Sigma Statistics and Research Limited

Wellington R Users Group: Data Analysis Tips in R

17 March 2014

David Lillis (Director)

e-mail: sigmastatistics@gmail.com

web-site: http://www.statisticalanalysis.co.nz/

WELLINGTON R:
DATA ANALYSIS TIPS IN R

In this session I discuss helpful tips that I have developed or picked up while using R. I hope that some or all of them are helpful for you too. If you have any questions about the material presented in this session, please feel free to e-mail me at: sigmastatistics@gmail.com or telephone me at: 0064-4-562-6247.

All text in green Courier New font refers to R syntax that you cut and paste into the R command line. Blue font gives the output that appears on your screen. Following each tip, I provide extension examples that you can try out for yourselves.

1. FIND COUNTS AND PERCENTAGES USING length(which())

Combining the length() and which() commands gives a handy method of counting elements that meet particular criteria.

```
b <- c(7, 2, 4, 3, -1, -2, 3, 3, 6, 8, 12, 7, 3)
b
```

Let's count the 3s in the vector b.

```
count3 <- length(which(b == 3))
count3
```

```
[1] 4
```

In fact, you can count the number of elements that satisfy almost any given condition.

```
length(which(b < 7))
```

```
[1] 9
```

Here is an alternative approach, also using the length() command, but also using square brackets for sub-setting:

```
length(b[ b < 7 ])
```

```
[1] 9
```

The square brackets allow us to subset. For such operations using square brackets, I like to use the words "such that". Here, we have the elements of b, such that the elements are less than 7.

R PROVIDES ANOTHER ALTERNATIVE THAT NOT EVERYONE KNOWS ABOUT

```
sum(b < 7)
```

```
[1] 9
```

This syntax gives a count rather than a sum. Be aware of the meaning of syntax like sum(b < 7). Both work on logical vectors whose elements are either TRUE or FALSE. Try entering b <- 7 at the console.

```
b < 7
```

```
[1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE
```

We see that sum(b < 7) counts the number of elements that are TRUE. There are nine such elements.

Now try:

```
mean(b < 7)
[1] 0.6923077
```

That syntax found the proportion of elements meeting the criterion.

Note that sum(), length() and length(which()) all provide mechanisms for counting elements.

Now find the percentage of 7s in b.

```
P7 <- 100 * length(which(b == 7)) / length(b)
P7
```

```
[1] 15.38462
```

EXTENSION EXAMPLE
You can find counts and percentages using functions that involve length(which()).

```
count <- function(x, n){ length((which(x == n))) }
perc  <- function(x, n){ 100*length((which(x == n))) / length(x) }

count(b, 3)
```

```
[1] 4
```

```
perc(b, 4)
```

```
[1] 7.692308
```

Note the syntax involved in setting up an R function.

2. TEST FOR THE EXISTENCE OF PARTICULAR VALUES USING THE any() AND all() COMMANDS

```
any(b == -4)
```

```
[1] FALSE
```

```
any(b < 5)
```

```
[1] TRUE
```

Both commands work on logical vectors. Use any() to check for missing data in a vector or an array.

```
d <- c(3, 2, NA, 5, 6, NA)
d
```

```
[1]  3  2 NA  5  6 NA
```

```
any(is.na(d))
```

```
[1] TRUE
```

Of course, we can check for non-missing data too.

```
any(!is.na(d))
```

```
[1] TRUE
```

The any() command is helpful when checking for particular values in large data sets.

You can use the all() command to check whether all elements in a given vector or array satisfy a particular condition. For example, let's see whether all non-missing values in d are less than 5. Here we note noting that the command is.na() identifies missing data and that the syntax !is.na() identifies non-missing data.

```
all(d[!is.na(d)] < 5)
```

```
[1] FALSE
```

Now check whether all non-missing elements are less than 7.

```
all(d[!is.na(d)] < 7)
```

```
[1] TRUE
```

The syntax above looks formidable. However, is.na() identifies missing elements by creating a logical vector whose elements are either TRUE or FALSE.

```
is.na(d)
```

```
[1] FALSE FALSE TRUE FALSE FALSE TRUE
```

The syntax !is.na(d) gives the opposite logical vector and counts non-missing data. Then, d[!is.na(d)] gives the elements of d that are-non missing. Finally, we apply the all() command, and include the condition that all elements are less than 7.

3. COUNT VALUES WITHIN CASES (I.E. ALONG ROWS)

SPSS has the Count Values within Cases option, but R does not. Here are two functions that you might find helpful, each of which counts values within cases inside a rectangular array. Note the syntax for creating a function, which I will not discuss here. Copy and paste the two functions into the R workspace.

```
countcases1 <- function(x, n) { apply(x, 1, function(r) sum(r == n)) }
```

OR

```
countcases2 <- function(x, n) { rowSums(x == n) }
```

Now let's use these function to count elements within a rectangular array. Let's use the array M.

```
M <- structure(c(1, 4, 5, 4, 5, 5, 3, 2, 5, 5, 5, 4, 5, 2, 5), .Dim =
c(3L, 5L), .Dimnames = list(c("g", "h", "j"), NULL))
```

```
M
```

```
   [,1] [,2] [,3] [,4] [,5]
g    1    4    3    5    5
h    4    5    2    5    2
j    5    5    5    4    5
```

Let's count the 5s along the rows of M.

```
countcases1(M, 5)
```

```
g h j
2 2 4
```

OR

```
countcases2(M, 5)
```

```
g h j
2 2 4
```

OK. Each of the two functions has produced a vector of counts. Now let's pick out the number of fives in the FIRST and SECOND rows using square brackets.

```
countcases1(M, 5)[1]
```

```
g
2
```

```
countcases1(M, 5)[3]
```

```
j
4
```

4. A NICE WAY OF SUBSETTING FOR ROWS

Let's provide summary tables on the following data set of tourists from different countries, the numbers of their children, and the amount of money they spent while on vacation. Copy and paste the following array into R.

```
A <- structure(list(COUNTRY = structure(c(3L, 3L, 3L, 3L, 1L, 3L, 2L, 3L, 1L, 3L, 3L,
1L, 2L, 2L, 3L, 3L, 3L, 2L, 3L, 1L, 1L, 3L,
1L, 2L), .Label = c("AUS", "JAPAN", "USA"), class = "factor"),GENDER = structure(c(2L,
1L, 2L, 2L, 1L, 2L, 1L, 2L, 1L, 2L, 2L, 1L, 1L, 1L, 1L, 2L, 1L, 1L, 2L, 2L, 1L, 1L,
1L, 2L), .Label = c("F", "M"), class = "factor"), CHILDREN = c(2L, 1L, 3L, 2L, 2L, 3L,
1L, 0L, 1L, 0L, 1L, 2L, 2L, 1L, 1L, 1L, 0L, 2L, 1L, 2L, 4L, 2L, 5L, 1L), SPEND =
c(8500L, 23000L, 4000L, 9800L, 2200L, 4800L, 12300L, 8000L, 7100L, 10000L, 7800L,
7100L, 7900L, 7000L, 14200L, 11000L, 7900L, 2300L, 7000L, 8800L, 7500L, 15300L, 8000L,
7900L)), .Names = c("COUNTRY", "GENDER", "CHILDREN", "SPEND"), class = "data.frame",
row.names = c(NA, -24L))
```

```
A
```

	COUNTRY	GENDER	CHILDREN	SPEND
1	USA	M	2	8500
2	USA	F	1	23000
3	USA	M	3	4000
4	USA	M	2	9800
5	AUS	F	2	2200
6	USA	M	3	4800
7	JAPAN	F	1	12300
8	USA	M	0	8000
9	AUS	F	1	7100
10	USA	M	0	10000
11	USA	M	1	7800
12	AUS	F	2	7100
13	JAPAN	F	2	7900
14	JAPAN	F	1	7000
15	USA	F	1	14200
16	USA	M	1	11000
17	USA	F	0	7900
18	JAPAN	F	2	2300
19	USA	M	1	7000
20	AUS	M	2	8800
21	AUS	F	4	7500
22	USA	F	2	15300
23	AUS	F	5	8000
24	JAPAN	M	1	7900

The generic form of the syntax we will use is as follows:

```
Z <- A[ A[ , num ] == val, ]
```

Note that we have two sets of square brackets and a comma just before the second closing bracket. Z gives all rows for which an indicator in column num has the value val. We can say it like this: "Z is the set of rows of A such that the elements of column num have the value val". OK. Let's subset for males.

```
Z <- A[ A[, 2] == "M", ]
Z
```

	COUNTRY	GENDER	CHILDREN	SPEND
1	USA	M	2	8500
3	USA	M	3	4000
4	USA	M	2	9800
6	USA	M	3	4800
8	USA	M	0	8000
10	USA	M	0	10000
11	USA	M	1	7800
16	USA	M	1	11000
19	USA	M	1	7000
20	AUS	M	2	8800
24	JAPAN	M	1	7900

Now isolate all rows for which the third column (number of children) is greater than 3.

```
Z <- A[ A[, 3] > 3, ]
Z
```

```
      COUNTRY  GENDER  CHILDREN    SPEND
21      AUS       F         4        7500
23      AUS       F         5        8000
```

5. RE-CODING

You can re-code an entire vector or array at once. Let's set up a vector that has missing values.

```
A <- c(3, 2, NA, 5, 3, 7, NA, NA, 5, 2, 6)
A
```

```
[1]  3  2 NA  5  3  7 NA NA  5  2  6
```

We can re-code all missing values by another number (such as zero) as follows:

```
A[ is.na(A) ] <- 0
A
```

```
[1] 3 2 0 5 3 7 0 0 5 2 6
```

Let's re-code all values less than 5 to the value 99.

```
A[ A < 5 ] <- 99
A
```

```
 [1] 99 99 99  5 99  7 99 99  5 99  6
```

You might want to re-code an array with character elements to numeric elements.

```
gender <- c("MALE","FEMALE","FEMALE","UNKNOWN","MALE")
gender
```

```
[1] "MALE"   "FEMALE"   "FEMALE"   "UNKNOWN"   "MALE"
```

Let's re-code males as 1 and females as 2. Very useful is the following re-coding syntax because it works in many practical situations. It involves repeated (nested) use of the ifelse() command.

```
ifelse(gender == "MALE", 1, ifelse(gender == "FEMALE", 2, 3))
```

```
[1] 1 2 2 3 1
```

The element with unknown gender was re-coded as 3. Make a note of this syntax. It's great for re-coding within R programmes.

Another example, this time using a rectangular array.

```
A <- data.frame(Gender = c("F", "F", "M", "F", "B", "M", "M"), Height =
c(154, 167, 178, 145, 169, 183, 176))

A
```

```
    Gender Height
1       F    154
2       F    167
3       M    178
4       F    145
5       B    169
6       M    183
7       M    176
```

We have deliberately introduced an error where gender is misclassified as B. This one gets re-coded to the value 99. Note that the Gender variable is located in the first column, or A[,1].

```
A[,1] <- ifelse(A[,1] == "M", 1, ifelse(A[,1] == "F", 2, 99))
A
```

```
    Gender Height
1        2    154
2        2    167
3        1    178
4        2    145
5       99    169
6        1    183
7        1    176
```

You can use the same approach to code as many different levels as you need to. Let's re-code for four different levels. My example is drawn from the films of the Lord of the Rings and the Hobbit. The sets where Peter Jackson produced these films are just a short walk from where I live, so the example is relevant for me.

```
S <- data.frame(SPECIES = c("ORC", "HOBBIT", "ELF", "TROLL", "ORC",
"ORC", "ELF", "HOBBIT"), HEIGHT
= c(194, 127, 178, 195, 149, 183, 176, 134))
```

```
S
```

```
    SPECIES HEIGHT
1     ORC    194
2  HOBBIT    127
3     ELF    178
4   TROLL    195
5     ORC    149
6     ORC    183
7     ELF    176
8  HOBBIT    134
```

We now use nested ifelse commands to re-code Orcs as 1, Elves as 2, Hobbits as 3, and Trolls as 4.

```
S[,1] <- ifelse(S[,1] == "ORC", 1, ifelse(S[,1] == "ELF", 2,
ifelse(S[,1] == "HOBBIT", 3, ifelse(S[,1] == "TROLL", 4, 99))))
S
```

```
     SPECIES HEIGHT
1       1    194
2       3    127
3       2    178
4       4    195
5       1    149
6       1    183
7       2    176
8       3    134
```

We can recode back to numeric just as easily.

```
S[,1] <- ifelse(S[,1] == 1, "ORC", ifelse(S[,1] == 2, "ELF",
ifelse(S[,1] == 3, "HOBBIT",  ifelse(S[,1] == 4, "TROLL", 99))))
S
```

The general approach is the same as before, but now you have a few additional sets of parentheses.

LESS EFFICIENT APPROACHES TO RECODING.
Other approaches to recoding variables involving operations on individual variables. They are less elegant and involve more syntax than the above approach. For example:

```
a <- c(2, 3, 4, 5)
b <- c(1, 2, 1, 2)
V <- cbind(a, b)
V <- data.frame(V)    # You must set up your array as a data frame.
```

```
V
```

```
  a b
1 2 1
2 3 2
3 4 1
4 5 2
```

```
V$b[V$b == 1] <- "Male"
V$b[V$b == 2] <- "Female"
V
```

```
   a    b
1  2   Male
2  3 Female
3  4   Male
4  5 Female
```

You may want to re-code from character to numeric. Take the array V and re-code back to numeric.

```
V$b[V$b == "Male"]   <- 1
V$b[V$b == "Female"] <- 2
V
```

```
  a b
1 2 1
2 3 2
3 4 1
4 5 2
```

6. MAKING TABLES

The aggregate() function splits your data into subsets and produces summary statistics on each subset. Let's use the aggregate function to provide summary tables on the following fictional data set of tourists from different countries, the numbers of their children and the amount of money they spent while on vacation.

```
A <- structure(list(COUNTRY = structure(c(3L, 3L, 3L, 3L, 1L, 3L, 2L, 3L, 1L, 3L, 3L,
1L, 2L, 2L, 3L, 3L, 3L, 2L, 3L, 1L, 1L, 3L,
1L, 2L), .Label = c("AUS", "JAPAN", "USA"), class = "factor"),GENDER = structure(c(2L,
1L, 2L, 2L, 1L, 2L, 1L, 2L, 1L, 2L, 2L, 1L, 1L, 1L, 1L, 2L, 1L, 1L, 2L, 2L, 1L, 1L,
1L, 2L), .Label = c("F", "M"), class = "factor"), CHILDREN = c(2L, 1L, 3L, 2L, 2L, 3L,
1L, 0L, 1L, 0L, 1L, 2L, 2L, 1L, 1L, 1L, 0L, 2L, 1L, 2L, 4L, 2L, 5L, 1L), SPEND =
c(8500L, 23000L, 4000L, 9800L, 2200L, 4800L, 12300L, 8000L, 7100L, 10000L, 7800L,
7100L, 7900L, 7000L, 14200L, 11000L, 7900L, 2300L, 7000L, 8800L, 7500L, 15300L, 8000L,
7900L)), .Names = c("COUNTRY", "GENDER", "CHILDREN", "SPEND"), class = "data.frame",
row.names = c(NA, -24L))

print(A)
attach(A)
```

Now look at each variable as a table using one line of syntax.

```
sapply(A, function(x) cbind(sort(table(x), decreasing = TRUE)))
```

Create a table of numbers of tourists by country.

```
numtourists <- aggregate(COUNTRY, list(COUNTRY = A$COUNTRY), length)
numtourists
```

```
   COUNTRY x
1      AUS  6
2    JAPAN  5
3      USA 13
```

Plot the numbers of tourists by country of origin.

```
plot(x ~ COUNTRY, numtourists, ylim = c(0,max(x)), ylab = "Numbers of
Tourists", bty="l", main = "Tourist Numbers by Country of Origin")
```

Create a table of numbers of tourists by gender.

```
tourists_gender <- aggregate(GENDER, list(GENDER = GENDER), length)
tourists_gender
```

Create a table of numbers of children.

```
tourists_children <- aggregate(CHILDREN, list(Children = CHILDREN), length)
tourists_children
```

Various tables.

```
aggregate(CHILDREN, by = list(A$COUNTRY), sum)

aggregate(CHILDREN, by = list(A$COUNTRY), mean)

aggregate(CHILDREN, by = list(A$COUNTRY), max)

aggregate(SPEND, by = list(A$COUNTRY), sum)

aggregate(SPEND, by = list(A$COUNTRY), mean)

aggregate(SPEND, by = list(A$COUNTRY), max)
```

Now suppose that you want to create a more complex breakdown. Let's say that you want to identify those tourists from each country who spent the greatest amount of money, and include the number of their children. You could try the following approach involving the aggregate function (using the tilde notation), and then the merge function:

```
maxs <- aggregate(SPEND ~ COUNTRY, data = A, FUN = max)
maxs
```

```
   COUNTRY SPEND
1      AUS  8800
2    JAPAN 12300
3      USA 23000
```

```
merge(maxs, A)
```

```
   COUNTRY SPEND GENDER CHILDREN
1      AUS  8800      M        2
2    JAPAN 12300      F        1
3      USA 23000      F        1
```

Create tables of numbers of tourists by country and gender (two categorical variables).

```
genderbycountry <- aggregate(COUNTRY, list(Country = COUNTRY, Gender =
GENDER), length)
genderbycountry
```

```
    Country Gender x
1      AUS      F 5
2    JAPAN      F 4
3      USA      F 4
4      AUS      M 1
5    JAPAN      M 1
6      USA      M 9
```

Compare the above tables with those created using the table() command. You can sort your tables as follows:

```
sort(table(A$COUNTRY), dec = TRUE)
COUNTRY
  USA   AUS JAPAN
   13     6     5
```

```
sort(table(GENDER), dec = TRUE)
GENDER
 F  M
13 11
```

However, David Lillis has developed a handy way of creating tables and including any number of columns with any kind of statistic. Let's use David's method on the array A.

```
numcountries <- length(unique(A$COUNTRY))

cat("THE NUMBER OF DIFFERENT COUNTRIES IS:", numcountries,"\n")
```

Now subset by individual countries and produce a table with six columns, including the Country, the number of records for each country, the mean number of children, the total number of children, the maximum spend and the mean spend. Observe closely how this is done. First initialise each array. The critical part of the process is to set up a separate subset for each level of the categorical variable.

```
COUNTRY  <- array(0,c(numcountries))
TABLE    <- array(0,c(numcountries,6))
count    <- array(0, numcountries)


for (k in 1: numcountries) {

COUNTRY[k]  <- as.character(unique(A$COUNTRY)[k])

Z <- A[A$COUNTRY ==  COUNTRY[k],]

count[k] <- nrow(Z)

meanchildren   <- round(mean(Z$CHILDREN),1)
totalchildren  <- sum(Z$CHILDREN)
maxspend       <- max(Z$SPEND)
meanspend      <- round(mean(Z$SPEND),1)


TABLE[k,] <- c(as.character(COUNTRY[k]), count[k], meanchildren,
totalchildren, maxspend, meanspend)}

TABLE

colnames(TABLE) <- c("Country", "Records","Mean Children", "Total Children",
"Max Spend", "Mean Spend")

TABLE
```

```
     Country Records Mean Children Total Children   Max Spend Mean Spend
[1,] "USA"   "13"      "1.3"          "17"           "23000"   "10100"
[2,] "AUS"   "6"       "2.7"          "16"           "8800"    "6783.3"
[3,] "JAPAN" "5"       "1.4"          "7"            "12300"   "7480"
```

7. CREATING HISTOGRAMS

Let's create a histogram from all the data in an array.

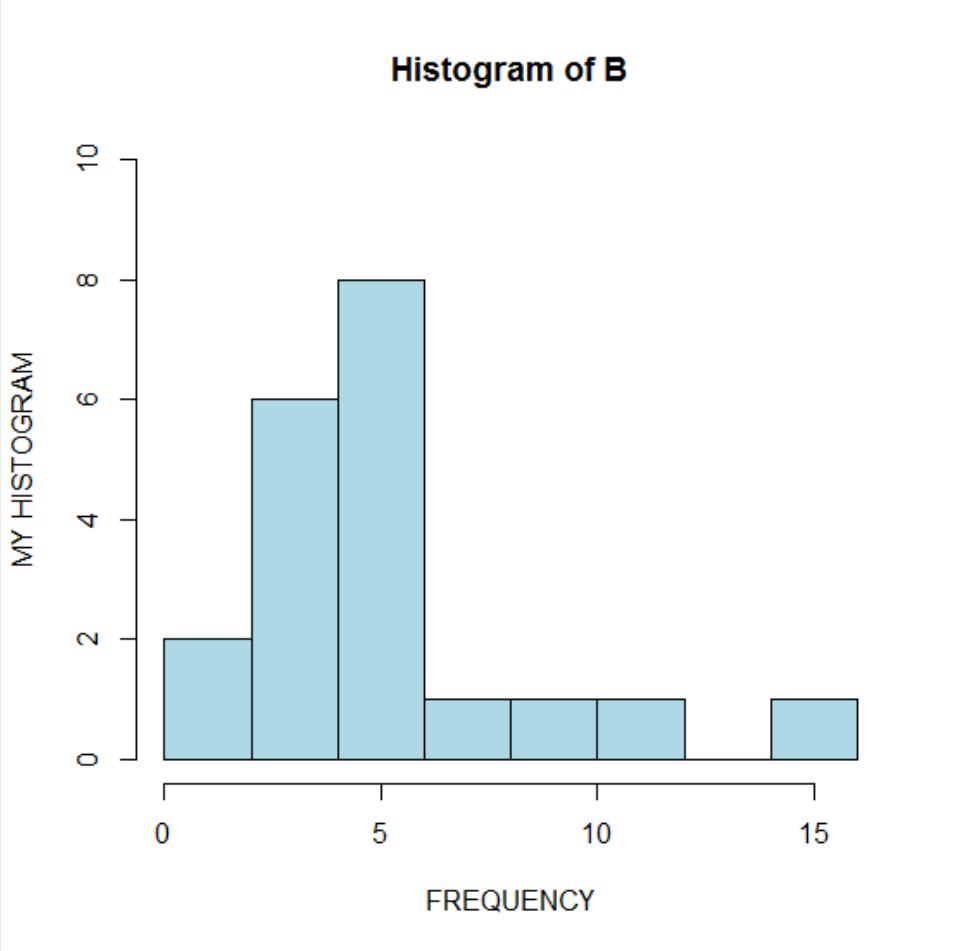
```
A <- structure(list(James = c(1L, 3L, 6L, 4L, 9L), Robert = c(2L, 5L,
4L, 5L, 12L), David = c(4L, 4L, 6L, 6L, 16L), Anne = c(3L, 5L,
6L, 7L, 6L)), .Names = c("James", "Robert", "David", "Anne"), class =
"data.frame", row.names = c(NA, -5L))
attach(A)
```

Transform the four vectors into a single vector and make a histogram of all elements.

```
B <- c(A$James, A$Robert, A$David, A$Anne)
```

Create a histogram in light blue.

```
hist(B, col="lightblue", ylim=c(0,10), ylab ="MY HISTOGRAM", xlab
="FREQUENCY")
```



Now a more complex example. Transform the four vectors, as before.

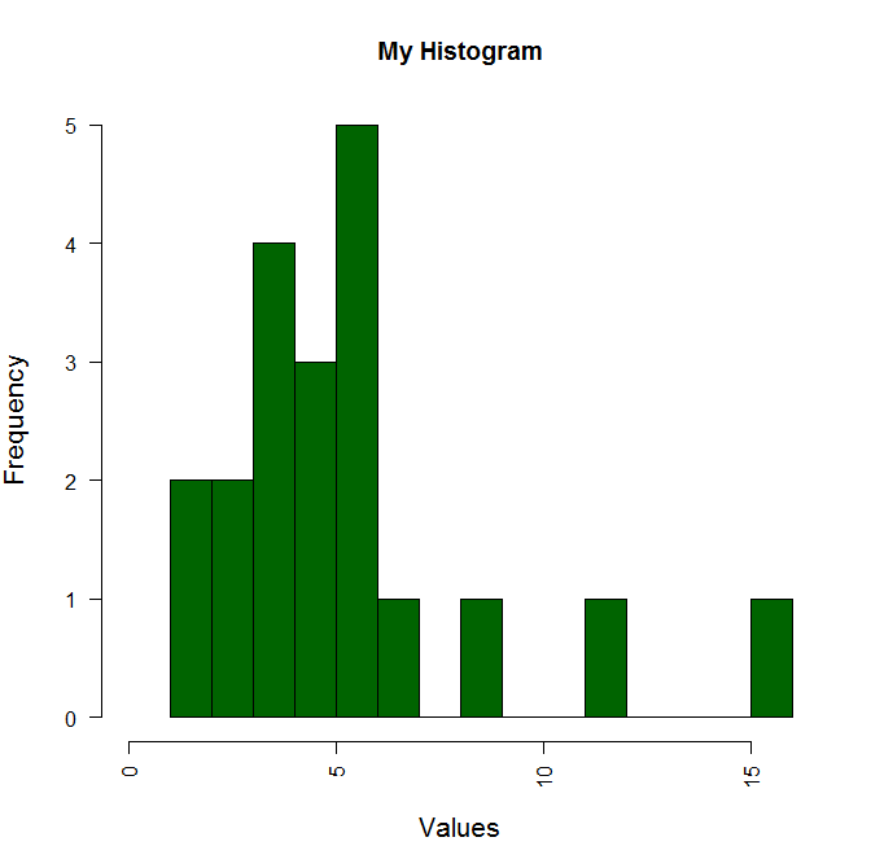
```
B <- c(A$James, A$Robert, A$David, A$Anne)
```

Find the maximum value in order to set the horizontal axis limits.

```
max <- max(B)
```

Make the horizontal axis labels perpendicular to the axis using las = 2.

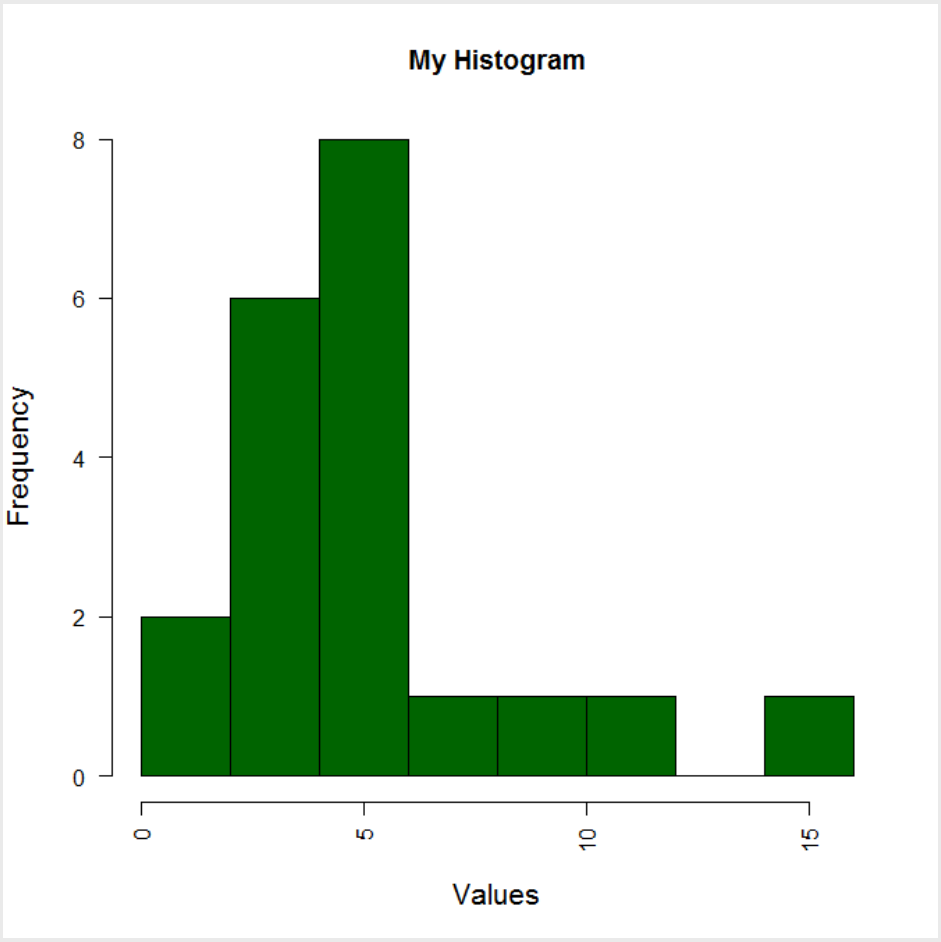
```
hist(B, col= "darkgreen", breaks=max, xlim=c(0,max),
main="My Histogram", las=2, xlab = "Values", cex.lab = 1.3)
```



Note the effect of the argument las=2. Compare it with the effect of las = 1.

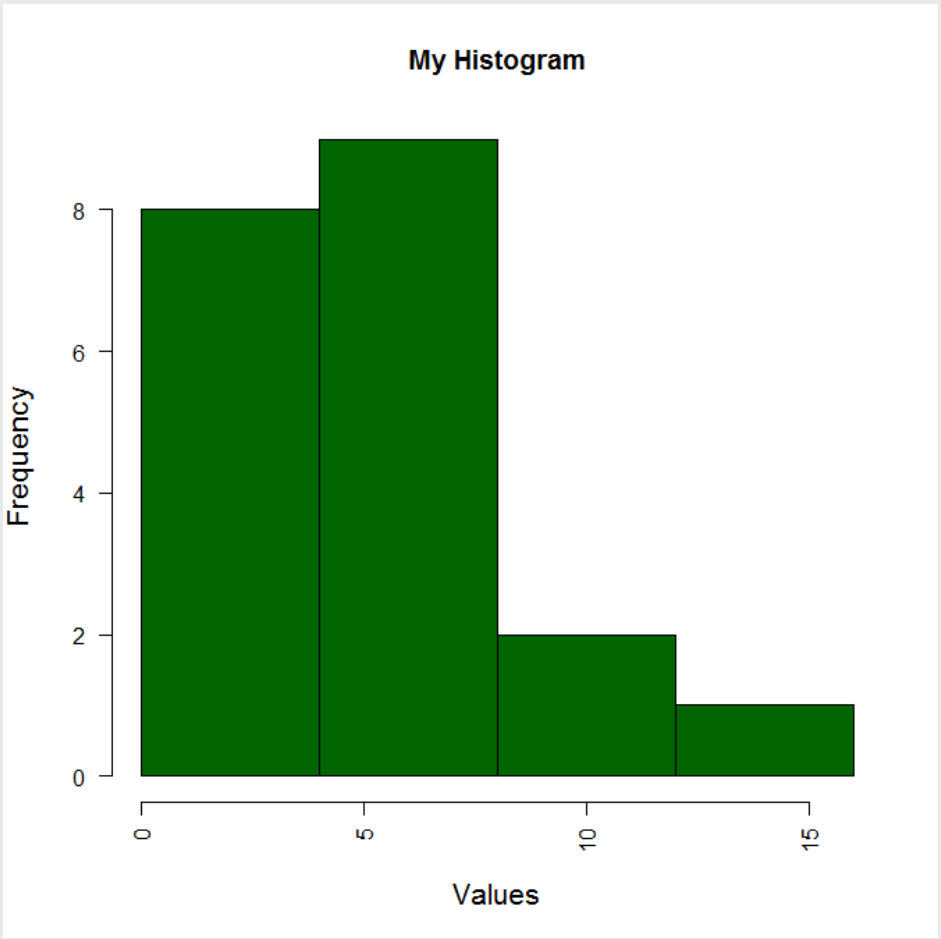
Try setting the number of bins at 6.

```
hist(B, col =  "darkgreen", breaks=6, xlim=c(0,max),
main="My Histogram", las=2, xlab = "Values", cex.lab = 1.3)
```



R has taken the number of bins (6) as indicative only. However, setting up histogram bins as a vector gives you more control over the output. Now we set up the bins as a vector of upper bounds and lower bounds, each bin four units wide.

```
bins <- c(0, 4, 8, 12, 16)

hist(B, col = "darkgreen", breaks=bins, xlim=c(0,max),
main="My Histogram", las=2, xlab = "Values", cex.lab = 1.3)
```


My Histogram
Frequency
0
2
4
6
8
0
5
10
15
Values

8. USE THE GREP COMMAND

Note the grep command, which selects columns on the basis of characters that make up the variable names. First let's practice the cat command:

Cut and paste the following syntax into the R workspace:

```
M <- structure(list(John = c(56L, 67L, 68L, 73L, 72L, 64L), Mary = c(73L,
82L, 80L, 78L, 79L, 80L), Dave = c(45L, 42L, 51L, 46L, 60L, 54L), Anne =
c(78L, 85L, 92L, 83L, 77L, 89L), Bob = c(51L, 49L, 58L, 54L, 62L, 68L)),
.Names = c("John", "Mary", "Dave", "Anne", "Bob"), class = "data.frame",
row.names = c("English", "History", "Maths", "Physics", "Chemistry",
"Biology"))
M

cat(" 'e' appears twice in", colnames(M),"\n")
```

Now identify those columns whose column names contain an e.

```
if(length(i <- grep("e",colnames(M))))  print(colnames(M)[i])

"Dave" "Anne"
```

Now create an array consisting of those same columns. Let's call it b.

```
if(length(i <- grep("e",colnames(M))))  b <- ((M)[i])
print(b)
            Dave Anne
 English      45   78
 History      42   85
 Maths        51   92
 Physics      46   83
 Chemistry    60   77
 Biology      54   89
```

9. DRAW REGRESSION LINES AND RESIDUALS

Let's use a function to draw in regression lines and plot residuals. First read in a data set.

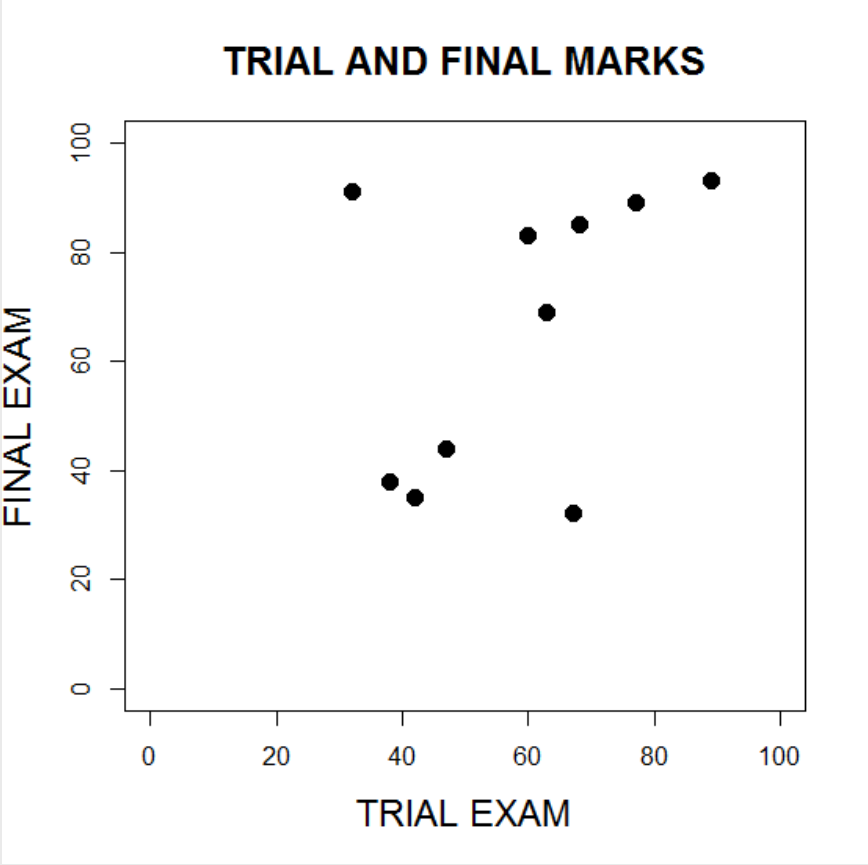
```
mod <- structure(list(Trial = c(68L, 47L, 63L, 38L, 60L, 89L, 42L, 77L,
32L, 67L), Final = c(85L, 44L, 69L, 38L, 83L, 93L, 35L, 89L, 91L,
32L)), .Names = c("Trial", "Final"), class = "data.frame", row.names =
c(NA, -10L))
attach(mod)
names(mod)
```

```
mod
```

```
 Trial Final
1      68    85
2      47    44
3      63    69
4      38    38
5      60    83
6      89    93
7      42    35
8      77    89
9      32    91
10     67    32
```

Let's plot the data.

```
plot(Trial, Final,pch=16,xlab="TRIAL EXAM",ylab="FINAL EXAM",
main="TRIAL AND FINAL MARKS", cex=1.5, cex.lab = 1.5, cex.main = 1.6,
xlim=c(0,100), ylim=c(0,100))
```

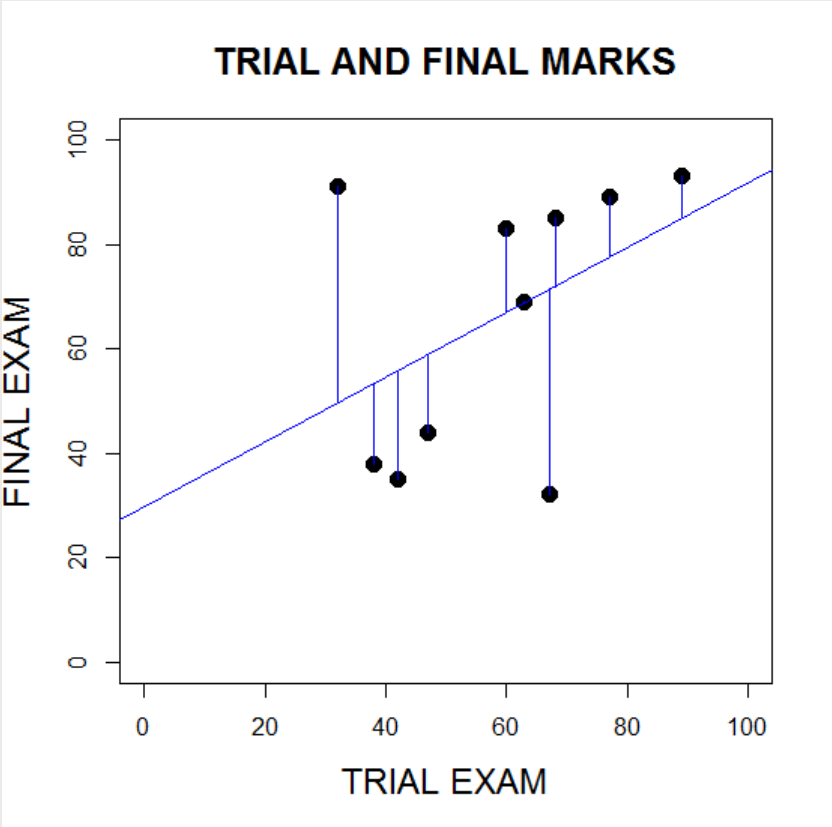


Now we draw both the regression line and the residuals, after loading the function into R.

```
drawresid <- function(X, Y, col)
{
abline(lm(Y ~ X), col = col)
regressionmodel <- predict(lm(Y ~ X))
for(k in 1:length(X)){ lines(c(X[k], X[k]), c(Y[k],
regressionmodel[k]), col = col) }
}
```

You enter the word drawresid at the command line, and then as the three arguments, you give the independent variable, the dependent variable, and finally the colour of the lines and the regression line. To illustrate, we will draw the regression line and the residuals in blue.

```
drawresid(Trial, Final, "blue")
```



10. STANDARDISED REGRESSION COEFFICIENTS

Here is a useful function for finding the standardised coefficients of a linear model. The function takes a linear model object as argument and returns the standardised coefficients. First copy the function into the R workspace.

```
std.coeffs   <- function(regression) {
unstd_coeffs <- summary(regression)$coef[-1, 1]
std_dev_x    <- sapply(regression$model[-1], sd)
std_dev_y    <- sapply(regression$model[1], sd)
std_coeffs   <- unstd_coeffs  * std_dev_x  / std_dev_y
return(std_coeffs)}
```

Let's try it on a data set.

```
sasquatch <- data.frame(height=c(245, 254, 142, 156, 262, 244, 267,
246, 252, 278),weight=c(254, 267, 114, 131, 298, 219, 301, 218, 298,
344), armspan=c(223, 244, 107, 135, 286, 237, 288, 256, 297, 299),
footsize = c(33, 35, 41, 42, 34, 40, 56, 36, 41, 44))
attach(sasquatch)

sasquatch
```

```
    height weight  armspan footsize
1      245    254      223       33
2      254    267      244       35
3      242    214      207       41
4      156    131      135       22
5      262    298      286       59
6      244    219      237       40
7      267    301      288       56
8      246    218      256       36
9      252    298      297       41
10     278    344      299       64
```

```
model1 <- lm(weight ~ height + armspan + footsize, data =
sasquatch)

summary(model1)
```

```
 Call:
 lm(formula = weight ~ height + armspan + footsize, data = sasquatch)

 Residuals:
     Min       1Q   Median       3Q      Max
 -42.660   -4.620    7.111   10.333   30.358

 Coefficients:
             Estimate Std. Error t value Pr(>|t|)
```

```
(Intercept) -86.2865    96.4524  -0.895    0.405
height        0.7062     0.7634   0.925    0.391
armspan       0.5839     0.5335   1.095    0.316
footsize      0.6592     1.4265   0.462    0.660

Residual standard error: 26.89 on 6 degrees of freedom
Multiple R-squared: 0.9144,     Adjusted R-squared: 0.8716
F-statistic: 21.36 on 3 and 6 DF,  p-value: 0.001327
```

OK, but what are the standardised coefficients? Luckily, the std.coeff() command finds all of the standardised coefficients at one time.

```
std.coeffs(model1)

    height    armspan   footsize
0.43742621 0.52107700 0.05882438
```

BY THE WAY – R HAS NICE SYNTAX FOR IMPROVING REGRESSION MODELS. Let's fit the model along with interactions.

```
model2 <- lm(weight ~ height * armspan * footsize, data =
sasquatch)

summary(model2)

Call:
lm(formula = weight ~ height * armspan * footsize, data = sasquatch)

Residuals:
       1        2        3        4        5        6        7        8
 3.95070 -6.34830 -0.91564  1.05458  2.83702  1.83034  0.08883 -2.28433
       9       10
-0.67875  0.46555

Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)             -2.394e+03  2.031e+03  -1.179   0.3598
height                   2.258e+01  8.059e+00   2.802   0.1073
armspan                 -2.056e+01  1.056e+01  -1.947   0.1910
footsize                 6.861e+01  4.973e+01   1.380   0.3017
height:armspan           3.463e-02  3.875e-02   0.894   0.4658
height:footsize         -5.961e-01  1.992e-01  -2.993   0.0959 .
armspan:footsize         4.437e-01  2.577e-01   1.722   0.2272
height:armspan:footsize -5.017e-04  9.495e-04  -0.528   0.6500
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6.13 on 2 degrees of freedom
Multiple R-squared: 0.9985,     Adjusted R-squared: 0.9933
F-statistic: 192.4 on 7 and 2 DF,  p-value: 0.005181
```

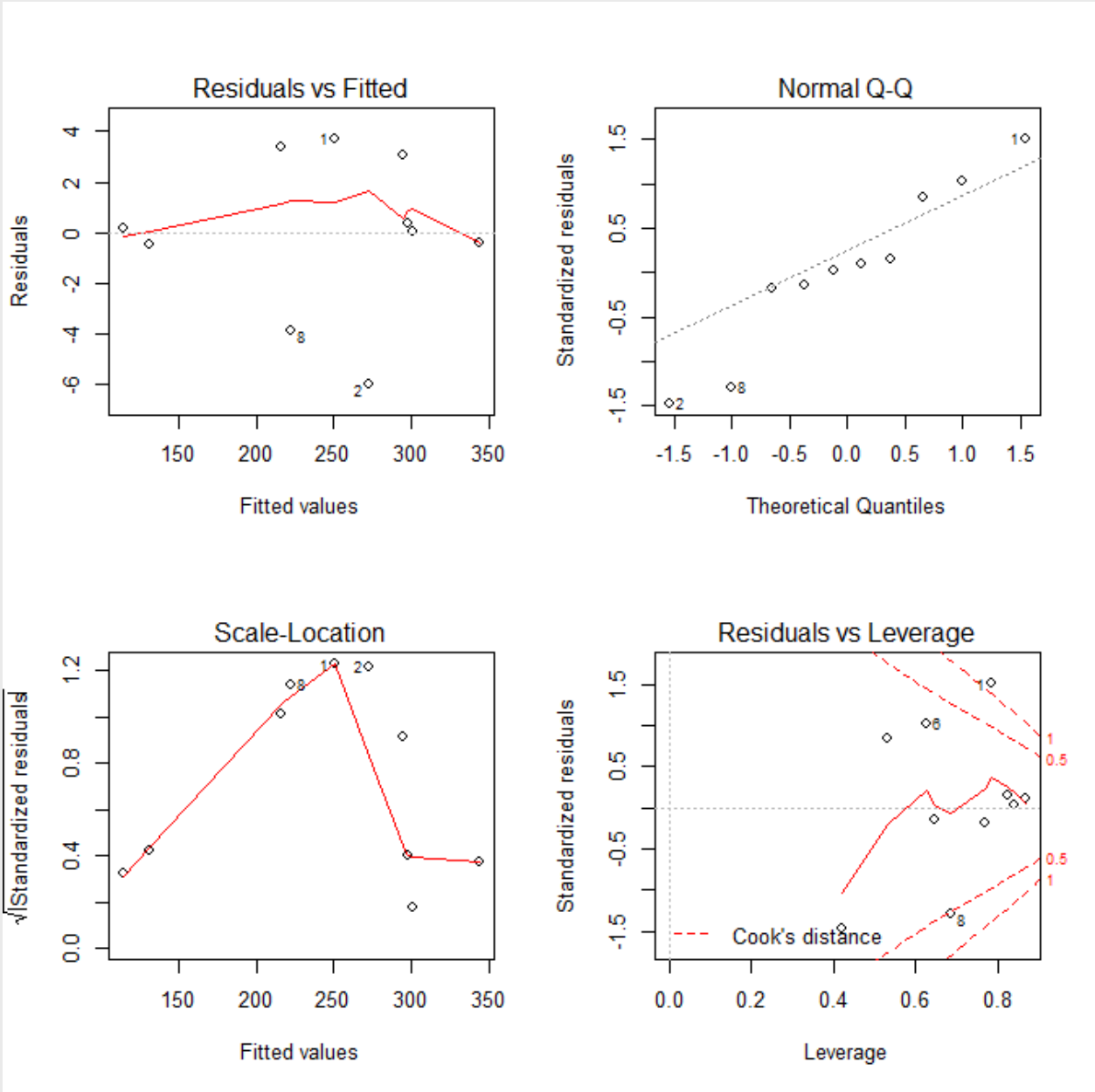
Now we eliminate the three-way interaction.

```
model3 <- update(model2,~. - height:armspan:footsize)
summary(model3)
```

```
Call:
lm(formula = weight ~ height + armspan + footsize + height:armspan +
    height:footsize + armspan:footsize, data = sasquatch)

Residuals:
       1        2        3        4        5        6        7        8
 3.70983 -5.97776  0.20116 -0.45317  3.05873  3.35230  0.06581 -3.87238
       9       10
 0.35564 -0.44014

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
(Intercept)      -3.404e+03  6.000e+02  -5.673  0.01085 *
height            2.630e+01  3.422e+00   7.684  0.00458 **
armspan          -1.506e+01  1.541e+00  -9.771  0.00228 **
footsize          9.334e+01  1.466e+01   6.369  0.00783 **
height:armspan    1.417e-02  1.254e-03  11.297  0.00149 **
height:footsize  -6.879e-01  8.503e-02  -8.089  0.00395 **
armspan:footsize  3.093e-01  3.574e-02   8.655  0.00324 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5.343 on 3 degrees of freedom
Multiple R-squared: 0.9983,     Adjusted R-squared: 0.9949
F-statistic: 295.3 on 6 and 3 DF,  p-value: 0.0003034
```

This is a better model. However, let's look for outliers and influential points.

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

plot(model3)
```



Let's eliminate point 8.

```
model4 <- update(model3, subset=(1:length(weight)!=8))
```

```
summary(model4)

Call:
lm(formula = weight ~ height + armspan + footsize + height:armspan +
    height:footsize + armspan:footsize, data = sasquatch, subset =
(1:length(weight) !=
    8))

Residuals:
       1        2        3        4        5        6        7        9
 2.68854 -4.39942 -1.55777  1.97464  1.17097  0.03427 -0.19597 -1.15117
      10
 1.43590

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)     -2.519e+03  7.383e+02  -3.413   0.0762 .
height           2.098e+01  4.343e+00   4.831   0.0403 *
armspan         -1.288e+01  1.854e+00  -6.944   0.0201 *
footsize         7.098e+01  1.840e+01   3.858   0.0611 .
height:armspan   1.286e-02  1.311e-03   9.805   0.0102 *
height:footsize -5.549e-01  1.083e-01  -5.124   0.0360 *
armspan:footsize 2.643e-01  4.050e-02   6.524   0.0227 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.342 on 2 degrees of freedom
Multiple R-squared: 0.9992,     Adjusted R-squared: 0.997
F-statistic: 440.8 on 6 and 2 DF,  p-value: 0.002265
```

Another way to assess the significance of the model is to use the F statistic. Our output gives us an F statistic of 155 on 6 and 2 degrees of freedom. This statistic can be written as F(6, 2). We can use the qf() function (quantiles of the F distribution) to find the critical value.

```
qf(0.95, 6, 2)

[1] 19.32953
```

Our calculated value of 155 is greater than this critical value, so we can be confident in rejecting the null hypothesis and in assuming that our results cannot have occurred by chance alone. Of course we can calculate the p-value directly – the probability of getting an F statistic as large as 155. We can use the pf() function (the distribution function) to do this.

```
1 - pf(155, 6, 2)

[1] 0.006423963
```

This is the p-value returned by R.

11. PLOTTING ERROR BARS

Here is my function for plotting error bars on a graph.

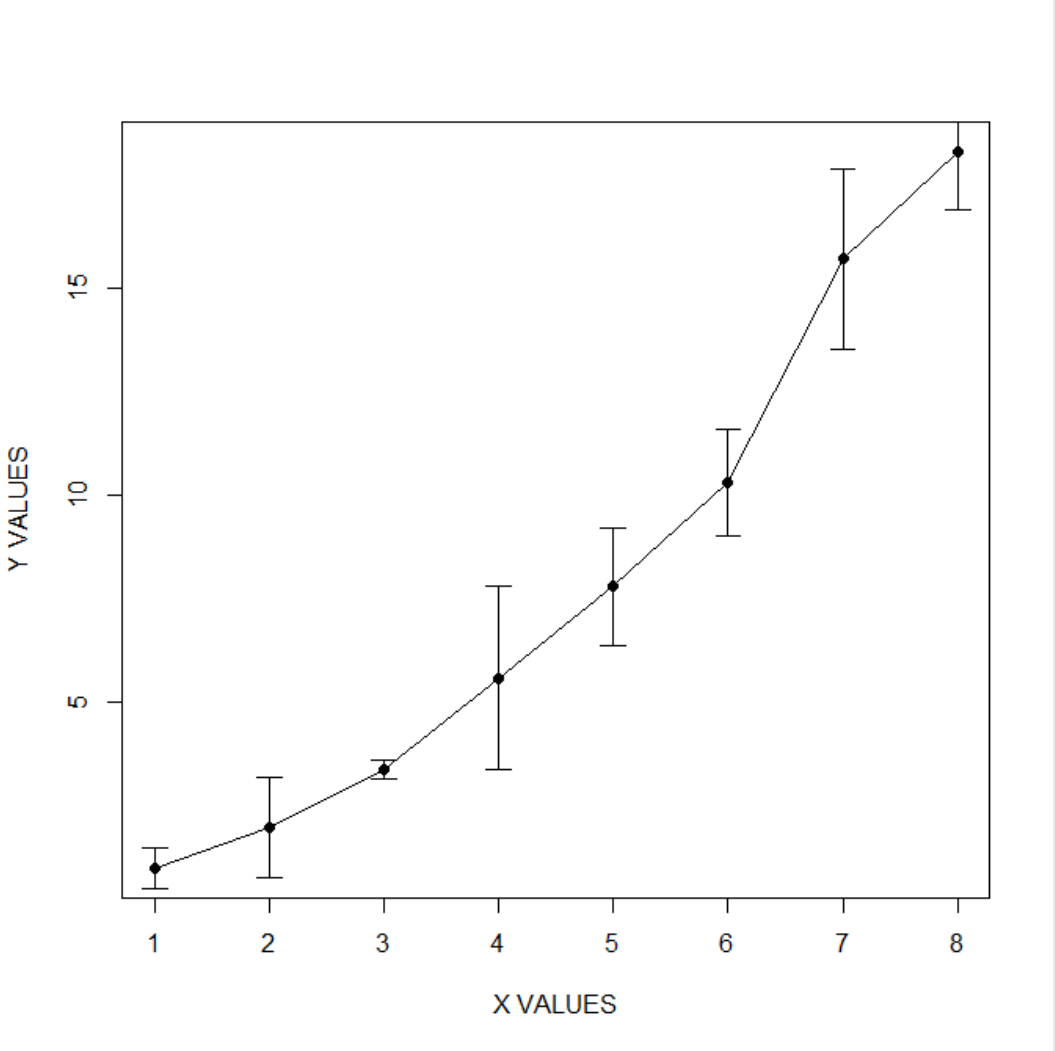
```
drawerrors <- function(w, z, err) {

zmax <- z + err
zmin <- z - err

HAT <- ( max(w) - min(w) ) / 65

for( k in 1:length(w) ) {

lines( c(w[k], w[k] ), c( z[k], zmax[k] ) , lwd = 0.8 )
lines( c(w[k], w[k] ), c( z[k], zmin[k] ) , lwd = 0.8 )

# THE FOLLOWING CODE CREATES THE HORIZONTAL BARS (THE "HAT") AT EACH
 END OF THE CONFIDENCE INTERVAL.

lines( c(w[k] - HAT, w[k] + HAT ), c( zmax[k], zmax[k] ) , lwd = 0.8 )
lines( c(w[k] - HAT, w[k] + HAT ), c( zmin[k], zmin[k] ) , lwd = 0.8 )
} }
```

Now set up some data and errors, and plot.

```
X <- c(1,2,3,4,5,6,7,8)
Y <- c(1,2,3.4,5.6,7.8,10.3, 15.7, 18.3)
ERROR <- c(0.5, 1.2, 0.23, 2.21, 1.43, 1.28 , 2.18, 1.41)


plot(X, Y, xlab = "X VALUES", ylab = "Y VALUES", pch = 16, cex=1.3)
lines(X, Y)
```

Now add in the error bars.

```
drawerrors(X, Y, ERROR)
```


Y VALUES
5
10
15
1
2
3
4
5
6
7
8
X VALUES

12. CONFIRMATORY FACTOR ANALYSIS USING LAVAAN

Here we take intelligence as a latent variable which can be measured on the basis of test scores in four areas: reading, writing, mathematics, and memory.

```
names(ability)
```

```
[1] "reading"  "writing"  "math"  "memory"
```

```
head(ability)
```

```
   reading  writing  math    memory
1    0.39     0.17    0.21     0.73
2    0.32     1.18    0.66    -0.25
3    0.22    -0.44    0.44     0.14
4    0.17     0.66   -0.29    -0.13
5    1.34     1.12    0.97    -0.03
6    1.40     0.54    0.31     0.66
```

```
library(lavaan)
```

Let's set up the model.

```
model <- ' intell  =~ reading + writing + math + memory '
```

Now fit the model.

```
fit <- cfa(model, data= ability)
fit
lavaan (0.5-13) converged normally after  24 iterations

  Number of observations                           100

  Estimator                                         ML
  Minimum Function Test Statistic               3.533
  Degrees of freedom                                2
  P-value (Chi-square)                          0.171
```

Print detailed output.

```
summary(fit, fit.measures=TRUE)
```

```
lavaan (0.5-13) converged normally after  24 iterations

  Number of observations                           100

  Estimator                                         ML
  Minimum Function Test Statistic                3.533
  Degrees of freedom                                 2
  P-value (Chi-square)                           0.171

Model test baseline model:

  Minimum Function Test Statistic              358.864
  Degrees of freedom                                 6
  P-value                                        0.000

Full model versus baseline model:

  Comparative Fit Index (CFI)                    0.996
  Tucker-Lewis Index (TLI)                       0.987

Loglikelihood and Information Criteria:

  Loglikelihood user model (H0)                -387.042
  Loglikelihood unrestricted model (H1)        -385.275

  Number of free parameters                          8
  Akaike (AIC)                                 790.083
  Bayesian (BIC)                               810.925
  Sample-size adjusted Bayesian (BIC)          785.659

Root Mean Square Error of Approximation:

  RMSEA                                          0.088
  90 Percent Confidence Interval          0.000  0.235
  P-value RMSEA <= 0.05                          0.245

Standardized Root Mean Square Residual:

  SRMR                                           0.012

Parameter estimates:

  Information                                 Expected
  Standard Errors                             Standard

                   Estimate  Std.err  Z-value  P(>|z|)
Latent variables:
  intell =~
    reading           1.000
    writing           0.895    0.070   12.726    0.000
    math              0.962    0.069   13.875    0.000
    memory          0.913    0.070   13.030    0.000

Variances:
    reading           0.221    0.043
    writing           0.214    0.039
    math              0.167    0.035
    memory            0.203    0.038
    intell            0.885    0.156
```

The output consists of three parts. The first six lines are called the header. The header contains the following information:

1. **The lavaan version number**
2. **Whether lavaan converged normally, and the number of iterations**
3. **The number of observations that were used in the analysis**
4. **The estimator used to obtain the parameter values (here: ML)**
5. **The model test statistic, the degrees of freedom, and a corresponding p-value.**

The next section contains additional fit measures, and is because we use the optional argument fit.measures = TRUE. It starts with the line Model test baseline model and ends with the value for the SRMR. The last section contains the parameter estimates. It starts with information about the standard errors (if the information matrix is expected or observed, and if the standard errors are standard, robust, or based on the bootstrap). Then, it tabulates all free (and fixed) parameters that were included in the model. Typically, first the latent variables are shown, followed by covariances and (residual) variances. The first column (Estimate) contains the (estimated or fixed) parameter value for each model parameter; the second column (Std.err) contains the standard error for each estimated parameter; the third column (Z-value) contains the Wald statistic (which is simply obtained by dividing the parameter value by its standard error), and the last column (P(>|z|)) contains the p-value for testing the null hypothesis that the parameter equals zero in the population.

Comparative Fit Index
The comparative fit index (CFI) assesses model fit through differences between the observations and the model. It adjusts for sample size inherent in the Chi-squared test of model fit, and the normed fit index. CFI ranges from 0 to 1. Large values indicate better fit. A CFI value of .90 or greater indicates a good fit.

Normed Fit Index and the Tucker-Lewis Index
The normed fit index (NFI) looks at the discrepancy between the Chi-squared value of the model and the Chi-squared value of the null model. However, the NFI is susceptible to sample size. The Tucker-Lewis index addresses some of the problems relating to sample size, though sometimes falls beyond zero and 1. Values for both the NFI and Tucker-Lewis should lie between 0 and 1. Values of .95 or greater indicate a good fit.

Root Mean Square Error of Approximation
The root mean square error of approximation (RMSEA) looks at the discrepancy between the hypothesized model and the population covariance matrix. The RMSEA ranges from 0 to 1. Smaller values indicate better model fit. A value of .06 or below suggests a good model fit.

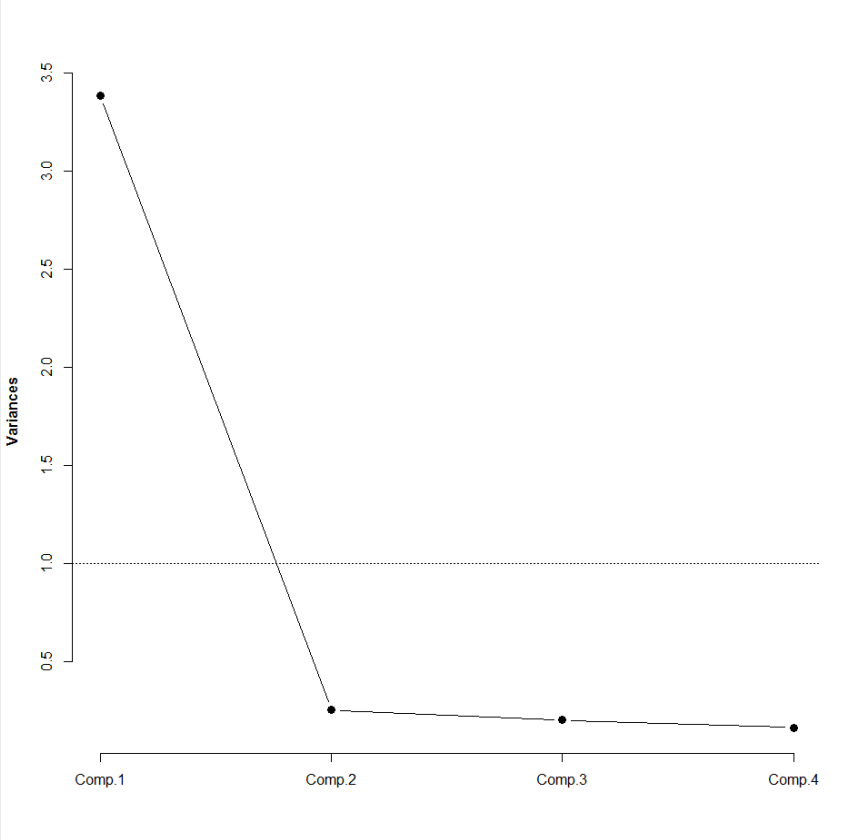
What does PCA tell us? Let's use princomp, R's default PCA function.

```
fitpca <- princomp(ability, cor=TRUE)
```

```
print(loadings(fitpca))

Loadings:
        Comp.1 Comp.2 Comp.3 Comp.4
reading -0.502 -0.164  0.794 -0.302
writing -0.495 -0.699 -0.296  0.423
math    -0.506  0.187 -0.530 -0.654
memory  -0.497  0.671         0.549


                Comp.1 Comp.2 Comp.3 Comp.4
SS loadings      1.00   1.00   1.00   1.00
Proportion Var   0.25   0.25   0.25   0.25
Cumulative Var   0.25   0.50   0.75   1.00


if(all(loadings(fitpca)[,1] < 0)) { loadings <- -1*loadings(fitpca) }

loadings
Loadings:
        Comp.1 Comp.2 Comp.3 Comp.4
reading  0.502  0.164 -0.794  0.302
writing  0.495  0.699  0.296 -0.423
math     0.506 -0.187  0.530  0.654
memory   0.497 -0.671        -0.549

                Comp.1 Comp.2 Comp.3 Comp.4
SS loadings      1.00   1.00   1.00   1.00
Proportion Var   0.25   0.25   0.25   0.25
Cumulative Var   0.25   0.50   0.75   1.00
```

Now draw a scree plot.

```
plot(fitpca, type="lines", col = "black", pch=16, cex=1.3, main= "",
font.lab=2)

abline(h=1.0, lty=3)
```



Let's try a PCA with rotation using the principal function within the psych package.

```
library(psych)
```

```
pcarot <- principal(ability, nfactors = ncol(ability), rotate =
"varimax", scores = T)
pcarot
```

```
Principal Components Analysis
Call: principal(r = ability, nfactors = ncol(ability), rotate =
"varimax",     scores = T)
Standardized loadings (pattern matrix) based upon correlation matrix
         RC2  RC1  RC3  RC4 h2      u2
reading 0.37 0.36 0.79 0.33  1 5.6e-16
writing 0.82 0.31 0.35 0.33  1 2.2e-16
math    0.37 0.39 0.34 0.77  1 5.6e-16
memory  0.32 0.81 0.34 0.35  1 2.2e-16

                       RC2  RC1  RC3  RC4
SS loadings           1.05 1.04 0.98 0.93
Proportion Var        0.26 0.26 0.25 0.23
Cumulative Var        0.26 0.52 0.77 1.00
Proportion Explained  0.26 0.26 0.25 0.23
```

```
Cumulative Proportion 0.26 0.52 0.77 1.00

Test of the hypothesis that 4 components are sufficient.

The degrees of freedom for the null model are  6  and the objective
function was  3.59
The degrees of freedom for the model are -4  and the objective function
was  0
The total number of observations was  100  with MLE Chi Square =  0
with prob <  NA
```

13. TETRACHORIC CORRELATIONS

To conduct tetrachoric correlations, we load the psych package from the CRAN web-site and use the tetrachoric command.

```
library(psych)
```

Let's perform a tetrachoric correlation on a small matrix.

```
M <- matrix(c(61661,1610,85,20),2,2)
M
      [,1]  [,2]
[1,] 61661   85
[2,]  1610   20


tetrachoric(M)

Call: tetrachoric(x = M)
tetrachoric correlation
[1] 0.35

 with tau of
[1] 1.9 2.9
```

Tau is defined as the normal equivalent of the cut points.

That correlation was very moderate. However, when both elements of the leading diagonal are similar and large, then we get a high correlation.

```
P <- matrix(c(61661,7652,7535,60089),2,2)
P
```

```
tetrachoric(P)

Call: tetrachoric(x = P)
tetrachoric correlation
[1] 0.94

 with tau of
[1] 0.013 0.015
```

In the next example, two raters have assessed a set of examination scripts and have returned the following set of results. Note how we set up the matrix and include column names and row names.

```
Pass <- c(40,20)
Fail <- c(10,30)
ratings <- as.matrix(c(Pass,Fail))
dim(ratings)<-c(2,2)
colnames(ratings) <- c("Pass", "Fail")
rownames(ratings) <- c("Pass", "Fail")
ratings

      Pass Fail
Pass    40   10
Fail    20   30
```

Now we compute the tetrachoric correlation.

```
tetrachoric(ratings)

Call: tetrachoric(x = ratings)
tetrachoric correlation
[1] 0.61

 with tau of
Pass Pass
0.00 0.25
```

14. POLYCHORIC CORRELATIONS

To conduct polychoric correlations, we load the polycor package from the CRAN web-site and use the polychor function. If you read the on-line manual that goes with the polycor package, you will see that this function computes the polychoric correlation (and its standard error) between two ordinal variables or from their contingency table, under the assumption that the ordinal variables dissect continuous latent variables that are bivariate normal. Both a maximum-likelihood estimator and a

faster two-step approximation are available through this package. For the ML estimator, the estimates of the thresholds and the covariance matrix of the estimates are also available.

The syntax for the polychor function is as follows:

```
polychor(x, y, ML = FALSE, control = list(), std.err = FALSE,
maxcor=.9999)
```

X is a contingency table of counts or an ordered categorical variable. The variable can be numeric, logical, a factor, or an ordered factor, but if it is a factor, then the levels should be in proper order.

y is a second ordered categorical variable.

ML: if TRUE, compute the maximum-likelihood estimate; if FALSE (the default) compute a quicker two-step approximation.
Control: optional arguments to be passed to the optim function. Please see the manual for further details.

std.err: if TRUE, return the estimated variance of the correlation (for the two-step estimator) or the estimated covariance matrix (for the ML estimator) of the correlation and thresholds; the default is FALSE.

maxcor: maximum absolute correlation (to insure numerical stability).

In the following example of polychoric correlation, we have data on the number of calves born to each breeding cow at a farm over two consecutive years.

```
Zeroes <- c(58,26,8)
Ones   <- c(52,62,12)
Twos   <- c(1,2,9)
calves <- as.matrix(c(Zeroes, Ones, Twos))
dim(calves) <-c(3,3)

colnames(calves) <- c("Zero", "One", "Two")
rownames(calves) <- c("Zero", "One", "Two")

calves
     Zero One Two
Zero   58  52   1
One    26  62   2
Two     8  12   9

library(polycor)

polychor(calves)
[1] 0.4258939
```

Again, when the elements of the leading diagonal are large, then the correlation is large.

15. SUB-SETTTING USING R'S SET MEMBERSHIP COMMANDS

R's set membership functions are very useful, particularly when you need to know about subsets of your data sets and the extent to which they overlap. Often you can get at such problems using the set membership functions on the indices of rows of the data.

Let's try out these functions on two small vectors.

```
x <- c(3, 4, 5, 6, 7)

y <- c(3, 5, 6, 7, 8, 9, 10)
```

Are any elements of x also in y?

```
any(x%in%y)
[1] TRUE
```

Are all elements of x also in y?

```
all(x%in%y)
[1] FALSE
```

The elements of x that are also in y are:

```
x[x%in%y]
[1] 3 5 6 7
```

OR: The intersection of x and y is:

```
intersect(x, y)
[1] 3 5 6 7
```

The elements of x that are not in y are:

```
x[!x%in%y]
[1] 4
```

OR: The difference (number of elements that are different) between x and y is:

```
setdiff(x, y)
 [1] 4
```

The indices of elements of x that are also in y are:

```
which(x%in%y)
[1] 1 3 4 5
```

The indices of elements of x that are not in y are:

```
which(!x%in%y)
[1] 2
```

The number of elements of x that are also in y is:

```
length(which(x%in%y))
[1] 4
```

The percentage of elements of x that are also in y is:

```
100*length(which(x%in%y))/length(x)
[1] 80
```

The number of elements of x that are not in y is:

```
length(which(!x%in%y))
[1] 1
```

The percentage of elements of x that are not in y is:

```
100*length(which(!x%in%y))/length(x)
[1] 20
```

The match of elements of x and y is:

```
match(x, y)
[1]  1 NA  2  3  4
```

The union of x and y is:

```
union(x, y)
[1]  3  4  5  6  7  8  9 10
```

Are the sets x and y the same?

```
setequal(x, y)
[1] FALSE
```

Here is a nice example using R's set membership functions.

```
setmemb <- function(x, y) { {
cat("\n")
cat("\n")
cat("Are any elements of x also in y?", any(x%in%y),"\n")
cat("\n")
cat("Are all elements of x also in y?", all(x%in%y),"\n")
cat("\n")
cat("The elements of x that are also in y are:", x[x%in%y],"\n")
cat("\n")
cat("The elements of x that are not in y are:", x[!x%in%y],"\n")
cat("\n")
cat("The indices of elements of x that are also in y are:",
which(x%in%y),"\n")
cat("\n")
cat("The indices of elements of x that are not in y are:",
which(!x%in%y),"\n")
cat("\n")
cat("The number of elements of x that are also in y is:",
length(which(x%in%y)),"\n")
cat("\n")
cat("The percentage of elements of x that are also in y is:",
100*length(which(x%in%y))/length(x),"%","\n")
cat("\n")
cat("The number of elements of x that are not in y is:",
length(which(!x%in%y)),"\n")
cat("\n")
cat("The percentage of elements of x that are not in y is:",
100*length(which(!x%in%y))/length(x),"%","\n")
cat("\n")
cat("The match of elements of x and y is:", match(x,y),"\n")
cat("\n")
cat("The intersection of x and y is:", intersect(x,y),"\n")
cat("\n")
cat("The difference between x and y is:", setdiff(x,y),"\n")
cat("\n")
cat("The union of x and y is:", union(x,y),"\n")
cat("\n")
cat("Are the sets x and y the same? :", setequal(x,y),"\n")
cat("\n")
}}

x <- c(3, 4, 5, 6, 7)
y <- c(3, 5, 6, 7, 8, 9, 10)
```

```
setmemb(x, y)

Are any elements of x also in y? TRUE

Are all elements of x also in y? FALSE

The elements of x that are also in y are: 3 5 6 7

The elements of x that are not in y are: 4

The indices of elements of x that are also in y are: 1 3 4 5

The indices of elements of x that are not in y are: 2

The number of elements of x that are also in y is: 4

The percentage of elements of x that are also in y is: 80 %

The number of elements of x that are not in y is: 1

The percentage of elements of x that are not in y is: 20 %

The match of elements of x and y is: 1 NA 2 3 4

The intersection of x and y is: 3 5 6 7

The difference between x and y is: 4

The union of x and y is: 3 4 5 6 7 8 9 10

Are the sets x and y the same? : FALSE
```

Let's look at a practical example that makes use of these functions. Let's set up the tourist data set again.

```
A <- structure(list(COUNTRY = structure(c(3L, 3L, 3L, 3L, 1L, 3L, 2L, 3L, 1L, 3L, 3L,
1L, 2L, 2L, 3L, 3L, 3L, 2L, 3L, 1L, 1L, 3L,
1L, 2L), .Label = c("AUS", "JAPAN", "USA"), class = "factor"),GENDER = structure(c(2L,
1L, 2L, 2L, 1L, 2L, 1L, 2L, 1L, 2L, 2L, 1L, 1L, 1L, 1L, 2L, 1L, 1L, 2L, 2L, 1L, 1L,
1L, 2L), .Label = c("F", "M"), class = "factor"), CHILDREN = c(2L, 1L, 3L, 2L, 2L, 3L,
1L, 0L, 1L, 0L, 1L, 2L, 2L, 1L, 1L, 1L, 0L, 2L, 1L, 2L, 4L, 2L, 5L, 1L), SPEND =
c(8500L, 23000L, 4000L, 9800L, 2200L, 4800L, 12300L, 8000L, 7100L, 10000L, 7800L,
7100L, 7900L, 7000L, 14200L, 11000L, 7900L, 2300L, 7000L, 8800L, 7500L, 15300L, 8000L,
7900L)), .Names = c("COUNTRY", "GENDER", "CHILDREN", "SPEND"), class = "data.frame",
row.names = c(NA, -24L))

A
```

	COUNTRY	GENDER	CHILDREN	SPEND
1	USA	M	2	8500
2	USA	F	1	23000
3	USA	M	3	4000
4	USA	M	2	9800
5	AUS	F	2	2200
6	USA	M	3	4800
7	JAPAN	F	1	12300
8	USA	M	0	8000
9	AUS	F	1	7100
10	USA	M	0	10000
11	USA	M	1	7800
12	AUS	F	2	7100
13	JAPAN	F	2	7900
14	JAPAN	F	1	7000
15	USA	F	1	14200
16	USA	M	1	11000
17	USA	F	0	7900
18	JAPAN	F	2	2300
19	USA	M	1	7000
20	AUS	M	2	8800
21	AUS	F	4	7500
22	USA	F	2	15300
23	AUS	F	5	8000
24	JAPAN	M	1	7900

Identify the indices of rows for males.

```
which(A$GENDER == "M")
```

```
[1]  1  3  4  6  8 10 11 16 19 20 24
```

Create a vector of these indices.

```
maleindices <- which(A$GENDER == "M")
```

Identify the indices of rows for people from the USA.

```
which(A$COUNTRY == "USA")
```

```
[1]  1  2  3  4  6  8 10 11 15 16 17 19 22
```

Create a vector of these indices.

```
countryindices <- which(A$COUNTRY == "USA")
```

The intersection of both sets (i.e. males from the USA) is:

```
intersect(maleindices, countryindices)
```

```
[1]  1  3  4  6  8 10 11 16 19
```

The elements of x that are also in y (i.e. again - males from the USA) are:

```
maleindices[maleindices %in% countryindices]
```

```
[1]  1  3  4  6  8 10 11 16 19
```

The number of elements of x that are also in y (males who are from the USA) is:

```
length(which(maleindices %in% countryindices))
```

```
[1] 9
```

The elements of x that are not in y (males who are not from the USA) are:

```
maleindices[!maleindices %in% countryindices]
```

```
[1] 20 24
```

The indices of elements of x that are not in y (again - indices for males who are not from the USA) are:

```
setdiff(maleindices, countryindices)
```

```
[1] 20 24
```

A Third Example

Here is another dataset, this time of patients receiving one of three medical treatments (A, B or C). We have data on their gender, their weight (actually, mass) in kg, whether or not they exercise, whether or not they smoke, and whether or not they recovered after treatment.

```
patients <- structure(list(GENDER = structure(c(2L, 2L, 1L, 1L, 1L, 2L, 2L,
1L, 2L, 2L, 1L, 1L, 1L, 2L, 2L, 2L, 1L, 1L, 2L, 2L, 2L, 1L, 1L,
1L, 2L, 2L, 1L), .Label = c("F", "M"), class = "factor"), TREATMENT = structure(c(1L,
1L, 2L, 1L, 2L, 2L, 3L, 2L, 1L, 1L, 2L, 2L, 2L, 1L, 3L, 2L, 2L, 1L, 3L, 1L, 1L, 2L,
3L, 2L, 2L, 2L, 3L), .Label = c("A", "B",
"C"), class = "factor"), WEIGHT = c(79L, 87L, 65L, 58L, 72L, 95L, 76L, 56L, 77L, 104L,
67L, 82L, 59L, 68L, 79L, 125L, 83L, 63L, 57L, 84L, 72L, 68L, 65L, 64L, 87L, 92L, 56L),
SMOKE = structure(c(2L, 2L, 1L, 2L, 1L, 1L, 1L, 1L, 2L, 1L, 1L, 2L, 1L, 2L, 1L, 2L,
1L, 2L, 1L, 2L, 2L, 1L, 1L, 1L, 2L, 1L, 2L), .Label = c("N", "Y"), class = "factor"),
EXERCISE = structure(c(2L, 2L, 1L, 1L, 1L, 1L, 2L, 2L, 2L,     1L, 2L, 2L, 1L, 1L, 2L,
1L, 1L, 1L, 1L, 1L, 2L, 1L, 1L, 1L,     1L, 2L, 1L), .Label = c("N", "Y"), class =
"factor"), RECOVER = structure(c(2L,     2L, 2L, 1L, 1L, 2L, 2L, 1L, 1L, 2L, 1L, 1L,
1L, 1L, 2L, 2L,     1L, 1L, 2L, 2L, 2L, 2L, 1L, 2L, 2L, 2L, 2L), .Label = c("N",
"Y"), class = "factor")), .Names = c("GENDER", "TREATMENT",
"WEIGHT", "SMOKE", "EXERCISE", "RECOVER"), class = "data.frame", row.names = c(NA, -
27L))
patients
```

	GENDER	TREATMENT	WEIGHT	SMOKE	EXERCISE	RECOVER
1	M	A	79	Y	Y	Y
2	M	A	87	Y	Y	Y
3	F	B	65	N	N	Y
4	F	A	58	Y	N	N
5	F	B	72	N	N	N
6	M	B	95	N	N	Y
7	M	C	76	N	Y	Y
8	F	B	56	N	Y	N
9	M	A	77	Y	Y	N
10	M	A	104	N	N	Y
11	F	B	67	N	Y	N
12	F	B	82	Y	Y	N
13	F	B	59	N	N	N
14	M	A	68	Y	N	N
15	M	C	79	N	Y	Y
16	M	B	125	Y	N	Y
17	F	B	83	N	N	N
18	F	A	63	Y	N	N
19	M	C	57	N	N	Y
20	M	A	84	Y	N	Y
21	M	A	72	Y	Y	Y
22	F	B	68	N	N	Y
23	F	C	65	N	N	N
24	F	B	64	N	N	Y
25	M	B	87	Y	N	Y
26	M	B	92	N	Y	Y
27	F	C	56	Y	N	Y

Let's create a few subsets. Let's subset for males who smoke and do no exercise. First we find the indices of those elements (rows).

```
males_smoke_notexercise <- which(patients$GENDER == "M" & patients$SMOKE
== "Y"  &  patients$EXERCISE == "N")

males_smoke_notexercise

[1] 14 16 20 25
```

Now we look at this subset using square brackets and the vector of indices. We insert the set of indices, followed by a comma.

```
patients[males_smoke_notexercise,]

     GENDER  TREATMENT WEIGHT SMOKE EXERCISE RECOVER
14      M         A      68     Y        N       N
16      M         B     125     Y        N       Y
20      M         A      84     Y        N       Y
25      M         B      87     Y        N       Y
```

We can also do it this way:

```
subset(patients, GENDER == "M" & SMOKE == "Y" &  EXERCISE == "N")
```

OR this way:

```
Z <- patients[patients$GENDER == "M" & patients$SMOKE == "Y" &
patients$EXERCISE == "N" , ]
Z
```

Let's subset for males over 85 kg who smoke and do no exercise.

```
males_over85_smoke_notexercise <- which(patients$GENDER == "M" &
patients$WEIGHT > 85 &  patients$SMOKE == "Y" &  patients$EXERCISE ==
"N")

males_over85_smoke_notexercise
[1] 16 25
```

Now we look at this subset using square brackets. We insert the set of indices, followed by a comma.

```
patients[males_over85_smoke_notexercise,]
     GENDER TREATMENT WEIGHT SMOKE EXERCISE RECOVER
16      M         B     125     Y        N       Y
25      M         B      87     Y        N       Y
```

Now we find the number of males who do not exercise using set membership syntax (use the syntax for the number of elements of x that are not in y):

```
males    <- which(patients$GENDER == "M")
exercise <- which(patients$EXERCISE == "Y")

length(which(males %in% exercise))

[1] 7
```

Now we find the percentage of smokers who do not exercise (use the percentage of elements of x that are not in y):

```
smoker <- which(patients$SMOKE == "Y")
noexercise <- which(patients$EXERCISE == "N")

100*length(which(smoker %in% noexercise))/length(smoker)

[1] 58.33333%
```

More Complex Analyses

In this section we find subsets relating to female smokers. First we identify the indices of rows for females.

```
femaleindices <- which(patients$GENDER == "F")
femaleindices

[1]  3  4  5  8 11 12 13 17 18 22 23 24 27
```

Similarly, we identify the indices of rows for people who smoke.

```
smokerindices <- which(patients$SMOKE == "Y")
smokerindices

[1]  1  2  4  9 12 14 16 18 20 21 25 27
```

Question: did any females smoke? To answer, we use: `any(x%in%y)`.

```
any(femaleindices %in% smokerindices)

[1] TRUE
```

Question: did all females smoke? Use the syntax: `all(x%in%y)`

```
all(femaleindices %in% smokerindices)
[1] FALSE
```

The intersection of both sets (i.e. females who smoke) is:

```
femaleswhosmoke <- intersect(femaleindices, smokerindices)
femaleswhosmoke
[1]  4 12 18 27
```

However, the following gives the result slightly differently (counting within the vector of indices rather than from within the original data set).

```
which(femaleindices %in% smokerindices)
[1]  2  6  9 13
```

Let's take a look at the subset we got from `femaleswhosmoke`:

```
patients[femaleswhosmoke,]
```

	GENDER	TREATMENT	WEIGHT	SMOKE	EXERCISE	RECOVER
4	F	A	58	Y	N	N
12	F	B	82	Y	Y	N
18	F	A	63	Y	N	N
27	F	C	56	Y	N	Y

Another way of getting the indices of females who smoke (i.e. using syntax for the elements of x that are also in y):

```
femaleindices[femaleindices %in% smokerindices]

[1]  4 12 18 27
```

The number of females who smoke is:

```
length(which(femaleindices %in% smokerindices))
[1] 4
```

The females who do not smoke are:

```
females_notsmoke <- femaleindices[!femaleindices %in% smokerindices]
females_notsmoke

[1]  3  5  8 11 13 17 22 23 24
```

Let's see the array.

```
patients[females_notsmoke, ]
```

```
       GENDER TREATMENT WEIGHT SMOKE EXERCISE RECOVER
 3        F         B     65     N        N       Y
 5        F         B     72     N        N       N
 8        F         B     56     N        Y       N
 11       F         B     67     N        Y       N
 13       F         B     59     N        N       N
 17       F         B     83     N        N       N
 22       F         B     68     N        N       Y
 23       F         C     65     N        N       N
 24       F         B     64     N        N       Y
```

Another way: the indices of elements of x that are not in y (again - indices for females who do not smoke) are:

```
females_notsmoke2 <- setdiff(femaleindices, smokerindices)
females_notsmoke2
1]  3  5  8 11 13 17 22 23 24
```

Further Development
Create a vector of the indices of rows for people who receive treatment B.

```
TREATMENTBindices <- which(patients$TREATMENT == "B")
TREATMENTBindices

[1]  3  5  6  8 11 12 13 16 17 22 24 25 26
```

Create a vector of the indices of rows for people who DO NOT receive treatment B using an exclamation mark.

```
TREATMENTNOTBindices <- which(!patients$TREATMENT == "B")
TREATMENTNOTBindices

[1]  1  2  4  7  9 10 14 15 18 19 20 21 23 27
```

Let's have a look at this subset:

```
patients[TREATMENTNOTBindices,]
```

```
   GENDER  TREATMENT  WEIGHT SMOKE EXERCISE RECOVER
1       M         A      79     Y        Y       Y
2       M         A      87     Y        Y       Y
4       F         A      58     Y        N       N
7       M         C      76     N        Y       Y
9       M         A      77     Y        Y       N
10      M         A     104     N        N       Y
14      M         A      68     Y        N       N
15      M         C      79     N        Y       Y
18      F         A      63     Y        N       N
19      M         C      57     N        N       Y
20      M         A      84     Y        N       Y
21      M         A      72     Y        Y       Y
23      F         C      65     N        N       N
27      F         C      56     Y        N       Y
```

Create a vector of the indices of rows for people who recover.

```
RECOVERindices <- which(patients$RECOVER == "Y")
RECOVERindices
```

```
[1]  1  2  3  6  7 10 15 16 19 20 21 22 24 25 26 27
```

Again, we take a look:

```
patients[RECOVERindices,]
```

```
   GENDER   TREATMENT WEIGHT SMOKE EXERCISE RECOVER
1       M         A      79     Y        Y       Y
2       M         A      87     Y        Y       Y
3       F         B      65     N        N       Y
6       M         B      95     N        N       Y
7       M         C      76     N        Y       Y
10      M         A     104     N        N       Y
15      M         C      79     N        Y       Y
16      M         B     125     Y        N       Y
19      M         C      57     N        N       Y
20      M         A      84     Y        N       Y
21      M         A      72     Y        Y       Y
22      F         B      68     N        N       Y
24      F         B      64     N        N       Y
25      M         B      87     Y        N       Y
26      M         B      92     N        Y       Y
27      F         C      56     Y        N       Y
```

The intersection of both sets (i.e. people receiving treatment B and who recover) is:

```
TREATMENTB_RECOVER <- intersect(TREATMENTBindices, RECOVERindices)
TREATMENTB_RECOVER
```

```
[1]  3  6 16 22 24 25 26
```

A quick look at the table.

```
patients[TREATMENTB_RECOVER,]
```

```
   GENDER   TREATMENT WEIGHT SMOKE EXERCISE RECOVER
3       F         B      65     N        N       Y
6       M         B      95     N        N       Y
16      M         B     125     Y        N       Y
22      F         B      68     N        N       Y
24      F         B      64     N        N       Y
25      M         B      87     Y        N       Y
26      M         B      92     N        Y       Y
```

Let's perform a slightly more complex subset in several steps. We subset for female smokers not receiving treatment B who recover. We begin by setting up four vectors of indices.

```
femaleindices <- which(patients$GENDER == "F")

smokerindices <- which(patients$SMOKE == "Y")

treatmentnotBindices <- which(!patients$TREATMENT == "B")

recoverindices <- which(patients$RECOVER == "Y")
```

Then we find the intersections of pairs of vectors.

```
female_smokers <- intersect(femaleindices, smokerindices)

treatmentnotB_recover <- intersect(treatmentnotBindices,
recoverindices)
```

Finally, we find the intersection of the two intersections!

```
female_smokers_treatmentnotB_recover <- intersect(female_smokers,
treatmentnotB_recover)

female_smokers_treatmentnotB_recover
[1] 27
```

Only one female smoker not receiving treatment B recovered. Let's look at the table.

```
patients[female_smokers_treatmentnotB_recover,]
```

```
   GENDER    TREATMENT WEIGHT SMOKE EXERCISE RECOVER
27      F          C        56     Y       N       Y
```