Basic Statistical Analyses using the R Statistical Package

Table of Contents

1. Introduction

R is a freely distributed software package for statistical analysis and graphics, developed and managed by the *R Development Core Team*. R can be downloaded from the internet site of the *Comprehensive R Archive Network* (CRAN) (http://cran.r-project.org). Check that you download the correct version of R for your operating system (for example, XP for the PC, Tiger or earlier versions of OSX for Macs). R is related to the S statistical language which is commercially available as S-PLUS.

R is an object-oriented language. For our basic applications, matrices representing data sets (where columns represent different variables and rows represent different subjects) and vectors representing variables (one value for each subject in a sample) are objects in R. Functions in R perform calculations on objects. For example, if 'cholesterol' was an object representing cholesterol levels from a sample, the function 'mean(cholesterol)' would calculate the mean cholesterol for the sample. For our basic applications, results of an analysis are displayed on the screen. Results from analyses can also be saved as objects in R, allowing the user to manipulate results or use the results in further analyses.

Data can be directly entered into R, but we will usually use MS Excel to create a data set. Data sets are arranged with each column representing a variable, and each row representing a subject; a data set with 5 variables recorded on 50 subjects would be represented in an Excel file with 5 columns and 50 rows. Data can be entered and edited using Excel. Excel can save files in 'comma delimited format', or .csv files; these .csv files can then be read into R for analysis.

R is an interactive language. When you start R, a blank screen appears with a '>', which is the ready prompt, on the first line of the screen. Analyses are performed through a series of commands; the user enters a command and R responds, the user then enters the next command and R responds. In this document, commands typed in by the user are given in red and responses from R are given in blue; R uses this same color scheme.

Some odds and ends. Entering an object name will generally print that object. R is case sensitive, so an object named Group must be referred to as Group, not group. The up and down arrow keys can be used to recall and scroll through past commands, which can save typing when fixing typos or modifying a command. R for the PC is a windows program, and you can use the mouse to cut and paste text into or out of R. You can also save text of your R session as a record of your work. There is a lot of R help out on the internet. For example, I was stuck trying to decipher the R help page for analysis of variance and so I googled 'Analysis of Variance R'. I found several sites offering examples. Finally, as with any software program, there usually is more than one way to do things through R. The methods in this handout are not the only way to perform these analyses through R, and you should feel free to experiment and explore.

2. The assign operator and inputting a data vector into R

The 'assign operator' in R is used to assign a name to an object. For example, suppose we have a sample of 5 infants with ages (in months) of 6, 10, 12, 7, 15. In R, these values can be represented as a column vector (as a data set, these values would be arranged in one column for the variable age, with 5 rows). To assign the name 'agemos' to these data, we can use the command:

```
> agemos <- c(6,10,12,7,14)
```

The '>' is the ready prompt given by R, indicating that R is ready for our input (R typed the >, I typed the rest of the line). Here, agemos is the name we are giving to the object that we will be creating. The '<-' is the assign operator, and the 'c(...)' is a function creating a column vector from the indicated values. So we are creating the object 'agemos' which is a data vector (or variable in a data set).

To print an object, just enter the object name:

```
> agemos
[1]  6 10 12  7 14
```

We can use this object name in later analyses. For example, the mean age of these 5 infants can be calculated using the 'mean( )' function:

```
> mean(agemos)
[1] 9.8
```

In R, object names are arbitrary and will generally vary to fit a particular application or study. Functions always involve parentheses to enclose the relevant arguments, and function names make up the R language. So, we might calculate mean age using mean(agemos) or mean cholesterol using mean(cholesterol); the function name is constant, but the object name varies to fit the particular study.

3. Bringing data into R from an Excel file or a text file and the Age at Walking example

3a. Bringing data into R from an Excel file

MS Excel is an excellent tool for entering and managing data from a small statistical study. Data are arranged with variables as columns and subjects as rows. The first row of the Excel file (the 'header') can be used to provide variable names (object names for vectors in R). For example, the following are data from the first 5 subjects in a study to compare age first walking between two groups of infants:

Subject	group	sexmale	agewalk
1	1	1	9.00
2	1	1	9.50
3	1	1	9.75
4	1	0	10.00
5	1	1	13.00

Here, 'Subject' is an id code; group is coded 1 or 2 for the two study groups; sexmale is coded 1 for males and 0 for females; and agewalk is the age when the infant first walked, in months. Note that I've used single-word (no spaces) variable names; using the underscore '_' or period '.' are nice ways to separate words in a variable (for example, age_years or age.years are one-word variable names).

To bring an Excel data file into R, it first has to be saved as a comma-delimited file. In Excel, click on 'Save as', and select '.csv' as the file type. The .csv file can then be brought into R as a matrix:

```
> kidswalk <- read.csv("C:/.../BS703/Data Sets/agewalk4R.csv",header=T)
```

Note that forward slashes (/) are used in giving the file directory, rather than the backslash (\ ) used by Windows (which R will not recognize). Here, the data set is being saved as a 'data frame' object named 'kidswalk' (R will use these object names to identify data, and so the same name cannot be used for both a data frame and a variable name). The function 'read.csv' reads in the specified .csv file and creates the corresponding R object. The 'header=T' command indicates to R that the first row of the file is a header containing variable names.

The 'read' command creates an object (dataframe) for the entire data set, but it does not create objects for the individual variables. So, while we could perform some analyses on this entire data set, we cannot yet perform analyses on specific variables. There are a couple of ways to select individual variables from the dataframe. One way is to specify both the dataframe name and the variable name using the format 'dataframe$variable'. For example, if we want to find the mean age at first walking (dataframe name kidswalk, variable name 'agewalk'), we could use the 'mean' function:

```
> mean(kidswalk$agewalk)
[1] 11.13
```

An alternative is to create objects for the individual variables in a dataframe using the 'attach( )' function:

```
> attach(kidswalk)
```

This function does not give any visible output, but creates objects (vectors) for each individual variable in the data set, using the variable names specified in the header as the object names – in this case it creates data vectors named Subject, group, sexmale, and agewalk. We could then use any of these variable objects in analyses:

```
> mean(agewalk)
[1] 11.13
```

3b. Bringing data into R from a space-delimited text file

The 'read.table( ) function creates an object (matrix) from a text file representing a data set.

```
> kidswalk <- read.table("C:/.../Data Sets/agewalk4R.txt",header=T)
```

Again, the read.table( ) function creates an object for the entire data set (a dataframe object), and the 'header=T' option indicates that the first row of the file contains variable names.

Creating a dataframe does not create individual objects for the variables in the dataframe. Individual variables can be referenced using the 'dataframename$variablename' format, or the attach( ) function can be used to create objects (data vectors) for the individual variables in the data set:

```
> attach(kidswalk)
```

3c. Viewing a data frame using the R data editor

An R dataframe can be viewed and edited as a spreadsheet using the R data editor. In R, click on the 'Editor' menu, then click on 'Data editor', which will lead to a prompt for the name of the dataframe to view/edit. Analyses cannot be performed while the Data editor is open.

3d. Specifying a folder for inputting data sets for PC and Mac

For PC: R will look for data sets in the folder were R is started. Before starting R, right click on the R icon, and click on Properties. In the 'Start in' field, specify the path of the folder containing the Excel .csv files you will import to R (the path name should be in quotes). After starting R, files in this folder can be specified just by their name rather than through the full path name. That is, the file can be read in by specifying

```
> kidswalk <- read.csv("agewalk4R.csv")
```
rather than
```
> kidswalk <- read.csv("C:/.../BS703/Data Sets/agewalk4R.csv",header=T)
```

For Macs: Open R and select the 'Misc' option. Then choose 'Set Working Directory'. Browse to the folder where you will store the Excel .csv data set files. After starting R, files in this folder can be specified just by their name rather than through the full path name. That is, the file can be read in by specifying

```
> kidswalk <- read.csv("agewalk4R.csv")
```
rather than
```
> kidswalk <- read.csv("C:/.../BS703/Data Sets/agewalk4R.csv",header=T)
```

4. The help( ) and help.search( ) functions

The help( ) function in R provides details for the different R commands. For example,

```
> help(read.csv)
```

gives details relating to the read.csv( ) function, while

```
> help (mean)
```

gives details for the mean( ) function.

The help( ) function only gives information on R functions. To search more broadly, you can use the 'help.search( ) function. For example,

```
> help.search("t test")
```

will search for the string 't test' and indicate R functions that reference this string.

5. Finding means and standard deviations

5a. Finding means and standard deviations for the overall sample

The 'mean( )' function calculates means from an object representing either a data matrix or a variable vector. For example, for the 'kidswalk' data set described above, we can calculate the means for all the variables in the data set (a dataframe object):

```
> mean(kidswalk)
 subjno  group    sex agewalk
  25.50   1.34   0.48   11.13
```

The mean( ) function can also be used to calculate the mean of a single variable (a data vector object):

```
> mean(agewalk)
[1] 11.13
```

The 'sd( )' function calculates standard deviations, either for all variables in a data set or for specific variables. The 'length( ) function returns the number of values (n, the sample size) in a data vector:

```
> sd(kidswalk)
   subjno     group       sex   agewalk
14.5773797 0.4785181 0.5046720 1.3583078

> sd(agewalk)
[1] 1.358308
```

5b. Finding means and standard deviations for subgroups

There are two ways to do subgroup analyses in R. First, we can create a new data frame for a particular subgroup using the 'subset( ) function, and then perform analyses on the subset. Second, the 'tapply( )' function can be used to perform analyses across a set of subgroups in a data frame.

The 'subset( )' function creates a new data frame, restricting observations to those that meet some criteria. For example, the following creates a new data frame for kids in Group 2 of the kidswalk data frame (named 'group2kids'), and finds the n and mean Age_walk for this subgroup:

```
> group2kids <- subset(kidswalk,Group==2)
> length(group2kids)
[1] 5
> mean(group2kids$Age_walk)
[1] 11.91176
```

When specifying the condition for inclusion in the subsample ('Group==2' in this example), two equal signs '==' are needed to indicate a value for inclusion. Less than (<) and greater than (>) arguments can also be used. For example,

```
> age65plus <- subset(allsubjects,age>64)
```

would create a dataframe of subjects aged 65 and older.

The 'tapply( )' function applies functions to subsets of a data set. The input for the tapply( ) function is the outcome variable (data vector) to be analyzed, the variable (data vector) that defines the subsets of subjects, and the function to be applied to the outcome variable. To find the means, standard deviations, and n's for the two study groups in the age first walk data set:

```
> tapply(agewalk, group, mean)
       1        2
10.72727 11.91176
> tapply(agewalk, group, sd)
       1        2
1.231684 1.277636
> tapply(agewalk, group, length)
 1  2
33 17
```

5c. Handling missing data in R

When setting up a dataset using Excel, missing data can be represented by a blank cell or by NA. In either case, data will be treated as missing when imported into R. Methods of handling missing data in analysis may vary for different types of analyses.

The 'na.omit( )' function will exclude missing data from an object. For example, the following example first sets up a variable (called 'xvar') measured on a sample of n=6 subjects, but the second subject is missing a value for this variable:

```
> xvar <- c(2,NA,3,4,5,8)
```

Printing the data set as a check on creating the variable and the missing value:

```
> xvar
[1]  2 NA  3  4  5  8
```

Printing with the na.omit( ) function will exclude the missing value:

```
> na.omit(xvar)
[1] 2 3 4 5 8
```

In trying to calculate the mean for these data, we get an error due to the missing value:

```
> mean(xvar)
[1] NA
```

We can calculate the mean using the 'na.omit( )' function:

```
> mean(na.omit(xvar))
[1] 4.4
```

Using the length( ) function gives the total number of subjects in the data set, while using the length( ) function with the na.omit( ) function gives the number of subjects with non-missing data:

```
> length(xvar)
[1] 6
> length(na.omit(xvar))
[1] 5
```

Some functions also have options to deal with missing data. For example, the mean( ) function has the 'na.rm=T' option to remove missing values from the calculation:

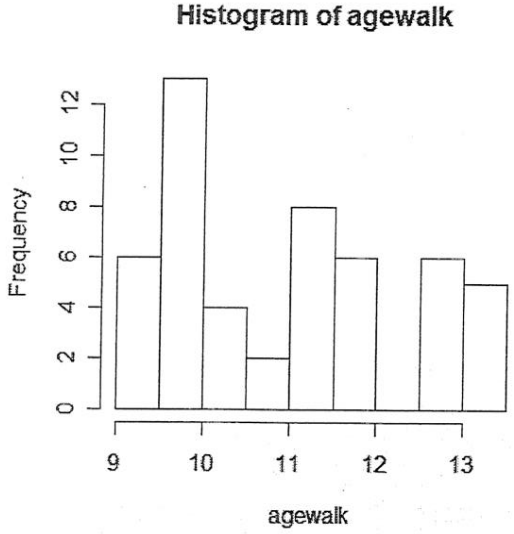
```
> mean(xvar, na.rm=T)
[1] 4.4
```

See the help documents for options for missing data for specific analyses.

6. Graphing histograms and box plots

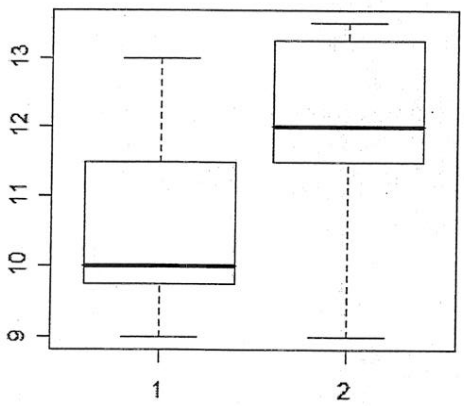
The 'hist( )' function draws a histogram of an object representing a variable vector. For a histogram of age of first walking from our example (I copied and pasted the histogram from the R window into this document):

> hist(agewalk)



For boxplots comparing the distributions of age of first walking for the two study groups:

> boxplot(agewalk ~ group)



7. Finding a confidence interval for a mean

The 't.test( )' function performs one-sample and two-sample t-tests. In performing a one-sample t-test, this function also gives a confidence interval for the population mean.

```
> t.test(agewalk)

        One Sample t-test

data:  agewalk
t = 57.9405, df = 49, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 10.74397 11.51603
sample estimates:
mean of x
   11.13
```

Using the t.test( ) function is the easiest way to calculate a confidence interval for a mean in R. But, as an example of using the programming capabilities of R, you can also calculate a confidence interval directly. The following commands use 5 R functions. The mean( ) function calculates the sample mean, the qt( ) function calculates critical values for from the t-table (the .975 percentile corresponds to the critical value for a 95% confidence interval), the length( ) function gives the number of subjects in the data set (the length of the column vector containing the data), the sd( ) function calculates the sample standard deviation, and the sqrt( ) function calculates the square root. After entering the first command, you can use the arrow key to recall the command and then just edit the command to change the + for the upper confidence limit to a – for the lower confidence limit.

```
> mean(agewalk)
[1] 11.13
> mean(agewalk)+qt(.975,length(agewalk)-1)*sd(agewalk)/sqrt(length(agewalk))
[1] 11.51603
> mean(agewalk)-qt(.975,length(agewalk)-1)*sd(agewalk)/sqrt(length(agewalk))
[1] 10.74397
```

8. Performing a one-sample t-test for a mean

The one-sample t-test compares the mean from one sample to some hypothesized value. The 't.test( )' function performs a one-sample t-test. For input, we need to specify the variable (vector) that we want to test, and the hypothesized mean value. To test whether the mean age at walking is equal to 12 months for the infants in our age of first walking example:

```
> t.test(agewalk,mu=12)

        One Sample t-test

data:  agewalk
t = -4.529, df = 49, p-value = 3.806e-05
alternative hypothesis: true mean is not equal to 12
95 percent confidence interval:
 10.74397 11.51603
sample estimates:
mean of x
   11.13
```

R performs a two-tailed test, as indicated by the two-tailed language in the alternative hypothesis. The p-value here is given in scientific notation, and the 'e-05' indicates that the decimal place should be moved 5 spaces to the left; 3.806e-05 is scientific notation for 0.00003806.

9. Performing an independent samples t-test to compare two means

The t.test( ) function can also be used to perform an independent samples t-test comparing means from two independent samples. For the following syntax, the underlying data set includes the subjects from both samples, with one variable indicating the dependent variable (the outcome variable) and another variable indicating which group a subject is in. To perform the independent samples t-test, we need to specify the object representing the dependent variable and the object representing the group information. For the usual pooled-variance version of the t-test:

```
> t.test(agewalk ~ group,var.equal=TRUE)

        Two Sample t-test

data:  agewalk by group
t = -3.1812, df = 48, p-value = 0.002571
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.9331253 -0.4358587
sample estimates:
mean in group 1 mean in group 2
      10.72727        11.91176
```

R reports a two-tailed p-value, as indicated by the two-tailed phrasing of the alternative hypothesis. Note that the output gives the means for each of the two groups being compared, but not the standard deviations or sample sizes. This additional information can be obtained using the tapply( ) function as described in Section 5b, Finding means and standard deviations for subgroups.

To perform an independent sample t-test using the unequal variance version of the t-test:

```
> t.test(agewalk ~ group,var.equal=FALSE)

        Welch Two Sample t-test

data:  agewalk by group
t = -3.1434, df = 31.39, p-value = 0.003635
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.9526304 -0.4163536
sample estimates:
mean in group 1 mean in group 2
      10.72727        11.91176
```

10. Performing a paired samples t-test for a mean difference

The 't.test( )' function can also be used to perform a paired-sample t-test. In this situation, we need to specify the two data vectors representing the two variables to be compared. The following example compares the means of a pre-test score (variable vector score1) and a post-test score (variable vector score2) from a sample of 5 subjects. The t.test( ) function does not give the means of the two underlying variables (it does give the mean difference) and so I used the mean( ) function to get this descriptive information:

```
> mean(score1)
[1] 20.2
> mean(score2)
[1] 21
> t.test(score1,score2,paired=TRUE)

        Paired t-test

data:  score1 and score2
t = -0.4377, df = 4, p-value = 0.6842
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -5.874139  4.274139
sample estimates:
mean of the differences
                  -0.8
```

The confidence interval here is the confidence interval for the mean difference.

11. Performing a chi-square goodness-of-fit test

The following gives the syntax needed to calculate a chi-square goodness-of-fit test from a set of tabled frequencies. As an example, 45 subjects are asked which of 3 screening tests they prefer; 10 subjects prefer Test A, 15 prefer test B, and 20 prefer Test C. We wish to test the null hypothesis that the three screening tests are equally preferred, or equivalently, that 1/3 of subjects prefer each test. The data:

Preference	Observed Frequency	Expected Proportion under the null
Test A	10	.333
Test B	15	.333
Test C	20	.333

To analyze these data in R, first create an object (arbitrarily named 'obsfreq' in the example) that contains the observed frequencies. Second, we create an object that contains the expected probabilities under the null (arbitrarily named 'nullprobs'; the third probability was rounded to .334 because the probabilities must sum to 1.00). Third, we compare the observed frequencies to the expected probabilities through the chisq.test( ) function:

```
> obsfreq <- c(10,15,20)
> nullprobs <- c(.333,.333,.334)
> chisq.test(obsfreq,p=nullprobs)

        Chi-squared test for given probabilities

data:  x
X-squared = 3.3018, df = 2, p-value = 0.1919
```

12. Contingency table analysis and the chi-square test of independence

12a. The chi-square test of independence from per-subject data

From the Age at Walking example, suppose we want to compare the percent of males (coded sexmale=1) between the two groups in our age first walking example. We can first use the 'table( )' function to get the observed counts for the underlying frequency table:

```
> table(group,sexmale)

    sexmale
group  0  1
   1  17 16
   2   9  8
```

In group 1, there are 16 males and 17 females, so 48.5% (16/33) of group 1 is male. In group 2, 47.1% (8/17) are male.

The 'summary( )' function applied to a table object compares these two percentages through the chi-square test of independence:

```
> summary(table(group,sex))

Number of cases in table: 50
Number of factors: 2
Test for independence of all factors:
      Chisq = 0.009141, df = 1, p-value = 0.9238
```

12b) The chi-square test of independence from tabled data

R can also perform a chi-square test on frequencies from a contingency table. For example, suppose we want to compare percent of subjects testing positive on a marker for an exposure across three groups:

	Group 1	Group 2	Group 3
Test Positive	20 (40.0%)	5 (33.3%)	40 (50.0%)
Test Negative	30	10	40

First, we create an object ('obsfreq' in the example) containing the observed frequencies from the observed table. I printed the object as a check that it was created correctly:

```
> obsfreq <- matrix(c(20,30, 5,10, 40,40),nrow=2,ncol=3)
> obsfreq
     [,1] [,2] [,3]
[1,]   20    5   40
[2,]   30   10   40
```

The 'chisq.test( )' function will then calculate the chi-square statistic for the test of independence for this table:

```
> chisq.test(obsfreq)

        Pearson's Chi-squared test

data:  obsfreq
X-squared = 2.1378, df = 2, p-value = 0.3434
```

13. Calculating a confidence interval for a proportion

I don't know how to do this directly in R. The following calculates a confidence interval for a proportion from basic principles (there must be a better way…), calculating the percent of females from our Age at first walking example.

First, the table( ) function givest the observed frequencies of males and females:

```
> table(sexmale)
sexmale
 0  1
26 24
```

Second, we can save the sample proportion as an object (arbitrarily named 'p.hat'). I also printed p.hat as a check on the calculation:

```
> p.hat <- 26/50
> p.hat
[1] 0.52
```

Third, we can calculate the upper and lower confidence limits:

```
> p.hat+qnorm(.975)*sqrt(p.hat*(1-p.hat)/50)
[1] 0.6597092
> p.hat-qnorm(.975)*sqrt(p.hat*(1-p.hat)/50)
[1] 0.3802908
```

The 'qnorm( )' function gives critical z-values; specifying the 97.5^{th} percentile gives the critical z-value for a 95% confidence interval.

14. Correlation and Regression

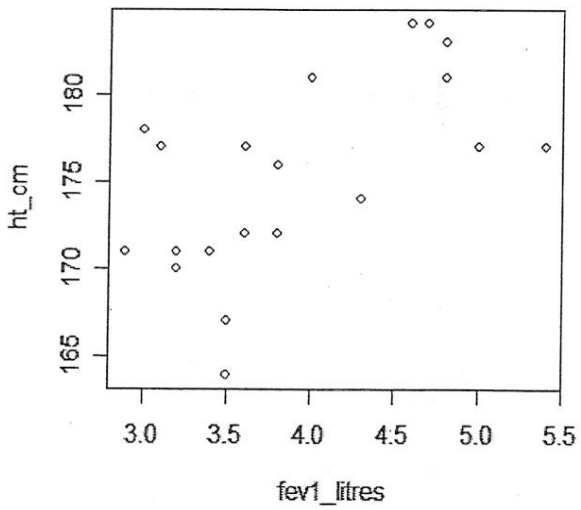
As an example of a study examining the association between two measurement variables, we will look at the association between forced expiratory volume (FEV1, a measure of lung function) and height (measured in centimeters) in a sample of 20 young adults. Data for the first 5 subjects:

ID	sexM	ht_cm	fev1_litres
1	1	174	4.3
2	1	181	4.8
3	0	184	4.7
4	1	177	5.4
5	1	177	3.1

14a. Graphing a scatterplot

The plot( ) function will graph a scatter plot. To plot FEV1 (the dependent or outcome variable) on the Y axis, and height (the independent or predictor variable) on the X axis:

```
> plot(ht_cm ~ fev1_litres)
```



14b. Correlation

The 'cor( ) function calculates correlation coefficients between the variables in a data set (vectors in a matrix object). For our height and lung function example, where 'fevheight' is the matrix object representing the data set:

```
> cor(fevheight)
                    ID        sexM      ht_cm fev1_litres
ID           1.00000000  0.02726935 -0.1624661  -0.4339991
sexM         0.02726935  1.00000000  0.1044337  -0.1196384
ht_cm       -0.16246613  0.10443368  1.0000000   0.5973320
fev1_litres -0.43399905 -0.11963840  0.5973320   1.0000000
```

The 'cor.test( )' function gives more detail around the correlation coefficient between two measurement variables, testing the null hypothesis of zero correlation (no association) and giving a CI for the correlation coefficient. For our height and lung function example:

```
> cor.test(ht_cm,fev1_litres)

        Pearson's product-moment correlation

data:  ht_cm and fev1_litres
t = 3.16, df = 18, p-value = 0.005419
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.2104363 0.8224525
sample estimates:
    cor
0.597332
```

14c. Simple regression analysis

Regression analysis is performed through the 'lm( )' function. LM stands for Linear Models, and this function can be used to perform simple regression, multiple regression, and Analysis of Variance.

For simple regression (with just one independent or predictor variable), predicting FEV1 from height:

```
> summary(lm(fev1_litres ~ ht_cm) )

Call:
lm(formula = fev1_litres ~ ht_cm)

Residuals:
     Min       1Q   Median       3Q      Max
-1.12043 -0.36014 -0.02043  0.32223  1.35898

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -10.01429    4.40863  -2.272  0.03562 *
ht_cm         0.07941    0.02513   3.160  0.00542 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.6148 on 18 degrees of freedom
Multiple R-Squared: 0.3568,     Adjusted R-squared: 0.3211
F-statistic: 9.985 on 1 and 18 DF,  p-value: 0.005419
```

The syntax here is actually calling two functions, the lm( ) function performs the regression analysis, and the summary( ) function prints selected output from the regression.

14d. Multiple regression analysis

Multiple regression analysis is also performed through the 'lm( )' function. The syntax is the same as for simple regression except that more than one predictor variable is specified:

```
> summary(lm(fev1_litres ~ ht_cm + sexM))

Call:
lm(formula = fev1_litres ~ ht_cm + sexM)

Residuals:
     Min       1Q   Median       3Q      Max
-1.02900 -0.33864 -0.08322  0.36404  1.45297

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -10.27991    4.42528  -2.323  0.03284 *
ht_cm         0.08196    0.02531   3.238  0.00484 **
sexM         -0.28059    0.29034  -0.966  0.34738
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.6159 on 17 degrees of freedom
Multiple R-Squared: 0.3903,     Adjusted R-squared: 0.3186
F-statistic: 5.441 on 2 and 17 DF,  p-value: 0.01491
```

15. Predicting a Yes/No outcome - logistic regression

In R, logistic regression is performed using the glm( ) function, for general linear model. This function can fit several regression models, and the syntax specifies the request for a logistic regression model.

As an example, we will look at factors associated with smoking among a sample of n=300 high school students from the Youth Risk Behavior Survey. The outcome variable is 'eversmokedaily1', coded as 1 for those who have smoked vs. 0 for those who have not. As a preliminary analysis, we calculate the percent of respondents who have ever smoked daily:

```
> table(eversmokedaily1)
eversmokedaily1
  0   1
229  69
> 69/(69+229)
[1] 0.2315436
```

In the following syntax, the glm( ) function performs the logistic regression, and the summary( ) function requests the default output summarizing the analysis. In entering this command, I hit the 'return' to type things in over 2 lines; R will allow you to continue a command onto a second or third line. The 'family=binomial(link=logit)' syntax specifies a logistic regression model:

```
> summary(glm(eversmokedaily1 ~ age + sex1F2M + bmi_under + bmi_over +
  bmi_obese + alcdays30, family=binomial(link=logit)))

Call:
glm(formula = eversmokedaily1 ~ age + sex1F2M + bmi_under + bmi_over +
    bmi_obese + alcdays30, family = binomial(link = logit))

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.0011  -0.5522  -0.3952  -0.2429   2.5496

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -0.4139     2.1764  -0.190   0.8492
age          -0.2336     0.1390  -1.681   0.0928 .
sex1F2M       0.3481     0.3507   0.992   0.3210
bmi_under     0.0535     0.6103   0.088   0.9302
bmi_over      0.7000     0.4172   1.678   0.0934 .
bmi_obese    -0.2981     0.6498  -0.459   0.6464
alcdays30     0.8256     0.1161   7.113 1.14e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)
```

```
    Null deviance: 292.83  on 268  degrees of freedom
Residual deviance: 217.99  on 262  degrees of freedom
AIC: 231.99

Number of Fisher Scoring iterations: 5
```

In logistic regression, slopes can be converted to odds ratios for interpretation. Below we calculate the odds ratio for those in the BMI overweight category, and we calculate the OR and the 95% CI for the OR for those having had a drink in the past month vs. those not having had a drink in the past month (the # indicates a comment that is ignored by R):

```
> exp(0.7000)    # OR for BMI overweight
[1] 2.013753

> exp(0.8256)    # OR for alcdays30
[1] 2.283250

> exp(0.8256+1.96*.1161)    # Upper CL for OR for alcdays30
[1] 2.866684

> exp(0.8256-1.96*.1161)    # Lower CL for OR for alcdays30
[1] 1.818558
```

16. One factor Analysis of Variance to compare means

As an example, suppose we want to compare the mean days to healing for 5 different treatments for fever blisters. We create a matrix object for the data set (names 'feverblister') that includes two variable vectors. 'DaysHeal' is the number of days to healing (fewer days indicate more effective medication) and our outcome variable. 'Treatment' is a group variable coded 1 through 5 for the 5 treatments. There are 6 subjects given each of the 5 treatments, for a sample of 30 subjects overall.

The first step is to create a categorical version of the Treatment variable (otherwise R will think Treatment is a measurement variable and that we are trying to do a regression):

```
> TreatmentF <- factor(feverblister$Treatment)
```

We can then use this categorical 'TreatmentF' variable in our ANOVA. Here I've saved the results of the ANOVA as an object named 'fever_anova':

```
> fever_anova <- aov(DaysHeal ~ TreatmentF)
```

We can now request different summary results about the analysis using the results of this analysis. To see the means that are being compared:

```
> print(model.tables(fev_anova,"means",digits=3) )

Tables of means
Grand mean

5.633333

 TreatmentF
TreatmentF
    1     2     3     4     5
7.500 5.000 4.333 5.167 6.167
```

The tapply( ) function can be used to get standard deviations and sample sizes for each group, as described in Section 5b: Finding means and standard deviations for subgroups.

To request the ANOVA table and p-value for the overall ANOVA comparing means across the 5 groups:

> summary(fever_anova)

```
            Df Sum Sq Mean Sq F value  Pr(>F)
TreatmentF   4 36.467   9.117   3.896 0.01359 *
Residuals   25 58.500   2.340
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Given the overall ANOVA shows significance, we can request pairwise comparisons using Tukey's multiple comparison procedure:

> TukeyHSD(fever_anova)

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = DaysHeal ~ TreatmentF)

$TreatmentF
          diff        lwr         upr     p adj
2-1 -2.5000000 -5.0937744  0.09377442 0.0627671
3-1 -3.1666667 -5.7604411 -0.57289224 0.0113209
4-1 -2.3333333 -4.9271078  0.26044109 0.0927171
5-1 -1.3333333 -3.9271078  1.26044109 0.5660002
3-2 -0.6666667 -3.2604411  1.92710776 0.9410027
4-2  0.1666667 -2.4271078  2.76044109 0.9996956
5-2  1.1666667 -1.4271078  3.76044109 0.6811222
4-3  0.8333333 -1.7604411  3.42710776 0.8770466
5-3  1.8333333 -0.7604411  4.42710776 0.2614661
5-4  1.0000000 -1.5937744  3.59377442 0.7881333
```

17. Sample Size and Power Calculations

This section describes how to calculate necessary sample size or power for a study comparing two groups on either a measurement outcome variable (through the independent sample t-test) or a categorical outcome variable (through the chi-square test of independence).

A. For a study comparing two means, independent samples t-test

The power.t.test( ) function will calculate either the sample size needed to achieve a particular power (if you specify the difference in means, the standard deviation, and the required power) or the power for a particular scenario (if you specify the sample size, difference in means, and standard deviation).

The input for the function is:

n – the sample size in each group
delta – the difference between the means of the two populations
sd – the standard deviation
power – the desired power, as a proportion (between 0 and 1)

To find the required sample size to achieve a specified power, specify delta, sd, and power. To find the power for a specified scenario, specify n, delta, and sd. R assumes you are testing at the two-tailed p=.05 level; you can over-ride these defaults by including sig.level=xx or 'alternative='one.sided'.

Finding required sample size:

```
>power.t.test(delta=.25,sd=0.7,power=.80)

     Two-sample t test power calculation

              n = 124.0381
          delta = 0.25
             sd = 0.7
      sig.level = 0.05
          power = 0.8
    alternative = two.sided

NOTE: n is number in *each* group
```

Finding power:

```
> power.t.test(n=50,delta=.25,sd=0.7)

     Two-sample t test power calculation

              n = 50
          delta = 0.25
             sd = 0.7
      sig.level = 0.05
          power = 0.4239677
    alternative = two.sided

 NOTE: n is number in *each* group
```

B. For a study comparing two proportions, independent samples

The power.prop.test( ) function in R calculates required sample size or power for studies comparing two groups on a proportion through the chi-square test. The input for the function is:

n – sample size in each group
p1 – the underlying proportion in group 1 (between 0 and 1)
p2 – the underlying proportion in group 2 (between 0 and 1)
power – the power of the test

To find the necessary sample size, specify p1, p2, and power. To find the power for a particular situation, specify n, p1, and p2. R assumes you are testing at the two-tailed p=.05 level; you can over-ride these defaults by including sig.level=xx or 'alternative='one.sided'.

Examples:

Finding power:

```
> power.prop.test(n=100,p1=.2,p2=.1)

     Two-sample comparison of proportions power calculation

              n = 100
             p1 = 0.2
             p2 = 0.1
      sig.level = 0.05
          power = 0.5081911
    alternative = two.sided

 NOTE: n is number in *each* group
```

Finding necessary sample size:

```
> power.prop.test(p1=.2,p2=.1,power=.8)

     Two-sample comparison of proportions power calculation

              n = 198.9634
             p1 = 0.2
             p2 = 0.1
      sig.level = 0.05
          power = 0.8
    alternative = two.sided

 NOTE: n is number in *each* group
```