Aggregation

Data Aggregation: The process of taking some data and putting it into a form that lends itself easily to summary, e.g. replace groups of observations with summary statistics.

Data Restructuring: Change the structure of the data so that its new form is more convenient for a specific purpose (usually analysis).

R has many ways to do either. Before you start writing code to do some of these activities check around to see what functions exist. Chances are there are some good tools available. We will explore many of them in this session.

Aggregation

So all of this is basically a lesson that there are functions available in R that can help with aggregation.

Command	Purpose
table, xtabs	Summarize grouping variables with grouping variables - Create Contingency Tables
tapply	Summarize a continuous variable by grouping variables
aggregate	Summarize continuous variable(s) by grouping variables
melt, cast	Reshape data and summarize continuous variable(s) by grouping variables

Aggregation - tables

One of the most basic forms of aggregation is to put things into tables for easy counting. This is usually done with categorical variables to obtain "count" information. We can then do things like Chi-Square tests.

```
letters[1:10]                          # Built in letters char vector

 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"


# sample 20 times from the 1st 10 letters with replacement

set.seed(222)

(my.sample = sample(letters[1:10],20,replace=TRUE) )

[1] "j" "a" "e" "a" "j" "j" "d" "e" "f" "b" "e" "a" "b" "h" "j" "a" "g" "a"
"b"

[20] "c"


table(my.sample)

my.sample

a b c d e f g h j

5 3 1 1 3 1 1 1 4
```

Aggregation - tables

```
prop.table(table(my.sample))
my.sample
   a    b    c    d    e    f    g    h    j 
0.25 0.15 0.05 0.05 0.15 0.05 0.05 0.05 0.20 

sum(prop.table(table(my.sample)))
[1] 1
```

Aggregation - tables

```
set.seed(123)
coinsamp = sample(c('H','T'),1000,T)

table(coinsamp)
coinsamp
  H   T 
507 493 
chisq.test(table(coinsamp),p=c(0.5,0.5))

	Chi-squared test for given probabilities

data:  table(coinsamp) 
X-squared = 0.196, df = 1, p-value = 0.658
```

Aggregation - tables

```
set.seed(123)
coinsamp = sample(c('H','T'),1000,T,prob=c(.70,.30))
chisq.test(table(coinsamp),p=c(0.5,0.5))


	Chi-squared test for given probabilities


data:  table(coinsamp)
X-squared = 168.1, df = 1, p-value < 2.2e-16
```

Aggregation - tables

One of the most basic forms of aggregation is to put things into tables for easy counting. This is usually done with categorical variables to obtain "count" information. We can then do things like Chi-Square tests.

```
table(mtcars$am)

 0  1

19 13
```

```
table(mtcars$cyl)

 4  6  8

11  7 14
```

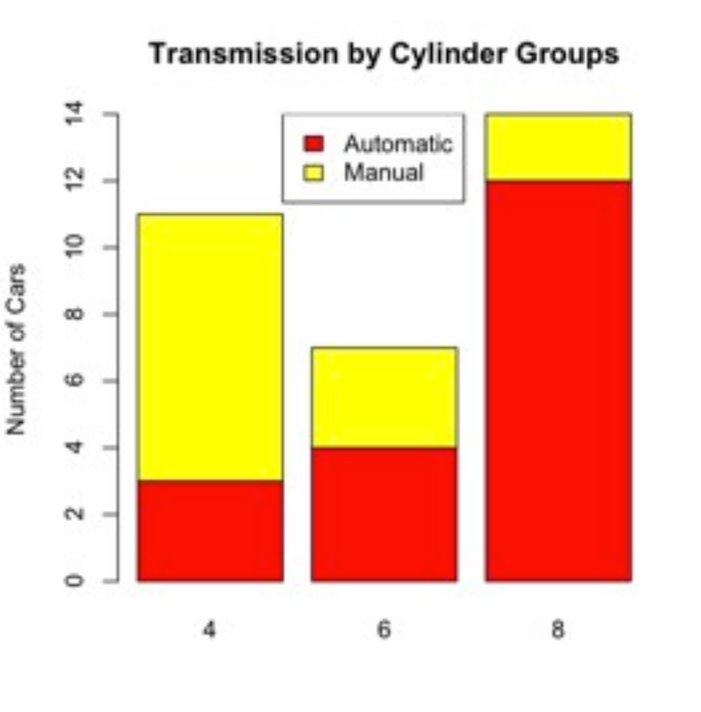
Aggregation - tables

However, we usually create tables out of a data frame that contains some categories or factors. Using the popular mtcars data set let's find out how many 4,6,8 and cylinder cars there are for each category of transmission (auto or manual).

```
table(mtcars$am,mtcars$cyl)
     4  6  8
  0  3  4 12
  1  8  3  2

table(Trans=mtcars$am,Cyl=mtcars$cyl)
     Cyl
Trans  4  6  8
    0  3  4 12
    1  8  3  2
```

Aggregation - tables

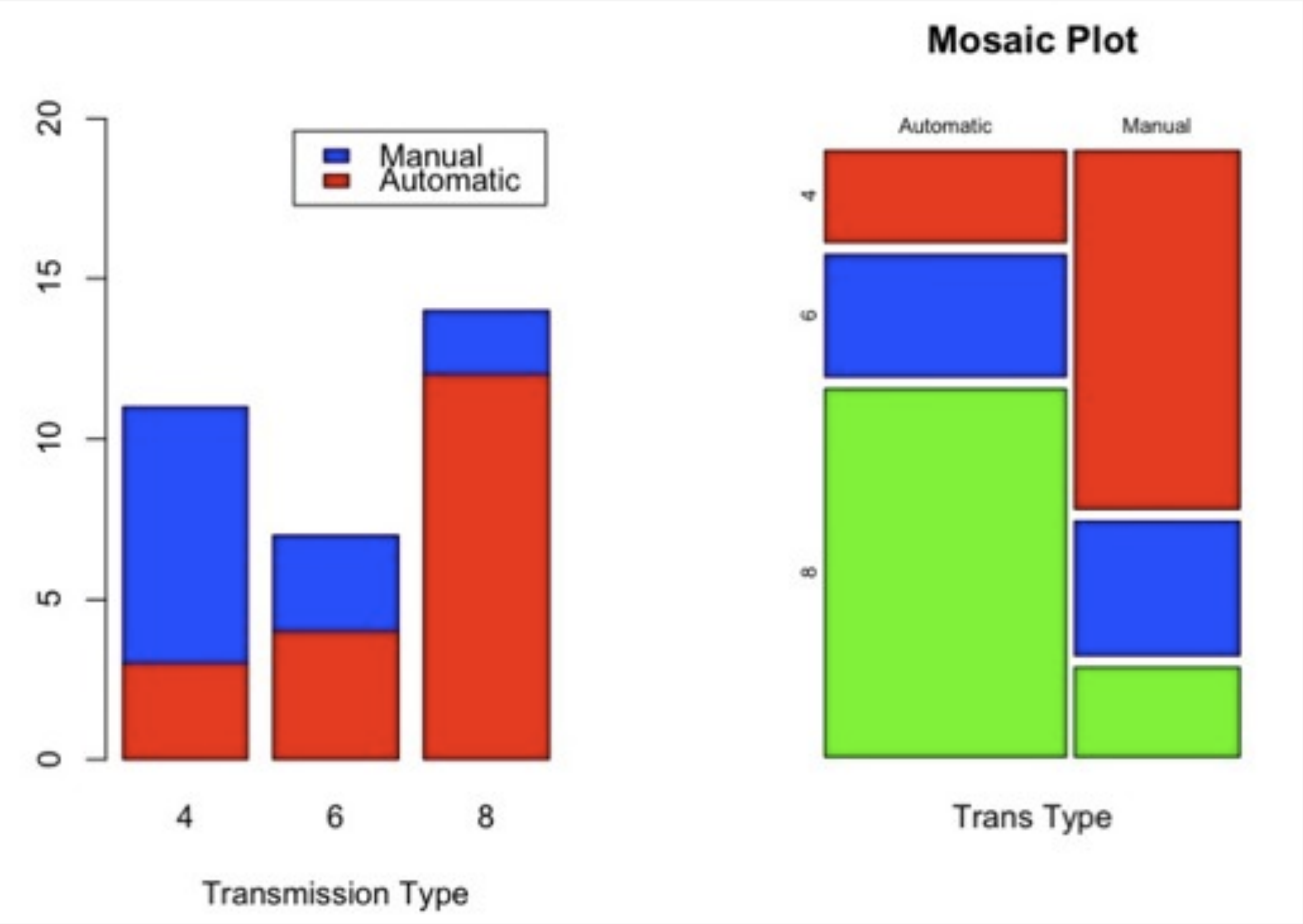
```
mytable = table(mtcars$am, mtcars$cyl)
barplot(mytable, main="Transmission by Cylinder Groups",
        ylab="Number of Cars",col=heat.colors(2))

legend(1.2,14,fill=heat.colors(2),c("Automatic","Manual"))
```



Aggregation - tables

```
mtcars = transform(mtcars, am = factor(am,labels=c("Automatic","Manual")))

my.table = table(mtcars$am,mtcars$cyl)

             4  6  8
  Automatic  3  4 12
  Manual     8  3  2

par(mfrow=c(1,2))

barplot(my.table,legend=T,col=c("red","blue","green"),
          xlab="Transmission Type",ylim=c(0,20))

# Here we get a Mosaic Plot

plot(my.table,col=c("red","blue","green"),main="Mosaic Plot",
      xlab="Trans Type", ylab="Number of Gears")
```

Aggregation - tables



Aggregation - tables

The table function can work on really big data frames. Remember the large Chicago crime data frame ?

```
url = "http://www.bimcore.emory.edu/BIOS560R/DATA.DIR/chi_crimes.csv"
chi = read.table(url,header=T,sep=",")
nrow(chi)
[1] 334141

crime.table = table(chi$Arrest,chi$Ward)
```

Aggregation - tables

```
names(chi)
 [1] "Case.Number"          "ID"
 [3] "Date"                 "Block"
 [5] "IUCR"                 "Primary.Type"
 [7] "Description"          "Location.Description"
 [9] "Arrest"               "Domestic"
[11] "Beat"                 "District"
[13] "Ward"                 "FBI.Code"
[15] "X.Coordinate"         "Community.Area"
[17] "Y.Coordinate"         "Year"
[19] "Latitude"             "Updated.On"
[21] "Longitude"            "Location"
```

Aggregation - tables

```
crime.table = table(chi$Arrest,chi$District)
           1     2     3     4     5     6
  false  8811 10319 12806 15422 10974 13464
  true   3296  3129  4843  4367  4284  5768

           7     8     9    10    11    12
  false 13877 16422 12140 10359 12821  6858
  true   6273  5964  4516  4657  8977  1916

          13    14    15    16    17    18
  false  5624  9980  8704  8868  7728 11008
  true   1460  2557  5681  1885  1945  3170

          19    20    22    24    25    31
  false 12378  4333  8342  7070 14099     0
  true   3230  1341  2403  2428  5559     6
```

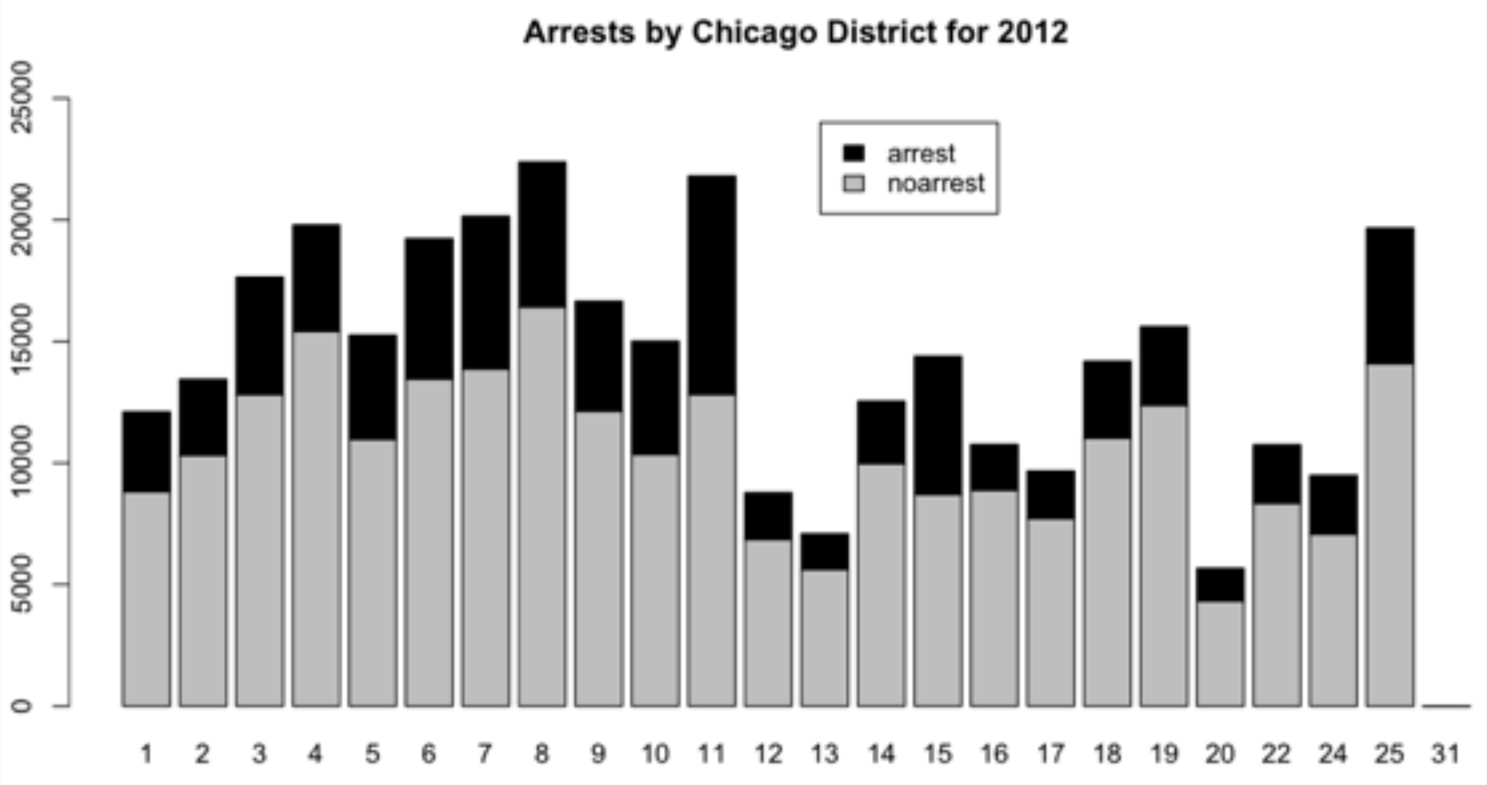
Aggregation - tables

The table function can work on really big data frames. Remember the large Chicago crime data frame ?

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

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

nrow(chi)

[1] 334141

crime.table = table(chi$Arrest,chi$District)


barplot(crime.table,col=c("gray","black"),
        ylim=c(0,25000),
        main="Arrests by Chicago District for 2012")

legend(15,24000,fill=c("black","grey"),c("arrest","noarrest"))
```

Aggregation - tables

The table function can work on really big data frames. Remember the large Chicago crime data frame ?



Aggregation - tables

Remember that we can take continuous variables and turn them into categories.

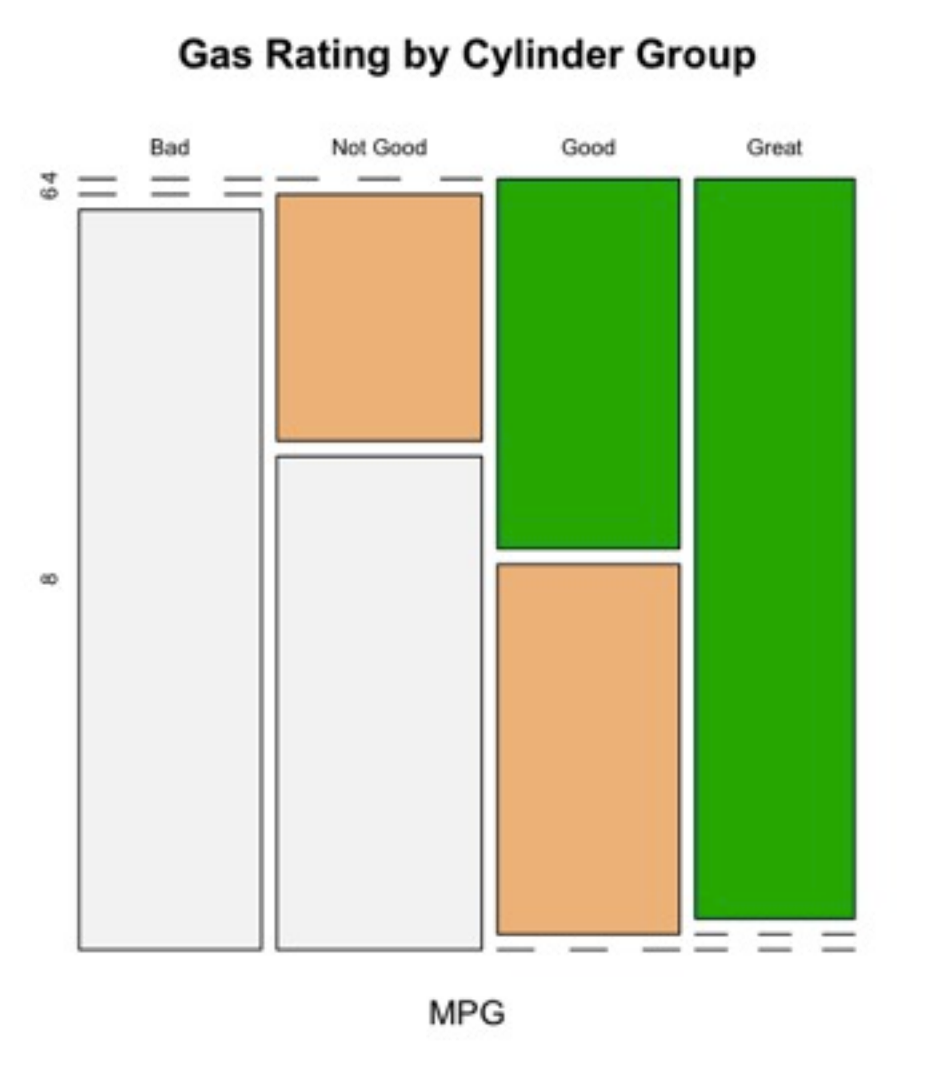
```
 my.cut = cut(mtcars$mpg,breaks=quantile(mtcars$mpg),
          labels=c("Bad","Not Good","Good","Great"),include.lowest=T)

my.cut
 [1] Good     Good     Good     Good     Not Good
 [6] Not Good Bad      Great    Good     Not Good
[11] Not Good Not Good Not Good Bad      Bad
[16] Bad      Bad      Great    Great    Great
[21] Good     Not Good Bad      Bad      Not Good
[26] Great    Great    Great    Not Good Good
[31] Bad      Good
Levels: Bad Not Good Good Great


table(MPG=my.cut,Cyl=mtcars$cyl)
          Cyl
MPG        4 6 8
  Bad      0 0 8
  Not Good 0 3 6
  Good     4 4 0
  Great    7 0 0
```

Aggregation - tables

```
plot(table(MPG=my.cut,mtcars$cyl),col=terrain.colors(3),
     main="Gas Rating by Cylinder Group")
```

Aggregation - tables

```
   Cyl
MPG          4 6 8
  Bad        0 0 8
  Not Good   0 3 6
  Good       4 4 0
  Great      7 0 0
```



Aggregation - margins

Its also a very common thing to want margins for the rows or columns or maybe even both. Once we have a table we have some flexibility.

```
mtcars = transform(mtcars, am = factor(am,labels=c("Automatic","Manual")))

my.table = table(mtcars$am,mtcars$cyl)

              4  6  8
  Automatic   3  4 12
  Manual      8  3  2
```

```
http://www.cyclismo.org/tutorial/R/tables.html#data
```

Aggregation - margins

But this is too hard. Let’s look in R to see if there is a function that will already do this for us.

```
> ??margins
base::apply         Apply Functions Over Array Margins
base::margin.table Compute table margin
base::prop.table   Express Table Entries as Fraction of Marginal Table
..
..

AND A BUNCH OF OTHER STUFF
..

stats::addmargins  Puts Arbitrary Margins on Multidimensional Tables or Arrays
```

Aggregation - margins

```
addmargins(my.table,1)      # Place the sums of each column as a new row

             4  6  8
  Automatic  3  4 12
  Manual     8  3  2
  Sum       11  7 14


addmargins(my.table,2)    # Place sums of each row as a new column
             4  6  8 Sum
  Automatic  3  4 12  19
  Manual     8  3  2  13


addmargins(my.table,c(1,2))  # Get Both

             4  6  8 Sum
  Automatic  3  4 12  19
  Manual     8  3  2  13
  Sum       11  7 14  32
```

Aggregation - margins

But remember that we found some interesting functions before:

```
Help files with alias or concept or title matching ‘margins’ using fuzzy
matching:


base::apply                 Apply Functions Over Array Margins

base::margin.table          Compute table margin

base::prop.table            Express Table Entries as Fraction of Marginal Table

car::mmps                   Marginal Model Plotting

gbm::plot.gbm               Marginal plots of fitted gbm objects
```

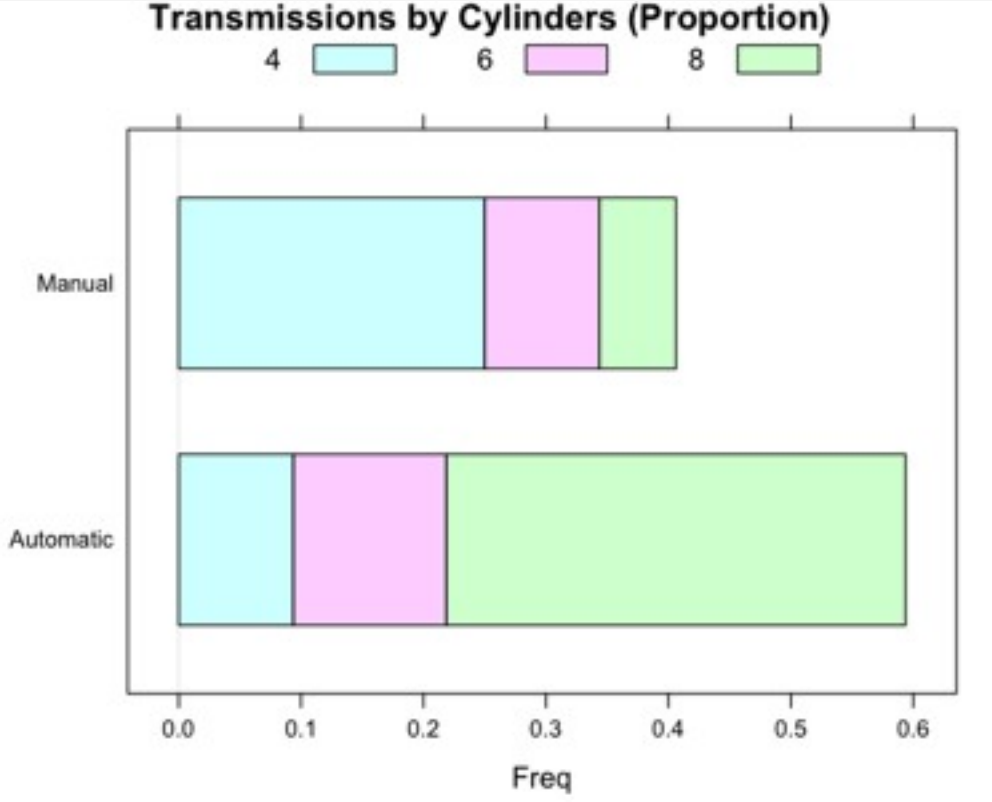
Aggregation - margins

But there is an easier way. Use the prop.table command, which, if you look at the help pages for it, is simply an alias to the sweep command without all the confusing arguments.

```
prop.table(my.table,1)
                    4         6         8
  Automatic 0.1578947 0.2105263 0.6315789
  Manual    0.6153846 0.2307692 0.1538462

rowSums(prop.table(my.table,1))   # The rows add up to one as they should
Automatic    Manual
        1         1
```

Aggregation - margins

```
library(lattice)
barchart(prop.table(my.table),auto.key=list(columns=3),
         main="Transmissions by Cylinders (Proportion)")
```



Aggregation - 3D Tables

You should also note that tables can be 3-dimensional though this can get tricky when viewing.

```
my.3d.table = table(mtcars$am,mtcars$gear,mtcars$cyl)
, ,  = 4

             3  4  5
  Automatic  1  2  0
  Manual     0  6  2


, ,  = 6

             3  4  5
  Automatic  2  2  0
  Manual     0  2  1


, ,  = 8

             3  4  5
  Automatic 12  0  0
  Manual     0  0  2
```

Aggregation - 3D Tables

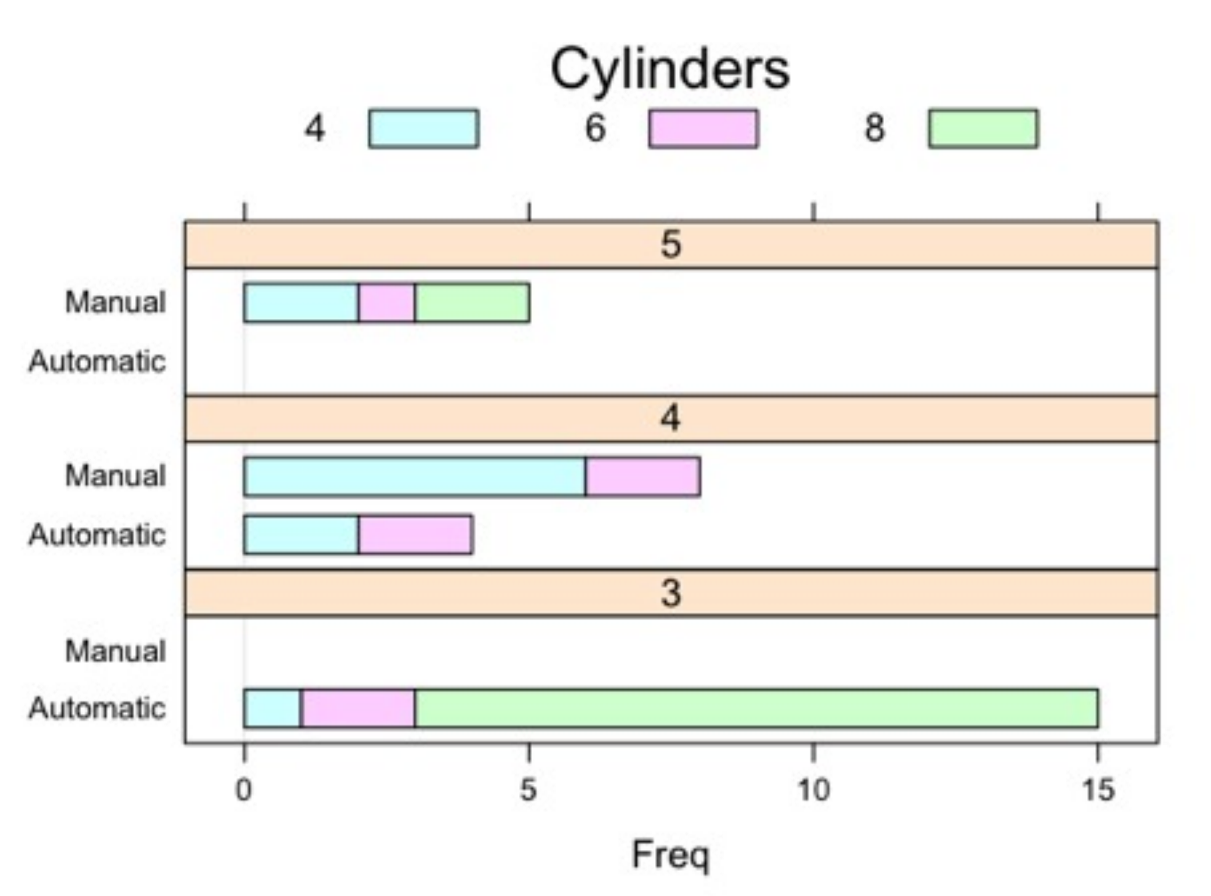
There is a function called ftable that flattens your table for ease of viewing:

```
ftable(my.3d.table)
              4  6  8

Automatic 3   1  2 12
          4   2  2  0
          5   0  0  0
Manual    3   0  0  0
          4   6  2  0
          5   2  1  2
>
```

Aggregation - 3D Tables

The lattice graphics package can help here:

```
barchart(my.3d.table,layout=c(1,3),auto.key=list(title="Cylinders",columns=3))
```



Aggregation - xtabs

This wouldn’t be R unless there were some other function that does pretty much the same thing as table. There is another function called “xtabs”. Some people feel that it is superior to the table function because it supports a formula interface when creating tables. The formula interface requires the "+" symbol to be on the right of any grouping variable.

```
xtabs(~am,mtcars)     # Transmission type by number of cylinders
am
 0  1
19 13


                      # Number in each cylinder group
xtabs(~cyl,mtcars)
cyl
 4  6  8
11  7 14
```

Aggregation - xtabs

This wouldn’t be R unless there were some other function that does pretty much the same thing as table. There is another function called “xtabs”. Some people feel that it is superior to the table function because it supports a formula interface when creating tables. The formula interface requires the "+" symbol to be on the right of any grouping variable.

```
xtabs(~am + cyl, mtcars)     # Transmission type by number of cylinders
           cyl
am            4  6  8
  Automatic   3  4 12
  Manual      8  3  2


xtabs(~cyl + am, mtcars)     # Number of cylinders by transmission type
   am
cyl Automatic Manual
  4         3      8
  6         4      3
  8        12      2
```

Aggregation - xtabs

A three dimensional table is possible

```
xtabs(~gear + cyl + am, mtcars)
, , am = Automatic

    cyl
gear  4  6  8
   3  1  2 12
   4  2  2  0
   5  0  0  0

, , am = Manual

    cyl
gear  4  6  8
   3  0  0  0
   4  6  2  0
   5  2  1  2
```

Aggregation - xtabs

Part of the reason we might use xtabs is because the formula interface is also used when testing for mutual independence. If we create a table with xtabs then we use the same formula in the loglm function.

```
mytab = xtabs(~gear + cyl + am, mtcars)

library(MASS)
loglm(~gear + cyl + am, mytab)

Call:
loglm(formula = ~gear + cyl + am, data = mytab)

Statistics:
                      X^2 df     P(> X^2)
Likelihood Ratio 51.94755 12 6.333779e-07
Pearson          49.24950 12 1.891657e-06
```

Aggregation - xtabs

Also, xtabs allows us to use a subset argument to pull out certain records:

```
xtabs(~gear + cyl, mtcars)
    cyl
gear  4  6  8
   3  1  2 12
   4  8  4  0
   5  2  1  2


xtabs(~gear + cyl, mtcars, subset=mpg < 20)
    cyl
gear  6  8
   3  1 12
   4  2  0
   5  1  2
```

Aggregation - tapply

Summarizing a continuous variable in terms of one or more grouping variables

Aggregation - tapply

Up until now we've been working with tables created from categorical variables. Even in our example that used a continuous variable we wound up using the cut command to group the continuous variables into categories.

We can easily summarize continuous variables in terms of a grouping variable using the **tapply** and **by** commands.

```
       continuous   factor      summary function
```

```
tapply(mtcars$mpg, mtcars$am, mean)
Automatic    Manual
 17.14737  24.39231

tapply(mtcars$mpg, list(Transmision=mtcars$am, Cyl=mtcars$cyl), mean)
           Cyl
Transmision    4    6    8
          0 22.9 19.1 15.1
          1 28.1 20.6 15.4
```

Aggregation - tapply

One limitation of the tapply command is that we can't summarize more than one continuous variable at a time.

```
This works:

tapply(mtcars$mpg, list(mtcars$am, mtcars$vs), mean)
              0         1
Automatic 0.8008991 0.9339769
Manual    1.6366124 1.7982418

But this does not:


tapply(list(mtcars$mpg,mtcars$hp), list(mtcars$am, mtcars$cyl), mean)
Error in tapply(list(mtcars$mpg, mtcars$hp), list(mtcars$am, mtcars$cyl),  :
  arguments must have same length
```

Aggregation - The aggregate command

This is the more general command for aggregation of continuous data in terms of some factors or categories:

```
aggregate(mtcars['mpg'],list(Transmission=mtcars$am),mean)

  Transmission      mpg
1    Automatic 17.14737
2       Manual 24.39231
```

```
aggregate(mtcars[c('mpg','hp')],list(Transmission_Type=mtcars$am),mean)

  Transmission_Type      mpg       hp
1         Automatic 17.14737 160.2632
2            Manual 24.39231 126.8462
```

Aggregation - The aggregate command

```
aggregate(mtcars[c('mpg','hp')],
          list(Transmission_Type = mtcars$am,
          Cylinders = mtcars$cyl), mean)

  Transmission_Type Cylinders      mpg        hp
1         Automatic         4 22.90000  84.66667
2            Manual         4 28.07500  81.87500
3         Automatic         6 19.12500 115.25000
4            Manual         6 20.56667 131.66667
5         Automatic         8 15.05000 194.16667
6            Manual         8 15.40000 299.50000
```

Aggregation - The aggregate command

The aggregate command also has a formula interface if you find that to be more convenient. Many do since it gives you an argument to specify the data frame you are trying to summarize. This saves typing.

```
aggregate(mpg ~ am, mtcars, mean)

      am      mpg
1   auto 17.14737
2 manual 24.39231

aggregate(mpg ~ am + cyl, mtcars, mean)

      am cyl      mpg
1   auto   4 22.90000
2 manual   4 28.07500
3   auto   6 19.12500
4 manual   6 20.56667
5   auto   8 15.05000
6 manual   8 15.40000
```

Aggregation - The aggregate command

The aggregate command also has a formula interface if you find that to be more convenient. Many do since it also gives you an argument to specify the data frame you are trying to summarize. This saves typing.

```
aggregate( cbind(mpg,wt,hp) ~ am + cyl, mtcars, mean)

         am cyl  mpg   wt    hp
1 Automatic   4 22.9 2.94  84.7
2    Manual   4 28.1 2.04  81.9
3 Automatic   6 19.1 3.39 115.2
4    Manual   6 20.6 2.75 131.7
5 Automatic   8 15.1 4.10 194.2
6    Manual   8 15.4 3.37 299.5
```

Aggregation - The aggregate command

So here is how this would look using the older bracket style:

```
aggregate( mtcars[c('mpg','wt','hp')],
           list(trans=mtcars$am, cyl=mtcars$cyl), mean)

      trans cyl  mpg   wt    hp
1 Automatic   4 22.9 2.94  84.7
2    Manual   4 28.1 2.04  81.9
3 Automatic   6 19.1 3.39 115.2
4    Manual   6 20.6 2.75 131.7
5 Automatic   8 15.1 4.10 194.2
6    Manual   8 15.4 3.37 299.5
```

Aggregation - The aggregate command

```
my.snps = read.table("http://www.bimcore.emory.edu/nsnps.csv",header=F,sep=" ")

names(my.snps)=c("X1","X2","SNP1","SNP2")

new.snps = cbind(my.snps,co=sample(c("case","control"),30,TRUE))



head(new.snps)
      X1   X2 SNP1 SNP2      co
1 -0.737 2.77   AA   BB control
2 -1.839 5.70   AA   BB    case
3 -0.470 4.89   AA   BB    case
4  0.851 6.38   AA   BB control
5 -1.690 8.11   AA   BB    case
6 -1.480 4.33   AA   BB control


# Let's summarize the mean of X1 and X2 across the "co" column

aggregate(cbind(X1,X2)~co,new.snps,mean)
       co      X1   X2
1    case -0.0371 4.77
2 control -0.0163 5.15
```

Aggregation - The aggregate command

```
head(new.snps)
      X1   X2 SNP1 SNP2      co
1 -0.737 2.77   AA   BB control
2 -1.839 5.70   AA   BB    case
3 -0.470 4.89   AA   BB    case
4  0.851 6.38   AA   BB control
5 -1.690 8.11   AA   BB    case
6 -1.480 4.33   AA   BB control


# Let's summarize the mean of X1 and X2 across the SNP1 column

aggregate(cbind(X1,X2)~SNP1,new.snps,mean)
  SNP1     X1   X2
1   aa  0.222 4.88
2   Aa  0.279 4.98
3   AA -0.573 5.19
```

Aggregation - The melt and cast commands

* melt and cast belong to a package called "reshape" which is an add on package for doing data frame manipulation and , if desired, aggregation.

* Some feel that these commands are superior to the built in commands such as aggregate, tapply, and by. They do provide a general way to break down data into a general format and then reassemble it so we discuss it here.

* Let’s look at a simple data set. You will need to tell melt which variables are “id” variables, (usually factors and/or characters), and which are “measured” variables, (continuous variables that are measurements on the id variables).

Aggregation - The melt and cast commands

First we need to install the reshape package:

```
   install.packages("reshape")
   library(reshape)
```

Second we need to understand the difference between "long" format and "wide" format.

```
data(smiths)
smiths

     subject time age weight height
1 John Smith    1  33     90   1.87
2 Mary Smith    1  NA     NA   1.54
```

This is "wide" format - all the data relating to an observation is on one line only

Aggregation - The melt and cast commands

Wide format is the most convenient for humans. Everything is on one line only.

```
smiths
     subject time age weight height
1 John Smith    1  33     90   1.87
2 Mary Smith    1  NA     NA   1.54
```

Long format is better for doing certain statistical tests such as ANOVA

```
     subject variable value
1 John Smith     time  1.00
2 Mary Smith     time  1.00
3 John Smith      age 33.00
4 Mary Smith      age    NA
5 John Smith   weight 90.00
6 Mary Smith   weight    NA
7 John Smith   height  1.87
8 Mary Smith   height  1.54
```

Aggregation - The melt and cast commands

We need ways to conveniently move back and forth between the two. The reshape package includes two commands to help with this: melt and cast

```
melt(smiths)
Using subject as id variables
     subject variable value
1 John Smith     time  1.00
2 Mary Smith     time  1.00
3 John Smith      age 33.00
4 Mary Smith      age    NA
5 John Smith   weight 90.00
6 Mary Smith   weight    NA
7 John Smith   height  1.87
8 Mary Smith   height  1.54
```

Melt can figure out what the appropriate id variable is - in this case "Subject". We could specify this explicitly:

```
melt(smiths, id.var="subject")
```

Aggregation - The melt and cast commands

The other important argument to melt is "measure.var". this tells melt what variables you want to use for measuring and or analysis.

```
melt(smiths,measure.var=c("time","age","weight","height"))
     subject variable value
1 John Smith     time  1.00
2 Mary Smith     time  1.00
3 John Smith      age 33.00
4 Mary Smith      age    NA
5 John Smith   weight 90.00
6 Mary Smith   weight    NA
7 John Smith   height  1.87
8 Mary Smith   height  1.54
```

Aggregation - The melt and cast commands

```
my.df = data.frame(id=paste("P",1:4,sep=""),
                     age=c(30.0,36.0,43.0,29.0),
                     pulse=c(78.2,74.4,82.3,91.9),
                     height=c(72,68,69,71),
                     group=c("ctrl","ctrl","treat","treat"))

my.df
  id age pulse height group
1 P1  30  78.2     72  ctrl
2 P2  36  74.4     68  ctrl
3 P3  43  82.3     69 treat
4 P4  29  91.9     71 treat



sapply(my.df,class)
       id       age     pulse    height     group
 "factor" "numeric" "numeric" "numeric"  "factor"
```

If we don’t give melt any information then it will use “id” and “group” as identification variables.

Aggregation - The melt and cast commands

```
> sapply(my.df,class)
       id       age     pulse    height     group 
 "factor" "numeric" "numeric" "numeric"  "factor" 

> melt(my.df)

Using id, group as id variables
   id group variable value
1  P1  ctrl      age  30.0
2  P2  ctrl      age  36.0
3  P3 treat      age  43.0
4  P4 treat      age  29.0
5  P1  ctrl    pulse  78.2
6  P2  ctrl    pulse  74.4
7  P3 treat    pulse  82.3
8  P4 treat    pulse  91.9
9  P1  ctrl   height  72.0
10 P2  ctrl   height  68.0
11 P3 treat   height  69.0
12 P4 treat   height  71.0

my.melt = melt(my.df)
```

Aggregation - The melt and cast commands

```
melt(my.df)

# Would be the same as

melt(my.df, id.vars = c("id","group"))

# Would be the same as

melt(my.df, id.vars = c("id","group"),
       measure.vars = c("age","pulse","height"))


# Would be the same as

melt(my.df, measure.vars = c("age","pulse","height"))
```

Aggregation - The melt and cast commands

```
cast(my.melt, group~variable, mean)

  group age pulse height
1  ctrl  33  76.3     70
2 treat  36  87.1     70
```

Aggregation - The melt and cast commands

```
cast(my.melt, group~variable, mean)

  group age pulse height
1  ctrl  33  76.3     70
2 treat  36  87.1     70



cast(my.melt, group~variable, mean, subset=variable==c("height","age"))

  group age height
1  ctrl  36     72
2 treat  29     69
```

Aggregation - The melt and cast commands

```
url = "http://www.bimcore.emory.edu/BIOS560R/DATA.DIR/sixsmiths.csv"
sixsmiths = read.table(url,header=T,sep=",")

sixsmiths
     subject time age weight height
1 John Smith    1  33     90   1.87
2 Mary Smith    1  28     85   1.54
3 John Smith    2  33     90   1.87
4 Mary Smith    2  28     89   1.54
5 John Smith    3  33     90   1.87
6 Mary Smith    3  28     92   1.54
```

Aggregation - The melt and cast commands

```
my.melt = melt(sixsmiths,id.vars=c("subject","time"))

      subject time variable value
1  John Smith    1      age 33.00
2  Mary Smith    1      age 28.00
3  John Smith    2      age 33.00
4  Mary Smith    2      age 28.00
5  John Smith    3      age 33.00
6  Mary Smith    3      age 28.00
7  John Smith    1   weight 90.00
8  Mary Smith    1   weight 85.00
9  John Smith    2   weight 90.00
10 Mary Smith    2   weight 89.00
11 John Smith    3   weight 90.00
12 Mary Smith    3   weight 92.00
13 John Smith    1   height  1.87
14 Mary Smith    1   height  1.54
15 John Smith    2   height  1.87
16 Mary Smith    2   height  1.54
17 John Smith    3   height  1.87
18 Mary Smith    3   height  1.54
```

Aggregation - The melt and cast commands

```
cast(my.melt,time~variable,mean)

  time  age weight height
1    1 30.5   87.5   1.71
2    2 30.5   89.5   1.71
3    3 30.5   91.0   1.71



cast(my.melt,time~variable,mean,subset=variable==c("weight"))
  time weight
1    1   87.5
2    2   89.5
3    3   91.0
```

Aggregation - Summary

So all of this is basically a lesson that there are functions available in R that can help with aggregation.

Command	Purpose
table, xtabs	Create Contingency Tables
tapply, by, split	Summarize a continuous variable by a grouping variables
aggregate	Summarize continuous variable(s) by grouping variables
melt, cast	Reshape data and summarize continuous variable(s) by grouping variables

Aggregation - Summary

So what are the rules here ?

If you want to look at count based data across one or more categories you can use tables or the xtabs functions.

If you want to look at count based data across one or more categories and you anticipate doing lots of stat tests on them then use the xtabs function.

If you want to look at one continuous variable summarized in some way across multiple categories then use tapply or by.

If you want to look at continuous variable(s) summarized in some way across multiple categories then use aggregtate or melt/cast.