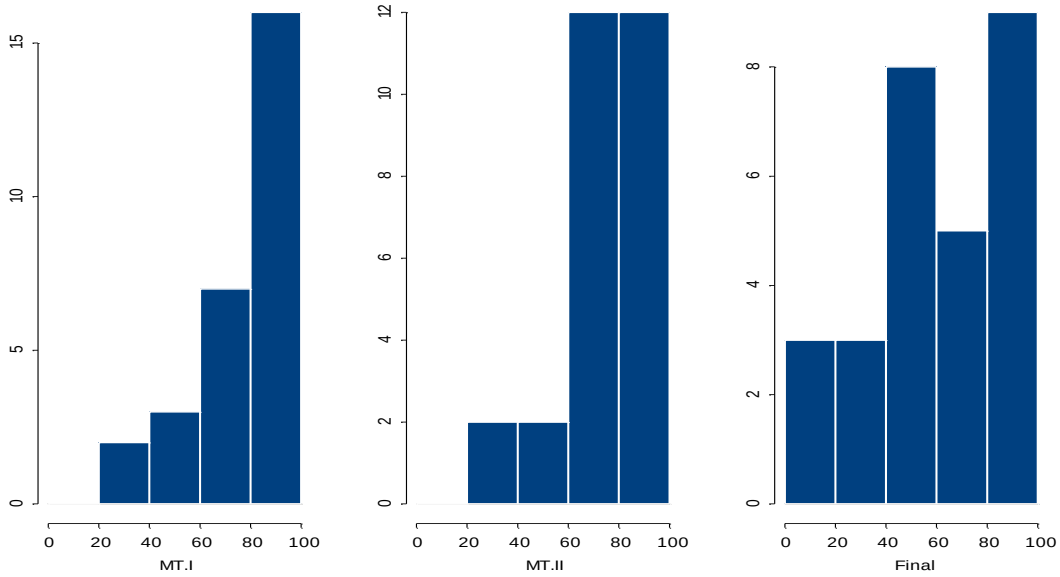
Multiple Plots: par( ) function

You can display several small plots as a table or matrix format. Use the `par(mfrow=c(`*nrow,ncol*`))` function. A 2×3 matrix of plot can be created by

```
> par(mfrow=c(2,3))
```

Try this out.

```
> par(mfrow=c(1,3))
> hist(MT.I, break=c(0,20,40,60,80,100))
> hist(MT.II, break=c(0,20,40,60,80,100))
> hist(Final, break=c(0,20,40,60,80,100))
```



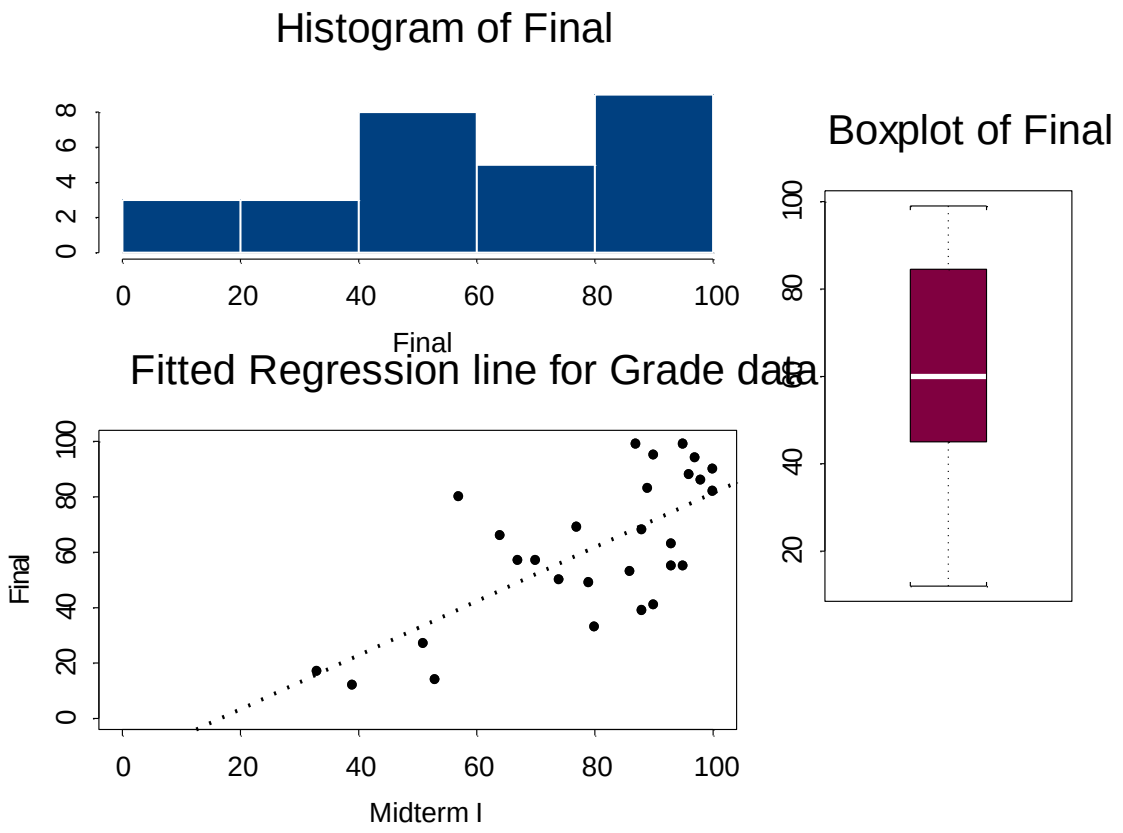
To restore the default setting type > par(mfrow=c(1,1)).

We may want customize the display of the multiple plots. It can be done using

>par(fig=c(*x1,x2,y1,y2*)

where the parameter x1, x2 are the plot location of left corner and right corner of x axis. Similarly for y axis. Note that the coordinate of the plot are set to (0,0) for the lower left corner and (1,1) for the top right corner. Let's consider the following example.

```
frame()     #opening new graph
par(fig=c(0,0.7,0,0.6))
plot(MT.I,Final, main="Fitted Regression line for Grade data", xlab="Midterm
    I", ylab="Final", xlim=c(0,100), ylim=c(0,100), pch=16)
fit_lm(Final~MT.I)
abline(fit, lty=2,lwd=4)
par(fig=c(0,0.7,0.55,1))
hist(Final,main="Histogram of Final")
par(fig=c(0.65,1,0.15,0.85))
boxplot(Final,main="Boxplot of Final")
```

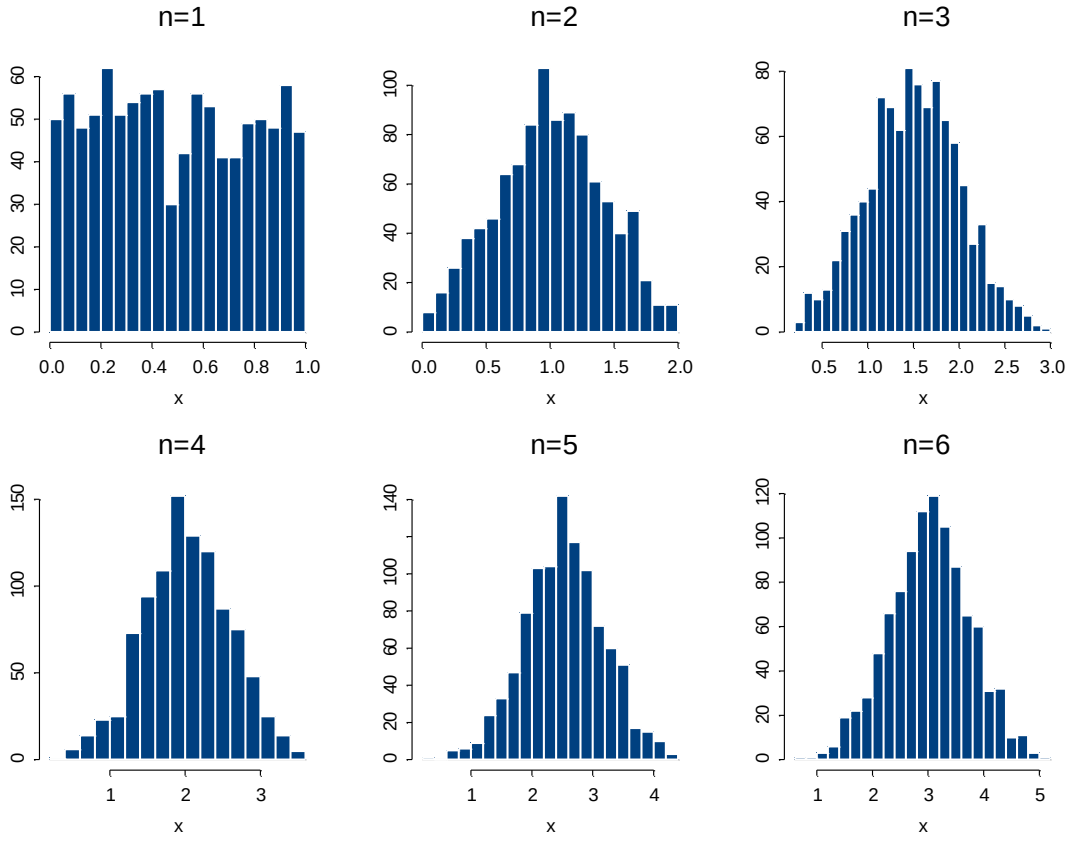


More example…

Following example demonstrates the approximate normality of the sum of continuous uniform r.v.'s.

```
par(mfrow=c(2,3))
x_runif(1000)         # random number of size 1000 from Unif(0,1)
hist(x,nclass=20,main="n=1")
x_runif(1000)+x
hist(x,nclass=20,main="n=2")
```

```
x_runif(1000)+x
hist(x,nclass=20,main="n=3")
x_runif(1000)+x
hist(x,nclass=20,main="n=4")
x_runif(1000)+x
hist(x,nclass=20,main="n=5")
x_runif(1000)+x
hist(x,nclass=20,main="n=6")
```



TRELLIS GRAPHICS

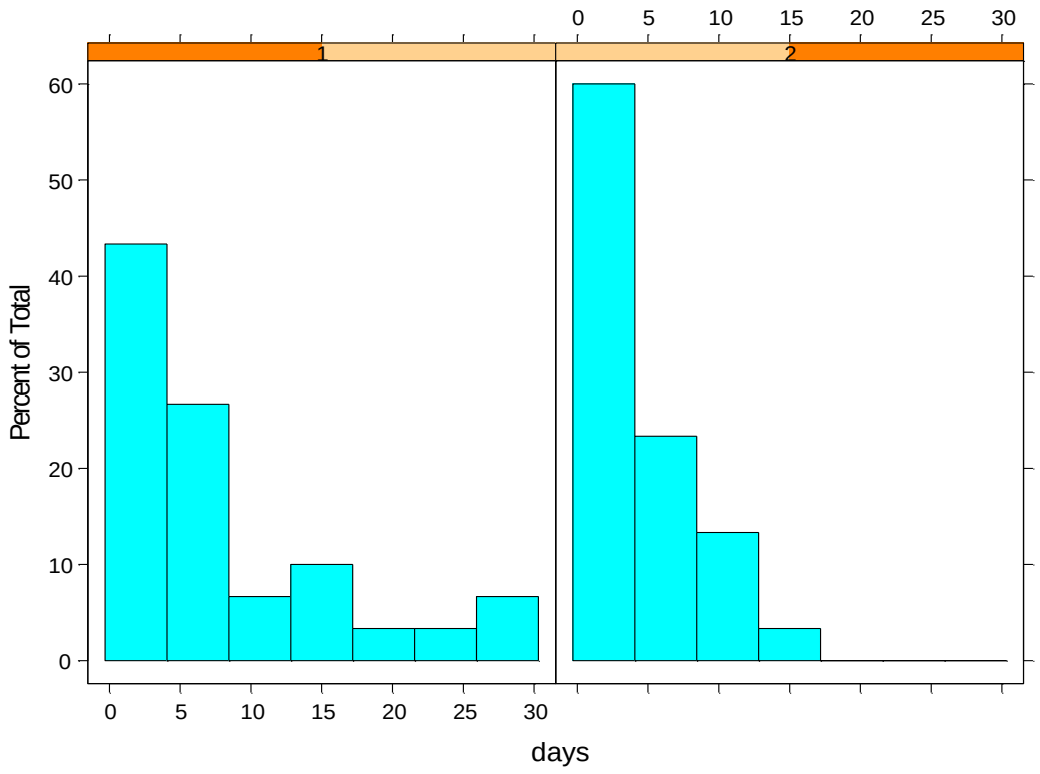
Once data is categorized by one or more factors, we might want to have separated graphs conditioned on one or more factors. Let's consider an example of drawing a Trellis histogram.

Example.

An experiment is designed to study the effects of treatment duration (factor A) and weight gain between treatment (factor B) on the number of days hospitalized (response, named *days*) for kidney failure patients. The factor A has two levels (1.short, 2.long) and the factor B has three levels (1.Mild, 2.Moderate, 3.Severe). A random sample of 10 patients per each of the 6 groups is collected. You can download the data from the class web site (kidney.dat).

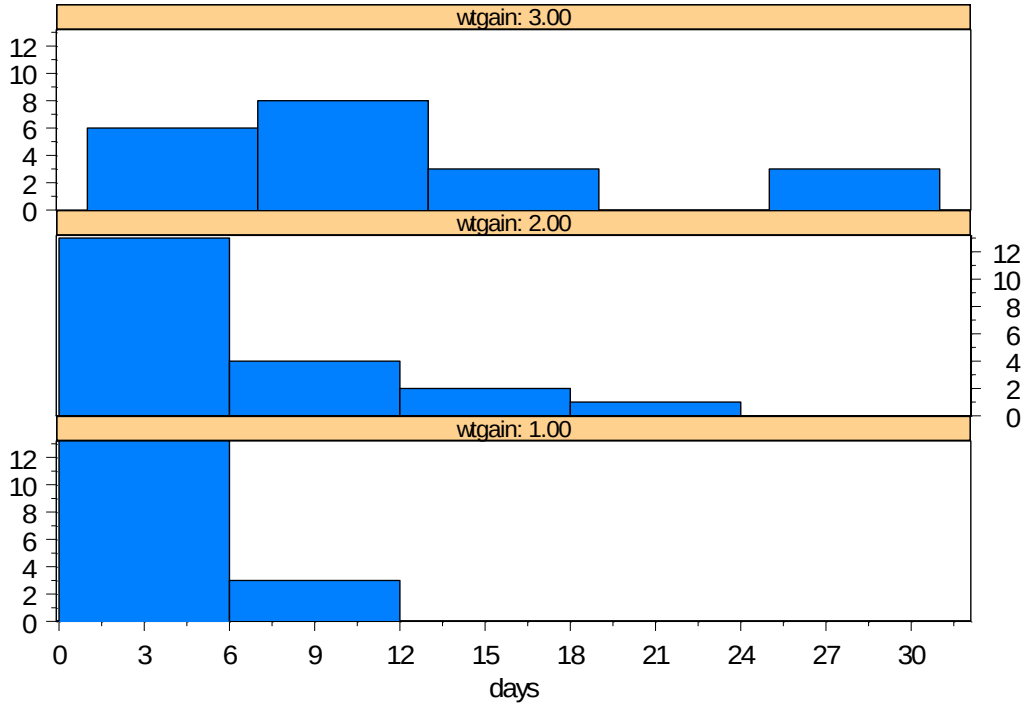
First, we would like to see the distribution of *days* conditioned on the *duration.*

```
> histogram(~days|duration, data=kidney)
```

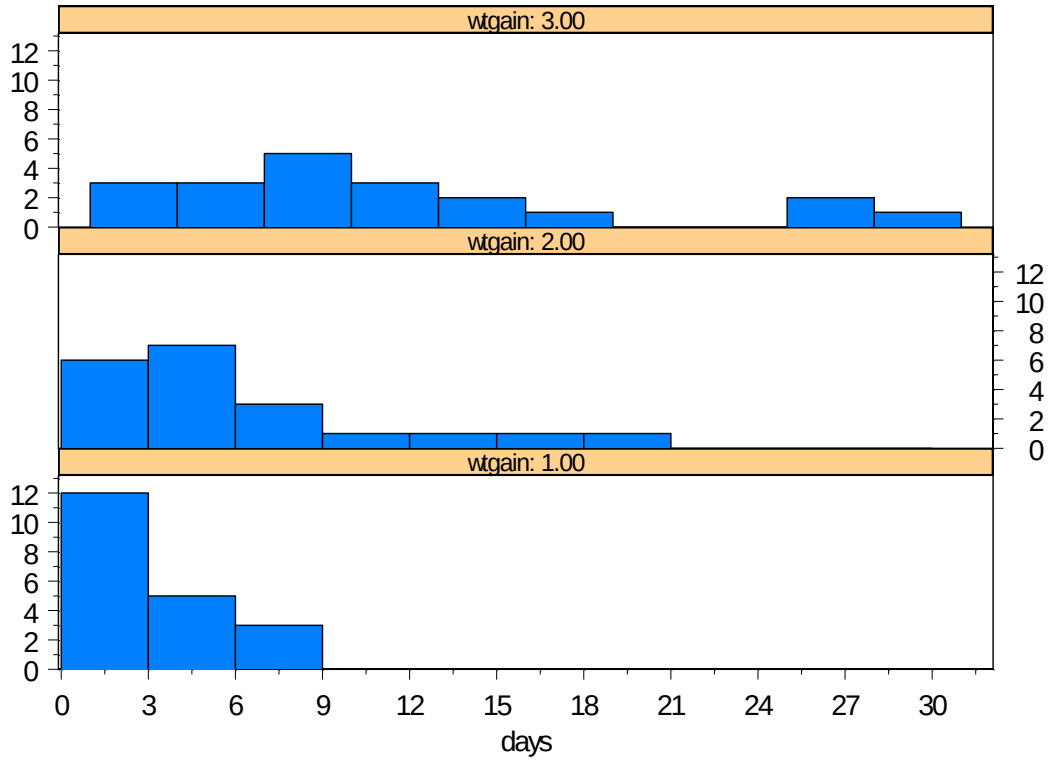


This also can be easily done using the user interface. First, open the data object, then mark the first two columns (days and duration). Then, choose the *Plots 2D* button on the menu bar then *Set Conditioning Model* button. You may see small yellow bars appeared on the *Plots 2D* palette. Then choose 1 for *the # of conditioning columns*. Then, choose *Histogram* on the *Plots 2D* palette. Once you have the Trellis plot via the interface you can easily change the conditioning column by just interacting between the Data Window and the Graph Window. To change the conditioning columns mark the columns in the Data Window and drag to the top of the Graph Window. You may choose multiple columns. We will discuss more in class.

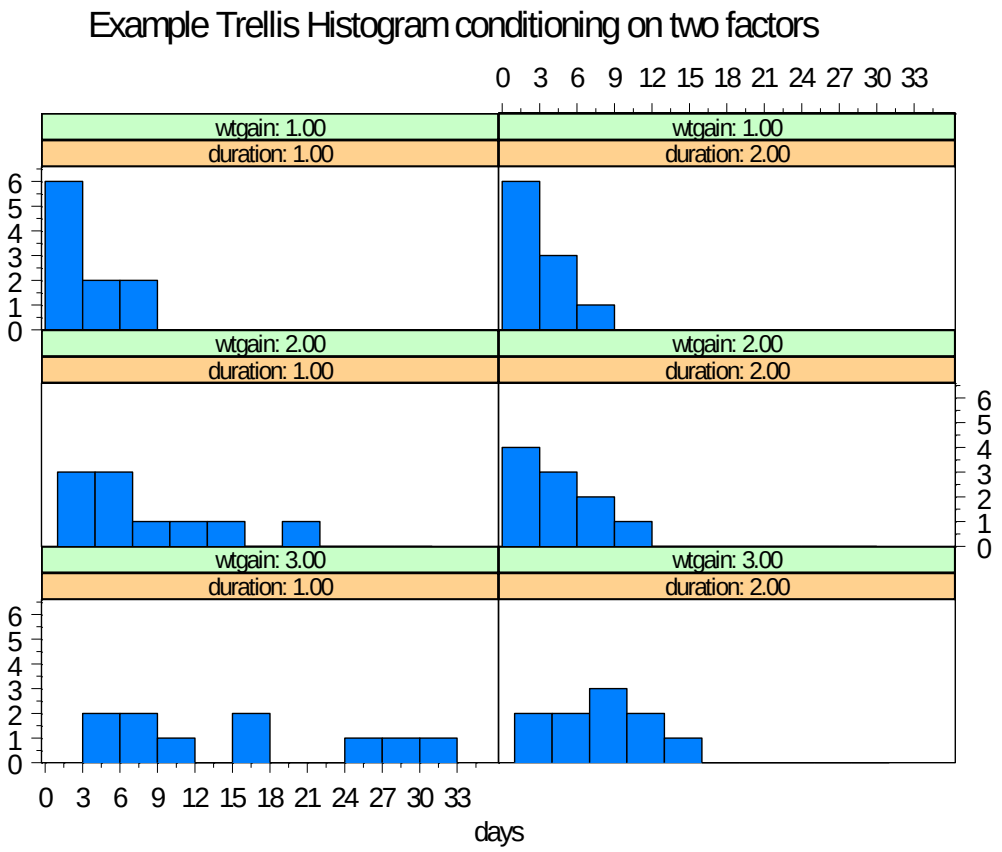
Trellis Histogram conditioning on factor B (weight gain) using the user interface.



To change the number of class right-click on any bar of the histogram, then *Options*, then type the number you want for the *Number of Bars*. Click OK.



Now, let choose both factors as conditioning factors. Choose both columns in the Data Window and press and hold the mouse and drag the cursor to the top of the graph.



Note that the above Trellis graph has the order as

(1,3)	(2,3)
(1,2)	(2,2)
(1,1)	(2,1)

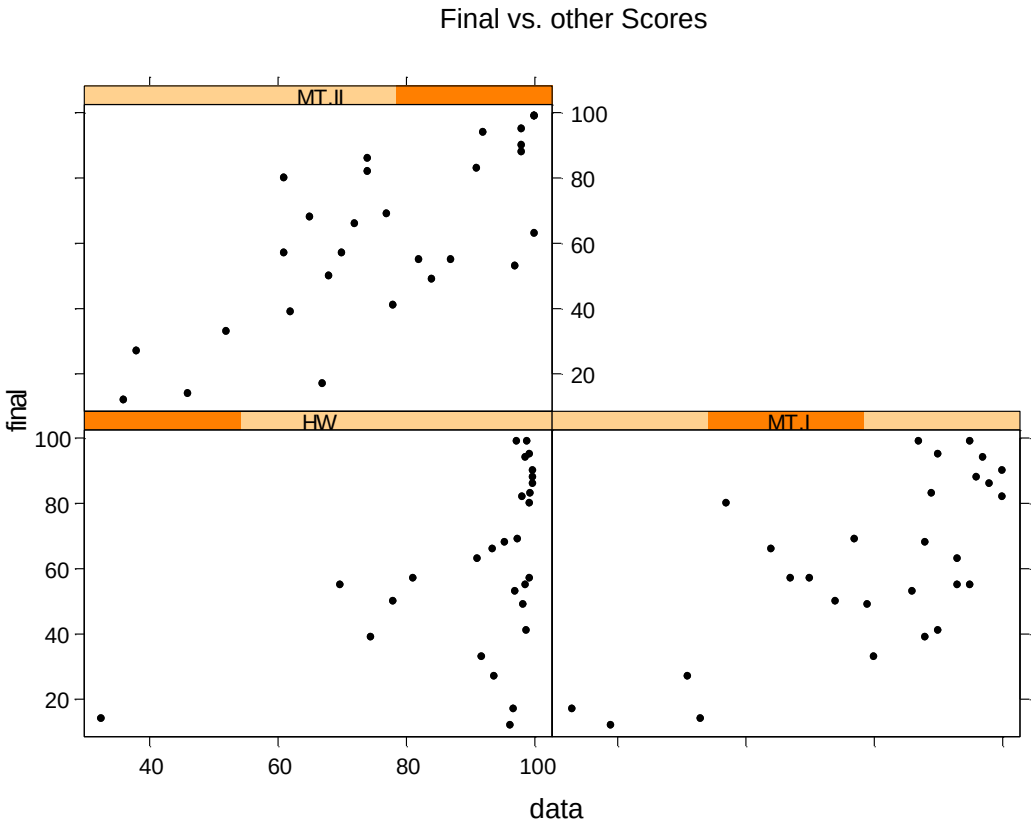
Here (1,2) means (factor A level 1, factor B level 2).
To reverse the order right-click on any white space in the graph then choose *Multipanel* then change the *Panel Order* (bottom of the menu) to *Table Order*. To add the title of the graph choose *Insert* then *Title*.

Data Preparation to Use for Trellis

If data consists of several columns and if we want to generate a Trellis graph conditioning on the column variables, we can use *make.groups* to make a group data and to ready for a Trellis graph. For better understanding let's revisit our STA131 grade data. The data has five numeric columns (HW, MT.I, MT.II, Final, Total) and we would like to generate scatter plots of Final versus each of HW, MT.I and MT.II.

```
> grp_make.groups(HW,MT.I,MT.II)
> final_rep(Final,3)
> grp.trel_data.frame(final,grp)
> grp.trel
   final    data which
 1    27  93.750    HW
 2    33  91.750    HW
  ..      ..         ..
29    27  51.000  MT.I
30    33  80.000  MT.I
```

```
..        ..      ..
57    27  38.000 MT.II
58    33  52.000 MT.II
..        ..      ..
84    63 100.000 MT.II

> xyplot(final~data|which, pch=16, col=1, data=grp.trel)
```

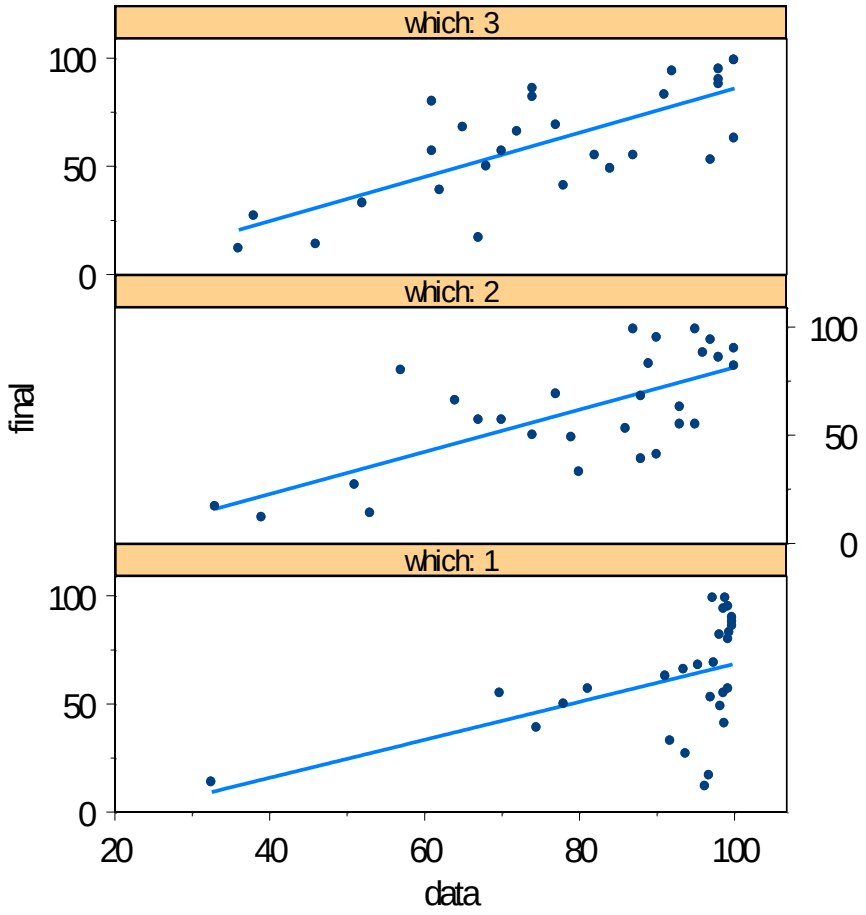


We have generated a Trellis scatter plot using the user interface or

> xyplot (y~x | z)

where x and y are the two variable for the scatter plot and z is the conditioning variable.

We can obtain better layout using Interface.


which: 3
which: 2
which: 1
final
data
100
50
0
20
40
60
80
100

DESCRIPTIVE STATISTICS

The *summary* function provides some basic statistics of the data or variables. Consider the grade data.

```
> summary(grade.data)
                Name           HW              MT.I
       Zhang, Li: 1      Min.:32.50      Min.: 33.00
    Wong, Gildas: 1   1st Qu.:93.06   1st Qu.: 69.25
 White, Jennifer: 1    Median:97.31    Median: 87.50
     Wai, Newton: 1      Mean:91.92      Mean: 79.61
 Sundstrom, Kurt: 1   3rd Qu.:98.97   3rd Qu.: 93.50
   Su, Xiao-Gang: 1      Max.:99.75      Max.:100.00
         (Other):22
      MT.II            Final           Total
    Min.: 36.00      Min.:12.00      Min.:31.80
 1st Qu.: 64.25   1st Qu.:47.00   1st Qu.:62.53
  Median: 75.50    Median:60.00    Median:73.50
    Mean: 76.00      Mean:61.46      Mean:71.86
 3rd Qu.: 93.25   3rd Qu.:83.75   3rd Qu.:87.50
    Max.:100.00      Max.:99.00      Max.:98.20
```

The *stem* function gives the stem-and-leaf display of a variable and the *quantile* function gives the quantiles of the variable at the specified values.

```
> attach(grade.data)
> stem(Total)

N = 28   Median = 76.4125
Quartiles = 65.7125, 89.2625

Decimal point is 1 place to the right of the colon

   3 : 5
   4 : 299
   5 :
   6 : 1456
   7 : 22344589
   8 : 12589
   9 : 0555668

 > quantile(Final, c(0.25, 0.5, 0.95))
  25% 50%  95%
  47  60 97.6

> grade.data[Total>=90,]
                Name     HW MT.I MT.II Final Total
15     Lin, Dongqing 99.250   90    98    95  95.0
16        Love, Brad 99.750   96    98    88  93.2
17       Peck, Laura 98.625   97    92    94  94.7
20 Singh, Ramanpreet 99.750  100    98    90  95.0
22     Su, Xiao-Gang 97.250   95   100    99  98.2
24       Wai, Newton 98.875   87   100    99  96.5

> apply(grade.data,2,mean)
 Name       HW     MT.I MT.II    Final    Total
   NA 91.91518 79.60714    76 61.46429 71.86429
```

Now, consider the kidney data. The data is categorized by two factors and we can have the summary of the response variable conditioning on factors.

```
> attach(kidney)
> summary(days[duration==1])
  Min. 1st Qu. Median  Mean 3rd Qu.  Max.
  0.00  2.25    5.00   8.20 11.50   30.00

> summary(days[duration==1 & wtgain==2])
  Min. 1st Qu. Median  Mean 3rd Qu.  Max.
  1.00  3.25    4.50   7.30 10.75   20.00

> tapply(days,list(duration,wtgain),mean)
    1   2    3
1 2.7 7.3 14.6
2 2.2 3.7  7.5
```

The *tapply* (*t* stands for table) splits the data by the given factors and apply the function (*mean* this case) to each subgroup. Another useful function for a categorized data is *by.* The *by* function is very convenient to obtain numerical summaries of the categorized data.

```
> by(days, duration, summary)
INDICES:1
         x
    Min.: 0.00
 1st Qu.: 2.25
  Median: 5.00
    Mean: 8.20
 3rd Qu.:11.50
    Max.:30.00
------------------------------------------------------------
INDICES:2
         x
    Min.: 0.000
 1st Qu.: 1.000
  Median: 3.500
    Mean: 4.467
 3rd Qu.: 7.000
    Max.:15.000
```

Similarly,

```
> by(days,wtgain, summary)


> by(days,list(duration,wtgain), summary)
:1
:1
         x
    Min.:0.00
 1st Qu.:0.25
  Median:2.00
    Mean:2.70
 3rd Qu.:4.50
    Max.:8.00
------------------------------------------------------------
:2
:1
         x
    Min.:0.00
 1st Qu.:0.25
  Median:1.50
    Mean:2.20
 3rd Qu.:3.75
```

```
    Max.:7.00
------------------------------------------------------
:1
:2
          x
    Min.: 1.00
 1st Qu.: 3.25
  Median: 4.50
    Mean: 7.30
 3rd Qu.:10.75
    Max.:20.00
------------------------------------------------------
:2
:2
          x
    Min.:0.00
 1st Qu.:1.25
  Median:3.00
    Mean:3.70
 3rd Qu.:5.75
    Max.:9.00
------------------------------------------------------
:1
:3
          x
    Min.: 3.00
 1st Qu.: 7.25
  Median:12.50
    Mean:14.60
 3rd Qu.:22.75
    Max.:30.00
------------------------------------------------------
:2
:3
          x
    Min.: 1.00
 1st Qu.: 4.50
  Median: 7.50
    Mean: 7.50
 3rd Qu.: 9.75
    Max.:15.00
>
```

DISTRIBUTION RELATED FUNCTIONS

Example:

```
> z_qnorm(.95)              # 95% quantiles under N(0,1)
> z
[1] 1.644854
> pnorm(z)                  # cumulated density at the 95% quantile
[1] 0.95
> qnorm(c(.05,.95))         # 5% and 95% quantiles under N(0,1)
[1] -1.644854  1.644854

> qt(.95,10)                # 95% quantile under t(10)
[1] 1.812461
> pt(2.0,10)                # cumulated density
[1] 0.963306
```

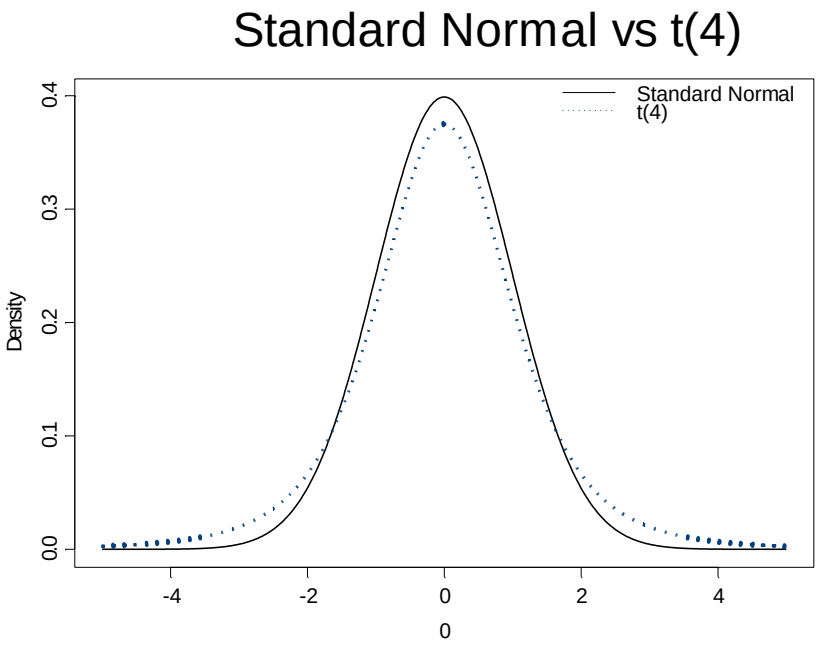
Character Type

- **d** : Distribution Function
- **p** : Cumulated Density Function
- **q** : Quantile
- **r** : Random number generation

The distribution names suffix one of the character types to define the S+ functions. For the list of names and necessary parameters see the handout in class.

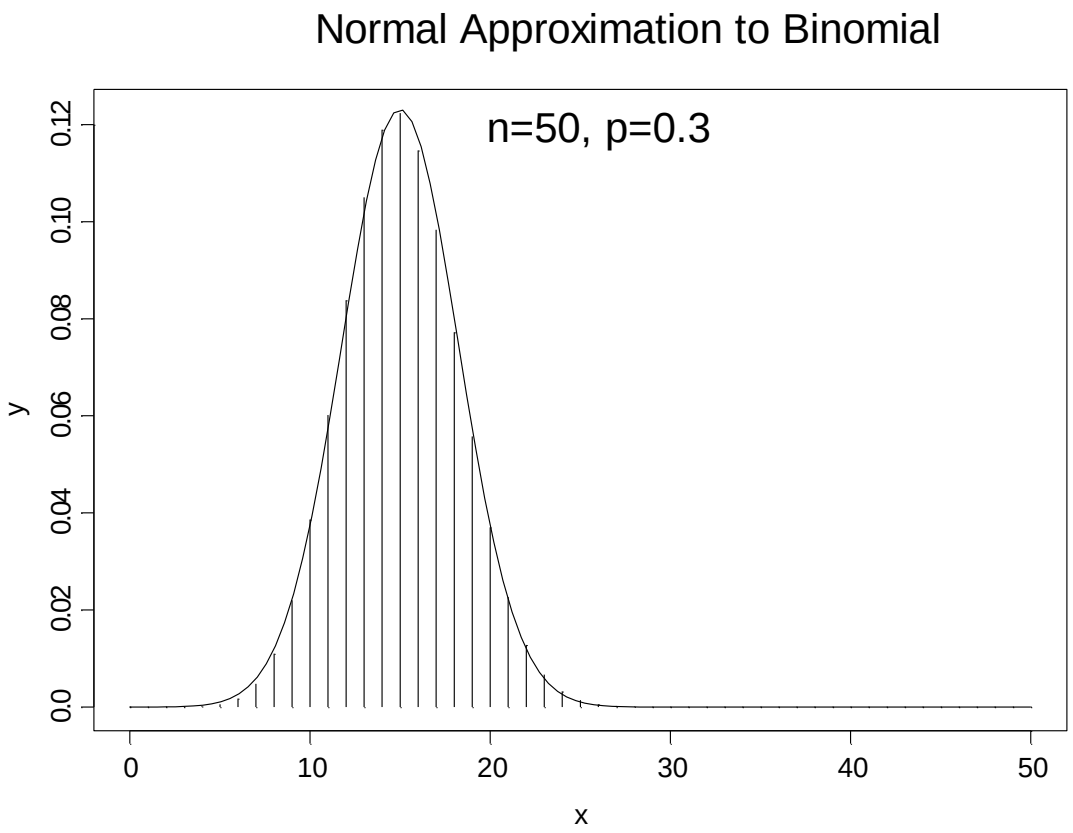
Let's consider the following S+ code to compare the normal distribution and t with 5 df.

```
pts_seq(-5,5,length=1000)
pts.nor_dnorm(pts)
pts.t_dt(pts,4)
yrange_range(pts.nor,pts.t)
plot(0,0,type="n",xlim=c(-5,5), ylim=yrange,ylab="Density")
lines(pts,pts.nor,col=1,lty=1,lwd=2)
lines(pts,pts.t,col=2,lty=2,lwd=4)

key(lines=list(lty=1:2, col=1:2),text=list(paste(c("Standard Normal","t(4)"))))
title("Standard Normal vs t(4)",cex=1.5)
```



For another example of using the distribution related functions, let's consider a graphical display of the normal approximation to the binomial distribution. Let X be binomial with n=50 and p=.3. The density of the Binomial distribution and the normal distribution with mean of *n*p=15* and the sd of *sqrt(n*p*(1-p))=3.24* are given in the following graph.

```
n_50; p_0.3; x_0:n
y_dbinom(x,n,p)
plot(x,y,type="h")

x_seq(0,n,length=100)
m=n*p;s=sqrt(n*p*(1-p))
f_dnorm(x,m,s)
lines(x,f)
title("Normal Approximation to Binomial")
text(25,.12,"n=50, p=0.3",cex=1.5)
```



To calculate P(X ≥ 20) we use either *pbinom* (exact prob.) or *pnorm* (normal approx.). First, the exact probability from the Binomial distribution is;

```
> 1 - pbinom(19, n, p)
[1] 0.0848026
```

The exact probability can be compared with a Normal approximation;

```
> z <- (19.5 - n * p)/sqrt(n * p * (1 - p))
> 1 - pnorm(z)
[1] 0.08245741
```

A list of the distributions and their S+ function names can be found in the text p114.

HYPOTHESIS TESTING

We have discussed one sample t-test earlier. S+ provides many functions for hypothesis testing. The names of functions are self-explanatory. To see the necessary arguments, for example t-test, type

```
 > args(t.test)
function(x, y = NULL, alternative = "two.sided", mu = 0, paired = F,
       var.equal = T, conf.level = 0.95)
```

We may also type >help(t.test) for more details.

Let's consider the two sample t-test comparing the two different durations in the kidney data. If the data is paired (not this case), we need to specified *paired=T* and do the paired t test.

```
> my.ttest_t.test(days[duration==1], days[duration==2], conf.level = .90)
> my.ttest

       Standard Two-Sample t-Test

data:  days[duration == 1] and days[duration == 2]
t = 2.2319, df = 58, p-value = 0.0295
alternative hypothesis: true difference in means is not equal to 0
90 percent confidence interval:
 0.9372859 6.5293808
sample estimates:
 mean of x mean of y
       8.2  4.466667
```

The resulted p-value of the test is .0295, which is compared with the specified significance level of 0.1. Since the p-value is less than the significance level, the test for equal means is rejected.

Note that the function *t.test* returns a list type of object with some statistics and information for the test. To see the available information;

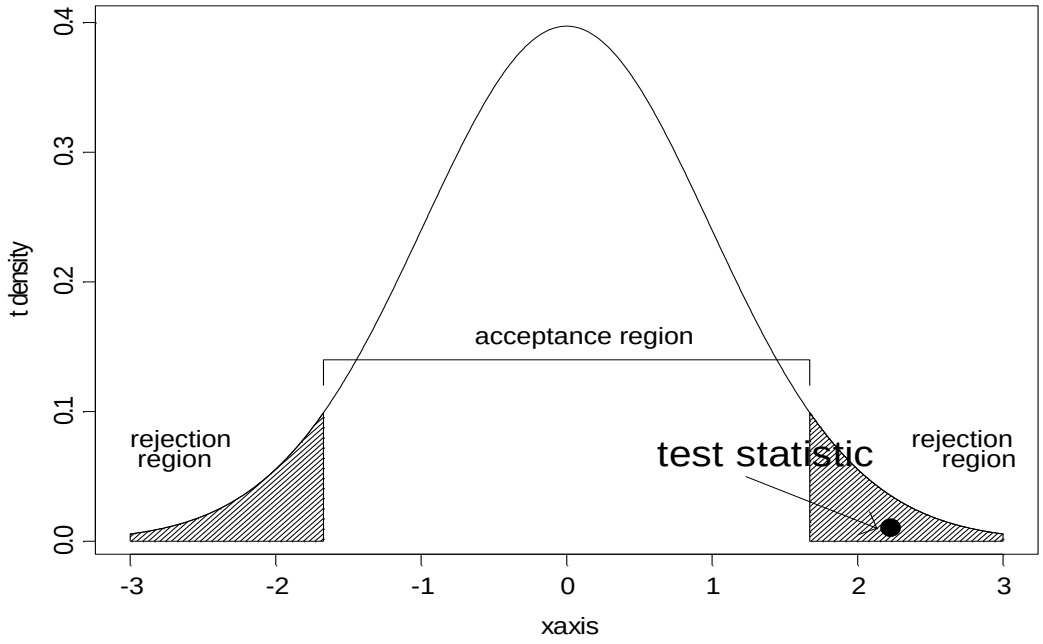
```
> summary(my.ttest)

            Length Class      Mode
  statistic 1      numeric
 parameters 1      numeric
    p.value 1      numeric
   conf.int 2      numeric
   estimate 2      numeric
 null.value 1      numeric
alternative 1      character
     method 1      character
  data.name 1      character
```

We now try to access some specific information.

```
> my.ttest$statistic
        t
 2.231888
> my.ttest$p.value
[1] 0.02950016
> my.ttest$conf.int
[1] 0.9372859 6.5293808
attr(, "conf.level"):
[1] 0.9
> my.ttest$data.name
[1] "days[duration == 1] and days[duration == 2]"
```

```
> my.ttest$null.value
 difference in means
                   0

> ttest$alternative
[1] "two.sided"
```

The testing procedure and the result are well demonstrated in the following graph. The S+ code generating the graph is copied from the recommended text by Krause and Olson. I found it very useful for writing report for non-statisticians. We will discuss the S+ code in class.




```
attach(kidney)
x_days[duration==1]
y_days[duration==2]
df_length(x)+length(y)-2
alpha_.1

bound.left_-3
bound.right_3
xaxis_seq(bound.left, bound.right, length=1000)
yaxis_dt(xaxis,df)
plot(xaxis,yaxis,type="l", ylab="t density")

critical.left_qt(alpha/2,df)
critical.right_qt(1-alpha/2,df)
xaxis_seq(bound.left,critical.left,length=100)
yaxis_c(dt(xaxis,df),0,0)
xaxis_c(xaxis,critical.left,bound.left)
polygon(xaxis,yaxis,density=25)

xaxis_seq(critical.right, bound.right,length=100)
yaxis_c(dt(xaxis,df),0,0)
xaxis_c(xaxis,bound.right,critical.right)
polygon(xaxis,yaxis,density=25)

test.stat_t.test(x, y, conf.level = 1-alpha)$statistic
```

```
points(test.stat,.01,cex=2,adj=0.5, pch=16)
arrows(test.stat-1,.05,test.stat-.1,.01,open=T,rel=T)
text(test.stat-1,.07,"test statistic",adj=.5,cex=1.5)

text(bound.left, 0.08, "rejection \n region", adj=0)
text(bound.right, 0.08, "rejection \n region", adj=1)
text((bound.left + bound.right)/2, 0.16, "acceptance region")
xaxis_c(rep(critical.left,2),rep(critical.right,2))
yaxis_c(0.12,0.14,0.14,0.12)
```

Other statistical test in S+: binom.test, chisq.gof, chisq.test, cor.test, var.test, wilcox.test, etc.

REGRESSION

In this section we will discuss fitting a regression model using the *lm* (linear model) function. As we have discussed earlier, the syntax of the model is

```
lm(response~predictor1+predictor2+…, options)
```

For a detailed discussion let's consider the Real Estate data that we used for MATH 532 class.

Data

A real estate expert is interested in developing a regression model that relates the selling price of suburban residential properties to characteristics of properties. Her interest lies in new, large residential property development on the outskirts of a major city for which she has data on 30 properties that were sold recently.

X_1	Property taxes (annual taxes, in dollars)
X_2	House size (floor area, in square feet)
X_3	Lot size (in acres)
X_4	Lot size squared
X_5	Attractiveness index
X_6	Style (E, S, or M)

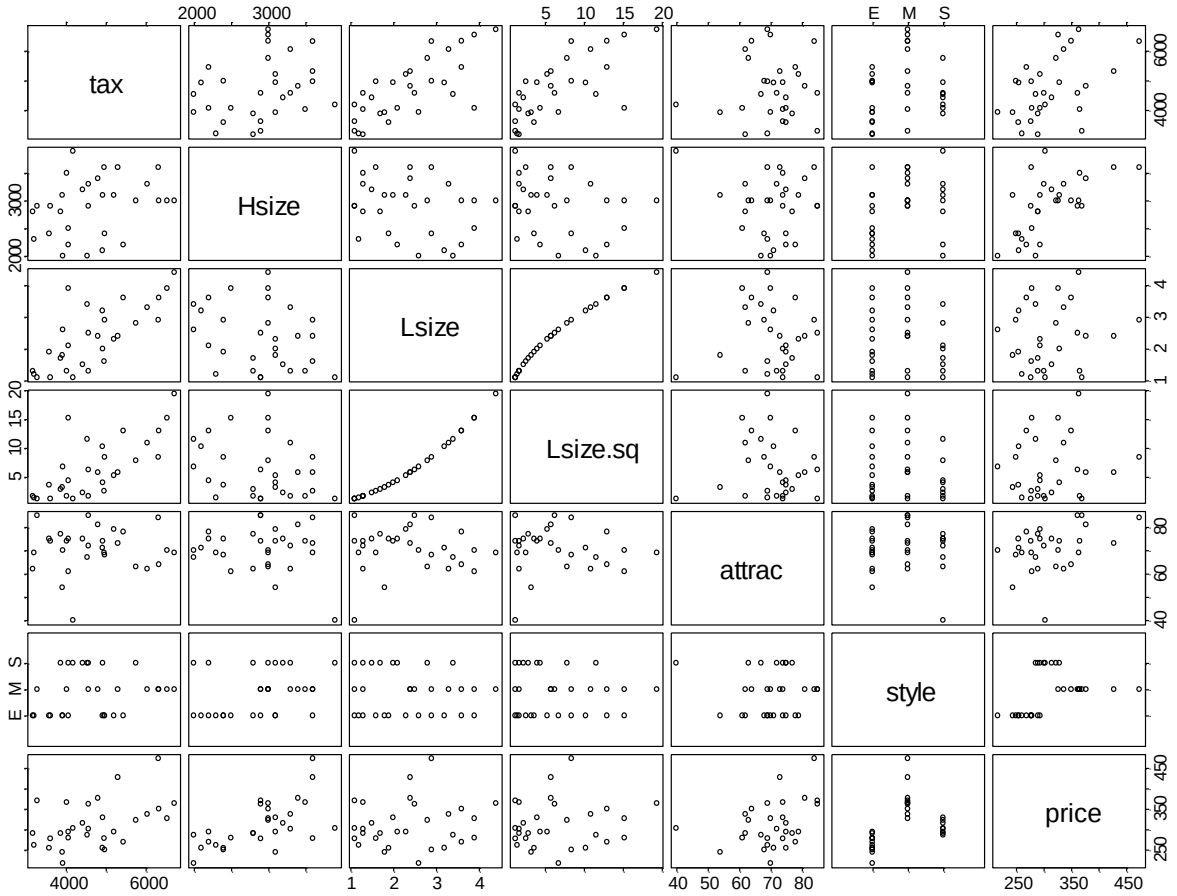
Data for the property selling prices

Property	Property Taxes	House Size	Lot Size	Lot Size Squared	Attrac-tiveness	Style	Selling Price (in $1000)
1	6337	3000	3.6	12.96	64	M	350
2	3204	2300	1.2	1.44	69	E	261
3	4574	3300	1.3	1.69	72	S	301
4	4924	2100	3.2	10.24	71	E	255
5	4182	3900	1.1	1.21	40	S	303
…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…
27	3917	3100	1.8	3.24	54	E	244
28	4068	2200	2.1	4.41	75	S	294
29	4068	2500	3.9	15.21	61	E	279
30	3612	2900	1.1	1.21	74	E	277

For the full data click on realestate.dat

For multivariate data, it is good idea to start the analysis with the pairwise scatterplots.

```
> pairs(realestate[,2:8])
```



Using the *cor* function, we can calculate the correlation matrix among variables.

```
> cor(realestate[,c(2:6,8)],realestate[,c(2:6,8)])

               tax      Hsize       Lsize    Lsize.sq      attrac     price
    tax 1.0000000  0.2812810  0.73968353  0.72108667  0.02879220 0.4693318
  Hsize 0.2812810  1.0000000 -0.28982630 -0.26804966 -0.12844139 0.6258176
  Lsize 0.7396835 -0.2898263  1.00000000  0.98477889 -0.01889162 0.1206351
Lsize.sq 0.7210867 -0.2680497  0.98477889  1.00000000 -0.06571432 0.1181183
 attrac 0.0287922 -0.1284414 -0.01889162 -0.06571432  1.00000000 0.3610034
  price 0.4693318  0.6258176  0.12063505  0.11811831  0.36100336 1.0000000
```

Note that the predictor *style* is qualitative and omitted from the correlation calculation. The pairwise scatterplot and the correlation suggest that *Hsize* has the strongest linear association with *price*. The variables *tax* and *attrac* also show moderately high correlation with the response.

Before we go any further, we need to note that we have a qualitative predictor, *style*, in the data. Many different codings of indicator variables are available. Here, we use the simple dummy coding (0,1 coding);

Style	Sty1	Sty2
E	1	0
M	0	1
S	0	0

Let add the two indicator variables in the data frame.

```
> sty1_rep(NA,30)
> sty2_rep(NA,30)
> sty1[style=="E"]_1
> sty1[style!="E"]_0
> sty2[style=="M"]_1
> sty2[style!="M"]_0
> realestate_data.frame(realestate,sty1,sty2)
```

At this moment you may want to recalculate the correlation matrix.

Simple Regression
First, consider a simple regression with the response variable, *price*, and the predictor, *Hsize*.

```
> fit1_lm(price~Hsize,data=realestate)
> summary(fit1)

Call: lm(formula = price ~ Hsize, data = realestate)
Residuals:
    Min     1Q Median    3Q   Max
 -81.44 -27.09 -2.379 30.81 114.7

Coefficients:
              Value Std. Error  t value Pr(>|t|)
(Intercept) 115.7431  46.9759     2.4639   0.0201
      Hsize   0.0676   0.0159     4.2457   0.0002

Residual standard error: 44.86 on 28 degrees of freedom
Multiple R-Squared: 0.3916
F-statistic: 18.03 on 1 and 28 degrees of freedom, the p-value
is 0.0002167

Correlation of Coefficients:
      (Intercept)
Hsize -0.9847
```

Specific information contained in the modeling can be accessed as before; *object$values.*

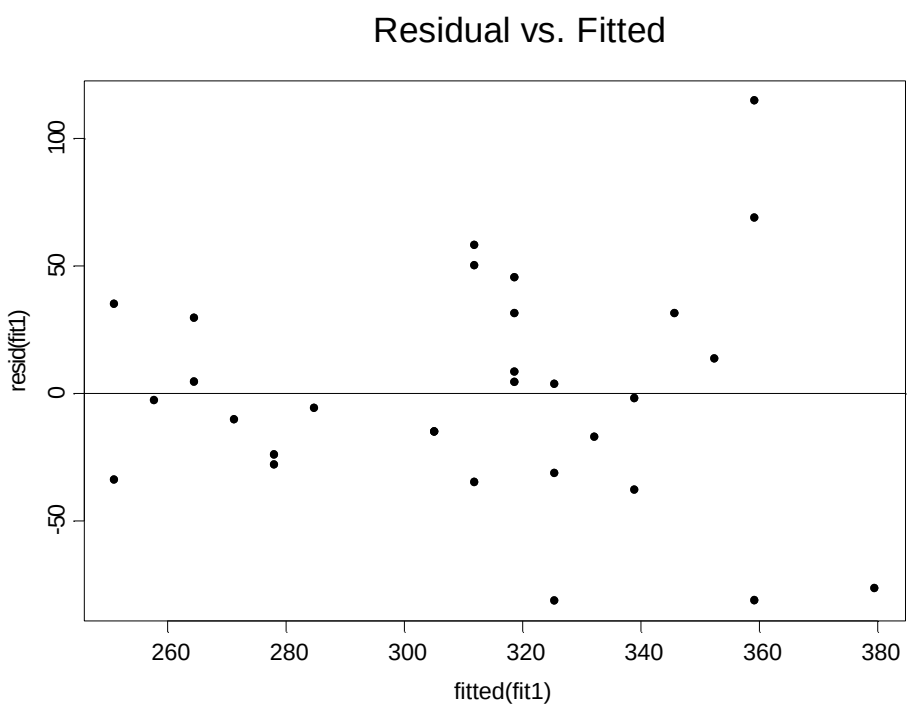
```
> fit1$coefficients
 (Intercept)      Hsize
    115.7431 0.06764303
> resid_fit1$residuals
> fitted_fit1$fitted.values
```

For the full list of available values see the help file for *lm.object*.

We can also access the residuals and the fitted values by *resid(fit1)* and *fitted(fit1)*. Let's plot the residuals versus the fitted values.

```
> plot(fitted(fit1),resid(fit1),pch=16)
> abline(h=0)
```

abline(h=0) will draw the horizontal line at 0.



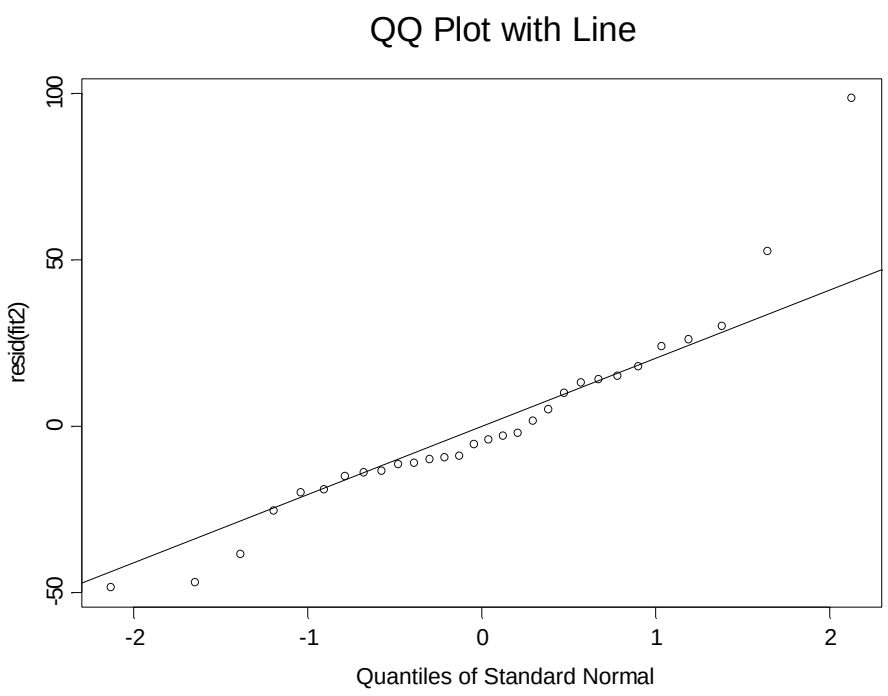
As discussed earlier,

```
> attach(realestate); plot(Hsize,price); abline(fit1)
```

will draw a scatterplot with the fitted line superimposed on.

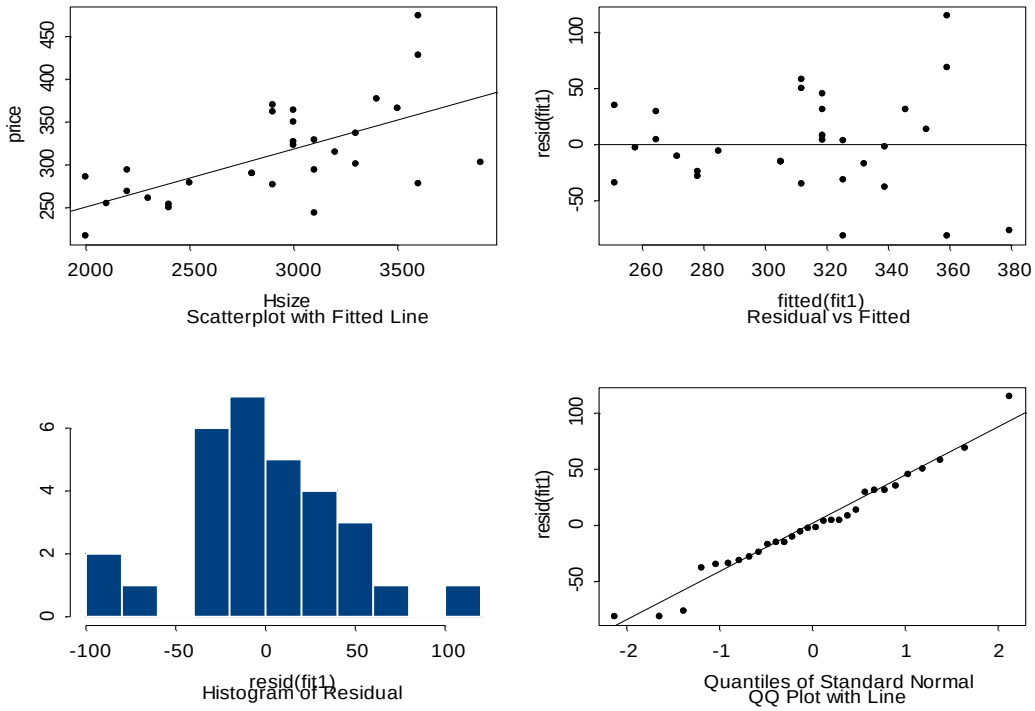
For a QQ plot

```
> qqnorm(resid(fit2),main="QQ Plot with Line")
> qqline(resid(fit2))
```



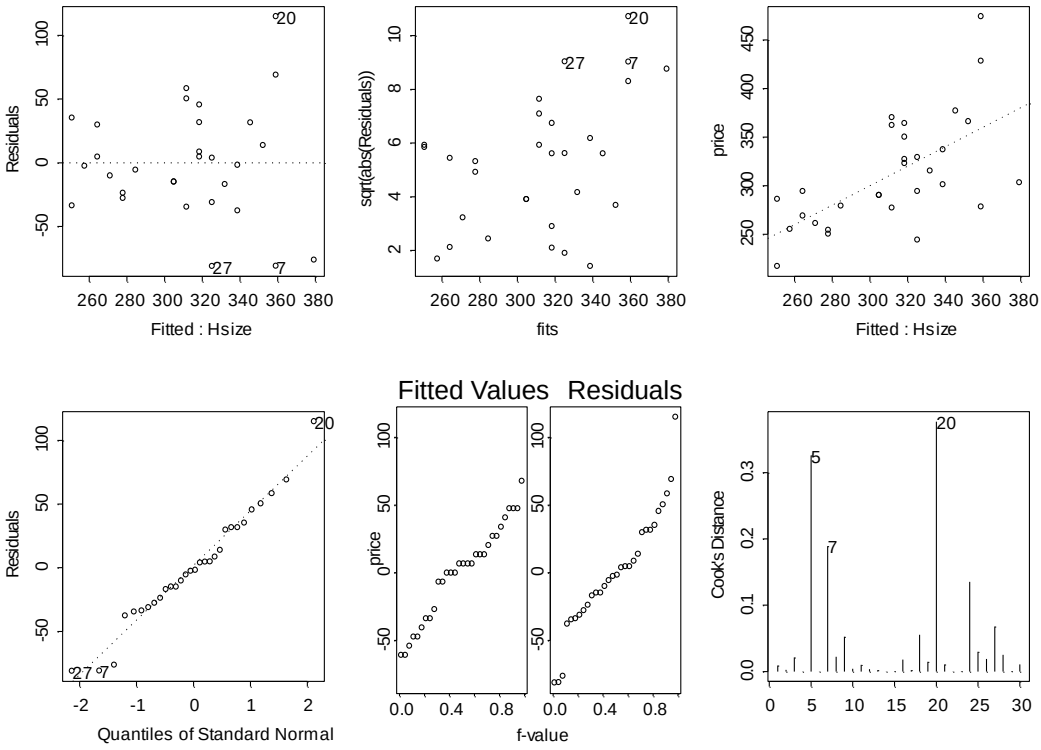
Let's put them together.

```
attach(realestate)
fit1_lm(price~Hsize,data=realestate)
par(mfrow=c(2,2))
plot(Hsize,price,pch=16,sub="Scatterplot with Fitted Line"); abline(fit1)
plot(fitted(fit1),resid(fit1),pch=16,sub="Residual vs Fitted"); abline(h=0)
hist(resid(fit1),sub="Histogram of Residual",nclass=10)
qqnorm(resid(fit1),sub="QQ Plot with Line",pch=16); qqline(resid(fit1))
```



We can also examine the S+ default plots:

```
> plot(fit1)
```



You can specify your choice of the plots to display by using the *ask* option:

```
> plot(fit1, ask=T)
```

Multiple Regression

Now, consider a multiple regression with two predictors. The predictor *Hsize* is in the model and we want to figure out which predictor should be considered for the second predictor in the model. From the pairwise scatterplot and the correlation matrix suggest *tax* or *attrac*. We may fit models with all possible multiple models of two predictors with *Hsize* already in the model. This can be done easily in S+:

```
> add1(fit1, .~. +tax+Lsize+Lsize.sq+attrac+sty1+sty2)
Single term additions

Model:
price ~ Hsize
         Df Sum of Sq      RSS       Cp
  <none>              56358.65 64409.88
     tax  1   8654.25 47704.39 59781.25
   Lsize  1   9224.91 47133.74 59210.59
Lsize.sq  1   8156.81 48201.84 60278.69
  attrac  1  18351.16 38007.49 50084.34
    sty1  1  20842.36 35516.29 47593.14
    sty2  1  32570.78 23787.87 35864.72
```

We see that the style variable (sty1 and sty2) has the largest sums of squares (and the lowest Cp) and thus these predictors should be considered in the model.

```
> fit2_lm(price~Hsize+sty1+sty2)
> summary(fit2)

Call: lm(formula = price ~ Hsize + sty1 + sty2)
Residuals:
    Min     1Q  Median    3Q   Max
 -42.12 -14.61 -0.7273 14.55 87.48

Coefficients:
               Value Std. Error   t value  Pr(>|t|)
(Intercept)  219.9120   34.2463     6.4215    0.0000
      Hsize    0.0290    0.0112     2.5929    0.0154
       sty1  -31.8181   12.9440    -2.4581    0.0209
       sty2   62.1801   13.3051     4.6734    0.0001

Residual standard error: 27.25 on 26 degrees of freedom
Multiple R-Squared: 0.7916
F-statistic: 32.93 on 3 and 26 degrees of freedom, the p-value is 5.24e-009

Correlation of Coefficients:
      (Intercept)   Hsize    sty1
Hsize -0.9596
 sty1 -0.4754      0.2773
 sty2  0.0243     -0.2375  0.4730
```

For the hypothesis testing for no style effect, we do the test for both coefficients of sty1 and sty2.

```
> by(price,style,mean)
INDICES:E
[1] 264
------------------------------------------------------------------------
INDICES:M
[1] 375.5
------------------------------------------------------------------------
```

```
    INDICES:S
    [1] 305.125
```

The model syntax

response~predictor1 + predictor2

will fit a multiple regression with the two main effects only and

response~predictor1 + predictor2+predictor1:predictor2

or

response~predictor1 predictor2*

will fit the model with two main effects and the interaction between the two predictors.

Note that for the qualitative variable with three levels, we may assign three indicator variables (one for each level) and drop the intercept in the regression model. This coding of indicator also has widely used in application. The model statement with –1 at the end will drop the intercept in the model

```
> lm(y~x1+x2+…-1)
```

More will come……

REGRESSION (con't)

CODING OF FACTORS

Last time we have discussed how to create dummy variables for the qualitative predictors in a regression model. We extend the discussion to the use of a convenient S+ function to create our own coding of factors. Let's consider the *style* variable in the *realestate* data.

```
> levels(style)
[1] "E" "M" "S"
```

We generate a matrix for the two dummy variables.

```
> my.cotr_matrix(c(1,0,0,0,1,0),ncol=2)
> my.cotr
     [,1] [,2]
[1,]    1    0
[2,]    0    1
[3,]    0    0
```

Now, use the *C* function in the model statement of the *lm* function. Note that *C* stands for "contrasts", a linear combination of the dummy variables.

```
> summary(lm(price~C(style,my.cotr)))

Call: lm(formula = price ~ C(style, my.cotr))
Residuals:
   Min     1Q Median    3Q  Max
 -48.5 -13.87 -4.812 13.75 98.5

Coefficients:
                      Value Std. Error   t value  Pr(>|t|)
       (Intercept)  305.1250   10.6052    28.7714    0.0000
C(style, my.cotr)1  -41.1250   13.6912    -3.0038    0.0057
C(style, my.cotr)2   70.3750   14.2283     4.9461    0.0000
```

```
Residual standard error: 30 on 27 degrees of freedom
Multiple R-Squared: 0.7378
F-statistic: 37.98 on 2 and 27 degrees of freedom, the p-value is 1.42e-008

Correlation of Coefficients:
                   (Intercept) C(style, my.cotr)1
C(style, my.cotr)1 -0.7746
C(style, my.cotr)2 -0.7454      0.5774
```

S+ provide four different built-in contrast functions; Helmert (default), Orthogonal polynomial, Sum, and Treatment. To see the matrix form for these function, for example, for a factor with three levels

```
> contr.helmert(3)
  [,1] [,2]
1   -1   -1
2    1   -1
3    0    2

> contr.treatment(3)
  2 3
1 0 0
2 1 0
3 0 1
```

If you want to use one of the built-in contrasts, use the name of the contrast in the second argument of the C function. For example

```
lm(price~C(style,treatment))
```

will use the treatment contrast for the variable *style*. Note that the default contrast is the Helmert. So if you call

```
lm(price~style)
```

it will return the result with the Helmert contrast.

Another possibility is the use of *I* (indicator) function. For example,

```
> lm(price~I(style=="E")+I(style=="M"),data=realestate)
```

The I function is very useful to assign indicator variables for a quantitative predictor. For example, if you are looking at the effect of "high" (attrac>75) attractiveness index versus "lower" (attrac<=75) instead of the effect of the attractiveness on a continuous scale,

```
> lm(price~I(attrac>75),data=realestate).
```

Model Selection: Stepwise Regression

Model selection procedure via Stepwise regression can be easily done using the user interface. We will discuss the procedure in details during the class and will not be illustrated in this note.

In the procedure we need to specify the full model (upper). We look at the correlation matrix to guess which interaction terms we need to consider for the full model.

```
> cor(realestate[,c(2:6,9,10)],realestate[,c(2:6,9,10)])
               tax      Hsize       Lsize    Lsize.sq      attrac        sty1       sty2
     tax  1.0000000  0.2812810  0.73968353  0.72108667  0.02879220 -0.37758018  0.4872225
   Hsize  0.2812810  1.0000000 -0.28982630 -0.26804966 -0.12844139 -0.45526549  0.4355324
   Lsize  0.7396835 -0.2898263  1.00000000  0.98477889 -0.01889162 -0.07395935  0.3030090
Lsize.sq  0.7210867 -0.2680497  0.98477889  1.00000000 -0.06571432 -0.07975938  0.3186872
  attrac  0.0287922 -0.1284414 -0.01889162 -0.06571432  1.00000000 -0.13282487  0.3093021
```

```
    sty1 -0.3775802 -0.4552655 -0.07395935 -0.07975938 -0.13282487  1.00000000 -0.5773503
    sty2  0.4872225  0.4355324  0.30300898  0.31868722  0.30930212 -0.57735027  1.0000000
```

The highlighted correlations seem to be high and important. We use all the main effects and the five interactions (we only considered two way interactions) for the upper model. Be sure to save the output object. For this example, the object was saved as *step.real*.

The resulted model from the stepwise procedure is

```
> summary(step.real)

Call: lm(formula = price ~ Hsize + attrac + sty1 + sty2 + Hsize:sty2, da
ta =
       realestate, na.action = na.exclude)
Residuals:
    Min     1Q Median    3Q   Max
 -36.67 -13.77  3.668 11.33 48.82

Coefficients:
               Value Std. Error   t value  Pr(>|t|)
(Intercept)  140.5705   52.0276     2.7018    0.0125
      Hsize    0.0272    0.0102     2.6562    0.0138
     attrac    1.2488    0.4854     2.5728    0.0167
       sty1  -34.0227   10.4913    -3.2429    0.0035
       sty2 -180.8893   87.0854    -2.0771    0.0487
 Hsize:sty2    0.0730    0.0273     2.6740    0.0133

Residual standard error: 21.94 on 24 degrees of freedom
Multiple R-Squared: 0.8753
F-statistic: 33.69 on 5 and 24 degrees of freedom, the p-value is 4.266e
-010

Correlation of Coefficients:
           (Intercept)   Hsize  attrac    sty1    sty2
     Hsize -0.7882
    attrac -0.8254      0.3328
      sty1 -0.3074      0.2929  0.0443
      sty2 -0.2706      0.3391  0.0972  0.1661
Hsize:sty2  0.3119     -0.3813 -0.1449 -0.1106 -0.9913
```

You need to make interpretations of the parameter estimates. I will leave it to you. For the summary of the ANOVA is

```
    > summary(aov(step.real))
               Df Sum of Sq  Mean Sq  F Value      Pr(F)
        Hsize   1  36282.82 36282.82 75.37984 0.00000001
       attrac   1  18351.16 18351.16 38.12569 0.00000222
         sty1   1  13881.09 13881.09 28.83884 0.00001633
         sty2   1   9132.76  9132.76 18.97388 0.00021368
    Hsize:sty2  1   3441.64  3441.64  7.15023 0.01327372
     Residuals 24  11552.00   481.33
```

Estimating Mean Response

To estimate the mean response with the final model

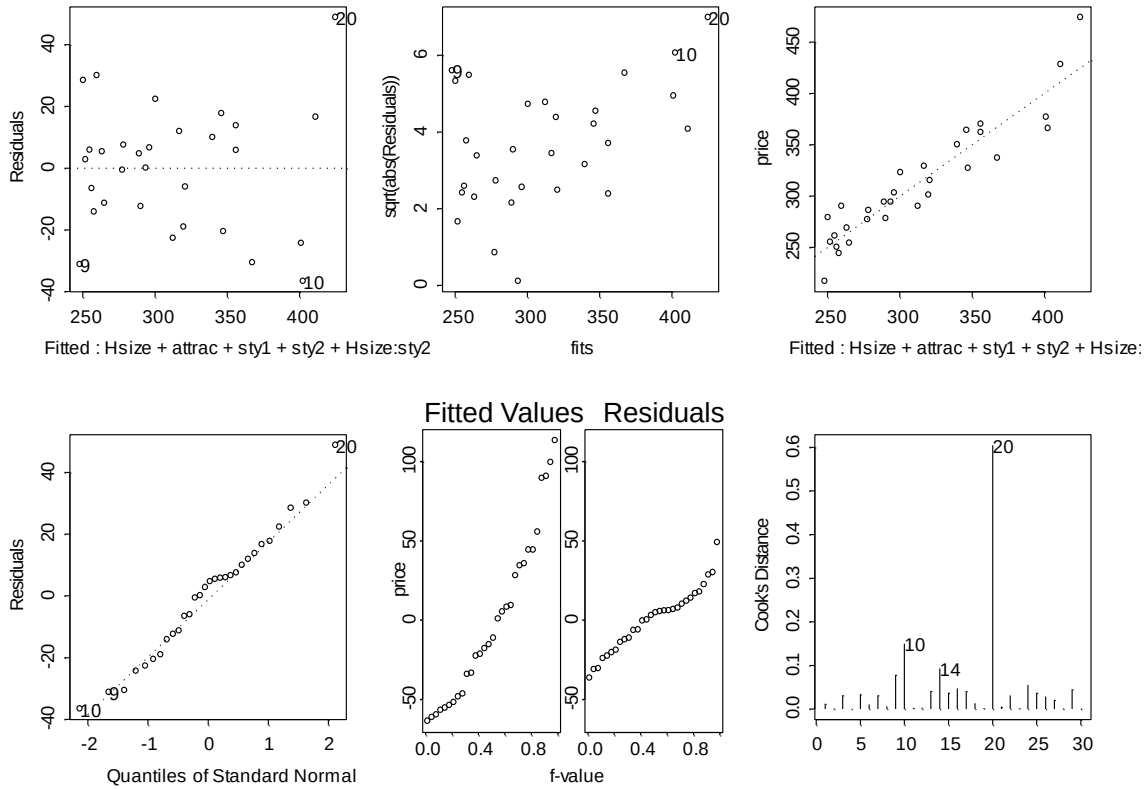
```
> prd1_predict.lm(step.real,data.frame(Hsize=2000,attrac=70,sty1=1,sty2=0),
       se.fit=T,ci.fit=T,pi.fit=T)
> prd1
$fit:
        1
 248.2893
```

```
$se.fit:
        1
 8.851564

$residual.scale:
[1] 21.93931

$df:
[1] 24

$ci.fit:
     lower   upper
1 230.0205 266.558
attr(, "conf.level"):
[1] 0.95

$pi.fit:
     lower    upper
1 199.4623 297.1162
attr(, "conf.level"):
```
[1] 0.95

It is important to note that the option *ci.fit=T* is used to obtain the interval estimation of the mean response at the values of predictors occurred in the data and *pi.fit=T* will give us the prediction interval of new observation. The prediction interval is always wider than the confidence interval because the variation in prediction has an extra source of variation due to fact that the response at the new point will not equal to its expectation. For more detailed discussion see your note for MATH 532 or Chapter 2 of the text by Neter et.al. Both intervals are pointwise and based on t distribution.

Graphical Summary
For the graphical summary of the final model

```
> par(mfrow=c(2,3))
> plot(step.real)
```



Simultaneous Confidence Band: Working-Hotelling

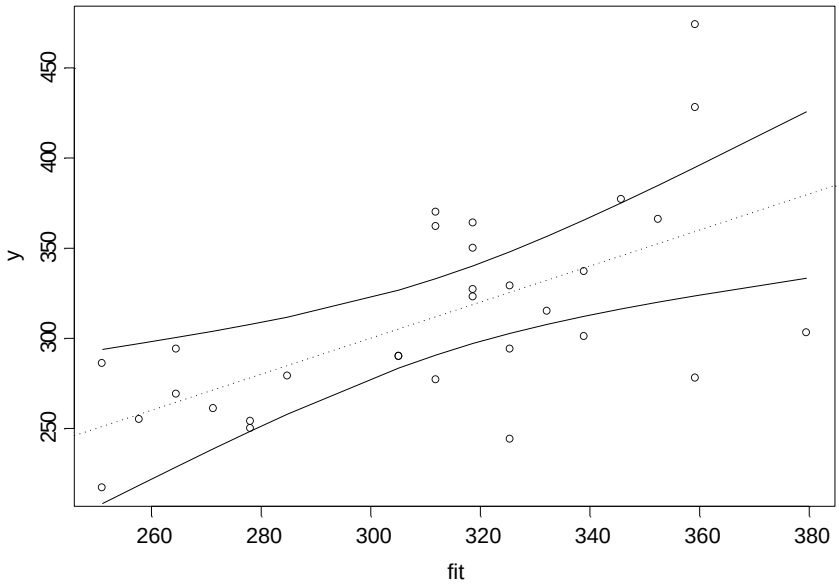
For simultaneous 1-α confidence intervals for entire responses in the data, we use the Working-Hetelling confidence band based on F distribution (see p234, Neter et.al.).

The code below can be used to create a simultaneous confidence interval for the mean response.

```
"confint.lm"_function(object,alpha=.05,plot.it=T,...)
    {
    f_predict(object,se.fit=T)
    p_length(coef(object))
    fit_f$fit
    adjust_(p*qf(1-alpha,p,length(fit)-p))^0.5*f$se.fit
    lower_fit-adjust
    upper_fit+adjust
    if(plot.it){
        y_fit+resid(object)
        plot(fit,y)
        abline(0,1,lty=2)
        ord_order(fit)
        lines(fit[ord],lower[ord])
        lines(fit[ord],upper[ord])
        invisible(list(lower=lower,upper=upper))
        }
    else list(lower=lower,upper=upper)
}
```

We use the function *confint.lm* to plot the simultaneous confidence intervals for the mean response from the simple regression model relating the *price* with *Hsize*.

```
> fit.Hsize_lm(price~Hsize)
> confint.lm(fit.Hsize)
```



Identifying Outliers in predictors

The hat matrix, $X(X'X)^{-1}X'$, has been used to identify the outliers. The elements of the hat matrix, leverages, have the mean of p/n where *p* is the number of the parameters in the model and *n* is the sample size. The leverages larger than $2p/n$ are considered as an evidence of outliers. Bonferroni test may be used for formal test for outliers in response (see Neter et.al. p374), but most of the cases they can be well identified via residual plot.

We can identify the outliers in predictors via leverage as follows;

```
> rule_2*length(coef(step.real))/length(fitted(step.real))
> rule
[1] 0.4
> influ.real_lm.influence(step.real)
> cbind(realestate,lev=influ.real$hat)[influ.real$hat>rule,]
  property  tax Hsize Lsize Lsize.sq attrac style   price        lev
5        5 4182  3900   1.1     1.21     40     S   303    0.5224226
```

If we exam the fifth property, we can see that the property has the largest Hsize and the lowest attractiveness.

We may want fit the model without the outlier.

```
> step.new <- lm(price ~ Hsize + attrac + sty1 + sty2 + Hsize:sty2, data =
    realestate[-5,  ], na.action = na.exclude)
> summary(step.new)

Call: lm(formula = price ~ Hsize + attrac + sty1 + sty2 + Hsize:sty2, data =
```

```
   realestate[-5,  ], na.action = na.exclude)
Residuals:
    Min     1Q Median    3Q   Max
 -36.53 -12.56  1.683 12.18 47.57

Coefficients:
               Value Std. Error   t value  Pr(>|t|)
(Intercept)  133.8143   55.2885    2.4203    0.0238
      Hsize    0.0256    0.0110    2.3213    0.0295
     attrac    1.3903    0.5962    2.3318    0.0288
       sty1  -32.9926   10.9486   -3.0134    0.0062
       sty2 -184.1886   88.9544   -2.0706    0.0498
 Hsize:sty2    0.0744    0.0280    2.6595    0.0140

Residual standard error: 22.32 on 23 degrees of freedom
Multiple R-Squared: 0.8762
F-statistic: 32.54 on 5 and 23 degrees of freedom, the p-value is 1.056e-009

Correlation of Coefficients:
           (Intercept)   Hsize  attrac    sty1    sty2
     Hsize -0.6161
    attrac -0.8162      0.0739
      sty1 -0.3510      0.1956  0.1602
      sty2 -0.2329      0.3477  0.0312  0.1419
Hsize:sty2  0.2628     -0.3962 -0.0534 -0.0810 -0.9909
```

The Cock's distance provided in the graph above can be used to identify influential observations.

Correlated error terms

Durbin-Watson statistic (DW) can be employed to test for first-order correlation (autocorrelation) in error terms. The test is useful when the data is observed over a systematic manner, like time or space. S+ provide the function, *durbinWatson(x)*. The argument *x* can be either data vector or a object from fitting *lm*. The statistic is bounded between 0 and 4, and for independent residuals we expect the value around 2. If the value is close to 0, it indicates positive correlation in error terms and if close to 4, it means possible negative correlation. Neter et.al provides the critical values of the statistic (see Table B.7).

Let's check the DW statistics for the residuals from the final model of realestate data. Note that since the data was not collected over time or space, we expect the value close to 2.

```
> durbinWatson(step.real)
Durbin-Watson Statistic: 1.960674
```

It is interesting to calculate the DW statistic for our Ozone data, since the data is collected over time. Note the variable *volume* is a column vector of ozone volume stacked over time.

```
> durbinWatson(volume[!is.na(volume)])
Durbin-Watson Statistic: 0.1490061
Number of observations: 1455
```

As we expected the DW value is very close to 0, which indicates strong positive correlation in the raw volume data.

Multicollinearity: Variance Inflation Factor (VIF)

Variance Inflation Factor has been widely used to measure the presence of multicollinearity in the model. Muticollinearity and its effect have been discussed in MATH 532 class. We will briefly discuss the concept of the VIF in class.

Following code will calculate the VIF.

```
vif_function(y,x){
   n=length(y);p=ncol(x)
   x.std_matrix(nrow=n,ncol=p)
   y.std_(y-mean(y))/(stdev(y)*sqrt(n-1))
   for (i in 1:p){
    x.std[,i]_(x[,i]-mean(x[,i]))/(stdev(x[,i])*sqrt(n-1))
   }
   VIF_ginverse(crossprod(x.std))
   varif_matrix(nrow=p,ncol=1)
   for (i in 1:p){
    varif[i,]_VIF[i,i]
   }
varif
}
```

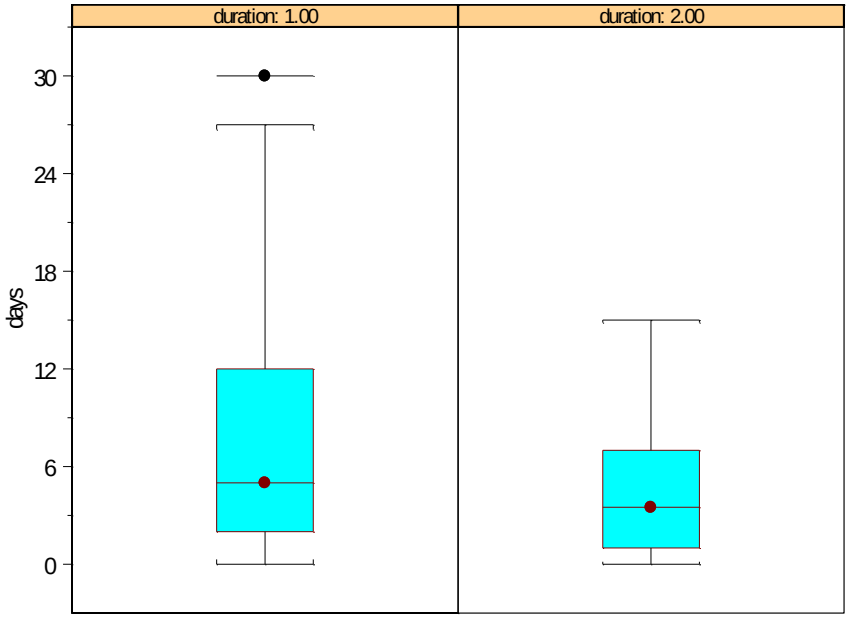
```
>attach(realestate)
> y_price
> x_data.frame(Hsize,attrac,sty1,sty2,Hsize*sty2)
> vif(y,x)
           [,1]
[1,]   1.723070
[2,]   1.248420
[3,]   1.646438
[4,] 105.039685
[5,] 108.206302
```

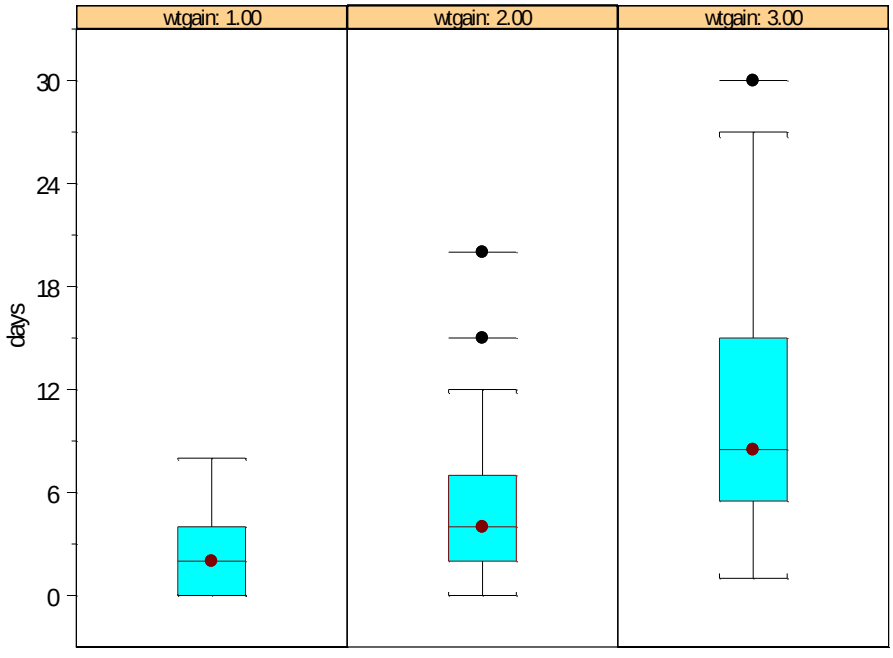
The VIF values considerably larger than 1 indicate muticollinearity problems.

ANALYSIS OF VARIANCE

Looking at the data: Kidney data revisited

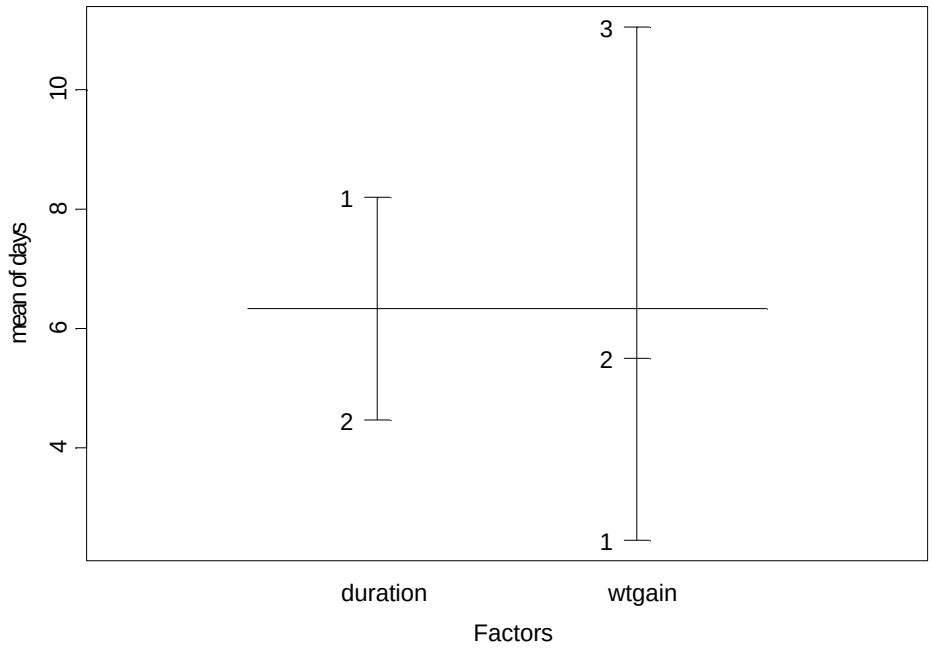
Trellis boxplot conditioning on duration and on wtgain, respectively.





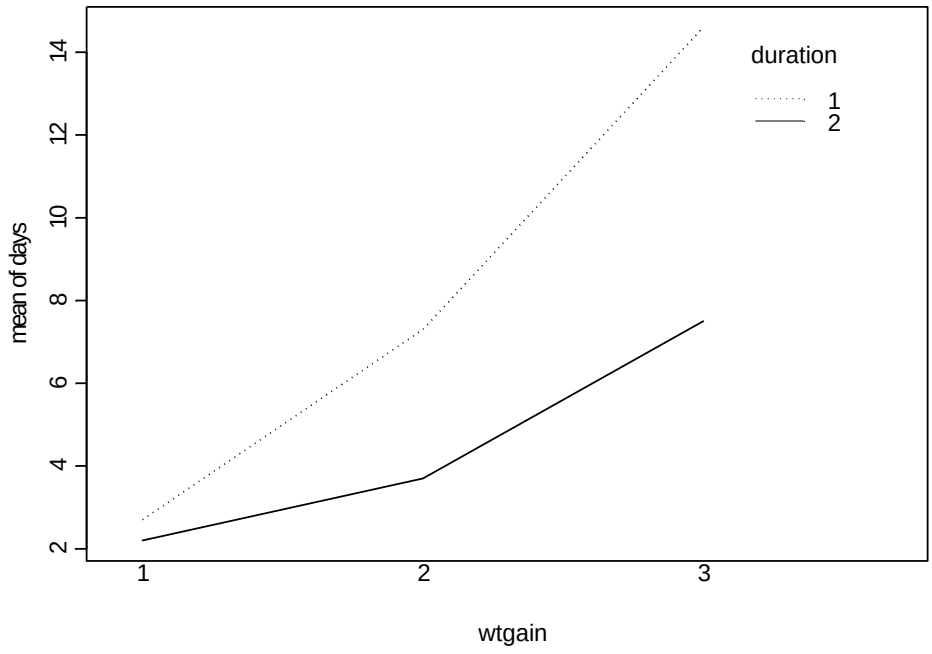
Other useful plots by plot.design and interaction.plot.

```
> plot.design(kidney[,1:3])
```



Main factor effects are evident.

```
> interaction.plot(wtgain,duration,days,data=kidney)
```



Interaction effect does not seem to significant.

Before applying the *aov* function, be sure that the data types of two factors, duration and wtgain, are "factor". If they are recorded as "factor", use *as* function or Data → Change Data Type on the menu bar.

One factor fixed effect ANOVA

Using aov function

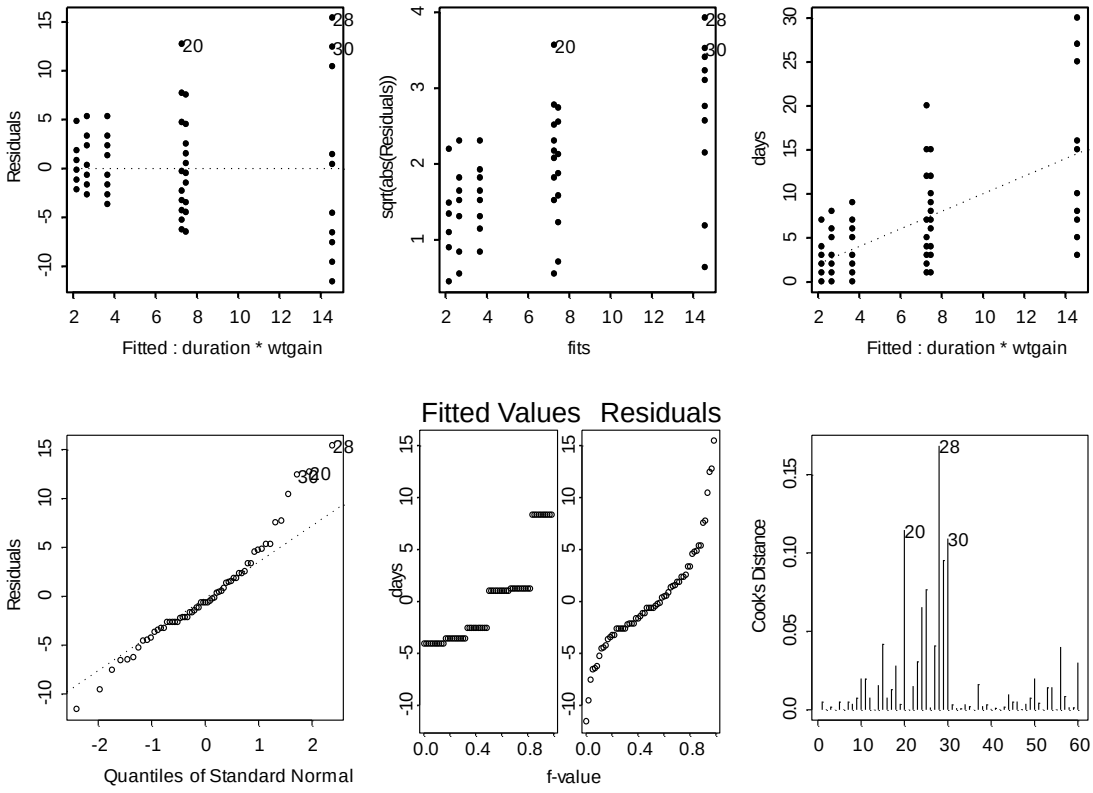
```
> duration.anova_aov(days~duration,data=kidney)
> summary(duration.anova)

            Df Sum of Sq  Mean Sq  F Value      Pr(F)
 duration    1   209.067 209.0667 4.981322 0.02950016
Residuals   58  2434.267  41.9701


> dummy.coef(duration.anova)
$"(Intercept)":
 (Intercept)
    6.333333

$duration:
         1         2
 1.866667 -1.866667

> contr.helmert(2)
  [,1]
1   -1
2    1


> duration.lm_lm(days~duration,data=kidney)
> summary(duration.lm)

Call: lm(formula = days ~ duration, data = kidney)
Residuals:
  Min     1Q Median    3Q  Max
 -8.2 -4.267 -1.467 2.533 21.8

Coefficients:
             Value Std. Error t value Pr(>|t|)
(Intercept)  6.3333  0.8364     7.5725  0.0000
   duration -1.8667  0.8364    -2.2319  0.0295

Residual standard error: 6.478 on 58 degrees of freedom
Multiple R-Squared: 0.07909
F-statistic: 4.981 on 1 and 58 degrees of freedom, the p-value is 0.0295


Correlation of Coefficients:
         (Intercept)
duration 0


> model.tables(duration.anova)

Tables of effects

 duration
       1       2
  1.8667 -1.8667

> model.tables(duration.anova,"means")
```

```
Tables of means
Grand mean

 6.3333

 duration
      1      2
 8.2000 4.4667

Using my cotrast

> my.cotr.l2_matrix(c(1,0),ncol=1)
> my.cotr.l2
     [,1]
[1,]    1
[2,]    0


> duration.anova.new_aov(days~C(duration,my.cotr.l2),data=kidney)
> summary(duration.anova.new)
                        Df Sum of Sq  Mean Sq  F Value      Pr(F)
C(duration, my.cotr.l2)  1   209.067 209.0667 4.981322 0.02950016
              Residuals 58  2434.267  41.9701

> dummy.coef(duration.anova.new)
$"(Intercept)":
 (Intercept)
    4.466667

$"C(duration, my.cotr.l2)":
        1 2
 3.733333 0



> wtgain.anova_aov(days~wtgain,data=kidney)
> summary(wtgain.anova)

          Df Sum of Sq  Mean Sq  F Value         Pr(F)
   wtgain  2   760.433 380.2167 11.51009 0.00006327712
Residuals 57  1882.900  33.0333

> summary(wtgain.anova,split=list(wtgain=list(L=1,Q=2)))

            Df Sum of Sq  Mean Sq  F Value      Pr(F)
     wtgain  2   760.433 380.2167 11.51009 0.00006328
  wtgain: L  1    93.025  93.0250  2.81609 0.09879855
  wtgain: Q  1   667.408 667.4083 20.20409 0.00003460
  Residuals 57  1882.900  33.0333
```

Two-factor fixed effects ANOVA

```
> kidney.anova_aov(days~duration*wtgain,data=kidney)
> summary(kidney.anova)

                Df Sum of Sq  Mean Sq  F Value     Pr(F)
       duration  1   209.067 209.0667  7.21472 0.0095871
         wtgain  2   760.433 380.2167 13.12097 0.0000227
duration:wtgain  2   109.033  54.5167  1.88133 0.1622404
      Residuals 54  1564.800  28.9778

      > model.tables(kidney.anova,"means")
```

```
Tables of means
Grand mean

 6.3333

 duration
      1      2
 8.2000 4.4667

 wtgain
     1     2     3
  2.45  5.50 11.05

duration:wtgain
Dim 1 : duration
Dim 2 : wtgain
     1    2    3
1  2.7  7.3 14.6
2  2.2  3.7  7.5

> par(mfrow=c(2,3))
> plot(kidney.anova)
> par(mfrow=c(1,1))
```



The residual plot shows a strong evidence of non-constant variance. As the fitted values increase the residuals increase. From the QQ plot we also suspect a skewness of the error distribution. This recommends a proper transformation of the response.

Transformation
A variance stabilizing transformation also corrects a non-normality in the error distribution. To suggest a appropriate transformation of the response we look at the relation between the treatment mean and the variance. Suggested transformations are

Relation	Transformation
$\sigma_i^2 = k\mu_i$	$Y' = \sqrt{Y}$
$\sigma_i^2 = k\mu_i^2$	$Y' = \log(Y)$
$\sigma_i = k\mu_i^2$	$Y' = 1/Y$
$\sigma_i^2 = \pi_i(1-\pi_i)/n_i,\ \pi_i = pr(ith\ level)$	$Y' = \arcsin(\sqrt{Y})$

Let's check which relation is most suitable for our kidney data.

```
> std.kidney_tapply(days,list(duration,wtgain),stdev)
> std.kidney
         1        2        3
1 2.790858 6.290204 9.720540
2 2.299758 2.945807 4.249183

> mean.kidney_tapply(days,list(duration,wtgain),mean)
> mean.kidney
    1   2    3
1 2.7 7.3 14.6
2 2.2 3.7  7.5

> sd.to.m_std.kidney/mean.kidney
> sd.to.m
         1         2         3
1 1.033651 0.8616718 0.6657904
2 1.045345 0.7961640 0.5665577

> sdsq.to.m_std.kidney^2/mean.kidney
> sdsq.to.m
         1        2        3
1 2.884774 5.420091 6.471842
2 2.404040 2.345345 2.407407

> sd.to.msq_std.kidney/mean.kidney^2
> sd.to.msq
          1         2          3
1 0.3828338 0.1180372 0.04560208
2 0.4751567 0.2151795 0.07554103
```

The ration σ_i / μ_i is the most stable, and this suggests a log transformation. Note that the response, *days*, is a count with some zero counts. The square root transformation also looks OK and thus we try the square root transformation below.
For more details, we may also consider a power transformation of the form (Box-Cox transformation):

$$Y' = Y^{\lambda}$$

To find the power, λ, we regress $\log\sigma_i$ on $\log\mu_i$, that is we fit

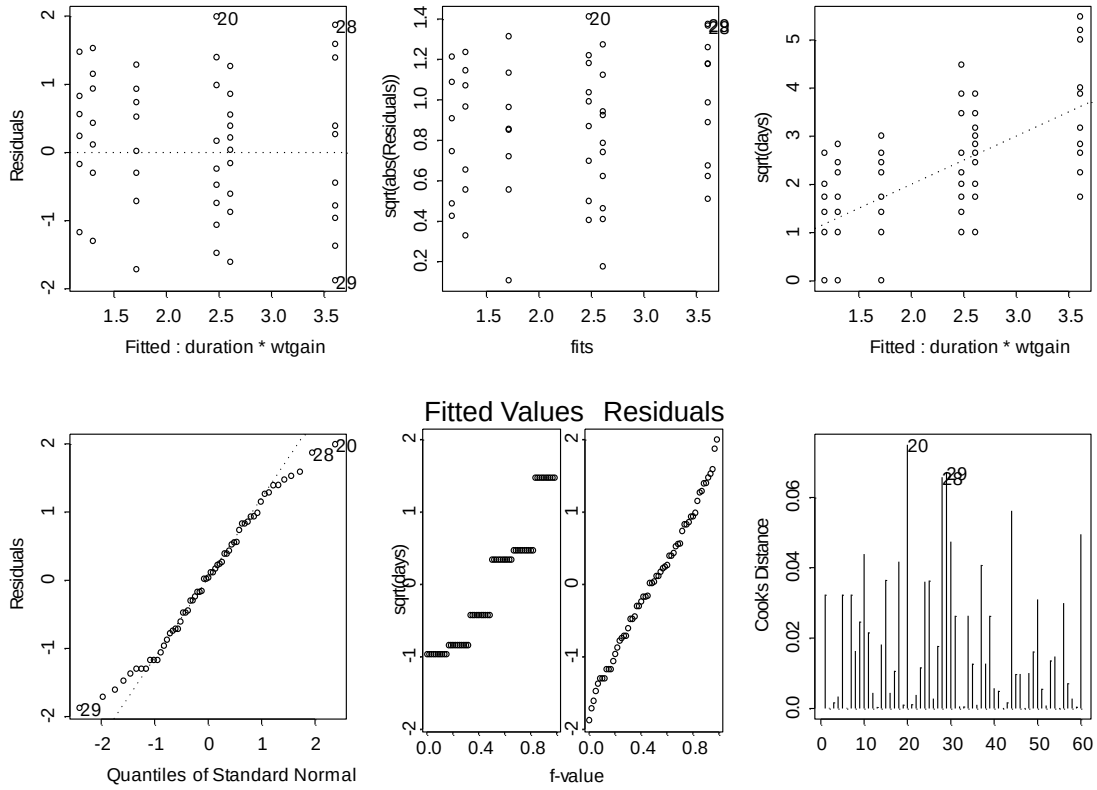
$$E(\log\sigma_i) = \hat{\beta}_0 + \hat{\beta}_1 \log\mu_i$$

We estimate λ by $\hat{\lambda} = 1 - \hat{\beta}_1$. The basic concept will be discussed in class.

```
> std.kidney_as(std.kidney,"vector")
> mean.kidney_as(mean.kidney,"vector")
> std.kidney
[1] 2.790858 2.299758 6.290204 2.945807 9.720540 4.249183
> plot(log(mean.kidney),log(std.kidney))
> log.fit_lsfit(log(mean.kidney),log(std.kidney))
> par(mfrow=c(1,1))
> plot(log(mean.kidney),log(std.kidney))
> abline(log.fit)
> log.fit$coef
 Intercept         X
 0.2176716 0.7359919
```

From above we estimate $\hat{\lambda} = .264$. The value is about in the middle of 0 and 0.5 and this suggests that we may use either square root or log transformation. Note the $\lambda = 0$ means log transformation by definition. For the same reason mentioned above, we take a square root transformation of the response, *days*. You may also consider $Y' = \log(Y+1)$.

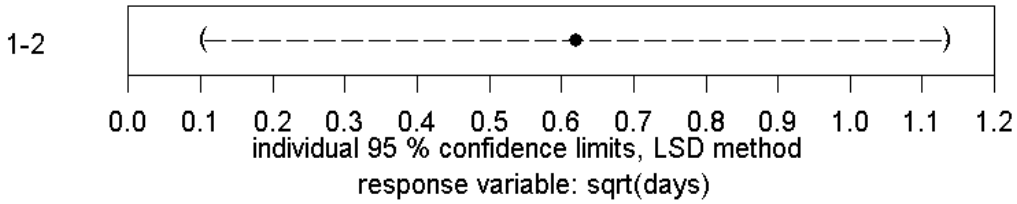
```
> sqrt.kidney.anova_aov(sqrt(days)~duration*wtgain)
> summary(sqrt.kidney.anova)
                Df Sum of Sq  Mean Sq  F Value     Pr(F)
      duration   1   5.95727  5.95727  5.48996 0.0228395
        wtgain   2  35.12196 17.56098 16.18343 0.0000031
duration:wtgain  2   2.02990  1.01495  0.93533 0.3987204
     Residuals  54  58.59654  1.08512
```



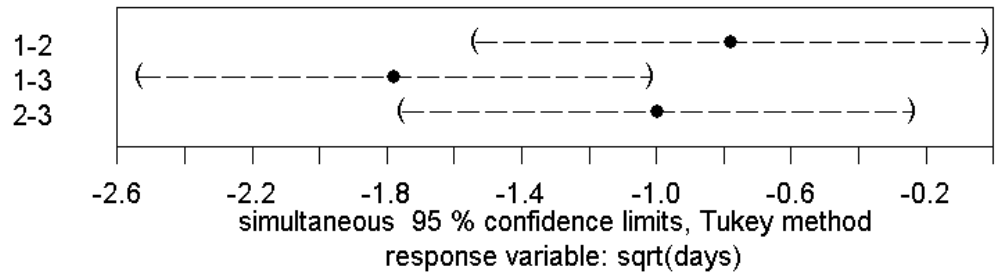
The interaction effect is not significant and thus we may fit the model without the interaction. Next time we will discuss Multiple comparison procedure and random and mix effect model.

Multiple Comparisons: all pairwise comparison

```
> multicomp(sqrt.fit,focus="duration",plot=T)

95 % non-simultaneous confidence intervals for specified
linear combinations, by the Fisher LSD method

critical point: 2.0049
response variable: sqrt(days)

intervals excluding 0 are flagged by '****'

    Estimate Std.Error Lower Bound Upper Bound
1-2     0.62     0.257       0.105        1.13 ****
```

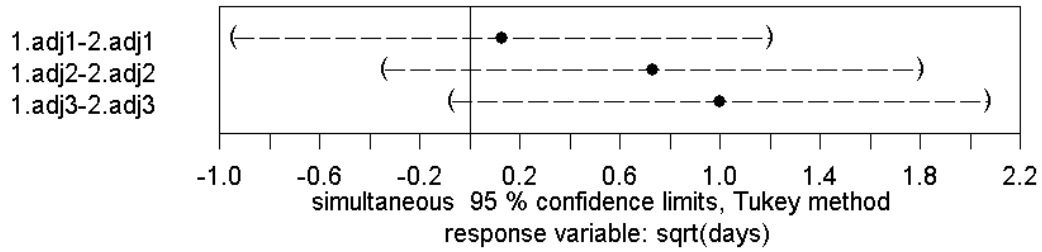



```
> multicomp(sqrt.fit,focus="wtgain",plot=T)

95 % simultaneous confidence intervals for specified
linear combinations, by the Tukey method

critical point: 2.41
response variable: sqrt(days)

intervals excluding 0 are flagged by '****'

    Estimate Std.Error Lower Bound Upper Bound
1-2   -0.780     0.314       -1.54     -0.0228 ****
1-3   -1.780     0.314       -2.53     -1.0200 ****
2-3   -0.997     0.314       -1.75     -0.2400 ****
```

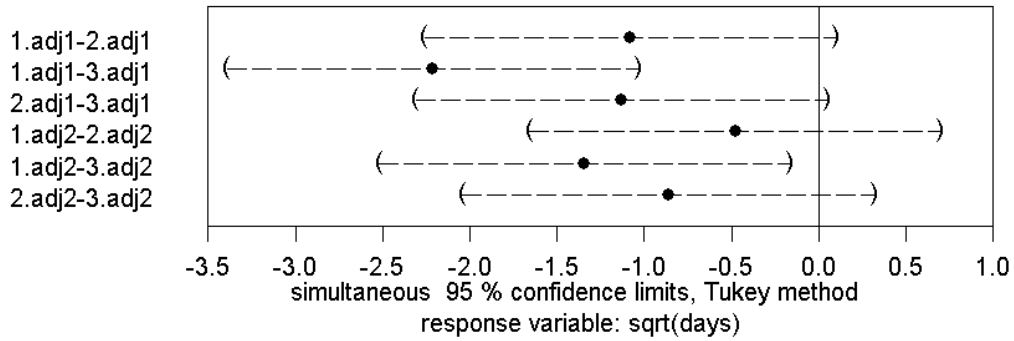



```
> multicomp(sqrt.fit,focus="duration",adjust=list(wtgain=seq(3)),plot=T)

95 % simultaneous confidence intervals for specified
linear combinations, by the Tukey method

critical point: 2.41
response variable: sqrt(days)
```

```
intervals excluding 0 are flagged by '****'

              Estimate Std.Error Lower Bound Upper Bound
1.adj1-2.adj1    0.128     0.444     -0.9430        1.20
1.adj2-2.adj2    0.731     0.444     -0.3400        1.80
1.adj3-2.adj3    1.000     0.444     -0.0712        2.07
```




```
> multicomp(sqrt.fit,focus="wtgain",adjust=list(duration=seq(2)),plot=T)

95 % simultaneous confidence intervals for specified
linear combinations, by the Tukey method

critical point: 2.6509
response variable: sqrt(days)

intervals excluding 0 are flagged by '****'

              Estimate Std.Error Lower Bound Upper Bound
1.adj1-2.adj1   -1.080     0.444       -2.26      0.0964
1.adj1-3.adj1   -2.210     0.444       -3.39     -1.0300 ****
2.adj1-3.adj1   -1.130     0.444       -2.31      0.0464
1.adj2-2.adj2   -0.479     0.444       -1.66      0.6990
1.adj2-3.adj2   -1.340     0.444       -2.52     -0.1640 ****
2.adj2-3.adj2   -0.863     0.444       -2.04      0.3150
```



S+ uses the Tukey's pairwise comparison as default. If you want to specify other methods (like Scheffe, Bonferroni, etc) use the option *method="scheffe"* for Scheffe method and *method="bon"* for Bonferroni method, etc.

```
> multicomp(sqrt.fit,focus="wtgain",method="scheffe",plot=T)
```

Inference for Linear Combinations of factor levels

We might want to construct a confidence interval for a linear combination of factor levels. Here, we will consider the interval for wtgain2- (wtgain1+wtgain3)/2. We have seen that the interaction is not significant and thus we fit the model with the two main effects only. Note that with the full model the linear combination is not estimable.

```
> main.fit_aov(sqrt(days)~duration+wtgain)
> summary(main.fit)
          Df Sum of Sq  Mean Sq  F Value      Pr(F)
 duration  1   5.75965  5.75965  5.83347 0.01901321
   wtgain  2  31.73837 15.86919 16.07258 0.00000305
Residuals 56  55.29134  0.98735

> lmat_matrix(c(0,0,0,-.5,1,-.5),ncol=1,dimnames=list(NULL,"wt2-(wt1+wt3)/2"))
> lmat
     wt2-(wt1+wt3)/2
[1,]             0.0
[2,]             0.0
[3,]             0.0
[4,]            -0.5
[5,]             1.0
[6,]            -0.5

> multicomp(main.fit,lmat=lmat,method="bon")

95 % non-simultaneous confidence intervals for specified
linear combinations, by the Fisher LSD method

critical point: 2.0032
response variable: sqrt(days)

intervals excluding 0 are flagged by '****'

                Estimate Std.Error Lower Bound Upper Bound
wt2-(wt1+wt3)/2   -0.109     0.272      -0.654       0.437
```

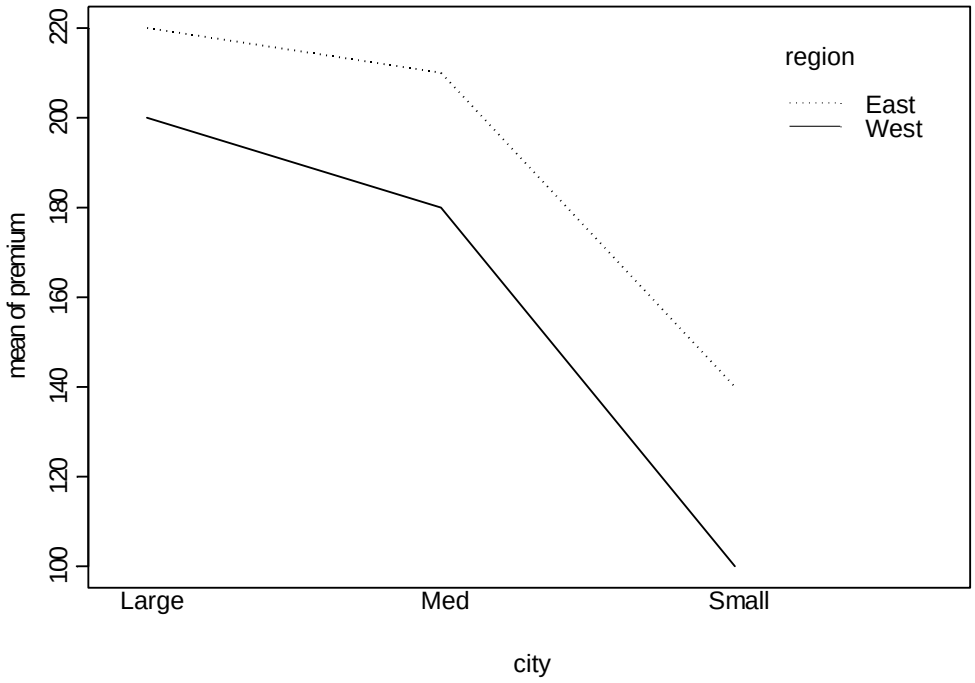
One case per treatment (no interaction model)

```
> premium_c(140,100,210,180,220,200)
> city_c("Small","Small","Med","Med","Large","Large")
> region_c("East","West","East","West","East","West")
> insur_data.frame(premium,city,region)
> insur
  premium  city region
1     140 Small   East
2     100 Small   West
3     210   Med   East
4     180   Med   West
5     220 Large   East
6     200 Large   West

> attach(insur)
> insur.anova_aov(premium~city+region)
> summary(insur.anova)
          Df Sum of Sq Mean Sq F Value      Pr(F)
     city  2      9300    4650      93 0.01063830
   region  1      1350    1350      27 0.03509872
Residuals  2       100      50

> model.tables(insur.anova,"means")

Tables of means
Grand mean
```

```
175

city
Large Med Small
210   195 120

region
East West
190  160
```

Test for Interaction effect

```
> interaction.plot(city,region,premium,lwd=2)
```



Tukey's 1df Test for no-interaction

Random and Mixed effects Models

Single-factor Random effect model

Model: $Y_{ij} = \mu_{\cdot} + \tau_i + \varepsilon_{ij},\ i = 1,...,r, j = 1,...,n$

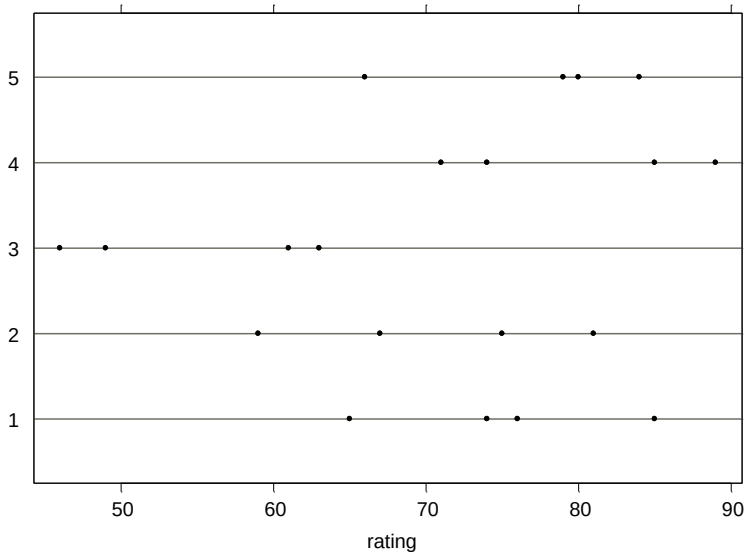
$\mu_{\cdot}$ is a constant, τ_i are iid $N(0,\sigma_{\mu}^2)$, ε_{ij} are iid $N(0,\sigma^2)$, τ_i and ε_{ij} are independent.

As an example, we adopt the Apex Enterprises Example in Neter, et.al., chapter 24 on the evaluating ratings of potential employees by its personnel officers. Five personnel officers were randomly selected and four prospective employees were assigned at random to each officer.

```
> apex$officer_as.factor(apex$officer)
```

```
> apex$employee_as.factor(apex$employee)
> attach(apex)
```

> dotplot(officer~rating,pch=16,col=1)



Test for the factor effect

For the random effect to test for the main effect we formulate the hypothesis in terms of the variance, σ_μ^2;

$$H_0 : \sigma_\mu^2 = 0 \text{ vs } H_1 : \sigma_\mu^2 > 0$$

For this single-factor random effect ANOVA model, testing procedure for the main effect is identical to that for the fixed effect. The different between the random and fixed effect appears in the expected mean square. The traditional ANOVA model is

Source	df	Sum of Squares	Mean Squares	E(MS)
Treatment	*r-1*	*SSTR*	*MSTR*	$\sigma^2 + n\sigma_\mu^2$
Residuals	*r(n-1)*	*SSE*	*MSE*	σ^2
Total	*rn-1*	*SSTO*		

We use the function *raov*, instead of *aov*, for the random effect model. From the ANOVA table below we have the estimated variances (E(MS) in the table above). Note that to use the *raov* function the design must be balance. For unbalance design we may use *varcomp* function that will be discussed later in this section.

```
> random.fit_raov(rating~officer)
> summary(random.fit)
          Df Sum of Sq  Mean Sq Est. Var.
  officer  4   1579.70 394.9250  80.41042
Residuals 15   1099.25  73.2833  73.28333
```

The calculated value of the test statistic is : $F^* = \frac{MSTR}{MSE} = \frac{394925}{732833} = 5.39$. Under the null hypothesis this statistic follows an F(r-1,n(r-1)).

```
> qf(.95,4,15)
[1] 3.055568
```

Since F* is larger than the critical value we reject the null hypothesis and conclude that at 0.05 level of significance the mean ratings of the personnel officers are different. The p-value of the test is 0.0068.

```
> 1-pf(5.39,4,15)
[1] 0.006797778
```

Inference for $\mu_{\cdot}$

We may want to construct a confidence interval for the overall mean, $\mu_{\cdot}$. The 95% CI is

$$\bar{Y}_{\cdot\cdot} \pm t(.975, r-1) * \frac{MSTR}{rn}$$

```
> model.tables(random.fit,"means")

Tables of means
Grand mean

 71.45

 officer
    1     2     3     4     5
75.00 70.50 54.75 79.75 77.25
```

For this example, the 95% CI for the overall mean is: $71.45 \pm 2.132(4.44) = (62,81)$.

CI for $\frac{\sigma_\mu^2}{\sigma_\mu^2 + \sigma^2}$

For the random effect model the above ratio is in interest as it measures the proportion of the total variability of the response that is accounted for by the variability of the level means. The lower and upper bound of the CI based on F distribution is;

$$L^* = \frac{L}{1+L} \text{ and } U^* = \frac{U}{1+U}$$

where

$$L = \frac{1}{n}\left[\frac{MSTR}{MSE}\left(F(1-\alpha/2; r-1, r(n-1))\right)^{-1} - 1\right]$$

and

$$U = \frac{1}{n}\left[\frac{MSTR}{MSE}\left(F(\alpha/2; r-1, r(n-1))\right)^{-1} - 1\right]$$

For the above example, MSTR=394.925, MSE=73.2833, r=5, n=4, F(.025,4,15)=3.804, F(.975,4,15)=0.116, then L=0.104 and U=11.36. The calculated 95% CI is (0.104/1.104, 11.36/12.36) = (0.09, 0.92).

You may want to construct CI's for σ^2 and σ_μ^2. Both intervals base on chi-square distribution and I will leave it to you (see Chapter 24, Neter, et.al.).

Two-factor Model : Model II(both random) and Model III (mixed)

Model II (balanced design)

Model: $Y_{ijk} = \mu_{\cdot\cdot} + \alpha_i + \beta_j + (\alpha\beta)_{ij} + \varepsilon_{ijk},\ i = 1,...,a,\ j = 1,...,b, k = 1,...,n$

$\mu_{\cdot}$ is a constant, α_i are iid $N(0,\sigma_\alpha^2)$, β_j are iid $N(0,\sigma_\beta^2)$, $(\alpha\beta)_{ij}$ are iid $N(0,\sigma_{\alpha\beta}^2)$, ε_{ijk} are iid $N(0,\sigma^2)$, $\alpha_i,\beta_j,(\alpha\beta)_{ij}$ and ε_{ij} are pairwise independent.

Model III (balanced design, factor A fixed, factor B random)

Model: $Y_{ijk}=\mu_{\cdot\cdot}+\alpha_i+\beta_j+(\alpha\beta)_{ij}+\varepsilon_{ijk},\ i=1,...,a,\ j=1,...,b,k=1,...,n$

$\mu_{\cdot}$ is a constant, α_i are constant with $\sum\alpha_i=0$, β_j are iid $N(0,\sigma_\beta^2)$, $(\alpha\beta)_{ij}$ are iid $N(0,\frac{a-1}{a}\sigma_{\alpha\beta}^2)$ with $\sum_i(\alpha\beta)_{ij}=0$, ε_{ijk} are iid $N(0,\sigma^2)$, $\alpha_i,\beta_j,(\alpha\beta)_{ij}$ and ε_{ij} are pairwise independent.

The expected mean squares and the test statistics are given in the following table.

Source	df	MS	E(MS)(M II)	E(MS)(M III)	F* (M II)	F* (M III)
A	a-1	MSA	$\sigma^2+nb\sigma_\alpha^2+n\sigma_{\alpha\beta}^2$	$\sigma^2+nb\frac{\sum\alpha_i^2}{a-1}+n\sigma_{\alpha\beta}^2$	$\frac{MSA}{MSAB}$	$\frac{MSA}{MSAB}$
B	b-1	MSB	$\sigma^2+na\sigma_\beta^2+n\sigma_{\alpha\beta}^2$	$\sigma^2+na\sigma_\beta^2$	$\frac{MSB}{MSAB}$	$\frac{MSB}{MSE}$
AB	(a-1)(b-1)	MSAB	$\sigma^2+n\sigma_{\alpha\beta}^2$	$\sigma^2+n\sigma_{\alpha\beta}^2$	$\frac{MSAB}{MSE}$	$\frac{MSAB}{MSE}$
Residual	ab(n-1)	MSE	σ^2	σ^2		
Total	abn-1					

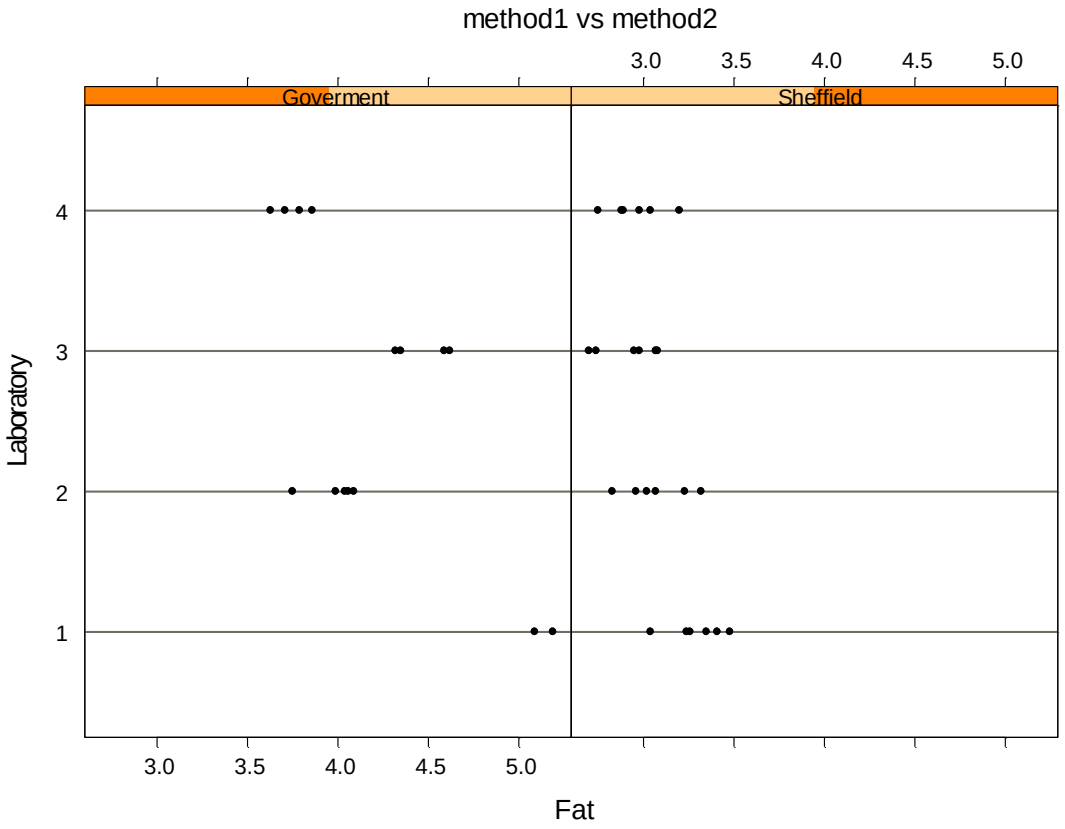
For the balanced Model II we can use *raov* function as before.

Mixed effects

For mixed effects ANOVA we use *varcomp* function. The *varcomp* function is also used for unbalanced designs. Note that for the balanced mixed effects designs the estimated mean squares and corresponding degrees of freedom are identical to that of the fixed effects. However, the testing procedures vary as in the table above. Hence, for testing hypothesis for factor effects we can simply use *aov* function and recalculate the test statistics. For the mixed effects we also interested in the variance structures for further analyses. We use *varcomp* function for this purpose. We discuss the use of *varcomp* function using an unbalanced mixed effects example. Note that for the unbalanced design (either fixed, random, or mixed) the ANOVA modeling is difficult. We used the general linear test approach based on the maximum likelihood method. Example: (Unbalanced Mixed effect)

We used the data in Table 24.11 of Neter, et.al. The milkfat contents of a company's yogurt product are measured using two different measurement methods (a=1) and randomly selected four laboratories (b=4). This is unbalanced mixed effect model; the factor Method is fixed and the factor Laboratory is random.

```
> fat.data$method_as.factor(fat.data$method)
> fat.data$lab_as.factor(fat.data$lab)
> fat.data$method[fat.data$method==1]_"Goverment"
> fat.data$method[fat.data$method==2]_"Sheffield"

> dotplot(lab~fat|method,ylab="Laboratory",xlab="Fat",main="method1 vs
method2",pch=16,col=1,data=fat.data)
```



The following procedure is necessary to define which factor is random and which is fixed.

```
> is.random(fat.data$lab)_T
> is.random(fat.data$method)_F
> is.random(fat.data)
 method lab
   F   T

> summary(varcomp(fat~method*lab,method="ml",data=fat.data))
Call:
varcomp(formula = fat ~ method * lab, data = fat.data, method = "ml")
Variance Estimates:
            Variance
       lab 0.05443933
method:lab 0.08570636
 Residuals 0.02325424
Method:  ml
Approximate Covariance Matrix of Variance Estimates:
                    lab  method:lab   Residuals (Intercept)      method
        lab  0.00595752 -0.00194176 -0.00000400  0.00000255 -0.00000171
 method:lab -0.00194176  0.00420470 -0.00001463 -0.00000070 -0.00000515
  Residuals -0.00000400 -0.00001463  0.00003545 -0.00000015  0.00000618
(Intercept)  0.00000255 -0.00000070 -0.00000015  0.00058123 -0.00000445
     method -0.00000171 -0.00000515  0.00000618 -0.00000445  0.01138762

Coefficients:
 (Intercept)    method
    3.694149 -0.632899
Approximate Covariance Matrix of Coefficients:
              (Intercept)        method
```

```
(Intercept)  0.0000135161 -0.0000001028
     method -0.0000001028  0.0002647853
```

Note that other possible methods in the estimation option of varcomp function are "reml", "minque0", and "winsor". Refer the text for the description of each method. From the output we can summary the results as

Parameter	MLE	SE	z*	p-value
$\mu_{..}$	3.694	0.024	large	0.00
α_1	0.633	0.107	5.916	0.00
σ^2_β	0.054	0.077	0.701	0.483
$\sigma^2_{\alpha\beta}$	0.086	0.065	1.32	0.187
σ^2	0.023	0.006	3.83	0.00

From the table, we see that the method effect is significant and the interaction effect between the factors is not quite significant.

Now let's consider the subset of the fat.data to study the balanced mixed effects. We used the data with the count 1 and 2 so that each treatment has two observarions.

```
> summary(varcomp(fat~method*lab,method="ml",subset=(count<3),data=fat.data))
Call:
varcomp(formula = fat ~ method * lab, data = fat.data, method = "ml", subset =
(count < 3))
Variance Estimates:
             Variance
       lab 0.06845940
method:lab 0.06776157
 Residuals 0.01849374
Method:  ml
Approximate Covariance Matrix of Variance Estimates:
                     lab   method:lab    Residuals (Intercept)      method
        lab  0.006461886 -0.001482569  0.000000000 0.000000000 0.000000000
 method:lab -0.001482569  0.002986884 -0.000043493 0.000000000 0.000000000
  Residuals  0.000000000 -0.000043493  0.000086986 0.000000000 0.000000000
(Intercept)  0.000000000  0.000000000  0.000000000 0.000494539 0.000000000
     method  0.000000000  0.000000000  0.000000000 0.000000000 0.009626054

Coefficients:
 (Intercept)    method
    3.740625 -0.618125
Approximate Covariance Matrix of Coefficients:
            (Intercept)       method
(Intercept) 9.1459e-006 0.0000000000
     method 0.0000e+000 0.0001780217
```

Using the output we may make inferences on the variances.

Analysis of Covariance (ANCOVA)

Covariance Model (Single factor) : $Y_{ij} = \mu_{.} + \tau_i + \gamma x_{ij} + \varepsilon_{ij},\ i = 1,...,r,\ j = 1,...,n_i$,

where $\mu_{.}$ is a constant (overall mean), τ_i is a fixed treatment effect, γ is a regression coefficient relating Y with x, x_{ij} is a mean-adjusted continuous predictor variable (concomitant variable), and ε_{ij} are iid $N(0,\sigma^2)$.

Example(Neter, et.al. Table 25.1): To study the effects of three types of promotions (treatments, r=3)) on sales (response) of crackers. The concomitant variable is sales from preceding period.

First to verify the model assumption of the equal regression coefficient, γ, over treatment, we need to plot Y versus X for each treatment level. Before further analysis it is convenient to make a groupedData data frame.

```
> cracker$period_cracker$period-mean(cracker$period)
> period.adj_cracker$period-mean(cracker$period)
> cracker_data.frame(cracker,period.adj)

> cracker.group_groupedData(sales~period.adj|treatment,data=cracker)
```

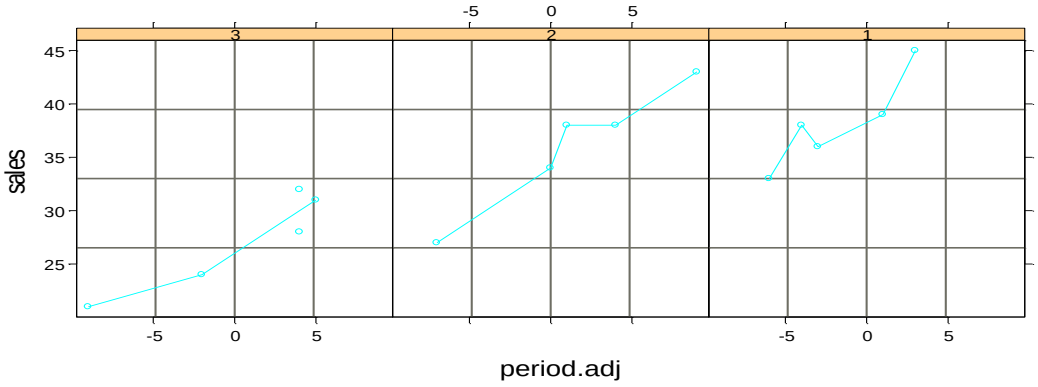
Now the object cracker.group is a groupedData object. The groupedData object can be summarized by using *gsummary* function.

```
    > gsummary(cracker.group)
      sales period treatment count period.adj
    3  27.2   25.4         3     3        0.4
    2  36.0   26.4         2     3        1.4
    1  38.2   23.2         1     3       -1.8
```

To make a scatterplot by group (treatment) simply type

```
> plot(cracker.group)
```

From the plot we see that there is a clear linear relation between Y and x and we don't see any evidence of changing slopes over treatment. For a formal test of parallel slopes we fit a regression model with interactions and test for the zero interaction coefficients.



Before the model fitting we assign a contrast variable (dummy) for the treatment.

```
> my.cotr_matrix(c(1,0,-1,0,1,-1),ncol=2)
> my.cotr
   [,1] [,2]
[1,]    1    0
[2,]    0    1
[3,]   -1   -1
```

```
> anova(lm(sales~C(treatment,my.cotr)*period.adj,data=cracker.group))
Analysis of Variance Table

Response: sales

Terms added sequentially (first to last)
                                Df Sum of Sq  Mean Sq  F Value     Pr(F)
          C(treatment, my.cotr)  2  338.8000 169.4000 48.36801 0.0000153
                     period.adj  1  269.0287 269.0287 76.81453 0.0000106
C(treatment, my.cotr):period.adj  2    7.0505   3.5252  1.00654 0.4031810
                      Residuals  9   31.5208   3.5023
```

The p-value for the no interactions test is very high (.4), and this implies that the parallel slopes are evident.

Now ANCOVA can be done using regression approach. Indeed, the ANCOVA model is a special case of regression model.

```
> cracker.lm_lm(sales~C(treatment,my.cotr)+period.adj,data=cracker.group)
> summary(cracker.lm)

Call: lm(formula = sales ~ C(treatment, my.cotr) + period.adj, data =
cracker.group)
Residuals:
    Min      1Q  Median      3Q     Max
 -2.435  -1.274  -0.3363  1.671  2.487
```

```
Coefficients:
                         Value Std. Error  t value Pr(>|t|)
           (Intercept)  33.8000   0.4835    69.9079   0.0000
C(treatment, my.cotr)1  -6.9594   0.6850   -10.1599   0.0000
C(treatment, my.cotr)2   0.9420   0.6987     1.3483   0.2047
            period.adj   0.8986   0.1026     8.7592   0.0000

Residual standard error: 1.873 on 11 degrees of freedom
Multiple R-Squared: 0.9403
F-statistic: 57.78 on 3 and 11 degrees of freedom, the p-value is 5.082e-007

Correlation of Coefficients:
                      (Intercept) C(treatment, my.cotr)1 C(treatment, my.cotr)2
C(treatment, my.cotr)1  0.0000
C(treatment, my.cotr)2  0.0000     -0.4761
            period.adj  0.0000     -0.0599                -0.2056

> anova(cracker.lm)
Analysis of Variance Table

Response: sales

Terms added sequentially (first to last)
                     Df Sum of Sq  Mean Sq  F Value         Pr(F)
C(treatment, my.cotr)  2  338.8000 169.4000 48.31052 3.566360e-006
           period.adj  1  269.0287 269.0287 76.72324 2.731033e-006
            Residuals 11   38.5713   3.5065
```

The result will be discussed in class.

We may fit a reduced model without the treatment effect and do a generalized F-test (reduced vs. full) to test for the treatment effect.

SIMPLE LOGISTIC REGRESSION

In many studies, the response variable may be a binary r.v. taking on the value 0 (failure) and 1 (success).

Example: In a longitudinal study of coronary heart disease as a function of age, gender, cholesterol level, etc. The response (Y) can be coded as 0 (no disease) and 1 (disease)

Simple Regression?

$$Y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$$

Let $Y_i = \begin{cases} 1 & for "success" \\ 0 & for "failure" \end{cases}$

Assumption $\varepsilon_i \sim N(0, \sigma^2)$???

Note that for the binary response ε_i can take only two values

$$\varepsilon_i = 1 - \beta_0 - \beta_1 x_i \quad if\ Y_i = 1$$
$$\varepsilon_i = -\beta_0 - \beta_1 x_i \quad if\ Y_i = 0.$$

Therefore ε_i can NOT be normally distributed!! The simple regression model can NOT be used for the binary response.

Logistic Response Function

Let's denote π_i to be the probability of success for the ith observation, that is $P(Y_i = 1) = \pi_i$, and $P(Y_i = 0) = 1 - \pi_i$. We want to relate π_i to a linear combination of the independent variables, X_i.

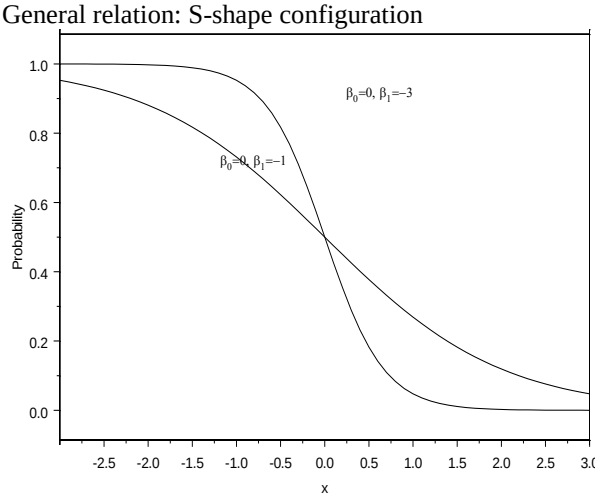
Note that generally for extremely large (or small) value of X_i the probability π_i is expected to be either 1 or 0.

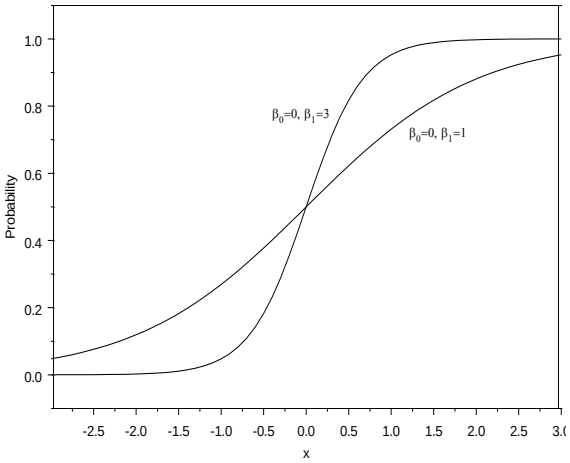
Example : $x_i = age, y_i = \begin{cases} 1 \ for "alive" \\ 0 \ for "death" \end{cases}$

For extremely large value of age, we expect the probability of survival to be 0.

What about the relationship between π_i and X_i for intermediate value of X_i? Can we expect a linear relationship? Pretty close

General relation: S-shape configuration





These shapes are very common in application.

Logistic Response Function :

$$\pi_i = \frac{\exp(\beta_0 + \beta_1 x_i)}{1 + \exp(\beta_0 + \beta_1 x_i)}$$

formulates the S-shape relationship.

Logistic Transformation

Logistic Regression Model:

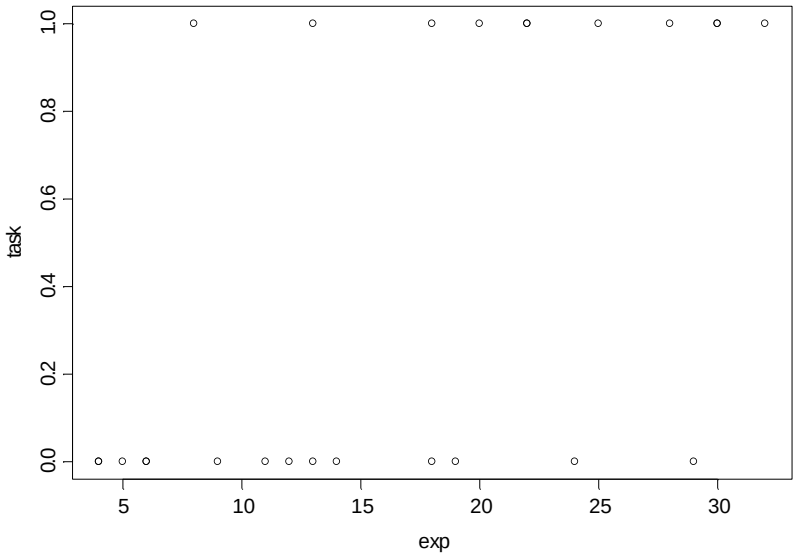
$$\log\left(\frac{\pi_i}{1-\pi_i}\right) = \beta_0 + \beta_1 x_i$$

We use the logistic transformation $\log\left(\frac{\pi_i}{1-\pi_i}\right)$ as a response in simple regression.

Note that since π_i is unknown, the response is unobservable. The parameters can be estimated via maximum likelihood method. We use *glm (generalized linear model)* function to fit the model with binomial family option.

The fitted response function is: $\hat{\pi}_i = \frac{\exp(b_0 + b_1 x_i)}{1+\exp(b_0 + b_1 x_i)}$.

Example (Neter, Table 14.1) We are interested in the effect of the experience on the ability to a complex programming task. For each of 25 individuals months of experience and the task result (1 for success, 0 for failure)




```
> fit.log_glm(task~exp,family=binomial,data=logist.data)
> summary(fit.log)

Call: glm(formula = task ~ exp, family = binomial, data = logist.data)
Deviance Residuals:
      Min         1Q     Median         3Q      Max
 -1.89916 -0.7508921 -0.4140038 0.7992195 1.962354

Coefficients:
                Value Std. Error   t value
(Intercept) -3.0596954  1.2589852 -2.430287
        exp  0.1614859  0.0649625  2.485833
(Dispersion Parameter for Binomial family taken to be 1 )

    Null Deviance: 34.29649 on 24 degrees of freedom
Residual Deviance: 25.42457 on 23 degrees of freedom

Number of Fisher Scoring Iterations: 4
```

```
Correlation of Coefficients:
    (Intercept)
exp -0.9214001

> 2*(1-pt(2.4858,23))
[1] 0.02062955
```

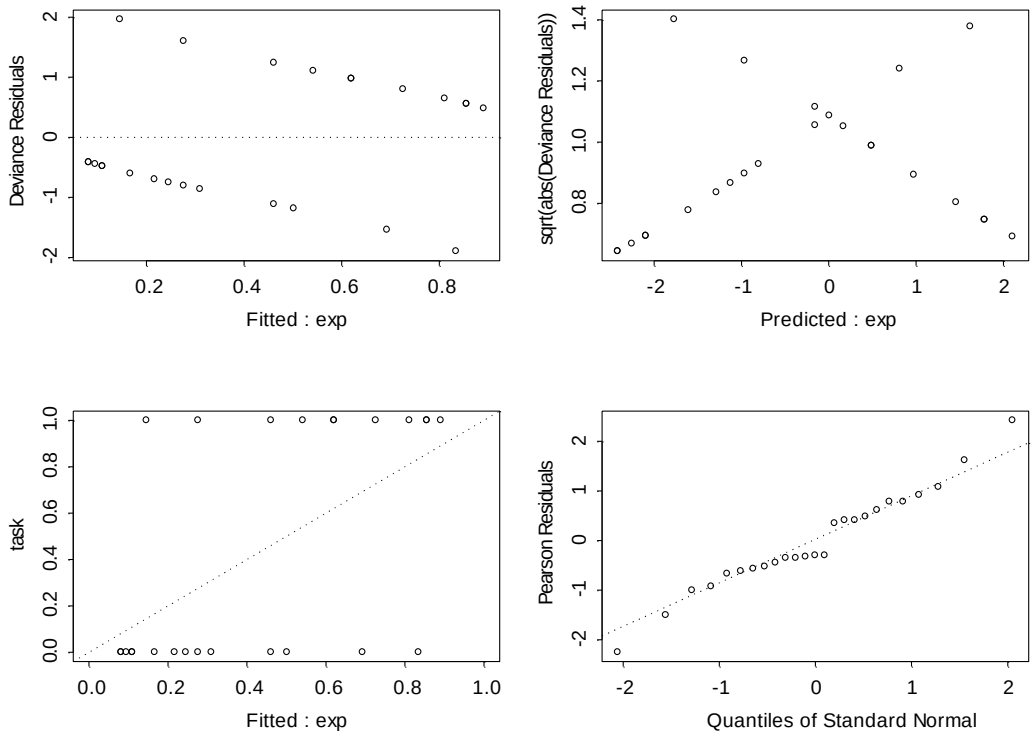
The p-value of 0.02 from a partial t-test testing the effect of the experience effect indicates that the effect is significant

We may do a chi-square test using the deviances (known as Person test) in the output. Again the null hypothesis is $Ho: \beta_1 = 0$. The null deviance is the value of the test statistic under the null hypothesis, reduced model with intercept only, and the residual deviance is the value of the test statistic under the full model, with intercept and slope. The subtraction of the two statistics will follow a chi-square distribution with one degree of freedom. For this example

$$G^2 = 34.29649 - 25.42457 = 8.872$$

The corresponding p-value of the test is 0.003, which is much less than the p-value for the t-test. The chi-square test is preferred to the t-test. More example will be discuss later.

```
> 1-pchisq(8.872,1)
[1] 0.002895777
> par(mfrow=c(2,2))
> plot(fit.log)
```



Note that unlike usual regression models, for the binary response the residuals are not normally distributed and the usual residual analyses are not applicable. The diagnostic plots above are not quite informative for

the logistic regression. To detect outliers or/and influential observations we can use leverages as before. Refer the example in the next section.

The fitted response function is $\hat{\pi}_i = \frac{\exp(-3.06+.1615x_i)}{1+\exp(-3.06+.1615x_i)}$.

We use the fitted response function to estimate the mean response at a given value of x. For example substituting 14 into x can obtain the estimated probability of successful performance for a person with 14 months experience. The estimated probability is 0.31.

The estimated odd ratio $OR = \exp(b_1) = \exp(.1615) = 1.175.$ So, with each additional month of experience increases the odds of completing the task by 17.5%.

Multiple Logistic Regression

In this section we will discuss the logistic regression with two or more predictors.

Example (Christensen 1997, Example 4.1.1)

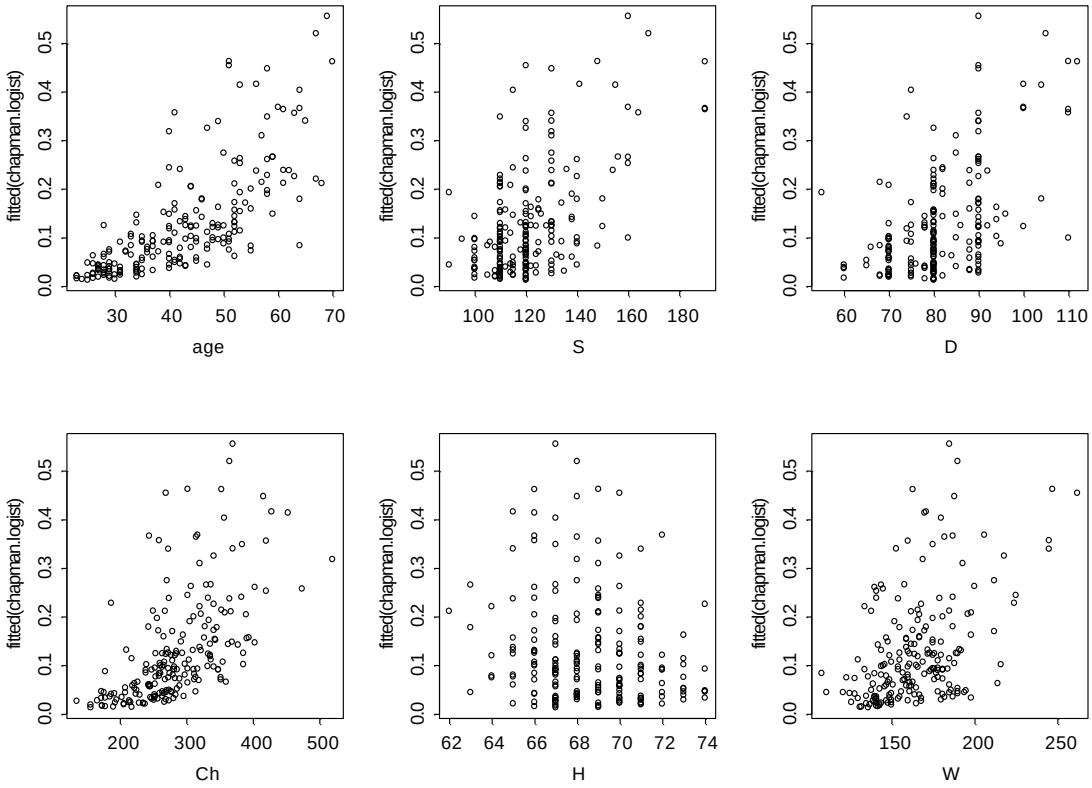
200 men are taken from the LA Heart Study to study the relations between the Coronary incident (CNT, 1 if an incident occurred, 0 otherwise) and six explanatory variables: age, Systolic blood pressure, Diastolic blood pressure, Cholesterol, Height, and Weight. The data can be downloaded from the class web (chapman.dat). We consider the logistic regression model:

$$\log\left(\frac{\pi_i}{1-\pi_i}\right) = \beta_0 + \beta_1 age_i + \beta_2 S_i + \beta_2 D_i + \beta_3 Ch_i + \beta_4 H_i + \beta_5 W_i$$

```
> summary(chapman.logist)

Call: glm(formula = CNT ~ age + S + D + Ch + H + W, family = binomial, d
ata
       = chapman)
Deviance Residuals:
       Min         1Q     Median         3Q      Max
 -1.112958 -0.5541206 -0.3906878 -0.2527301 2.681136

Coefficients:
                  Value  Std. Error    t value
(Intercept) -4.517319021 7.479489249 -0.6039609
        age  0.045899976 0.023529058  1.9507783
          S  0.006855721 0.020194525  0.3394842
          D -0.006936751 0.038343820 -0.1809092
         Ch  0.006306448 0.003631292  1.7366953
          H -0.074001542 0.106189623 -0.6968811
          W  0.020141537 0.009868974  2.0408948

(Dispersion Parameter for Binomial family taken to be 1 )

    Null Deviance: 154.5547 on 199 degrees of freedom

Residual Deviance: 134.8515 on 193 degrees of freedom

Number of Fisher Scoring Iterations: 5
```

```
Correlation of Coefficients:
    (Intercept)        age          S          D         Ch          H
age -0.2071983
  S  0.0130841  -0.2675552
  D -0.2094813   0.0126957 -0.7730439
 Ch -0.1406628  -0.1641129  0.0292113 -0.1133042
  H -0.9534236   0.1719066 -0.0194712  0.1426191  0.0203800
  W  0.2837116  -0.0023497  0.1522709 -0.3637363  0.1463836 -0.4411074
```

Based on the t-value, age, cholesterol, and weight effects seem to exist. The multiple scatterplot below also supports the result.



We want to fit the model with the three variables only.

```
> chapman.logist.reduced_glm(CNT~age+Ch+W,family=binomial,data=chapman)
> summary(chapman.logist.reduced)

Call: glm(formula = CNT ~ age + Ch + W, family = binomial, data = chapma
n)
Deviance Residuals:
       Min          1Q      Median          3Q      Max
 -1.104901 -0.5541154 -0.3776944 -0.2510398 2.700912

Coefficients:
                  Value  Std. Error   t value
(Intercept) -9.255888319 2.070916857 -4.469464
        age  0.053003641 0.020821917  2.545570
```

```
          Ch  0.006517925 0.003587892  1.816645
           W  0.017538629 0.008270296  2.120677

(Dispersion Parameter for Binomial family taken to be 1 )

    Null Deviance: 154.5547 on 199 degrees of freedom

Residual Deviance: 135.5233 on 196 degrees of freedom

Number of Fisher Scoring Iterations: 5

Correlation of Coefficients:
    (Intercept)         age          Ch
age -0.3660495
 Ch -0.5167738  -0.2383809
  W -0.7669655   0.0100841  0.1319159
```

We may compare the model with the full model. For the test statistic, we use the difference between the residual deviances of the full model and the reduced model.

G^2= 135.5233 – 134.8515 = 0.6718 with the degrees of freedom of 196-193=3

The p-valued of the test is 0.88 and we conclude that the reduced model with three predictors only is an adequate substitute of for the full model. From the above output the variable Ch does not seem significant. We can repeat the above testing procedure for the Ch effect. A measure analogy to R^2 can be obtained by

$$R^2 = \frac{null\,deviance - residual\,deviance}{null\,deviance}.$$

For the above model with age, Ch, and W, R^2 = (154.55-135.52)/154.55 = 0.12. Only about 12% of variability can be explained by the model. This is not surprising for the binary data. Unless the probability π_i is very close to 1 or 0, the R^2 will be much lower than you expected. The low R^2 does not necessary mean a poor fit. So, the direct interpretation of the R^2 value should not be suggested. We can use the R^2 value for a tool of model comparisons.

Formal model selection procedure using AIC or Cp can be conveyed, but will not be discussed in this class. S+ does not provide the automatic search procedure, but SAS does via Proc Logistic.

Fitted response function: $\hat{\pi}_i = \dfrac{\exp(-9.26 + .053age_i + .0065Ch_i + .0175W_i)}{1 + \exp(-9.26 + .053age_i + .0065Ch_i + .0175W_i)}$

For example, for 60 years old man with cholesterol reading of 300 and the weights 200 pounds, the estimated probability of a coronary incident is

$$\hat{\pi} = \frac{\exp(-9.26 + .053(60) + .0065(300) + .0175(200))}{1 + \exp(-9.26 + .053(60) + .0065(300) + .0175(200))} = 0.35.$$

We can identify the outliers and/or influential observations using leverages as follows;

```
>rule_2*length(coef(chapman.logist.reduced))/length(fitted(chapman.logis
t.reduced))
> rule
[1] 0.04
```

```
> influ.chapman_lm.influence(chapman.logist.reduced)
> cbind(age,Ch,W,leverage=influ.chapman$hat)[influ.chapman$hat>rule,]
    age  Ch   W   leverage
 18  40 302 225 0.04554945
 19  51 302 247 0.10435819
 38  52 474 145 0.08053879
 41  40 520 169 0.14885485
 44  56 428 171 0.04979886
 48  64 243 171 0.04341317
 51  65 370 153 0.04235409
 55  67 365 190 0.05636015
 60  68 268 138 0.05473981
 61  64 261 108 0.04480911
 64  58 416 188 0.05418758
 81  47 341 218 0.04645482
 84  41 259 245 0.07883004
 96  67 320 134 0.04994716
108  51 269 262 0.14683385
111  69 370 185 0.06171088
113  64 244 187 0.05093838
114  70 353 163 0.05816883
116  53 453 170 0.06694622
121  53 420 141 0.04030119
123  63 420 160 0.05607527
126  28 386 189 0.04183048
153  58 187 224 0.08980256
157  49 273 245 0.09279471
193  60 317 206 0.04371181
>
```

Observations with high leverages are highlighted. We need to exam those observations closely. For example, the 41th observation (the highest leverage) has an extremely high cholesterol value, etc.

BOOTSTRAPING, the basic

The bootstrap is a powerful computer-based tool for statistical inference in case standard approximations are not applicable. Suppose for example, we want to compare two groups by their medians. Unfortunately we do not have an explicit formula to calculate the standard errors of the median estimates (we do have it for means). For another example, for the inference about population means, the standard inferences rely on the large sample central limit theorem. Then, we may simply ask, "What if the sample size is very small with an unknown underline sampling distribution?" Theses questions can be answered by computer-based techniques, like bootstrap.

To illustrate the basic concept of the bootstrap, let's consider a simple example (I will do my best to avoid mathematical notations). Suppose we observed data with size eight: *x* = (23, 34, 12, 9, 42, 17, 8, 29). We are interested in the reciprocal of mean, $\theta = 1/\mu$. We can simply estimate it by $\hat{\theta} = 1/\bar{x}$, where $\bar{x}$ is the sample mean. Now, our question is how to calculate the standard error of the estimate. Intuitively thinking, if we have many estimates of the parameter, $\hat{\theta}_1$, $\hat{\theta}_2$, …, $\hat{\theta}_B$, with B large enough positive integer, we could approximate it by the sample standard deviation of the B estimates; ie,

(1) $$se = \left[\sum_i^B \left(\hat{\theta}_i - \bar{\hat{\theta}} \right)^2 / (B-1) \right]^{1/2}$$, where $\bar{\hat{\theta}}$ is the sample mean of the $\hat{\theta}_i$'s.

To have the B estimates we need B samples of size eight each. Since we don't have any other data source we generate each sample by randomly sampling eight times with replacement from the original data point and name it X^*. We do the sampling B times to generate $X^{*1}, X^{*2}, ..., X^{*B}$ (this is called bootstrap samples). From each of the B samples we estimate the parameter and using the B estimates we calculate the sample standard deviation in (1). The following bootstrap algorithm for estimating standard errors is adopted from *An Introduction to the Bootstrap*, by E. Efron and R. Tibshirani, Chapman & Hall.

Bootstrap standard error estimation

Consider a random sample $\boldsymbol{x}=(x_1, x_2, \ldots, x_n)$ from an unknown probability distribution. We are interested in a parameter θ. We calculate an estimate $\hat{\theta}$ from $\boldsymbol{x}$.

1. Select B independent bootstrap sample $\boldsymbol{x}^{*1}, \boldsymbol{x}^{*2}, \ldots, \boldsymbol{x}^{*B}$ each consisting of n data points randomly selected for the original data with replacement. S+ recommend at least 250 to estimate standard errors and 1000 to estimate percentiles.
2. From each bootstrap sample calculate the parameter estimates, $\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_B$, called bootstrap replication
3. Evaluate the standard error by the sample standard deviation of the B replications as in (1).

In S+ we use the function *bootstrap* for the calculation. For the example we used B = 100.

```
> x_c(23, 34, 12, 9, 42, 17, 8, 29)
> boot1_bootstrap(x,1/mean(x),B=100)
Forming replications  1  to  100

> summary(boot1)
Call:
bootstrap(data = x, statistic = 1/mean(x), B = 100)

Number of Replications: 100

Summary Statistics:
      Observed      Bias    Mean       SE
Param  0.04598 0.0007052 0.04668 0.008573

Empirical Percentiles:
         2.5%      5%    95%   97.5%
Param 0.03455 0.03568 0.0646 0.06787

BCa Confidence Limits:
         2.5%      5%     95%   97.5%
Param 0.03463 0.03625 0.06615 0.06878
```

The estimated parameter is 0.04598 with the bootstrap standard error of 0.008573. The empirical percentiles from 100 bootstrap estimates are given. A better confidence interval, named BCa (bias-corrected and accelerated), is given. The BCa CI is better in terms of accuracy of the interval estimates and accurate coverage probability. Other possible CIs are bootstrap-t interval, ABC interval, and parametric approach via normal theory. For more detailed discussion on the bootstrap CI, consult *An Introduction to the Bootstrap*, by E. Efron and R. Tibshirani.

We will close this section with an interesting example. Note that the following example is just to give you the basic idea of the use of bootstrap. This is not a complete analysis and should not refer to other purpose.

Example: Bootstrap Bioequivalence

A drug company has separately applied each of three hormone supplement medicinal patches, 'Approved (by FDA)', 'Placebo', and 'New', to eight patients who suffer from a hormone deficiency. Measurements

are blood pressure level after each patch wearing. The FDA requires proof of bioequivalence before it will approve for sale a previously approved product manufactured at a new facility. Data is give below;

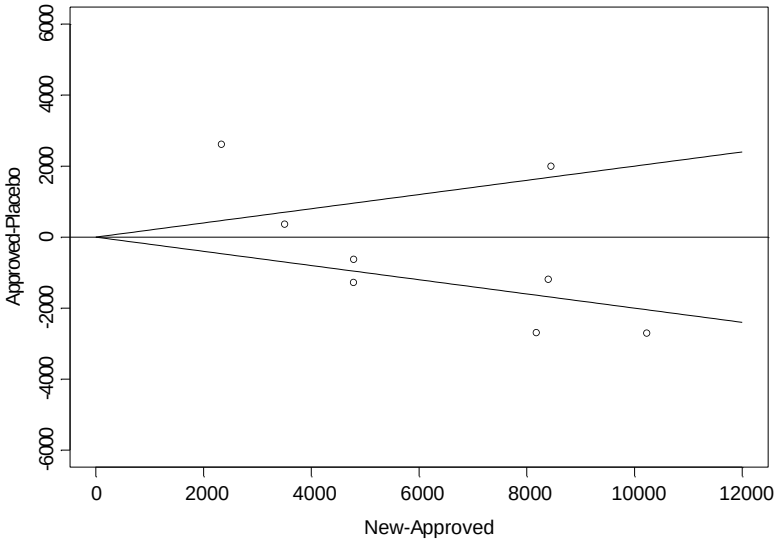
patient	placebo	approved	new
1	9243	17649	16449
2	9671	12013	14614
3	11792	19979	17274
4	13357	21816	23798
5	9055	13850	12560
6	6290	9806	10157
7	12412	17208	16570
8	18806	29044	26325

Let x = approved – placebo, y = new-approved. Define $\gamma = \nu / \mu$, where $\mu = E(x)$ and $\nu = E(Y)$. The FDA bioequivalence requirement is that a 90% CI for γ lie within [-0.2, 0.2]. Here the parameter estimate for r is -0.0713.

```
> mean(y)/mean(x)
[1] -0.0713061
```

Following plot shows the data. The wedge in the plot indicates the FDA requirement region and we see that four of them are outside of the region.

```
> plot(x,y,xlab="New-Approved",ylab="Approved-Placebo")
> plot(x,y,xlab="New-Approved",ylab="Approved-Placebo",xlim=c(0,12000),ylim=c(-6000,6000))
> xx_seq(0,12000,100)
> yy_0.2*xx
> lines(xx,yy,type="l")
> lines(xx,-yy,type="l")
> abline(h=0)
```



Are the FDA bioequivalence criteria satisfied by the data? To answer this question we use the bootstrap CIs.

```
> bot_bootstrap(data,mean(data[,"y"])/mean(data[,"x"]),2000)
```

```
> summary(bot)
```

Call:
bootstrap(data = data, statistic = mean(data[, "y"])/mean(data[, "x"]), B = 2000)

Number of Replications: 2000

Summary Statistics:
	Observed	Bias	Mean	SE
Param	-0.07131	0.008381	-0.06292	0.1037

Empirical Percentiles:
	2.5%	5%	95%	97.5%
Param	-0.2321	-0.2135	0.1245	0.1709

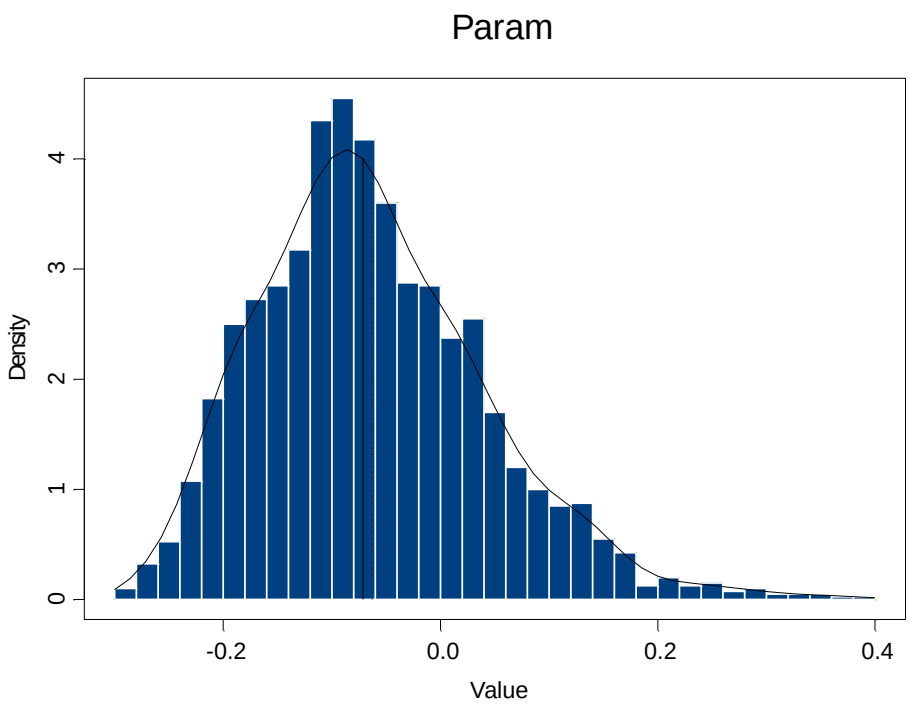
BCa Confidence Limits:
	2.5%	5%	95%	97.5%
Param	-0.2257	-0.2046	0.1382	0.1906

The estimated parameter is –0.07131 with a bootstrap standard error of 0.1037. Neither of the lower bounds of the CIs do not quite satisfy the FDA criteria.

Then, the next question would be how many patients should be measured in a future experiments so that the FDA requirement will have a good chance of being satisfied. This question related to power or sample size question and we will not discuss in this class.

Following is the histogram of the 2000 bootstrap estimates.

> plot(bot)



Next time we will discuss the use of bootstrap in regression analysis.

BOOTSTRAP REGEESION

Bootstrap regression is very useful when the regression function is non-linear in the parameters. In this section we will discuss how the bootstrap regression works in a simple regression model.

There are two different ways of bootstrap regression

1. **Bootstrapping pairs:** Random sample pairs (x,y) with replacement. For each bootstrap sample, we fit regression model and the based on the B estimates we make inferences.
2. **Bootstrapping residuals:** We replace the response using the fitted and resampled residuals. That is, we create a new data set by

$$y_i^* = b_0 + b_1 x_i + e_i^*$$

where e_i^* are resampled with replacement from the residuals e_i.

Example

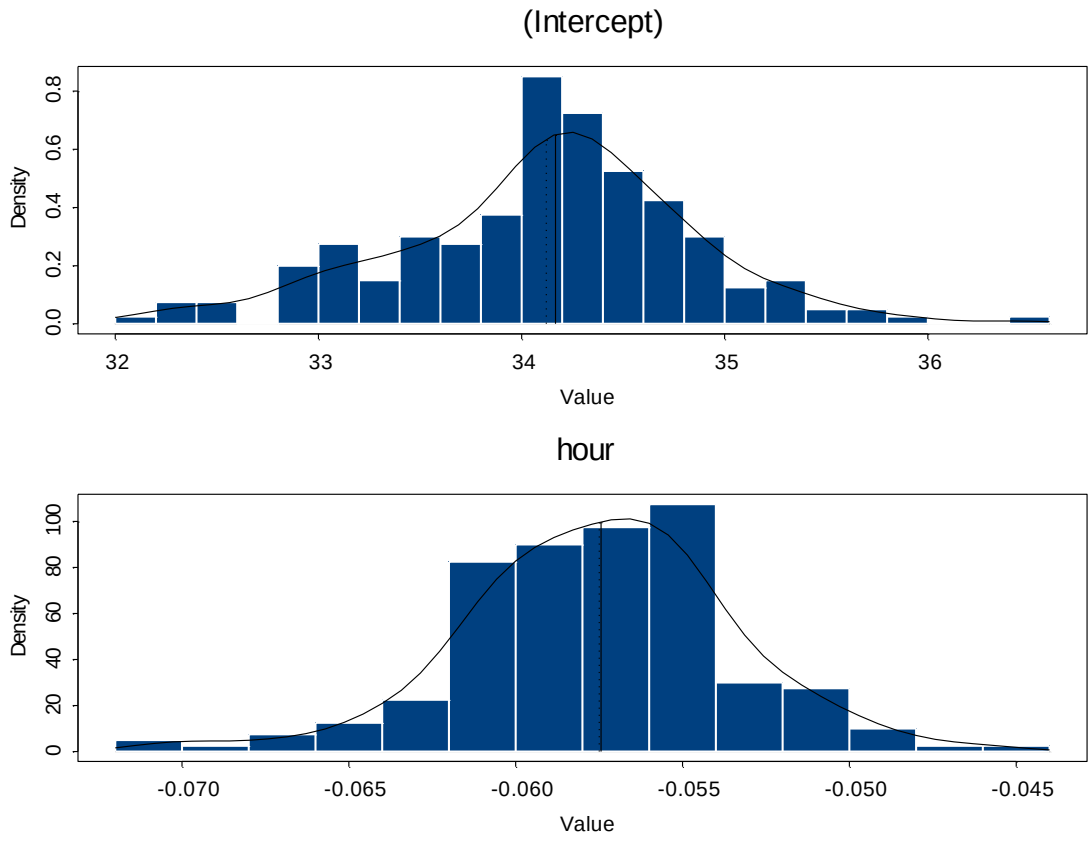
Hormone data: *n =27*

Response: *amount* (amount of milligrams of anti-inflammatory hormone remaining in 27 devices)
Predictor: *hour* (Number of hours of wear)
Group : *lot* (3 different manufacturing lots)

Regression model fit

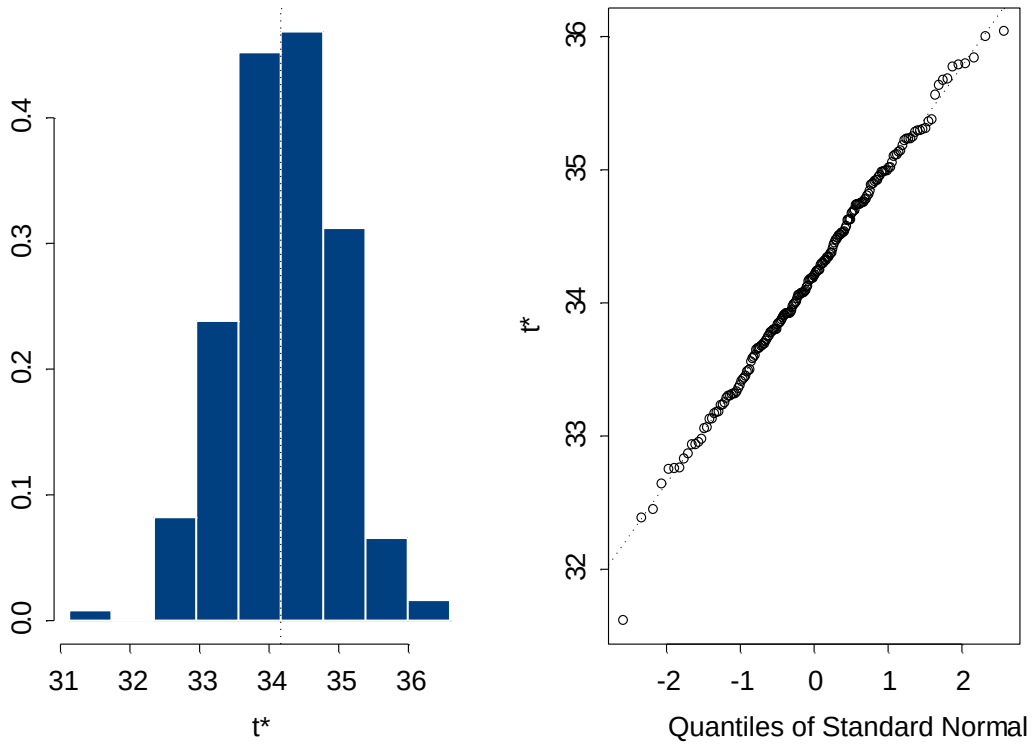
```
> fit.lm_lm(amount~hour,data=hormone)
> summary(fit.lm)

Call: lm(formula = amount ~ hour, data = hormone)
Residuals:
    Min     1Q   Median    3Q   Max
 -4.936 -1.728 -0.02287 1.739 3.732

Coefficients:
               Value Std. Error  t value Pr(>|t|)
(Intercept)  34.1675   0.8672    39.3999   0.0000
       hour  -0.0574   0.0045   -12.8683   0.0000

Residual standard error: 2.378 on 25 degrees of freedom
Multiple R-Squared: 0.8688
F-statistic: 165.6 on 1 and 25 degrees of freedom, the p-value is 1.584e
-012

Correlation of Coefficients:
     (Intercept)
hour -0.8494

#Bootstrap regression by bootstrapping paris

> hor.boot_bootstrap(hormone,coef(eval(lm.fit$call)),B=200)
Forming replications  1  to  100
Forming replications  101  to  200
> hor.boot

Call:
bootstrap(data = hormone, statistic = coef(eval(lm.fit$call)), B = 200)

Number of Replications: 200

Summary Statistics:
             Observed       Bias      Mean       SE
```

```
(Intercept)  34.16753  0.0057033  34.17323 0.770618
      hour  -0.05745 -0.0003323  -0.05778 0.004131

> plot(hor.boot)
```




```
# Bootstrap regression by bootstrapping residuals

> library(boot)
> hormone2_data.frame(hormone,resid=resid(lm.fit),fitted=fitted(lm.fit))
> hormone.fun_function(data,i){
+ d_data
+ d$amount_d$fitted+d$resid[i]
+ coef(update(fit.lm,data=d))
+ }

> hor.lm.boot_boot(hormone2,hormone.fun,R=200)
> hor.lm
Problem: Object "hor.lm" not found
Use traceback() to see the call stack
> hor.lm.boot

ORDINARY NONPARAMETRIC BOOTSTRAP


Call:
boot(data = hormone2, statistic = hormone.fun, R = 200)


Bootstrap Statistics :
```

```
       original        bias    std. error
t1*  34.1675282  0.0414358271 0.782883776
t2*  -0.0574463 -0.0002944581 0.004160554

> plot(hor.lm.boot)
```

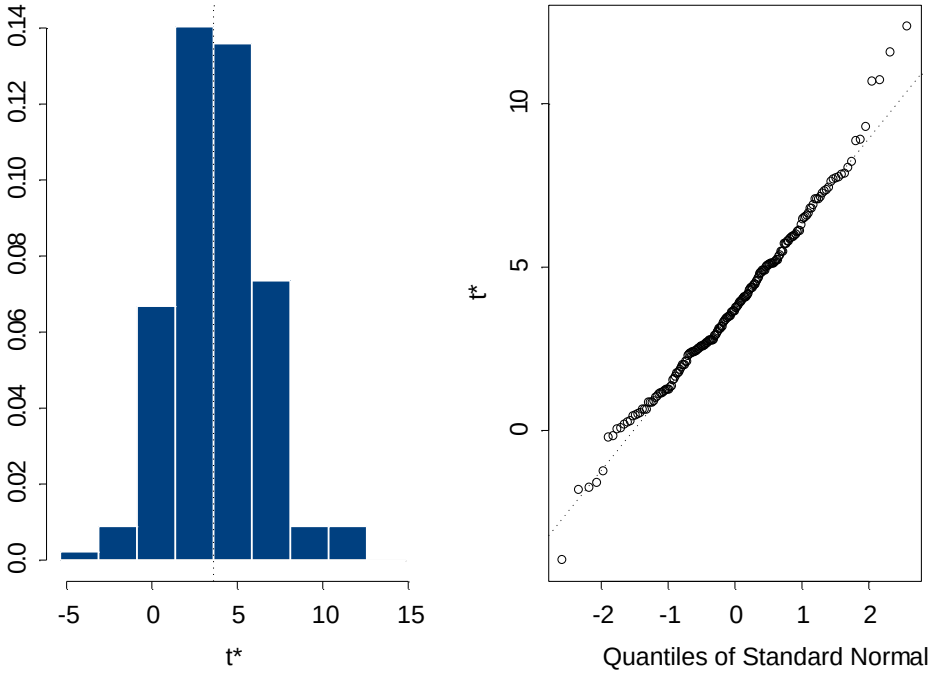


COMPARING TWO SAMPLE: BOOTSTRAP CI for the mean difference

We use our grade.data to compare the means of MT.I and MT.II.

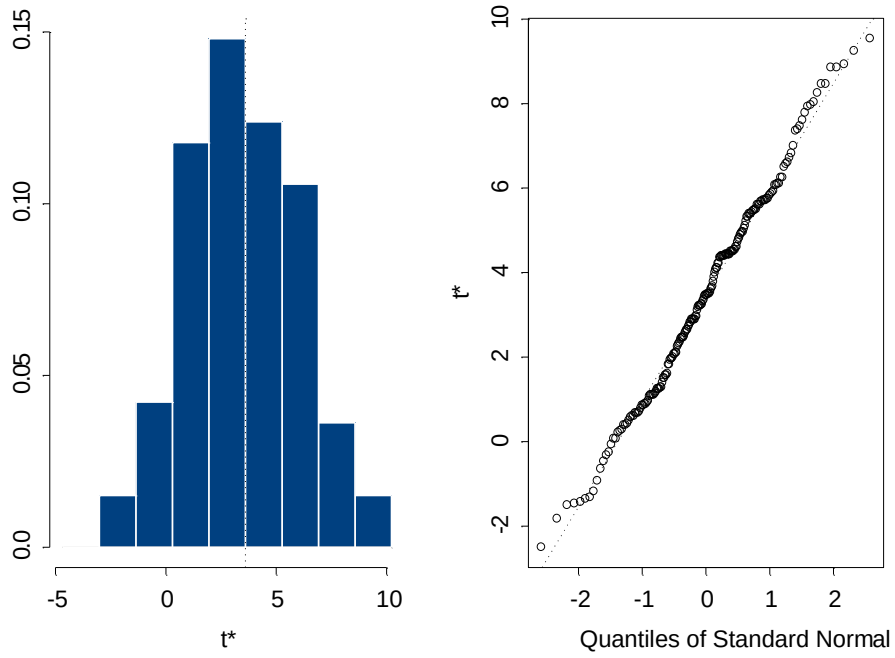
```
> attach(grade.data)
> t.test(MT.I-MT.II)

       One-sample t-Test

data:  MT.I - MT.II
t = 1.3713, df = 27, p-value = 0.1816
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 -1.789946  9.004232
sample estimates:
 mean of x
  3.607143

> grade.boot_boot(MT.I-MT.II,function(x,i) mean(x[i]),R=200)
> boot.ci(grade.boot,type=c("norm","basic","perc","bca"))
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 200 bootstrap replicates

CALL :
```

```
boot.ci(boot.out = grade.boot, type = c("norm", "basic", "perc", "bca")
       )

Intervals :
Level      Normal              Basic
95%   ( -1.609,   8.347 )   ( -2.061,   8.436 )

Level     Percentile            BCa
95%   ( -1.222,   9.275 )   ( -1.641,   8.870 )
Calculations and Intervals on Original Scale
Some basic intervals may be unstable
Some percentile intervals may be unstable
Some BCa intervals may be unstable
> plot(grade.boot)
```



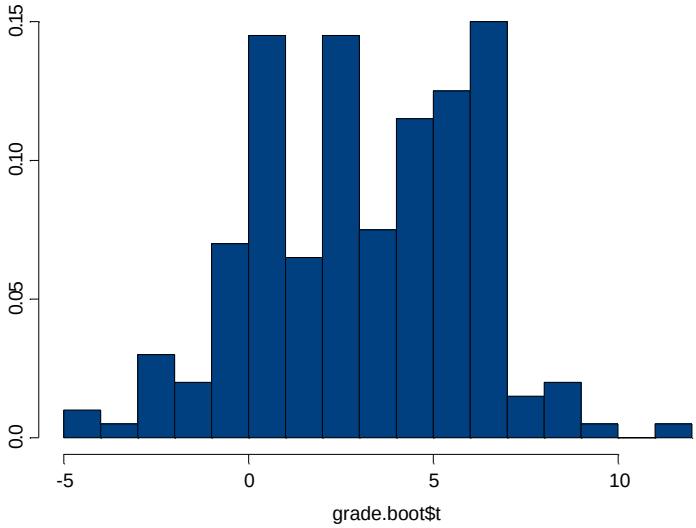
Another possible CI is the studentized CI which based on the studentized statistics. This CI is known as the most reliable. To calculate the studentized CI we need a variance estimate of the parameter estimate from each bootstrap sample.

```
> mean.fun_function(d,i){n_length(i);c(mean(d[i]),(n-1)*var(d[i])/n^2)}
> grade.boot.stud_boot(MT.I-MT.II,mean.fun,R=200)
> boot.ci(grade.boot.stud,type="stud")
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 200 bootstrap replicates

CALL :
boot.ci(boot.out = grade.boot.stud, type = "stud")

Intervals :
Level    Studentized
95%   ( -2.157,   8.808 )
Calculations and Intervals on Original Scale
Some studentized intervals may be unstable
> plot(grade.boot.stud)
```



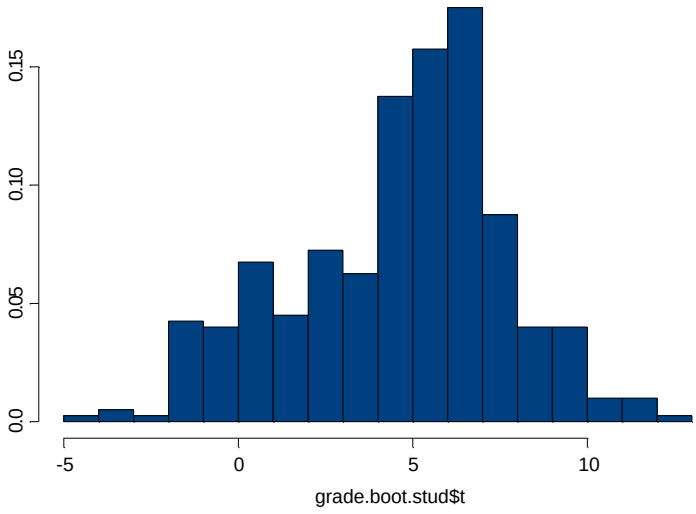
Comparing medians

```
> grade.boot_boot(MT.I-MT.II,function(x,i) median(x[i]),R=200)
> boot.ci(grade.boot,type=c("norm","basic","perc","bca"))
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 200 bootstrap replicates

CALL :
boot.ci(boot.out = grade.boot, type = c("norm", "basic", "perc", "bca")
       )

Intervals :
Level      Normal              Basic
95%   ( -3.606,   7.966 )   ( -3.000,   8.000 )

Level     Percentile            BCa
95%   ( -3.00,   8.00 )   ( -4.50,   6.38 )
Calculations and Intervals on Original Scale
Some basic intervals may be unstable
Some percentile intervals may be unstable
Some BCa intervals may be unstable

> truehist(grade.boot$t,h=1)
```



The Studentized CI can be obtained as follows;

```
> median.fun_function(d,i){n_length(i);c(median(d[i]),(n-1)*var(d[i])/n^2)}
> grade.boot.stud_boot(MT.I-MT.II,median.fun,R=200)
> boot.ci(grade.boot.stud,type="stud")
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 200 bootstrap replicates

CALL :
boot.ci(boot.out = grade.boot.stud, type = "stud")

Intervals :
Level    Studentized
95%   ( -1.785,   8.520 )
Calculations and Intervals on Original Scale
Some studentized intervals may be unstable
> truehist(grade.boot.stud$t,h=1)
```



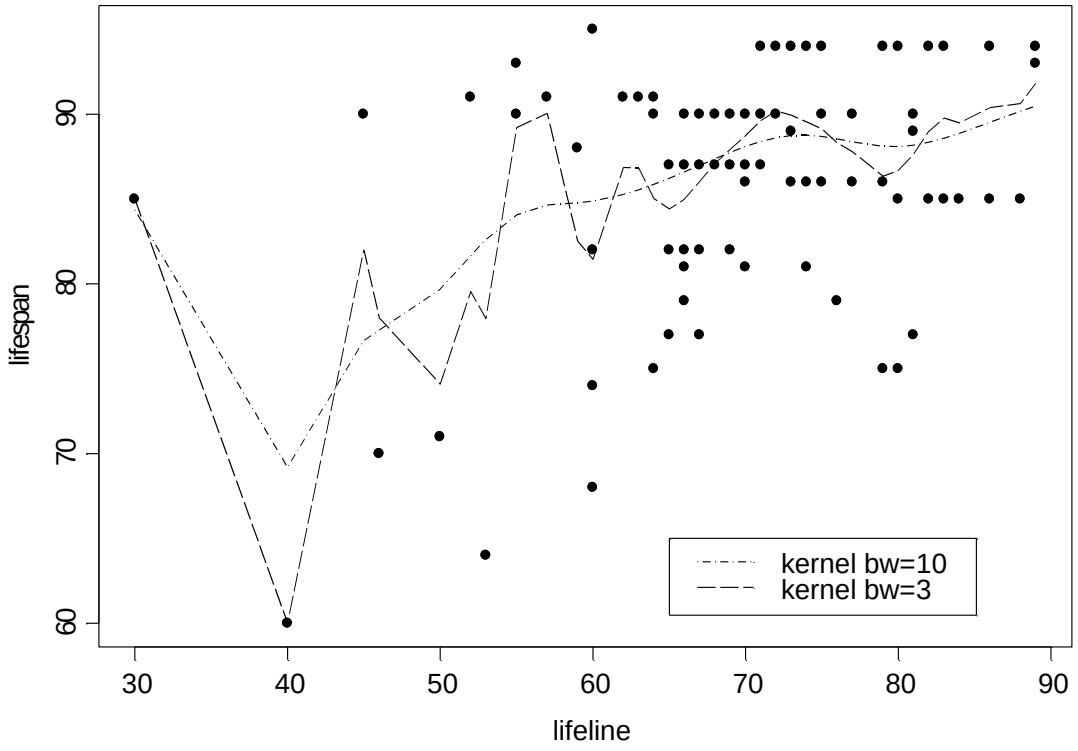
SMOOTH REGRESSION (Brief note)

Data: Random sample of size 100 to predict Life Span by the length of one's lifeline (left hand). Source: Newrick et.al. (1990) *Journal of the Royal Society of Medicine* and *Modern Regression Methods*, T.P. Ryan, Wiley Interscience.

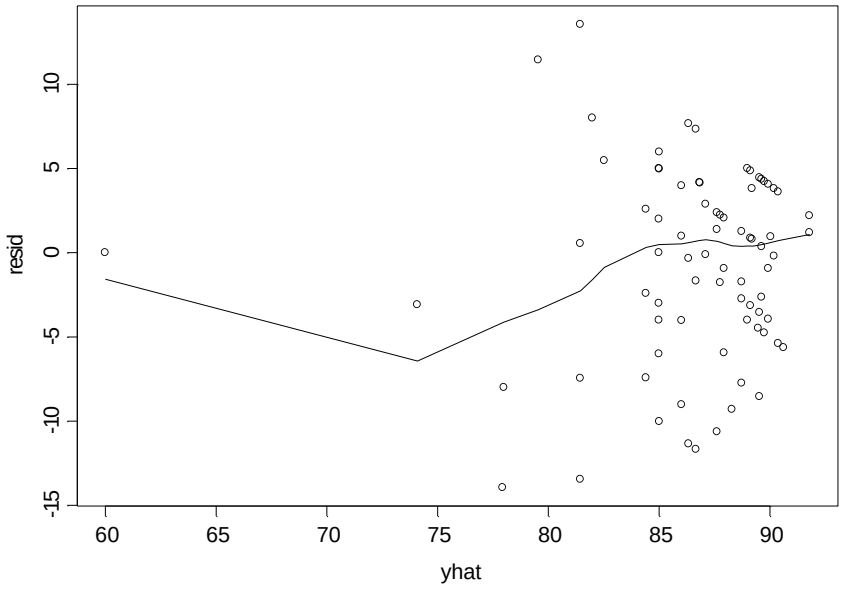
Kernel Regression

```
> plot(lifeline,lifespan,pch=16)
> lines(ksmooth(lifeline,lifespan,kernel="normal",bandwidth=10),lty=3)
> lines(ksmooth(lifeline,lifespan,kernel="normal",bandwidth=3),lty=4)
> legend(65,65,c("kernel bw=10","kernel bw=3"),lty=c(3,4))
```

The larger the bandwidth, the smoother the fit. For the more option of the kernel functions, refer the S+ help note.

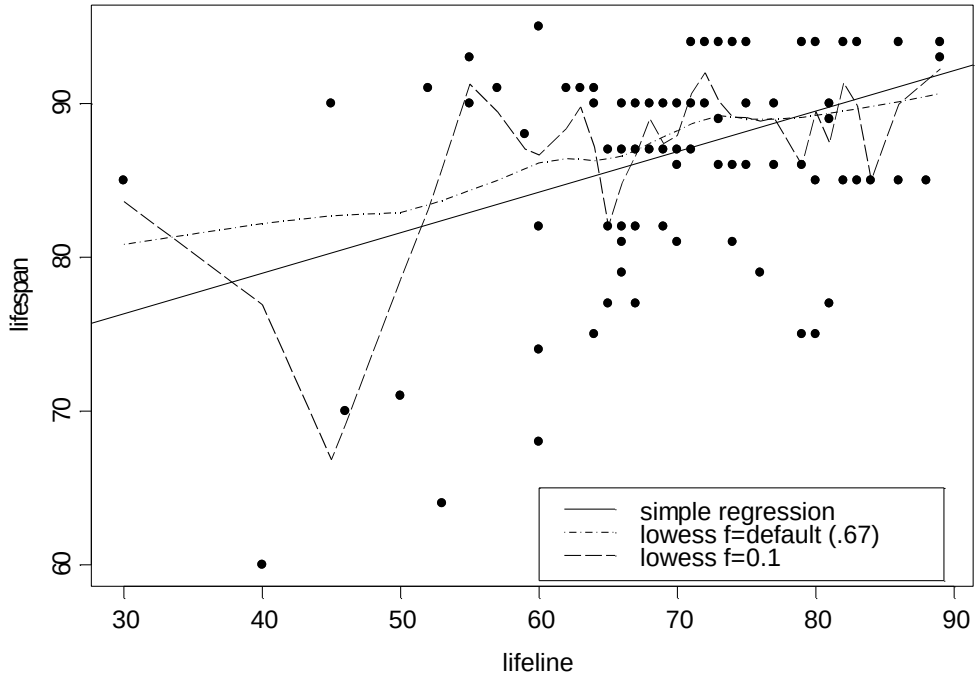



```
> yhat_approx(kernel.fit,xout=lifeline)$y
> resid <- lifespan-yhat
> plot(yhat,resid)
> lines(lowess(yhat,resid))
```



Locally weighted regression (Cleveland 1979)

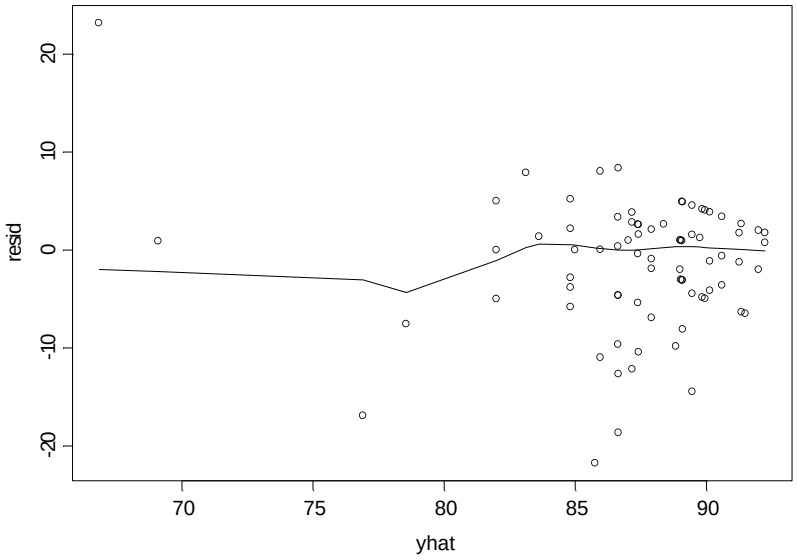
```
> plot(lifeline,lifespan,pch=16)
> abline(lm(lifespan~lifeline))
> lines(lowess(lifeline,lifespan),lty=3)
> lines(lowess(lifeline,lifespan,0.1),lty=4)
> legend(60,65,c("simple regression","lowess f=default (.67)","lowess
f=0.1"),lty=c(1,3,4))
```



Using interface,

Statistics → *Smoothing* → *Loess* → specify the data, x, and y → change the options for symbols and lines as you wish → *Smooth/Sort* option → *Span*: type in the smoothing parameter f (default is 0.75). The larger the value, the smoother the fit → *Degree*: 1 for linear 2 for quadratic fit → *Family*: Gauss for default, Symmetric for robust fitting

```
> yhat_approx(Lowess.fit,xout=lifeline)$y
> resid <- lifespan-yhat
> plot(yhat,resid)
> lines(lowess(yhat,resid))
```



Using the Interface: Statistics → Regression → Local (loess) → etc

Natural Spline

```
>  plot(lifeline,lifespan,pch=16)
> lines(lifeline, fitted(lm(lifespan~ns(lifeline,df=20))),lty=3)
> lines(lifeline, fitted(lm(lifespan~ns(lifeline,df=5))),lty=4)
> legend(65,65,c("Natural Spline df=20","Natural Spline df=5"),lty=c(3,4))
```