# BIOS 545 DataFrames and Control Statements

Steve Pittard wsp@emory.edu

## Merging Data Frames

Merging data frames is possible

- But we don't often encounter two or more similar enough data frames that we can merge them easily.
- Usually we pick and choose columns from a number of data frames to build a new data frame.
- But this isn't easy either since not all observations are from the same population.
- Always work with the *minimum* amount of data necessary for a given analysis
- There is no reason to create a big data frame if you are only going to be using 20 percent of the attributes/columns.

Merging data frames is possible. You have to select a "key'' that is common to both data frames to make this work. Pick a column(s) that links the two data frames:

```
df1 <- data.frame(indiv_id = 1:4,
                  snp1 = c(1,1,0,1),
                  snp2 = c(1,1,0,0))

df2 <- data.frame(indiv_id = c(1,3,4,6),
                  cov1 = c(1.14,4.50,0.80,1.39),
                  cov2 = c(74.6,79.4,48.2,68.1))
```

So think about what columns are in common

```
df1
```

```
##   indiv_id snp1 snp2
## 1        1    1    1
## 2        2    1    1
## 3        3    0    0
## 4        4    1    0
```

```
df2
```

```
##   indiv_id cov1 cov2
## 1        1 1.14 74.6
## 2        3 4.50 79.4
## 3        4 0.80 48.2
## 4        6 1.39 68.1
```

```
names(df1)
```

```
## [1] "indiv_id" "snp1"     "snp2"
```

```
names(df2)
```

```
## [1] "indiv_id" "cov1"     "cov2"
```

You can even use some functions to help

```
intersect(names(df1),names(df2))
```

```
## [1] "indiv_id"
```

```
merge(df1, df2, by="indiv_id", all=TRUE)
```

```
##   indiv_id snp1 snp2 cov1 cov2
## 1        1    1    1 1.14 74.6
## 2        2    1    1   NA   NA
## 3        3    0    0 4.50 79.4
## 4        4    1    0 0.80 48.2
## 5        6   NA   NA 1.39 68.1
```

```
merge(df1,df2,all=TRUE)
```

```
##   indiv_id snp1 snp2 cov1 cov2
## 1        1    1    1 1.14 74.6
## 2        2    1    1   NA   NA
## 3        3    0    0 4.50 79.4
## 4        4    1    0 0.80 48.2
## 5        6   NA   NA 1.39 68.1
```

What happens if you don't specify "all=TRUE" ?

```
merge(df1,df2)
```

```
##   indiv_id snp1 snp2 cov1 cov2
## 1        1    1    1 1.14 74.6
## 2        3    0    0 4.50 79.4
## 3        4    1    0 0.80 48.2
```

Pay attention to the `all.x'' and`all.y'' arguments as it helps you specify which records you want to the include

```
df1
```

```
##   indiv_id snp1 snp2
## 1        1    1    1
## 2        2    1    1
## 3        3    0    0
## 4        4    1    0
```

```
df2
```

```
##   indiv_id cov1 cov2
## 1        1 1.14 74.6
## 2        3 4.50 79.4
## 3        4 0.80 48.2
## 4        6 1.39 68.1
```

```
merge(df1, df2, by="indiv_id", all.y=T)
```

```
##   indiv_id snp1 snp2 cov1 cov2
## 1        1    1    1 1.14 74.6
## 2        3    0    0 4.50 79.4
## 3        4    1    0 0.80 48.2
## 4        6   NA   NA 1.39 68.1
```

Lastly, note that the merge columns do not have to be named the same thing in each data frame as long as they refer to the same thing

```
names(df2) <- c("id","cov1","cov2")

df1
```

```
##   indiv_id snp1 snp2
## 1        1    1    1
## 2        2    1    1
## 3        3    0    0
## 4        4    1    0
```

```
df2
```

```
##   id cov1 cov2
## 1  1 1.14 74.6
## 2  3 4.50 79.4
## 3  4 0.80 48.2
## 4  6 1.39 68.1
```

```
merge(df1,df2,by.x="indiv_id",by.y="id",all=TRUE)
```

```
##   indiv_id snp1 snp2 cov1 cov2
## 1        1    1    1 1.14 74.6
## 2        2    1    1   NA   NA
## 3        3    0    0 4.50 79.4
## 4        4    1    0 0.80 48.2
## 5        6   NA   NA 1.39 68.1
```

## Splitting Data Frames

The split function lets us break up a data frame based on a grouping variable

- Let's say we want to split up mtcars based on the number of cylinders which take on the values 4,6,8
- Use the split command which gives back a list with each element containing a part of the data frame corresponding to each cylinder group
- Without using split you could do:

```
eight.cyl <- mtcars[mtcars$cyl == 8,]

six.cyl   <- mtcars[mtcars$cyl == 6, ]

four.cyl  <- mtcars[mtcars$cyl == 4, ]
```

But what if we had 10 categories we wanted to split by ? The split function does the same thing as our manual example and scales to multiple categories.

```
hold <- split(mtcars, mtcars$cyl)

str(hold,max.level=1)
```

```
## List of 3
##  $ 4:'data.frame':   11 obs. of  11 variables:
##  $ 6:'data.frame':   7 obs. of  11 variables:
##  $ 8:'data.frame':   14 obs. of  11 variables:
```

```
hold[[1]] # Show the first 3 lines of the first list element
```

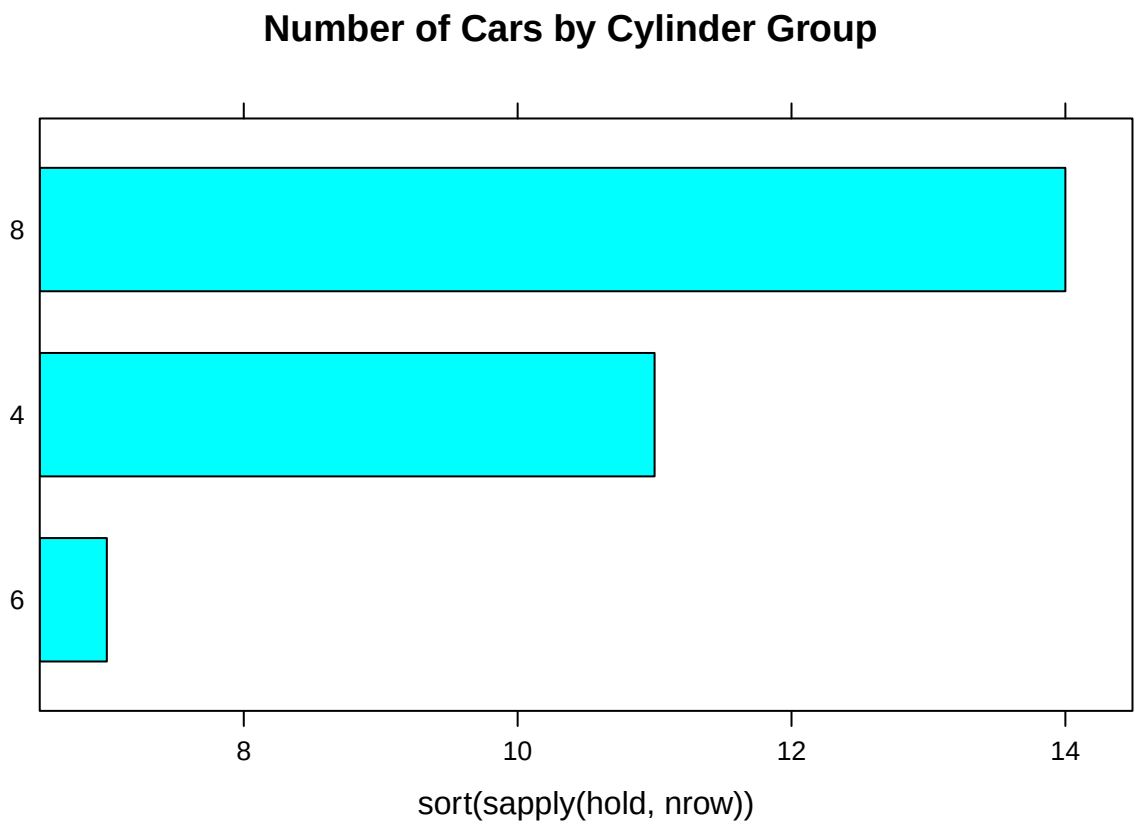
```
##                mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Fiat 128       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
## Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

Why is this useful ? Well we might want to do sime summary across each of the individual groups.

```
hold <- split(mtcars, mtcars$cyl)

sapply(hold,nrow)
```

```
##  4  6  8
## 11  7 14
```

```
library(lattice)
barchart(sort(sapply(hold,nrow)),main="Number of Cars by Cylinder Group")
```




Or we might want to focus in on only the cars occupying a certain cylinder group while ignoring the rest. So if we wanted only the 8 cylinder cars:

```
eight.cyl <- hold$`8`

# -OR-

eight.cyl <- hold[[3]]
```

Because what we get back is a list we can use lapply to look at the first few records of each element which is a data frame

```
lapply(hold,head,3)
```

```
## $`4`
##             mpg cyl  disp hp drat   wt  qsec vs am gear carb
## Datsun 710 22.8   4 108.0 93 3.85 2.32 18.61  1  1    4    1
## Merc 240D  24.4   4 146.7 62 3.69 3.19 20.00  1  0    4    2
## Merc 230   22.8   4 140.8 95 3.92 3.15 22.90  1  0    4    2
##
## $`6`
##                 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  1    4    4
## Mazda RX4 Wag  21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Hornet 4 Drive 21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
##
## $`8`
##                   mpg cyl  disp  hp drat   wt  qsec vs am gear carb
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.44 17.02  0  0    3    2
## Duster 360        14.3   8 360.0 245 3.21 3.57 15.84  0  0    3    4
## Merc 450SE        16.4   8 275.8 180 3.07 4.07 17.40  0  0    3    3
```

We could write our own summary function. While it is an advanced idea at this point, it is good for you too see this kind of approach as it is common in R. This example gives the mean MPG for each cylinder group:

```
# We create our own function and apply it to each element of "hold"

hold <- split(mtcars,mtcars$cyl)

# Here we use and anonymous function which gets applied
# to every element of hold which are the three individual "splits"
# containing the rows of data corresponding cylinder values of 4, 6, and 8

sapply(hold, function(x) mean(x$mpg))
```

```
##        4        6        8
## 26.66364 19.74286 15.10000
```

The following is the equivalent to the above and is perhaps more readable and understandable until you get used to the idea of anonymous functions which are functions that have no name.

They are typically used in cased where you don't anticipate needing the function beyond that specific moment so you don't even bother to give it a name. Either approach is fine and you should typically do whatever makes the most sense to you.

The more advanced R programmers like to use anonymous functions because it is rumored that they provide better performance although that is not necessarily true.

```
mymean <- function(x) {
  return(mean(x$mpg))
}
```

```
sapply(hold, mymean)
```

```
##        4        6        8
## 26.66364 19.74286 15.10000
```

One useful example of sorting data relates to examining coefficients emerging a linear regression:

```
mylm <- lm(mpg~., mtcars)
mylm$coefficients
```

```
## (Intercept)         cyl        disp          hp        drat          wt
## 12.30337416 -0.11144048  0.01333524 -0.02148212  0.78711097 -3.71530393
##        qsec          vs          am        gear        carb
##  0.82104075  0.31776281  2.52022689  0.65541302 -0.19941925
```

The magnitude of coefficients might be and indicator of how important or significant a variable might be. So arranging them by magnitute might be helpful.

```
sort(abs(mylm$coefficients),decreasing=TRUE)
```

```
## (Intercept)          wt          am        qsec        drat        gear
## 12.30337416  3.71530393  2.52022689  0.82104075  0.78711097  0.65541302
##          vs        carb         cyl          hp        disp
##  0.31776281  0.19941925  0.11144048  0.02148212  0.01333524
```

## Ordering Data

When considering how to arrange or sort data, most people assume that there is a **sort** function which is true. After all, we have used it to sort numeric vectors from lowest to highest or highest to lowest.

```
# Get 10 random numbers between 10 and 20 inclusive
set.seed(1232)
(some_numbers <- sample(10:20))
```

```
##  [1] 15 20 16 18 11 19 10 14 17 12 13
```

```
sort(some_numbers) # low to high
```

```
##  [1] 10 11 12 13 14 15 16 17 18 19 20
```

```
sort(some_numbers,TRUE)   # high to low
```

```
##  [1] 20 19 18 17 16 15 14 13 12 11 10
```

Let's try this out on the mtcars data frame. We want to sort the rows by the **mpg** column withe the idea being see cars with the best gas mileage (the highest) listed first and those with the worst gas mileage (the lowest) listed last. You might first try:

```
sort(mtcars$mpg)
```

```
##  [1] 10.4 10.4 13.3 14.3 14.7 15.0 15.2 15.2 15.5 15.8 16.4 17.3 17.8 18.1 18.7
## [16] 19.2 19.2 19.7 21.0 21.0 21.4 21.4 21.5 22.8 22.8 24.4 26.0 27.3 30.4 30.4
## [31] 32.4 33.9
```

But this is just the vector corresponding to only the mpg column. Out of desperation you might try something like this:

```
sort(mtcars)
```

If you look at the help page for the **sort** function you will be steered towards the **order** function. Let's take a look at what it does does.

```
order(mtcars$mpg)
```

```
##  [1] 15 16 24  7 17 31 14 23 22 29 12 13 11  6  5 10 25 30  1  2  4 32 21  3  9
## [26]  8 27 26 19 28 18 20
```

What do we get back here ? These are definitely not the MPG values in the data frame ? No, they are row numbers of the data frame which correspond to the lowest MPG to the highest. So record #15 must be the lowest MPG automobile in the set. And record #20 must have the highest MPG. Check this out to be sure:

```
mtcars[15,]
```

```
##                     mpg cyl disp  hp drat   wt  qsec vs am gear carb
## Cadillac Fleetwood 10.4   8  472 205 2.93 5.25 17.98  0  0    3    4
```

```
mtcars[20,]
```

```
##                 mpg cyl disp hp drat    wt qsec vs am gear carb
## Toyota Corolla 33.9   4 71.1 65 4.22 1.835 19.9  1  1    4    1
```

```
range(mtcars$mpg)
```

```
## [1] 10.4 33.9
```

So, a better way to get this information is as follows:

```
# sort by mpg (ascending)
mtcars[order(mtcars$mpg),]
```

```
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## 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
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
```

```
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
```

Or we could sort the data frame rows from highest to lowest MPG. When sorting data we refer to whatever column we are using to sort upon as the **key**.

```
mtcars[rev(order(mtcars$mpg)),]
```

```
##                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## 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
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## 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
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
```

```
mtcars[order(mtcars$mpg,decreasing=TRUE),]
```

```
##                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## 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
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
```

```
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
## 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
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## 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
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
```

```
mtcars[order(-mtcars$mpg),]
```

```
##                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## 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
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
## 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
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## 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
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
```

```
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
```

## Sampling from Data Frames

Just as we can sample from a vector of data we can also sample rows from a data frame. However, it isn't immediately obvious how to do this. For a vector, the sample command will look like this:

```
# sample 5 values from the vector of numbers from 10 to 20.
# Do not sample with replacement

sample(10:20, 5, replace=FALSE)
```

```
## [1] 10 13 20 11 14
```

That was easy enough so let's try this with the data frame

```
sample(mtcars,5, replace=FALSE)
```

```
##                     hp    wt vs  qsec gear
## Mazda RX4          110 2.620  0 16.46    4
## Mazda RX4 Wag      110 2.875  0 17.02    4
## Datsun 710          93 2.320  1 18.61    4
## Hornet 4 Drive     110 3.215  1 19.44    3
## Hornet Sportabout  175 3.440  0 17.02    3
## Valiant            105 3.460  1 20.22    3
## Duster 360         245 3.570  0 15.84    3
## Merc 240D           62 3.190  1 20.00    4
## Merc 230            95 3.150  1 22.90    4
## Merc 280           123 3.440  1 18.30    4
## Merc 280C          123 3.440  1 18.90    4
## Merc 450SE         180 4.070  0 17.40    3
## Merc 450SL         180 3.730  0 17.60    3
## Merc 450SLC        180 3.780  0 18.00    3
## Cadillac Fleetwood 205 5.250  0 17.98    3
## Lincoln Continental 215 5.424  0 17.82    3
## Chrysler Imperial  230 5.345  0 17.42    3
## Fiat 128            66 2.200  1 19.47    4
## Honda Civic         52 1.615  1 18.52    4
## Toyota Corolla      65 1.835  1 19.90    4
## Toyota Corona       97 2.465  1 20.01    3
## Dodge Challenger   150 3.520  0 16.87    3
## AMC Javelin        150 3.435  0 17.30    3
## Camaro Z28         245 3.840  0 15.41    3
## Pontiac Firebird   175 3.845  0 17.05    3
## Fiat X1-9           66 1.935  1 18.90    4
## Porsche 914-2       91 2.140  0 16.70    5
## Lotus Europa       113 1.513  1 16.90    5
## Ford Pantera L     264 3.170  0 14.50    5
## Ferrari Dino       175 2.770  0 15.50    5
## Maserati Bora      335 3.570  0 14.60    5
## Volvo 142E         109 2.780  1 18.60    4
```

What is this ? Well it's not what we wanted that's for sure. So maybe we can sample from a vector that represents all the row numbers in the data frame.

```
set.seed(321)
(idx <- sample(1:nrow(mtcars), 5, replace = FALSE))
```

```
## [1] 22 18 29 13 25
```

Let's now use this information as input to the bracket notation

```
mtcars[idx,]
```

```
##                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Dodge Challenger 15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## Fiat 128         32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Ford Pantera L   15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Merc 450SL       17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Pontiac Firebird 19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
```

We can combine this into one go:

```
set.seed(321)
mtcars[sample(1:nrow(mtcars),5,FALSE),]
```

```
##                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Dodge Challenger 15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## Fiat 128         32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Ford Pantera L   15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Merc 450SL       17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Pontiac Firebird 19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
```

## Control Structures

Let's start with the for-loop structure. This is a looping construct that let's you do some things for a specific number of times. `name'' is some index variable that takes on values returned by`expr_1'', which is almost always some type of sequence. It could represent the length of a vector or rows of a matrix or data frame.

```
for (name in expr_1) {
   expr_2
}
```

```
for (ii in 1:3) {
  print(ii)
}
```

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

Better to generalize this - use the **length()** function so the loop will work with a vector of any size:

```
x <- rnorm(3)
for (ii in 1:length(x)) {
  print(ii)
}
```

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

Here we access the actual values of x

```
x <- rnorm(3)
for (ii in 1:length(x)) {
   print(x[ii])
}
```

```
## [1] -0.3508749
## [1] -1.694481
## [1] -0.8358054
```

Consider the example wherein we have a x values that we want to provide as input into some function that will generate y values.

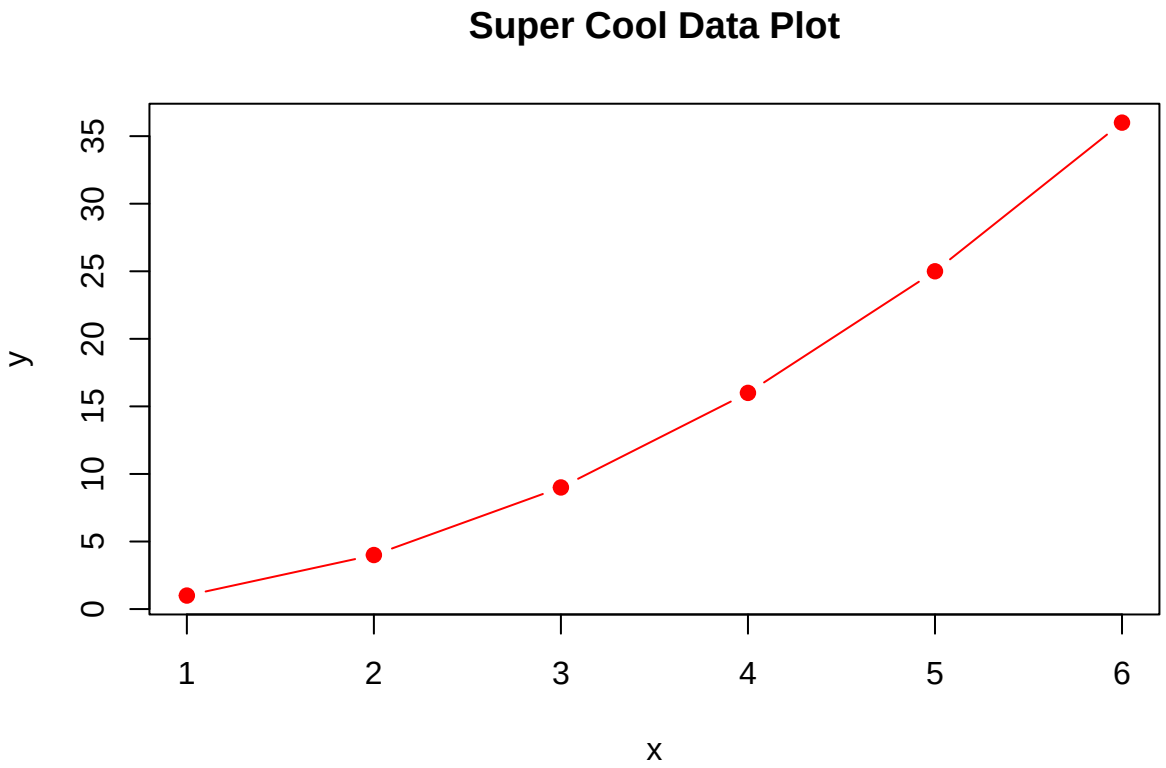
```
y <- vector()  # A blank vector
x <- 1:6
for (ii in 1:length(x)) {
  y[ii] <- x[ii]^2
}

x
```

```
## [1] 1 2 3 4 5 6
```

```
y
```

```
## [1]  1  4  9 16 25 36
```

```
plot(x,y,main="Super Cool Data Plot",type="b",pch=19,col="red")
```




Here we use a for-loop to add up the elements in a vector and find the average. There are already functions in R to do this but knowing how to do it yourself is important:

```
x <-rnorm(1000,20,4)  # 1,000 random elements from a N(20,4)
```

```
mysum <- 0
for (ii in 1:length(x)) {
  mysum <- mysum + x[ii]
}
avg <- mysum / length(x)
cat("The average of this vector is:",avg,"\n")
```

```
## The average of this vector is: 20.128
```

We could clean up the output a little bit

```
cat("The average of this vector is:",round(avg,2),"\n")
```

```
## The average of this vector is: 20.13
```

Given a vector find the smallest value without using the "min" function:

```
set.seed(188)
x <- rnorm(1000)  # 1,000 random elements from a N(20,4)

# Set the min to the first element of x. Unless we are very lucky then
# this will change as we walk through the vector

mymin <- x[1]
for (ii in 1:length(x)) {
  if (x[ii] < mymin) {
     mymin <- x[ii]
  }
}

mymin
```

```
## [1] -3.422185
```

```
min(mymin)
```

```
## [1] -3.422185
```

Here is a more simple example

```
x <- c(1,-1,8)

mymin <- x[1]

for (ii in 1:length(x)) {
  mystr <- paste("The loop counter is:",ii,"and x[ii] is:",x[ii])
  print(mystr)
  if (x[ii] < mymin) {
     mymin <- x[ii]
   }
}
```

```
## [1] "The loop counter is: 1 and x[ii] is: 1"
## [1] "The loop counter is: 2 and x[ii] is: -1"
## [1] "The loop counter is: 3 and x[ii] is: 8"
```

```
mymin
```

```
## [1] -1
```

We can loop through data frames also. Let's see if we can compute the mean of the MPG for all cars. Note that we use the **nrow** function to get the number of rows to loop over

```
mpgsum <- 0
for (ii in 1:nrow(mtcars)) {
  mpgsum <- mpgsum + mtcars[ii,"mpg"]
}

mpgmean <- mpgsum/nrow(mtcars)   # Divide the sum by the # of records

cat("Mean MPG for all cars is:",mpgmean,"\n")
```

```
## Mean MPG for all cars is: 20.09062
```

```
mean(mtcars$mpg)
```

```
## [1] 20.09062
```

Remember the split command ? We can work with the output of that also. Relative to mtcars we let's split up the data frame by cylinder number, which is (4,6, or 8)

```
mysplits <- split(mtcars, mtcars$cyl)

str(mysplits, max.level=1)
```

```
## List of 3
##  $ 4:'data.frame':   11 obs. of  11 variables:
##  $ 6:'data.frame':   7 obs. of  11 variables:
##  $ 8:'data.frame':   14 obs. of  11 variables:
```

We get back a list that contains 3 elements each of which has a data frame corresponding to the number of cylinders. We could summarize each of these data frame elements using a for loop

```
mysplits
```

```
## $`4`
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## 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
## 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
## Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
##
## $`6`
##                 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
## Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Valiant        18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Merc 280       19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C      17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
```

```
##
## $`8`
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
```

```
mysplit <- split(mtcars,mtcars$cyl)

for (ii in 1:length(mysplit)) {
   print(nrow(mysplit[[ii]]))
}
```

```
## [1] 11
## [1] 7
## [1] 14
```

This is equivalent to

```
sapply(mysplit, nrow)
```

```
##  4  6  8
## 11  7 14
```

```
mysplit <- split(mtcars,mtcars$cyl)

for (ii in 1:length(mysplit)) {
   splitname <- names(mysplit[ii])
   cat("mean for",splitname,"cylinders is",mean(mysplit[[ii]]$mpg),"\n")
}
```

```
## mean for 4 cylinders is 26.66364
## mean for 6 cylinders is 19.74286
## mean for 8 cylinders is 15.1
```

This is basically equivalent to

```
sapply(mysplit, function(x) mean(x$mpg))
```

```
##        4        6        8
## 26.66364 19.74286 15.10000
```

What about looping over each split and pulling out only those cars with an manual transmission ? (am == 1) data(mtcars)

```
mysplit <- split(mtcars,mtcars$cyl)

mylist <- list() # Setup a blank list to contain the subset results
```

```
for (ii in 1:length(mysplit)) {
  mylist[[ii]] <- subset(mysplit[[ii]], am == 1)
}

mylist
```

```
## [[1]]
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## 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
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
##
## [[2]]
##                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  1    4    4
## Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Ferrari Dino  19.7   6  145 175 3.62 2.770 15.50  0  1    5    6
##
## [[3]]
##                 mpg cyl disp  hp drat   wt qsec vs am gear carb
## Ford Pantera L 15.8   8  351 264 4.22 3.17 14.5  0  1    5    4
## Maserati Bora  15.0   8  301 335 3.54 3.57 14.6  0  1    5    8
```

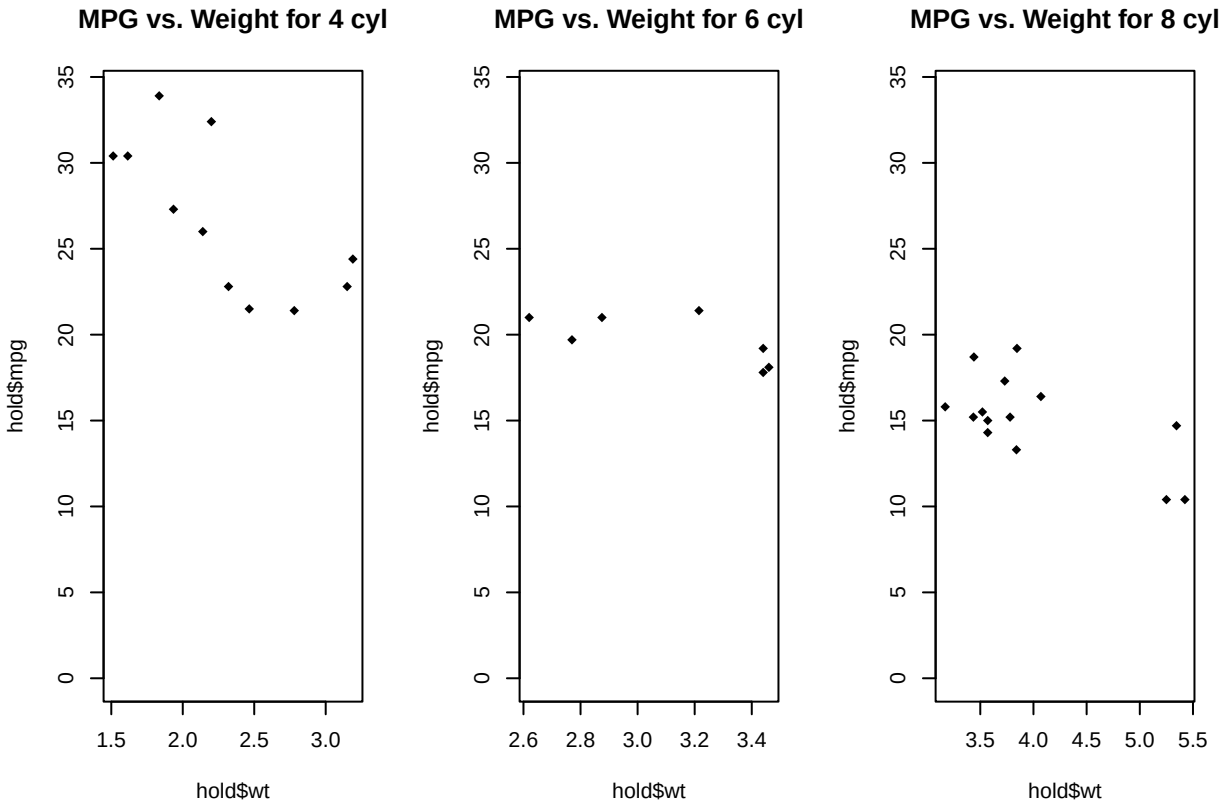
Equivalent to:

```
lapply(mysplit, subset, am == 1)
```

```
## $`4`
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## 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
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
##
## $`6`
##                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  1    4    4
## Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Ferrari Dino  19.7   6  145 175 3.62 2.770 15.50  0  1    5    6
##
## $`8`
##                 mpg cyl disp  hp drat   wt qsec vs am gear carb
## Ford Pantera L 15.8   8  351 264 4.22 3.17 14.5  0  1    5    4
## Maserati Bora  15.0   8  301 335 3.54 3.57 14.6  0  1    5    8
```

Let's say we want to plot MPG vs. Weight for each cylinder group. Check it out:

```
mysplits <- split(mtcars, mtcars$cyl)

par(mfrow=c(1,3))    # This relates to plotting

for (ii in 1:length(mysplits)) {
  hold <- mysplits[[ii]]
  plot(hold$wt, hold$mpg, pch = 18, main=paste("MPG vs. Weight for",
       names(mysplits[ii]), "cyl",sep=" "),ylim=c(0,34))
}
```




The for loop structure generalizes to matrices.

```
set.seed(123)
mymat <- matrix(round(rnorm(6),2),3,2)

for (ii in 1:nrow(mymat)) {
  cat("The sum of row",ii,"is",sum(mymat[ii,]),"\n")
}
```

```
## The sum of row 1 is -0.49
## The sum of row 2 is -0.1
## The sum of row 3 is 3.28
```

## Using the "if" statement

This is an easy structure. It tests a logical expression, which results in either or a TRUE or FALSE condition, and, based on that, executes a specific block of code.

```
if (logical_expression) {
   do something
```

```
   ...
}

#
if (logical_expression) {
  do something
  ..
} else {
  do something else
  ...
}
```

Here is a basic example

```
( x <- 3)
```

```
## [1] 3
```

```
if (is.numeric(x)) {
   print("x is a number")
}
```

```
## [1] "x is a number"
```

```
if (x != 3) {
     print("x is not equal to 3")
} else {
    print("guess what ? x is in fact equal to 3")
}
```

```
## [1] "guess what ? x is in fact equal to 3"
```

Here is a more involved if statement that tests for several conditions. It uses the `else'' keyword in addition toif''`. Note that an `if'' statement does not require an`else" statement but an `else'' statement requires a`parent'' if statement.

```
some.num <- 3

if (some.num < 3) {        # A more involved if statement
     print("Less than 3")
} else if (some.num > 3) {
     print("Greater than 3")
} else {
     print("Must be equal to 3")
}
```

```
## [1] "Must be equal to 3"
```

The if/else statements show up a lot in functions. Checking for valid arguments is a common practice.

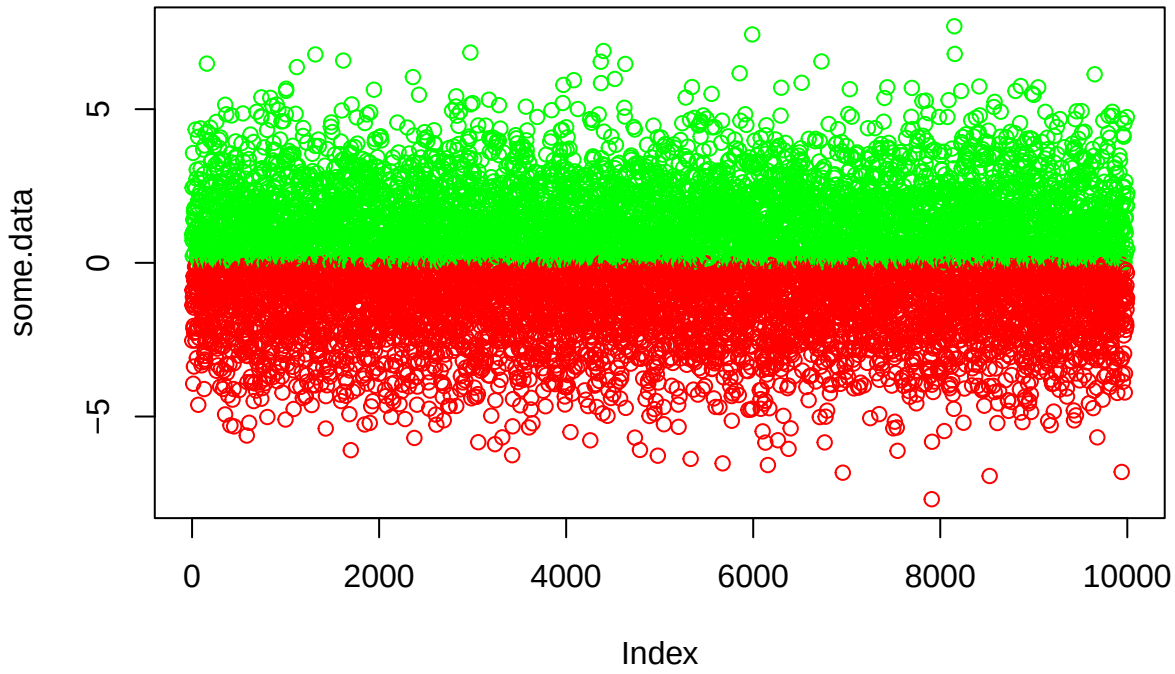
```
x <- 4
y <- 5

if (!is.numeric(x) | !is.numeric(y)) {
   stop("I need numeric values to do this")
} else {
   if (x == y) {
       print("Equal")
   } else {
```

```
      print("Not equal")
   }
}
```

```
## [1] "Not equal"
```

- However, R supports a command called **ifelse** that is desgined to work specifcally on vectors.
- It works well for very large vectors. The format is **ifelse(test,yes,no)** where "test'' is a logical expression to be evaluated.
- If it is TRUE then the action specified in the `yes'' position will be executed. If the evaluated expression is FALSE then the action specified in the`no'' position is executed.

```
some.data = rnorm(10000,0,2)
colors = ifelse(some.data < 0,"RED","GREEN")
plot(some.data,col=colors)
```




```
# This would be the same as:

for (ii in 1:length(some.data)) {
    if (some.data[ii] < 0) {
       colors[ii] = "RED"
    } else {
       colors[ii] = "GREEN"
    }
}
```

We can use ifelse when we want to turn some continuous quantity within a data frame into a factor that we can then use to group by

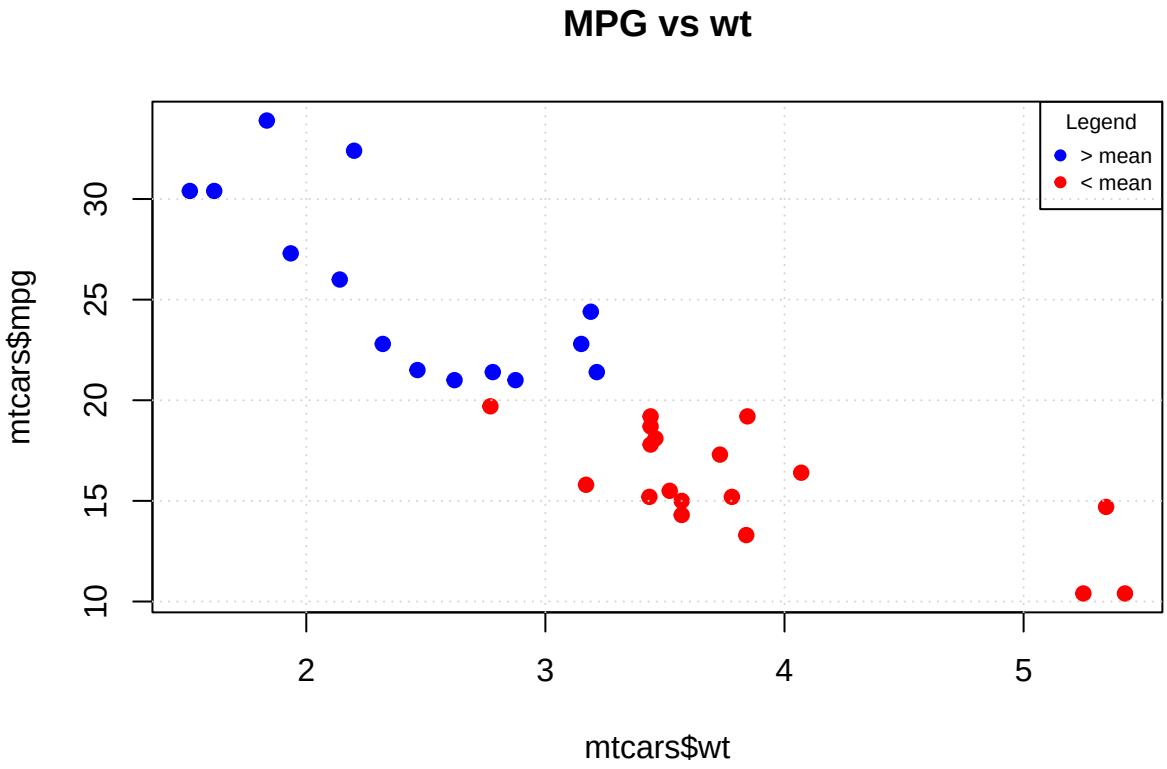
```
mtcars$rating <- ifelse(mtcars$mpg >= mean(mtcars$mpg), "blue", "red")
```

```
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb rating
```

```
## Mazda RX4          21.0   6  160 110 3.90 2.620 16.46  0  1    4    4   blue
## Mazda RX4 Wag      21.0   6  160 110 3.90 2.875 17.02  0  1    4    4   blue
## Datsun 710         22.8   4  108  93 3.85 2.320 18.61  1  1    4    1   blue
## Hornet 4 Drive     21.4   6  258 110 3.08 3.215 19.44  1  0    3    1   blue
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2    red
## Valiant            18.1   6  225 105 2.76 3.460 20.22  1  0    3    1    red
```

```
plot(mtcars$mpg~mtcars$wt,col=mtcars$rating,pch=19, main="MPG vs wt")

grid()

legend("topright", c("> mean","< mean"), pch=19,
       col=c("blue","red"),title="Legend",cex=0.7)
```




Many times we see for loops used in conjunction with if statements

```
score <- c(74,68,98,90,100,67,59)    # Exam scores to be graded
for (ii in 1:length(score)) {
  if (score[ii] == 100) {
      grade <- "A+"
  } else if (score[ii] >= 90) {
      grade <- "A"
  } else if (score[ii] >= 80) {
      grade <- "B"
  } else if (score[ii] >= 70) {
      grade <- "C"
  } else if (score[ii] >= 60) {
      grade <- "D"
  }
  else {
    grade <- "F"
```

```
  }
  print(grade)
}
```

```
## [1] "C"
## [1] "D"
## [1] "A"
## [1] "A"
## [1] "A+"
## [1] "D"
## [1] "F"
```

```
set.seed(123)
x <- round(runif(9,1,20))

for (ii in 1:length(x)) {
    if (x[ii] %% 2 == 0) {
        print(TRUE)
    }
    else {
        print(FALSE)
} }
```

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

This example mimics the bracket notation

```
set.seed(123)
x <- round(runif(10,1,20))
logvec <- vector()
for (ii in 1:length(x)) {
     if (x[ii] %% 2 == 0) {
       logvec[ii] <- TRUE
     }
     else {
      logvec[ii] <- FALSE
     }
# Setup an empty vector
}
logvec
```

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

```
x[logvec]
```

```
## [1]  6 16 18  2 18 10
```

One can easily "break" out of a for loop based on some condition. Normally you should clean your data before processing but perhaps not Let's say that you are processing elements of a vector and if you encounter

a value of NA then you want to stop the for loop

```
my.vec <- c(1,2,3,NA,5,6,7,8,9,10)
for (ii in 1:length(my.vec)) {
  if (is.na(my.vec[ii])) {
          break
  }
    cat("element is ",ii,"\n")
}
```

```
## element is  1
## element is  2
## element is  3
```

Here we want to"catch"the the missing value and then"skip over it". To do this we would use the "next" statement.

```
my.vec <- c(1,2,3,NA,5,6,7,8,9,10)
for (ii in 1:length(my.vec)) {
  if (is.na(my.vec[ii])) {
       next
  }
  cat("element is ",ii,"\n")
 }
```

```
## element is  1
## element is  2
## element is  3
## element is  5
## element is  6
## element is  7
## element is  8
## element is  9
## element is  10
```

Here is an example that will be useful when processing things like genetic sequences. Let's say we have a string of text we wish to "encode"by changing all vowels to something else. This would not be a tough code to break but let's see what is involved. In our code we:

```
We'll change
        a to s
        e to t
        i to u
        o to v
        u to w
```

So a string like:

```
sequence <- "Hello my would come out like: Ed. Happy to meet you"
```

would come out like:

```
"Htllv my nsmt us td. Hsppy tv mttt yvw"
```

```
sequence <- "Hello my would come out like: Ed. Happy to meet you"
(seq <- unlist(strsplit(sequence,"")))
```

```
##  [1] "H" "e" "l" "l" "o" " " "m" "y" " " "w" "o" "u" "l" "d" " " "c" "o" "m" "e"
## [20] " " "o" "u" "t" " " "l" "i" "k" "e" ":" " " "E" "d" "." " " "H" "a" "p" "p"
## [39] "y" " " "t" "o" " " "m" "e" "e" "t" " " "y" "o" "u"
```

```
sequence <- "Hello my name is Ed. Happy to meet you"
 seq < unlist(strsplit(sequence,""))
 for (ii in 1:length(seq)) {
     # Write code to inspect each element of seq to determine if it is
     # a candidate for changing.
}
```

### Using a while loop

The while loop is similar to the for loop. We have some expression that must be evaluated until some condition is met. This is useful when we are writing code that must converge on something.

```
sum <- 0
n <- 1000
i <- 1
while (i <= n) {
   sum <- sum + i
   i <- i + 1
}
sum
```

```
## [1] 500500
```

```
# The following is equivalent

sum <- 0
n <- 1000
for (i in 1:n) {
   sum <- sum + 1
}
sum
```

```
## [1] 1000
```

Taking the square root of a number and then taking the square root of that result and so will eventually converge to 1. We can use a while loop to do this

```
num <- 13
sqrtval <- sqrt(num)
# Loop until the sqrt value becomes equal to 1
while ( sqrtval != 1) {
  sqrtval <- sqrt(sqrtval)
  # sprintf allows us to format a variable according to a pattern
  # See http://www.cookbook-r.com/Strings/Creating_strings_from_variables/
  print(sprintf("%2.12f",sqrtval)) }
```

```
## [1] "1.898828922116"
## [1] "1.377980015137"
## [1] "1.173873934942"
## [1] "1.083454629849"
## [1] "1.040891267064"
## [1] "1.020240788767"
## [1] "1.010069695005"
## [1] "1.005022236075"
## [1] "1.002507973073"
## [1] "1.001253201280"
## [1] "1.000626404449"
```

```
## [1] "1.000313153192"
## [1] "1.000156564340"
## [1] "1.000078279106"
## [1] "1.000039138787"
## [1] "1.000019569202"
## [1] "1.000009784553"
## [1] "1.000004892265"
## [1] "1.000002446129"
## [1] "1.000001223064"
## [1] "1.000000611532"
## [1] "1.000000305766"
## [1] "1.000000152883"
## [1] "1.000000076441"
## [1] "1.000000038221"
## [1] "1.000000019110"
## [1] "1.000000009555"
## [1] "1.000000004778"
## [1] "1.000000002389"
## [1] "1.000000001194"
## [1] "1.000000000597"
## [1] "1.000000000299"
## [1] "1.000000000149"
## [1] "1.000000000075"
## [1] "1.000000000037"
## [1] "1.000000000019"
## [1] "1.000000000009"
## [1] "1.000000000005"
## [1] "1.000000000002"
## [1] "1.000000000001"
## [1] "1.000000000001"
## [1] "1.000000000000"
## [1] "1.000000000000"
## [1] "1.000000000000"
## [1] "1.000000000000"
## [1] "1.000000000000"
## [1] "1.000000000000"
## [1] "1.000000000000"
## [1] "1.000000000000"
## [1] "1.000000000000"
## [1] "1.000000000000"
```