Matrices - Doing Calculations

Let's look at some examples involving calculations on matrices:

```
set.seed(123)

X = matrix(rpois(9,1.5),nrow=3)

colnames(X) = c("aspirin","paracetamol","nurofen")

rownames(X) = paste("Trial",1:3,sep=".")

X
        aspirin paracetamol nurofen
Trial.1       1           3       1
Trial.2       2           4       3
Trial.3       1           0       1

mean(X[,3])  # Mean of the 3rd column
[1] 1.666667

var(X[3,]) # Variance of the 3rd row
[1] 0.3333333
```

Matrices - Doing Calculations

Let's look at some examples involving calculations on matrices. But there are some general functions to help with this kind of thing:

```
X
        aspirin paracetamol nurofen
Trial.1       1           3       1
Trial.2       2           4       3
Trial.3       1           0       1

rowSums(X)
Trial.1 Trial.2 Trial.3
      5       9       2

colSums(X)
aspirin paracetamol     nurofen
   4           7           5
```

Maybe columns represent protein expression and you are trying to determine if there are differences between the mean expression levels.

The R Book - Michael J. Crawley

Matrices - Doing Calculations

But there are some general functions to help with this kind of thing:

```
rowMeans(X)
  Trial.1   Trial.2   Trial.3
1.6666667 3.0000000 0.6666667

colMeans(X)
    aspirin paracetamol     nurofen
   1.333333    2.333333    1.666667

colMeans(X)[3]
 nurofen
1.666667
```

These are fast and can work on very large matrices. Though be careful if you have missing values in your data.

The R Book - Michael J. Crawley

Matrices - Doing Calculations - apply

Its worth pointing out that you can do similar things with the apply function. It allows you to plug in any function - not just the mean function.

```
X
        aspirin paracetamol nurofen
Trial.1       3           1       3
Trial.2       2           2       2
Trial.3       2           0       5

apply(X,1,summary)                 # 1 is for rows
        Trial.1 Trial.2 Trial.3
Min.      1.000     2.0  0.0000
1st Qu.   1.000     2.5  0.5000
Median    1.000     3.0  1.0000
Mean      1.667     3.0  0.6667
3rd Qu.   2.000     3.5  1.0000
Max.      3.000     4.0  1.0000

apply(X,2,summary)                  # 2 is for columns
        aspirin paracetamol nurofen
Min.      1.000       0.000   1.000
1st Qu.   1.000       1.500   1.000
Median    1.000       3.000   1.000
Mean      1.333       2.333   1.667
3rd Qu.   1.500       3.500   2.000
Max.      2.000       4.000   3.000
```

The R Book - Michael J. Crawley

Matrices - Doing Calculations - apply

Its worth pointing out that you can do similar things with the apply function. It allows you to plug in any function - not just the mean function.

```
apply(X,1,mean)
  Trial.1   Trial.2   Trial.3
2.0000000 2.3333333 0.3333333

# This is equivalent to:

rowMeans(X)
  Trial.1   Trial.2   Trial.3
2.0000000 2.3333333 0.3333333

# Let's "scale"/"center" the values in the rows. We subtract each value from the mean of its row

apply(X,1,function(x) (x-mean(x)))    # Scale the values in the rows

            Trial.1    Trial.2    Trial.3
aspirin          -1  1.6666667  0.6666667
paracetamol       0 -0.3333333 -0.3333333
nurofen           1 -1.3333333 -0.3333333
```

Matrices - Doing Calculations - apply

Let's find what rows have values greater than 2. Let's also find the row that has the largest number of values greater than 2. Sound hard ? Not really.

```
X > 2
        aspirin paracetamol nurofen
Trial.1   FALSE       FALSE    TRUE
Trial.2    TRUE       FALSE   FALSE
Trial.3   FALSE       FALSE   FALSE

apply(X > 2, 1, sum)            # Its a tie it seems
Trial.1 Trial.2 Trial.3
      1       1       0

max(apply(X > 2, 1, sum))
[1] 1

which(apply(X > 2,1,sum) == max(X>2) )
Trial.1 Trial.2
      1       2
```

This works because we can sum TRUE and FALSE values since R gives a value of "1" and "0" respectively.

```
as.numeric(TRUE)
[1] 1

as.numeric(FALSE)
[1] 0
```

Matrices - Linear Algebra

R supports common linear algebra operations also.

```
A = matrix(c(1,3,2,2,8,9),3,2)
A
     [,1] [,2]
[1,]    1    2
[2,]    3    8
[3,]    2    9
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 8 \\ 2 & 9 \end{bmatrix}^{\top} = \begin{bmatrix} 1 & 3 & 2 \\ 2 & 8 & 9 \end{bmatrix}$$

```
t(A)
     [,1] [,2] [,3]
[1,]    1    3    2
[2,]    2    8    9
```

http://bendixcarstensen.com/APC/linalg-notes-BxC.pdf

Matrices - Linear Algebra

```
A
     [,1] [,2]
[1,]    1    2
[2,]    3    8
[3,]    2    9

B = matrix(c(5,8,4,2),2,2)

A %*% B
     [,1] [,2]
[1,]   21    8
[2,]   79   28
[3,]   82   26
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 8 \\ 2 & 9 \end{bmatrix} \begin{bmatrix} 5 & 4 \\ 8 & 2 \end{bmatrix} = \left[\begin{bmatrix} 1 & 2 \\ 3 & 8 \\ 2 & 9 \end{bmatrix} \begin{bmatrix} 5 \\ 8 \end{bmatrix} : \begin{bmatrix} 1 & 2 \\ 3 & 8 \\ 2 & 9 \end{bmatrix} \begin{bmatrix} 4 \\ 2 \end{bmatrix} \right]$$

$$= \begin{bmatrix} 1 \cdot 5 + 2 \cdot 8 & 1 \cdot 4 + 2 \cdot 2 \\ 3 \cdot 5 + 8 \cdot 8 & 3 \cdot 4 + 8 \cdot 2 \\ 2 \cdot 5 + 9 \cdot 8 & 2 \cdot 4 + 9 \cdot 2 \end{bmatrix} = \begin{bmatrix} 21 & 8 \\ 79 & 28 \\ 82 & 26 \end{bmatrix}$$

http://bendixcarstensen.com/APC/linalg-notes-BxC.pdf

Matrices - Linear Algebra

The inverse of a n x n matrix A is the matrix B (which is also n x n) that when multiplied by A gives the identity matrix.

```
A = matrix(1:4,2,2)
A
     [,1] [,2]
[1,]    1    3
[2,]    2    4

B = solve(A)

B
     [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5

A %*% B             # We get the identity matrix
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

http://bendixcarstensen.com/APC/linalg-notes-BxC.pdf

Matrices - Linear Algebra

Suppose you have the following system of equations. This can be represented as:

$$\begin{aligned} x_1 + 3x_2 &= 7 \\ 2x_1 + 4x_2 &= 10 \end{aligned}$$

$$\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 7 \\ 10 \end{bmatrix} \quad \text{i.e. } Ax = b$$

```
A
     [,1] [,2]
[1,]    1    3
[2,]    2    4

b = c(7,10)
x = solve(A) %*% b
x
     [,1]
[1,]    1
[2,]    2
```

Since $A^{-1}A = I$ and since $Ix = x$ we have

$$x = A^{-1}b = \begin{bmatrix} -2 & 1.5 \\ 1 & -0.5 \end{bmatrix} \begin{bmatrix} 7 \\ 10 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

http://bendixcarstensen.com/APC/linalg-notes-BxC.pdf

Lists - Intro

* Lists address the situation where we need to store information of different types in a single structure.

* Remember that vectors and matrices restrict us to only one data type at a time.

* Many functions in R return lists.

Lists - Creating

```
family1 = list(husband="Fred", wife="Wilma", numofchildren=3, agesofkids=c(8,11,14))

length(family1)  # Has 4 elements
[1] 4

family1
$husband
[1] "Fred"

$wife
[1] "Wilma"

$numofchildren
[1] 3

$agesofkids
[1]  8 11 14

str(family1)
List of 4
 $ husband      : chr "Fred"
 $ wife         : chr "Wilma"
 $ numofchildren: num 3
 $ agesofkids   : num [1:3] 8 11 14
```

Lists - Indexing

```
family1 = list(husband="Fred", wife="Wilma", numofchildren=3, agesofkids=c(8,11,14))

family1$agesofkids    # If the list elements have names then use "$" to access the element
[1]  8 11 14

family1$agesofkids[1:2]
[1]  8 11

sapply(family1,class)
      husband          wife numofchildren    agesofkids
  "character"   "character"     "numeric"     "numeric"

sapply(family1,length)
      husband          wife numofchildren    agesofkids

            1             1             1             3
```

If the list elements have no names then you have to use numeric indexing

```
family2 = list("Barney","Betty",2,c(4,6))

[[1]]
[1] "Barney"

[[2]]
[1] "Betty"

[[3]]
[1] 2

[[4]]
[1] 4 6
```

Lists - Indexing

If the list elements have no names then you have to use numeric indexing. But try to create lists with names as its easier to work with.

```
family2 = list("Barney","Betty",2,c(4,6))

family2[4]    # Accesses the 4th index and associated element
[[1]]
[1] 4 6

family2[[4]]   # Accesses the 4th element value only - more direct
[1] 4 6


family2[3:4]   # Get 3rd and 4th indices and associate values
[[1]]
[1] 2

[[2]]
[1] 4 6
```

Lists - Indexing

You can do "unlist" on any list to turn it into a vector. Since the list has mixed data types all of the elements of the vector will be converted to a single data type. In this case character

```
unlist(family1)
      husband          wife numofchildren   agesofkids1   agesofkids2
       "Fred"       "Wilma"           "3"           "8"          "11"
  agesofkids3
         "14"


as.numeric(unlist(family1))
[1] NA NA  3  8 11 14
```

Normally we don't create lists as a "standalone" object except in two major cases:

1) We are writing a function that does some interesting stuff and we want to return to the user a structure that has lots of information.

2) As a precursor to creating a a data frame which is a hybrid between a list and a matrix. We'll investigate this momentarily.

Lists - Functions

R has lots of statistical functions that return lists of information. In fact this is the norm.

```
data(mtcars)   # Load mtcars into the environment

mylm = lm(mpg ~ wt, data = mtcars)

print(mylm)

Call:
lm(formula = mpg ~ wt, data = mtcars)

Coefficients:
(Intercept)           wt
     37.285       -5.344

# But there is a lot more information

typeof(mylm)
[1] "list"
```

Lists - Functions

R has lots of statistical functions that return lists of information. In fact this is the norm.

```
str(mylm,give.attr=F)  # Lots of stuff here

List of 12
 $ coefficients : Named num [1:2] 37.29 -5.34
 $ residuals    : Named num [1:32] -2.28 -0.92 -2.09 1.3 -0.2 ...
 $ effects      : Named num [1:32] -113.65 -29.116 -1.661 1.631 0.111 ...
 $ rank         : int 2
 $ fitted.values: Named num [1:32] 23.3 21.9 24.9 20.1 18.9 ...
 $ assign       : int [1:2] 0 1
 $ qr           :List of 5
  ..$ qr   : num [1:32, 1:2] -5.657 0.177 0.177 0.177 0.177 ...
  ..$ qraux: num [1:2] 1.18 1.05
  ..$ pivot: int [1:2] 1 2
  ..$ tol  : num 1e-07
  ..$ rank : int 2
 $ df.residual  : int 30
 $ xlevels      : Named list()
 $ call         : language lm(formula = mpg ~ wt, data = mtcars)
 $ terms        :Classes 'terms', 'formula' length 3 mpg ~ wt
 $ model        :'data.frame':   32 obs. of  2 variables:
  ..$ mpg: num [1:32] 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
  ..$ wt : num [1:32] 2.62 2.88 2.32 3.21 3.44 ...
```

Lists - Functions

```
names(mylm)
 [1] "coefficients"  "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"        "qr"            "df.residual"
 [9] "xlevels"       "call"          "terms"         "model"

mylm$effects
 (Intercept)          wt
-113.6497374  -29.1157217   -1.6613339    1.6313943    0.1111305   -0.3840041
  -3.6072442    4.5003125    2.6905817    0.6111305   -0.7888695    1.1143917
   0.2316793   -1.6061571    1.3014525    2.2137818    6.0995633    7.3094734
   2.2421594    6.8956792   -2.2010595   -2.6694078   -3.4150859   -3.1915608
   2.7346556    0.8200064    0.5948771    1.7073457   -4.2045529   -2.4018616
  -2.9072442   -0.6494289

# Some use the $ notation to extract desired information they want straight from the function call

lm(mpg ~ wt, data = mtcars)$coefficients
(Intercept)          wt
  37.285126   -5.344472
```

Lists - Functions

When we create our own functions we can package things up into a list and return things.

```
my.summary <- function(x) {
      return.list = list()       # Declare the list

      return.list$mean = mean(x)
      return.list$sd = sd(x)
      return.list$var = var(x)

      return(return.list)
}

my.summary(1:10)

$mean
[1] 5.5

$sd
[1] 3.02765

$var
[1] 9.166667

names(my.summary(1:10))
[1] "mean" "sd"   "var"

my.summary(1:10)$var     # Here we exploit the $ notation to get only what we want
[1] 9.166667
```

Lists - Functions

Some other basic R functions will return a list - such as some of the character functions:

```
mystring = "This is a test"

mys = strsplit(mystring, " ")

str(mys)
List of 1
 $ : chr [1:4] "This" "is" "a" "test"

mys
[[1]]
[1] "This" "is"   "a"    "test"

mys[[1]][1]
[1] "This"

mys[[1]][1:2]
[1] "This" "is"

unlist(mys)
[1] "This" "is"   "a"    "test"
```

Lists - Twitter

```
delta.tweets = searchTwitter('@delta', n = 100)  # Uses the add-on twitteR package

class(delta.tweets)
[1] "list"

delta.tweets
[[1]]
[1] "sotsoy: Apparently if you use your frequent flier miles on @delta they stick you at the back
of the plane on every flight next to the bathroom"

[[2]]
[1] "ImTooNonFiction: My @Delta flight has been delayed for the last 2 hrs. We've been on plane at
gate for 2+ hours and no mention of a voucher or compensation"

[[3]]
[1] "ShaneNHara: @Delta and @DeltaAssist, thank you for a swift boarding process here at SEA en
route to LAX. Taking care of your loyal flyers = appreciated."

[[4]]
[1] "NaiiOLLG: RT @TheRealNickMara: ThankYou @Delta for a great flight!! #Work!!!"

[[5]]
[1] "forbeslancaster: @bsideblog @Delta just saw a commercial highlighting delta awesome service.
Totes NOT true"

..
```

Lists - Twitter

```
sapply(delta.tweets,function(x) x$getText())  # Pulls out the text of the tweet

[1] "Apparently if you use your frequent flier miles on @delta they stick you at the back of the
plane on every flight next to the bathroom"

[2] "My @Delta flight has been delayed for the last 2 hrs. We've been on plane at gate for 2+
hours and no mention of a voucher or compensation"

[3] "@Delta and @DeltaAssist, thank you for a swift boarding process here at SEA en route to LAX.
Taking care of your loyal flyers = appreciated."

[4] "RT @TheRealNickMara: ThankYou @Delta for a great flight!! #Work!!!"

[5] "@bsideblog @Delta just saw a commercial highlighting delta awesome service. Totes NOT true"

..
..
..
other results omitted due to obscenities...
```

Lists - sapply

Lastly, While we could use sapply to apply some statistical function across all elements of a list it might not make sense since you have different data types:

```
sapply(family1,mean)
      husband          wife numofchildren    agesofkids 
           NA            NA             3            11 
Warning messages:
1: In mean.default(X[[1L]], ...) :
  argument is not numeric or logical: returning NA
2: In mean.default(X[[2L]], ...) :
  argument is not numeric or logical: returning NA
```

We could write our own function to ignore non-numeric data:

```
sapply(family1, function(x) { if (is.numeric(x)) print(mean(x))})
[1] 3
[1] 11
$husband
NULL
$wife
NULL

$numofchildren
[1] 3

$agesofkids
[1] 11
```

Factors - Intro

R supports factors, which are a special data type for, among other things, managing categories of data.

"One of the most important uses of factors is in statistical modeling; since categorical variables enter into statistical models differently than continuous variables, storing data as factors insures that the modeling functions will treat such data correctly".

Identifying categorical variables is usually straightforward. These are the variables by which you might want to summarize some continuous data.

Categorical variables usually take on a definite number of values.

Factors - Intro

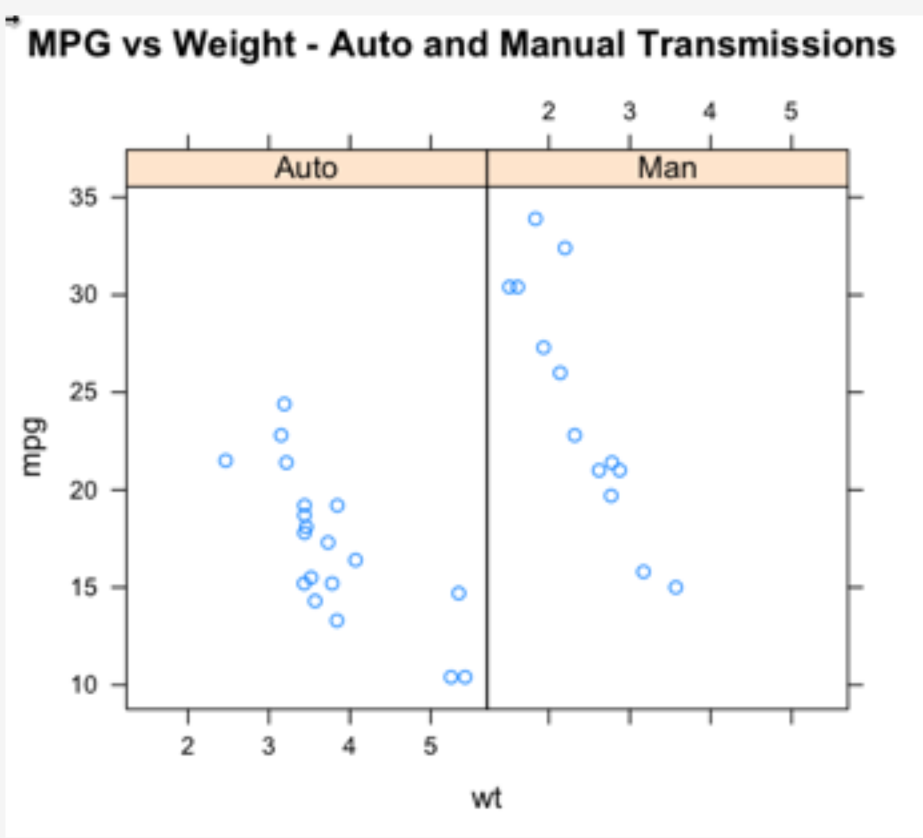
Let's say we have some automobile data that tells us if a car has an automatic transmission (0) or a manual transmission (1). We store this into a vector called transvec

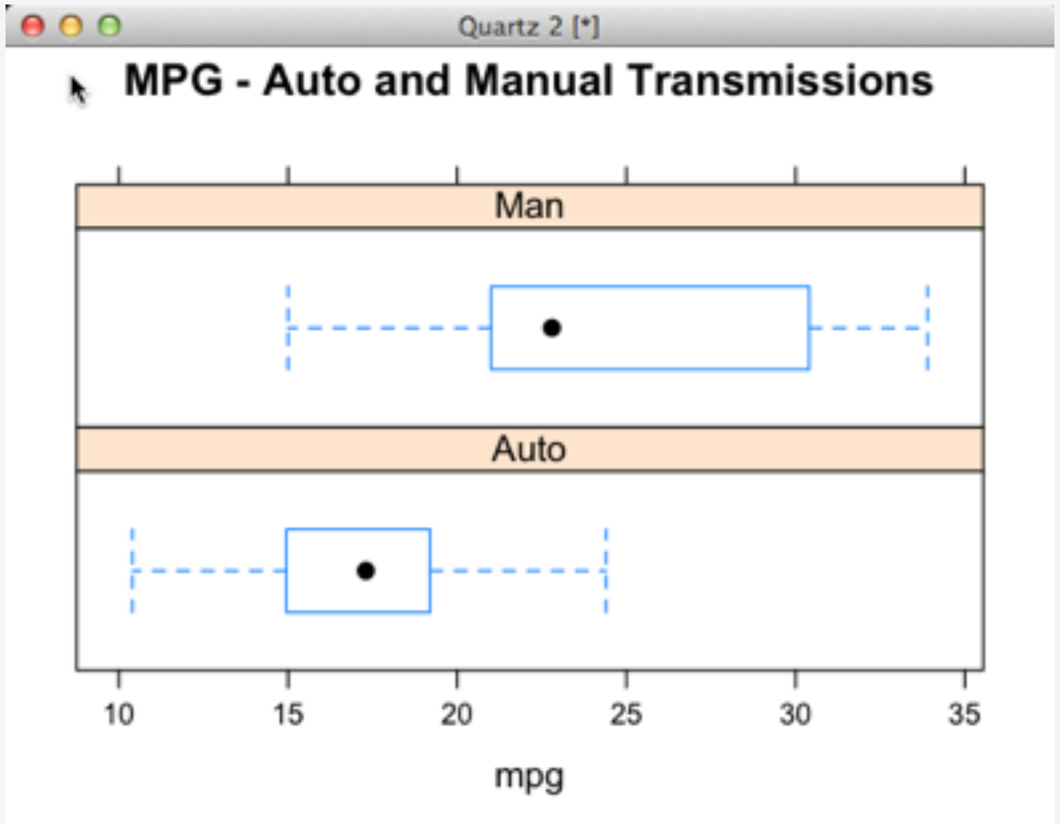
```
transvec = c(1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,1,1,1,1,1,1,1)

table(transvec)    # Count 'em up. Which are Auto and Manual ?
transvec
 0  1
19 13

mytransfac = factor(transvec, levels = c(0,1), labels = c("Auto","Man") )

table(mytransfac)
mytransfac
Auto  Man
  19   13

levels(mytransfac)
[1] "Auto" "Man"

mytransfac
[1] Man  Man  Man  Auto Auto Auto Auto Auto Auto Auto Auto Auto Auto Auto Auto
[16] Auto Auto Man  Man  Man  Auto Auto Auto Auto Auto Man  Man  Man  Man  Man
[31] Man  Man
Levels: Auto Man
```

Factors - Intro

R knows how to handle factors when doing plots. Here were get an X/Y plot and a Box Plot with very little work since R knows that mytransfac is a factor

```
library(lattice)
xyplot(mpg~wt | mytransfac, mtcars, main="MPG vs Weight - Auto and Manual Transmissions")

bwplot(~mpg|mytransfac, mtcars, main="MPG - Auto and Manual Transmissions",layout=c(1,2))
```





Factors - Aggregation Preview

With our knowledge of factors and vectors we can do some basic aggregation using the tapply command.

We have a factor vector called mytransfac. Let's summarize some MPG data that corresponds to the automobiles used in the mytransfac vector. So for each car we have its MPG figure and whether it has an automatic or manual transmission.

```
mympg = c(21,21,22.8,21.4,18.7,18.1,14.3,24.4,22.8,19.2,17.8,16.4,17.3,15.2,10.4,
        10.4,14.7,32.4,30.4,33.9,21.5,15.5,15.2,13.3,19.2,27.3,26,30.4,15.8,19.7,15,21.4)


tapply( continuous_value_to_summarize,  factor_or_grouping variable,  function_for_summary)

tapply(mympg,mytransfac,mean)
    Auto      Man
17.14737 24.39231
```

Factors - cut

It is sometimes useful to take a continuous variable and chop it up into intervals or categories for purposes of summary or grouping. R has a function to do this called “cut” to accomplish this. Let’s work through some examples to understand what is going on:

Let’s cut up the numbers between 1 and 10 into 4 intervals. It looks kind of messy:

```
cut(0:10,breaks=4)

[1] (-0.01,2.5] (-0.01,2.5] (-0.01,2.5] (2.5,5]     (2.5,5]     (2.5,5]     (5,7.5]     (5,7.5]
(7.5,10]    (7.5,10]
[11] (7.5,10]
Levels: (-0.01,2.5] (2.5,5] (5,7.5] (7.5,10]

table(cut(0:10,breaks=4))

(-0.01,2.5]     (2.5,5]     (5,7.5]    (7.5,10]
          3           3           2           3
```

Factors - cut

Well that was cool but people like to read labels:

```
my.cut = cut(0:10,breaks=4,labels=c("Q1","Q2","Q3","Q4"))
[1] Q1 Q1 Q1 Q2 Q2 Q2 Q3 Q3 Q4 Q4 Q4
Levels: Q1 Q2 Q3 Q4


table(my.cut)
my.cut
Q1 Q2 Q3 Q4
 3  3  2  3
```

Factors - cut

But you can to take finer-grained control over how the intervals are made.

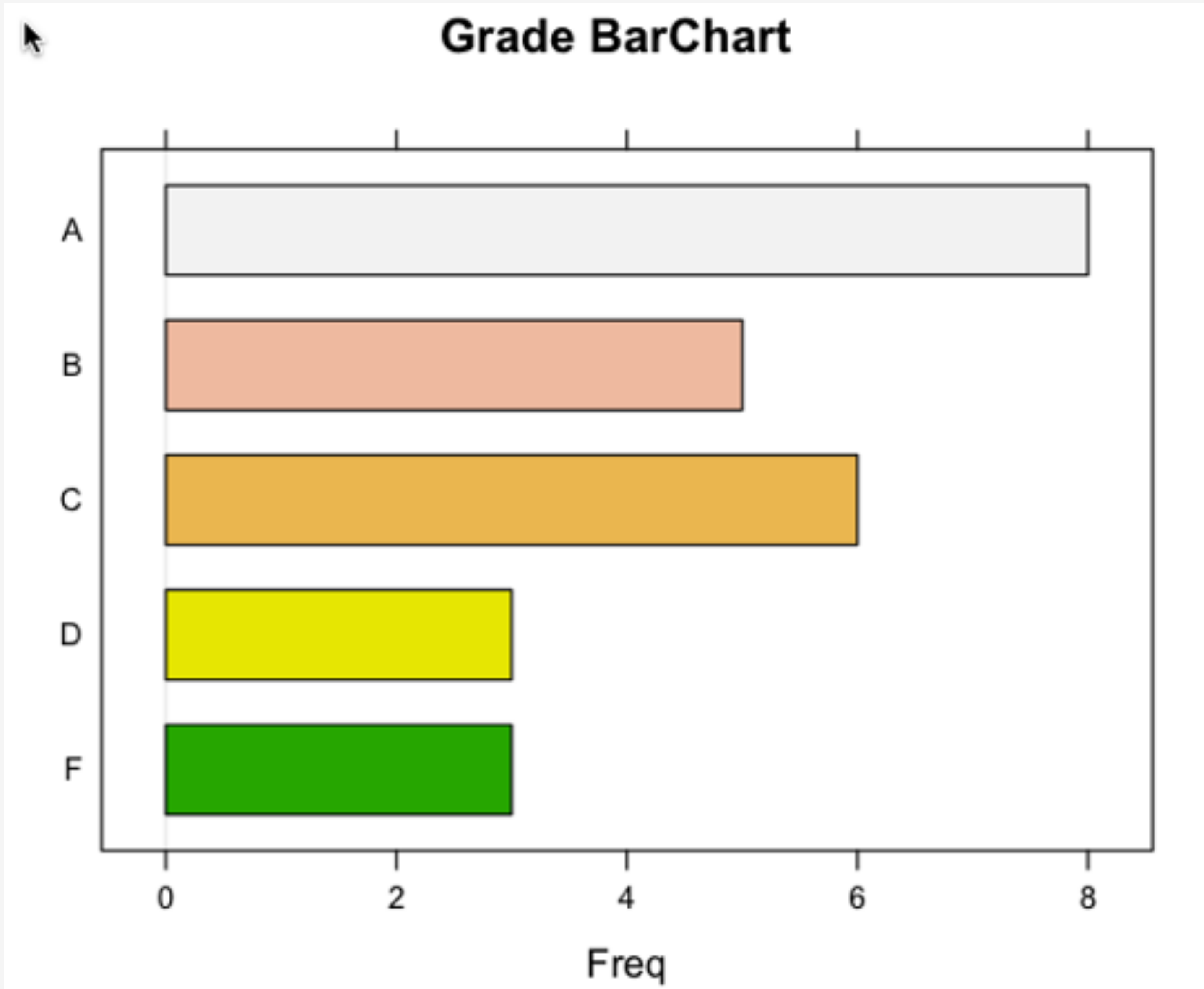
```
quantile(0:10)
  0%  25%  50%  75% 100%
 0.0  2.5  5.0  7.5 10.0

table(cut(0:10,breaks=quantile(0:10),include.lowest=TRUE))

 [0,2.5]  (2.5,5]  (5,7.5] (7.5,10]
       3        3        2        3
```

Factors - cut

Another example. Let's say we have some exam scores. Let's summarize them according to the typical US grading system. F: < 60, D: 60-70: C: 70-80: B:80-90 A:90-100

```
set.seed(123)
exam.score = runif(25,50,100)

cut(exam.score,breaks=c(50,60,70,80,90,100))
 [1] (60,70]  (80,90]  (70,80]  (90,100] (90,100] (50,60]  (70,80]  (90,100]
 [9] (70,80]  (70,80]  (90,100] (70,80]  (80,90]  (70,80]  (50,60]  (90,100]
[17] (60,70]  (50,60]  (60,70]  (90,100] (90,100] (80,90]  (80,90]  (90,100]
[25] (80,90]
Levels: (50,60] (60,70] (70,80] (80,90] (90,100]

cut(exam.score,breaks=c(50,60,70,80,90,100),labels=c("F","D","C","B","A"))
 [1] D B C A A F C A C C A C B C F A D F D A A B B A B
Levels: F D C B A

my.table = table(cut(exam.score,breaks=c(50,60,70,80,90,100),labels=c("F","D","C","B","A")))

F D C B A
3 3 6 5 8

barchart(my.table,main="Grade BarChart",col=terrain.colors(5))
```

Factors - cut



Factors - cut

We have a small problem in that the intervals don't exactly match the grading scheme. In this scheme someone getting a grade of 90 will get a B although we intend for them to get an A. This is where you should be paying attention to the (and] characters. To make the interval exclude the "right side" of the interval we specify the "right=F" argument.

```
cut(exam.score,breaks=c(50,60,70,80,90,100))
 [1] (60,70]  (80,90]  (70,80]  (90,100] (90,100] (50,60]  (70,80]  (90,100]
 [9] (70,80]  (70,80]  (90,100] (70,80]  (80,90]  (70,80]  (50,60]  (90,100]
[17] (60,70]  (50,60]  (60,70]  (90,100] (90,100] (80,90]  (80,90]  (90,100]
[25] (80,90]
Levels: (50,60] (60,70] (70,80] (80,90] (90,100]

cut(exam.score,breaks=c(50,60,70,80,90,100),right=F)
 [1] [60,70)  [80,90)  [70,80)  [90,100) [90,100) [50,60)  [70,80)  [90,100)
 [9] [70,80)  [70,80)  [90,100) [70,80)  [80,90)  [70,80)  [50,60)  [90,100)
[17] [60,70)  [50,60)  [60,70)  [90,100) [90,100) [80,90)  [80,90)  [90,100)
[25] [80,90)
Levels: [50,60) [60,70) [70,80) [80,90) [90,100)
```

Factors - cut

So if you don't think that the cut command doesn't do something interesting then here is how you would have had to the last example with the exams:

```
exam.score = runif(25,50,100)

acount = 0
bcount = 0
ccount = 0
dcount = 0
fcount = 0
exam.score = runif(25,50,100)
for (ii in 1:length(exam.score)) {

 if (exam.score[ii] < 60) {fcount = fcount + 1} else
   if ((exam.score[ii] >= 60) & (exam.score[ii] < 70)) {dcount = dcount + 1} else
     if ((exam.score[ii] >= 70) & (exam.score[ii] < 80)) {ccount = ccount +1} else
       if ((exam.score[ii] >= 80) & (exam.score[ii] < 90)) {bcount = bcount +1} else
          if ((exam.score[ii] >= 90) & (exam.score[ii] <= 100)) {acount = acount +1}
}
cat("acount bcount ccount dcount fcount")
cat(acount,bcount,ccount,dcount,fcount)
acount bcount ccount dcount fcount
8 5 7 3 2
```

Factors - Ordered

Sometimes we want our factors to be ordered. For example we intuitively know that January comes before February and so on. Can we get R to create ordered factors ?

```
mons =c("Jan","Feb","Mar","Apr","May","Jun","Jan","Feb","May","Jun", "Apr","Mar")

my.fact.mons = factor(mons)
 [1] Jan Feb Mar Apr May Jun Jan Feb May Jun Apr Mar
Levels: Apr Feb Jan Jun Mar May

my.fact.mons[1] < my.fact.mons[2]

Warning message:
In Ops.factor(my.fact.mons[1], my.fact.mons[2]) :
  < not meaningful for factors

levels(my.fact.mons)
[1] "Apr" "Feb" "Jan" "Jun" "Mar" "May"
```

http://www.stat.berkeley.edu/classes/s133/factors

Factors - Ordered

```
my.fact.mons = factor(mons, labels=c("Jan","Feb","Mar","Apr","May","Jun"),ordered=TRUE)

my.fact.mons
 [1] Mar Feb May Jan Jun Apr Mar Feb Jun Apr Jan May
Levels: Jan < Feb < Mar < Apr < May < Jun

my.fact.mons[1] < my.fact.mons[2]
[1] FALSE

table(my.fact.mons)
my.fact.mons
Jan Feb Mar Apr May Jun
  2   2   2   2   2   2

levels(my.fact.mons)                       # This is what we want !
[1] "Jan" "Feb" "Mar" "Apr" "May" "Jun"
```

http://www.stat.berkeley.edu/classes/s133/factors

Supplemental Factors - AOV example

Let's do an AOV on the mtcars data set variables MPG and number of gears the latter of which takes on the values 3,4,5. So it is well suited to be a factor.

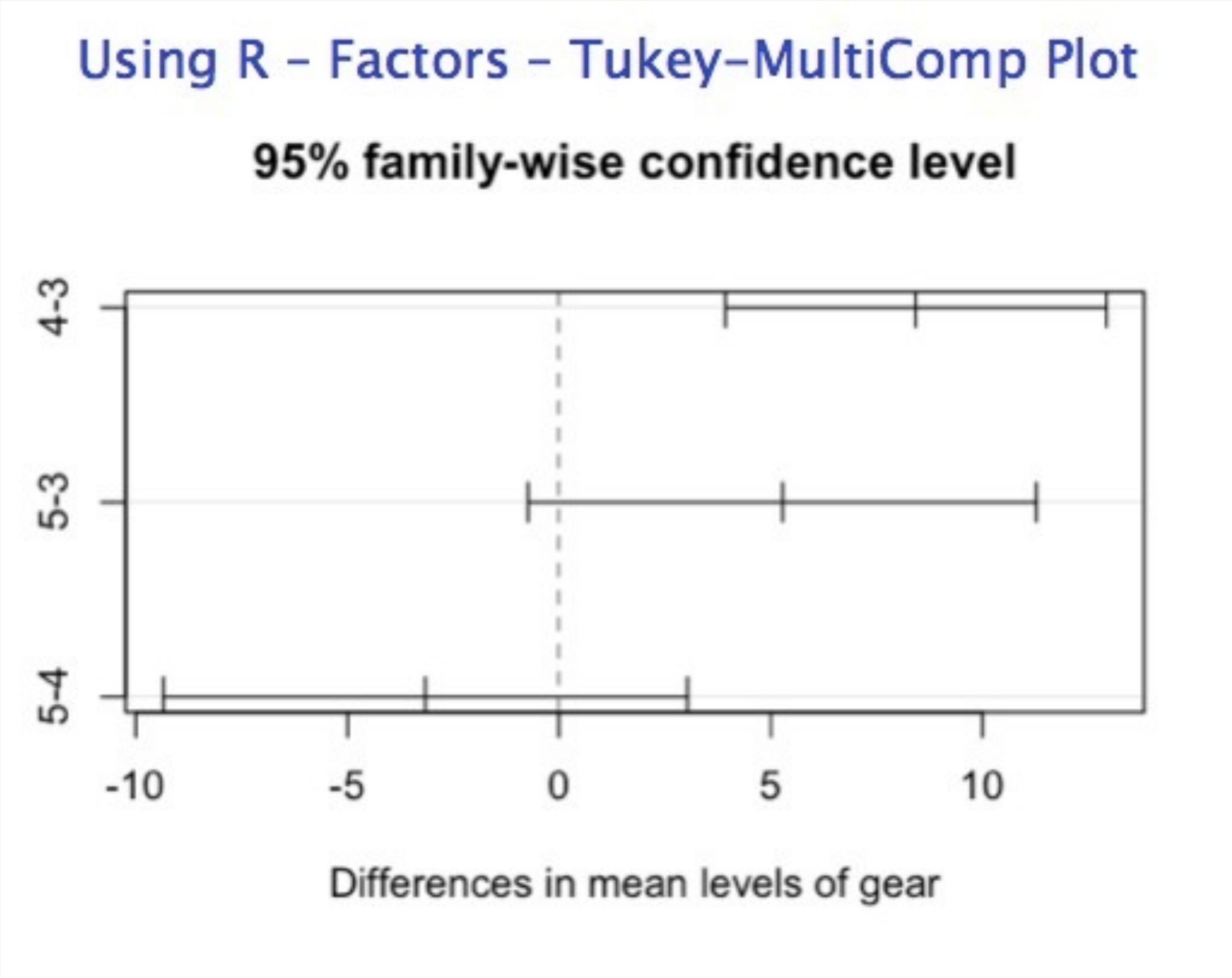
```
mtcars$gear = factor(mtcars$gear)  # Turn gear into a factor
aov.ex1 = aov(mpg ~ gear,mtcars)
summary(aov.ex1)
             Df Sum Sq Mean Sq F value    Pr(>F)
factor(gear)  2 483.24 241.622  10.901 0.0002948 ***
Residuals    29 642.80  22.166
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
> print(model.tables(aov.ex1,"means"))
Tables of means
Grand mean

20.09062

 gear
        3     4     5
    16.11 24.53 21.38
rep 15.00 12.00  5.00

par(mfrow=c(2,2))
plot(aov.ex1)
```

Supplemental Factors - AOV example

Let's do an AOV on the mtcars data set variables MPG and number of gears the latter of which takes on the values 3,4,5. So it is well suited to be a factor.

```
my.tukey = TukeyHSD(aov.ex1,"gear")  # Tukey Multiple Comparisons
my.tukey
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = mpg ~ gear, data = mtcars)

$gear
         diff        lwr       upr     p adj
4-3  8.426667  3.9234704 12.929863 0.0002088
5-3  5.273333 -0.7309284 11.277595 0.0937176
5-4 -3.153333 -9.3423846  3.035718 0.4295874

Differences between Gears are significant at 5% level if the confidence interval around the
estimation of the difference does not contain zero

plot(my.tukey)
```

Supplemental Factors - AOV example



Dataframes

Dataframes

A **data frame** is a special type of list that contains data in a format that allows for easier manipulation, reshaping, and open-ended analysis.

Data frames are tightly coupled collections of variables. It is one of the more important constructs you will encounter when using R so learn all you can about it.

A data frame is an analogue to the Excel spreadsheet. In general this is the most popular construct for storing, manipulating, and analyzing data.

Data frames can be constructed from existing vectors, lists, or matrices. Many times they are created by reading in comma delimited files, (CSV files), using the read.table command.

Once you become accustomed to working with data frames, R becomes so much easier to use.

Dataframes

Here we have 4 vectors two of which are character and two of which are numeric.
We could work with them in the following fashion if we wanted to do some type of summary on them.

```
names = c("P1","P2","P3","P4","P5")
temp = c(98.2,101.3,97.2,100.2,98.5)
pulse = c(66,72,83,85,90)
gender = c("M","F","M","M","F")

# We could write a for loop to get information for each patient but this isn’t # so convenient or scalable.

for (ii in 1:length(gender)) {
   print.string = c(names[ii],temp[ii],pulse[ii],gender[ii])
   print(print.string)
}

[1] "P1"   "98.2" "66"   "M"
[1] "P2"    "101.3" "72"    "F"
[1] "P3"   "97.2" "83"   "M"
[1] "P4"    "100.2" "85"    "M"
[1] "P5"   "98.5" "90"   "F"
```

Dataframes

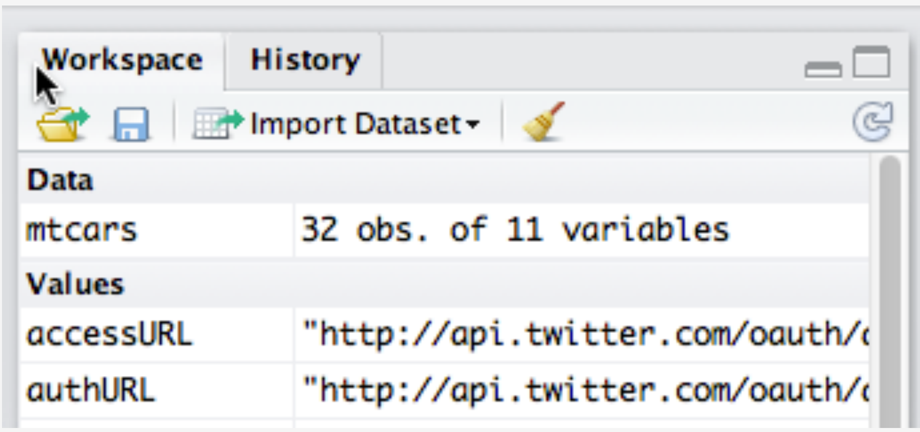
A data frame can be regarded as a matrix with columns possibly of differing modes and attributes. It may be displayed in matrix form, and its rows and columns extracted using matrix indexing conventions. Let's create a data frame:

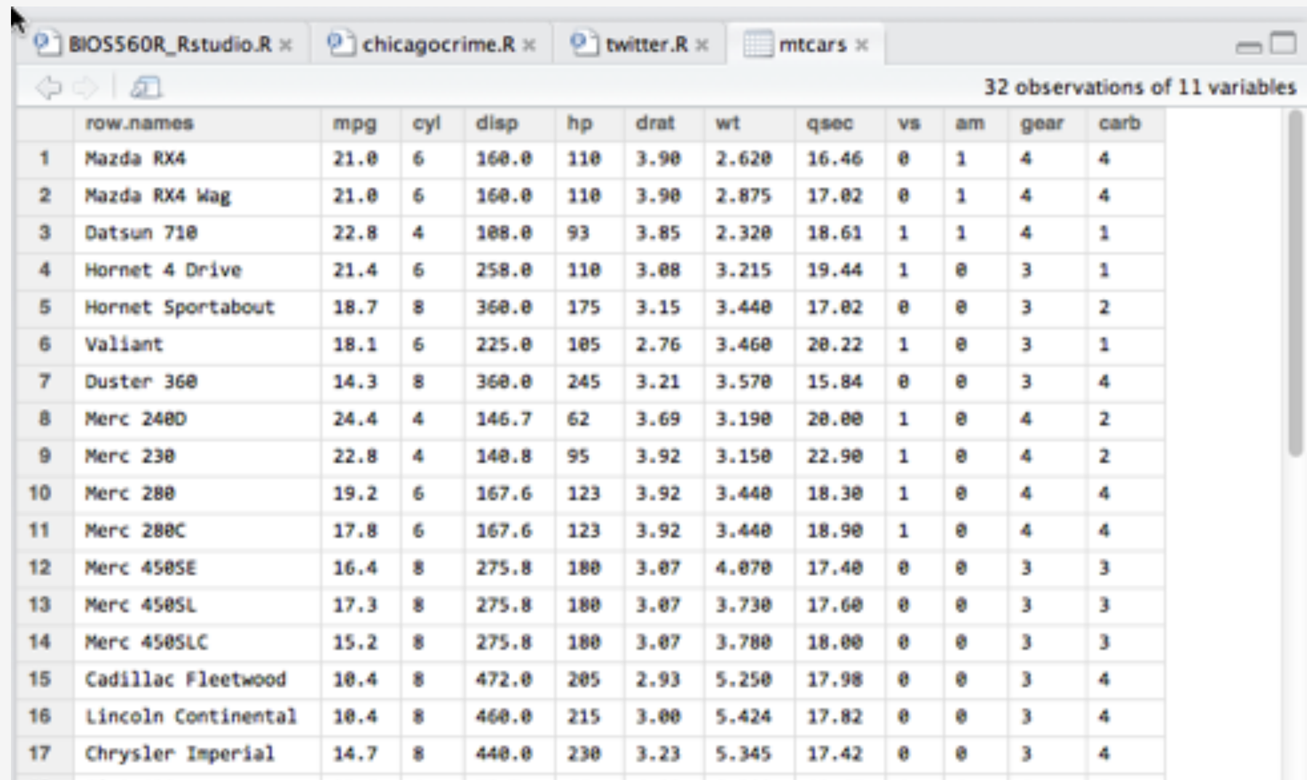
```
names=c("P1","P2","P3","P4","P5")
temp=c(98.2,101.3,97.2,100.2,98.5)
pulse=c(66,72,83,85,90)
gender=c("M","F","M","M","F")

my_df = data.frame(names,temp,pulse,gender) # Much more flexible
my_df

  names  temp pulse gender
1    P1  98.2    66      M
2    P2 101.3    72      F
3    P3  97.2    83      M
4    P4 100.2    85      M
5    P5  98.5    90      F

plot(my_df$pulse ~ my_df$temp,main="Pulse Rate",xlab="Patient",ylab="BPM")

mean(my_df[,2:3])
 temp pulse
99.08 79.20
```

Dataframes

Once you have the data frame you could edit it with a GUI editor. Or you can use the Workspace Viewer/ Editor in RStudio

```
data(mtcars)   # This will load a copy of mtcars into your workspace.
```





	row.names	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
1	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
2	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
3	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
4	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
5	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
6	Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
7	Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
8	Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
9	Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
10	Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
11	Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
12	Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
13	Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
14	Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
15	Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
16	Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
17	Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4

Dataframes

R comes with a variety of built-in data sets that are very useful for getting used to data sets and how to manipulate them.

```
library(help="datasets")

# Gives detailed descriptions on available data sets

AirPassengers          Monthly Airline Passenger Numbers 1949-1960
BJsales                Sales Data with Leading Indicator
BOD                    Biochemical Oxygen Demand
CO2                    Carbon Dioxide Uptake in Grass Plants
ChickWeight            Weight versus age of chicks on different diets
DNase                  Elisa assay of DNase
EuStockMarkets         Daily Closing Prices of Major European Stock
                       Indices, 1991-1998
Formaldehyde           Determination of Formaldehyde
HairEyeColor           Hair and Eye Color of Statistics Students

help(mtcars)    # Get details on a given data set
```

Dataframes

Let's focus on one of the built in sets. Its called "mtcars". The data was extracted from the 1974 _Motor Trend_ US magazine, and comprises fuel consumption and 11 aspects of automobile design and performance for 32 automobiles (1973-74 models).

```
        [, 1]  mpg   Miles/(US) gallon
[, 2]  cyl   Number of cylinders
[, 3]  disp  Displacement (cu.in.)
[, 4]  hp    Gross horsepower
[, 5]  drat  Rear axle ratio
[, 6]  wt    Weight (lb/1000)
[, 7]  qsec  1/4 mile time
[, 8]  vs    V/S
[, 9]  am    Transmission (0 = automatic, 1 = manual)
[,10]  gear  Number of forward gears
[,11]  carb  Number of carburetors
```

```
# One way to get the type of each column

> sapply(mtcars, class)
      mpg       cyl      disp        hp      drat        wt      qsec        vs 
"numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" "numeric" 
       am      gear      carb 
"numeric" "numeric" "numeric" 
```

Dataframes

The data was extracted from the 1974 _Motor Trend_ US magazine, and comprises fuel consumption and 11 aspects of automobile design and performance for 32 automobiles (1973-74 models).

```
str(mtcars)
'data.frame':    32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...

nrow(mtcars)     # How many rows does it have ?
[1] 32

ncol(mtcars)     # How many columns are there ?
[1] 11
```

Dataframes

```
rownames(mtcars)
 [1] "Mazda RX4"           "Mazda RX4 Wag"       "Datsun 710"
 [4] "Hornet 4 Drive"      "Hornet Sportabout"   "Valiant"
 ..
[19] "Honda Civic"         "Toyota Corolla"      "Toyota Corona"
[22] "Dodge Challenger"    "AMC Javelin"         "Camaro Z28"
[25] "Pontiac Firebird"    "Fiat X1-9"           "Porsche 914-2"
[28] "Lotus Europa"        "Ford Pantera L"      "Ferrari Dino"
[31] "Maserati Bora"       "Volvo 142E"


rownames(mtcars) = 1:32

head(mtcars)
   mpg cyl disp  hp drat   wt qsec vs transmission gear carb
1 21.0   6  160 110 3.90 2.62 16.5  0            1    4    4
2 21.0   6  160 110 3.90 2.88 17.0  0            1    4    4

rownames(mtcars) = paste("car",1:32,sep="_")
head(mtcars)
       mpg cyl disp  hp drat   wt qsec vs transmission gear carb
car_1 21.0   6  160 110 3.90 2.62 16.5  0            1    4    4
car_2 21.0   6  160 110 3.90 2.88 17.0  0            1    4    4
car_3 22.8   4  108  93 3.85 2.32 18.6  1            1    4    1
```

Dataframes

There are various ways to **select, remove, or exclude** rows and columns from a data frame.

```
mtcars[,-11]
                      mpg cyl disp  hp drat    wt  qsec vs am gear
Mazda RX4            21.0   6  160 110 3.90 2.620 16.46  0  1    4
Mazda RX4 Wag        21.0   6  160 110 3.90 2.875 17.02  0  1    4
Datsun 710           22.8   4  108  93 3.85 2.320 18.61  1  1    4

mtcars      # Notice that carb is included
                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1

mtcars[,-3:-5]   # Print all columns except for columns 3 through 5
                     mpg cyl    wt  qsec vs am gear      carb
Mazda RX4           21.0   6 2.620 16.46  0  1    4 0.6020600
Mazda RX4 Wag       21.0   6 2.875 17.02  0  1    4 0.6020600
Datsun 710          22.8   4 2.320 18.61  1  1    4 0.0000000

mtcars[,c(-3,-5)] # Print all columns except for colums 3 AND 5
                     mpg cyl  hp    wt  qsec vs am gear      carb
Mazda RX4           21.0   6 110 2.620 16.46  0  1    4 0.6020600
Mazda RX4 Wag       21.0   6 110 2.875 17.02  0  1    4 0.6020600
Datsun 710          22.8   4  93 2.320 18.61  1  1    4 0.0000000
```

Dataframes

There are various ways to **select, remove, or exclude** rows and columns from a data frame.

```
mtcars[mtcars$mpg >= 30.0,]
                mpg cyl disp  hp drat    wt  qsec vs am gear carb
Fiat 128       32.4   4 78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic    30.4   4 75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla 33.9   4 71.1  65 4.22 1.835 19.90  1  1    4    1
Lotus Europa   30.4   4 95.1 113 3.77 1.513 16.90  1  1    5    2

mtcars[mtcars$mpg >= 30.0,2:6]

                mpg cyl disp  hp drat
Fiat 128       32.4   4 78.7  66 4.08
Honda Civic    30.4   4 75.7  52 4.93
Toyota Corolla 33.9   4 71.1  65 4.22
Lotus Europa   30.4   4 95.1 113 3.77

mtcars[mtcars$mpg >= 30.0 & mtcars$cyl < 6,]
                mpg cyl disp  hp drat    wt  qsec vs am gear carb
Fiat 128       32.4   4 78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic    30.4   4 75.7  52 4.93 1.615 18.52  1  1    4    2
Toyota Corolla 33.9   4 71.1  65 4.22 1.835 19.90  1  1    4    1
Lotus Europa   30.4   4 95.1 113 3.77 1.513 16.90  1  1    5    2
```

Dataframes

Find all rows that correspond to Automatic and Count them

```
mtcars[mtcars$am==0,]
                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
..
..

nrow(mtcars[mtcars$am == 0,])
[1] 19

nrow(mtcars[mtcars$am == 1,])
[1] 13
```

Dataframes

Many times data will be read in from a comma delimited ,("CSV"), file exported from Excel. The file can be read from local storage or from the Web.

```
url = "http://www.bimcore.emory.edu/BIOS560R/DATA.DIR/hsb2.csv"

data1 = read.table(url,header=T,sep=",")

head(data1)
  gender  id race ses schtyp  prgtype read write math science socst
1      0  70    4   1      1  general   57    52   41      47    57
2      1 121    4   2      1   vocati   68    59   53      63    61
3      0  86    4   3      1  general   44    33   54      58    31
4      0 141    4   3      1   vocati   63    44   47      53    56
5      0 172    4   2      1 academic   47    52   57      53    61
6      0 113    4   2      1 academic   44    52   51      63    61


sapply(data1,class)    # Applies the "Class" function to all columns
 gender        id      race       ses    schtyp   prgtype      read     write
"integer" "integer" "integer" "integer" "integer"  "factor" "integer" "integer"

     math   science     socst
"integer" "integer" "integer"
```

Dataframes

Or you can use the "colClasses" argument when calling read.table() which allows you to set the variable type as you read in the data. It takes a bit of work up front but is worth it since it requires you to think about what you want/need your variable types. You can always change the types after the fact as in the previous example.

```
myclasses = c("character","integer","integer",
                "integer","character","factor","integer","integer",
                "integer","integer","numeric")

data1 = read.table("http://www.bimcore.emory.edu/BIOS560R/DATA.DIR/hsb2.csv",
                      header=T,
                      sep=",",
                      colClasses = myclasses)

sapply(data1,class)
     gender          id        race         ses      schtyp     prgtype
"character"   "integer"   "integer"   "integer" "character"    "factor"

       read       write        math     science       socst
  "integer"   "integer"   "integer"   "integer"   "numeric"
```

Dataframes

Back to the mtcars data frame. What columns appear to be candidates for a factor ?
It would be variables who have only "a few" number of different values. If we do something like this we can get an idea. Looks like the last 4 columns might be what they want.

```
str(mtcars)
'data.frame':  32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...

unique(mtcars$am)   # Tells us what the unique values are
[1] 1 0
```

Dataframes

Back to the mtcars data frame. What columns appear to be candidates for a factor ?
It would be variables who have only "a few" number of different values. If we do something like this we can get an idea. Looks like the last 4 columns might be what they want.

```
sapply(mtcars[,8:11], unique) # applies the unique function to columns 8-11 inclusive

$vs
[1] 0 1

$am
[1] 1 0

$gear
[1] 4 3 5

$carb
[1] 4 1 2 3 6 8
```

Dataframes

```
mtcars$am = factor(mtcars$am, levels = c(0,1), labels = c("Auto","Man") )

str(mtcars$am)
Factor w/ 2 levels "Auto","Man": 2 2 2 1 1 1 1 1 1 1 ...

# See what we have now !

head(mtcars,5)
                   mpg cyl disp  hp drat    wt  qsec vs   am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  Man    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  Man    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  Man    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1 Auto    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0 Auto    3    2

tapply(mtcars$mpg,mtcars$am,mean)
    Auto      Man
17.14737 24.39231

tapply(mtcars$mpg,mtcars$am,quantile)
$Auto
   0%    25%    50%    75%   100%
10.40 14.95 17.30 19.20 24.40

$Man
  0%   25%  50%   75% 100%
15.0 21.0 22.8 30.4 33.9
```