

BIOS 545 Spring 2021 Lecture 4

Lists, Data Frames

Steve Pittard, wsp@emory.edu

Lists

Lists provide a way to store information of different types within a single data structure

Remember that vectors and matrices restrict us to only one data type at a time. That is we cannot mix, for example, characters and numbers within a vector or matrix.

Many functions in R return information stored in lists

Consider the following example wherein we store information about a family. Not all this information is of the same type

```
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
```

If possible, always create named elements. It is easier for humans to index into a named list

```
family1 <- list(husband="Fred",
                wife="Wilma",
```

```

        numofchildren=3,
        agesofkids=c(8,11,14))
# If the list elements have names then use "$" to access the element
family1$agesofkids

```

```
## [1] 8 11 14
```

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
```

```
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
```

As newcomers to R we usually doesn't create lists 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 information of varying types
- 2) As a precursor to creating a a data frame, which represents a hybrid object with characteristics of a list and a matrix

As an example of the first case, R has lots of statistical functions that return lists of information.

```
data(mtcars) # Load mtcars into the environment
mylm <- lm(mpg ~ wt + am, data = mtcars)
print(mylm)
```

```
##
## Call:
## lm(formula = mpg ~ wt + am, data = mtcars)
##
```

```
## Coefficients:
## (Intercept)          wt          am
##    37.32155      -5.35281      -0.02362
```

But there is a lot more information

```
str(mylm,attr=FALSE)
```

```
## List of 12
## $ coefficients : Named num [1:3] 37.3216 -5.3528 -0.0236
## .. attr(*, "names")= chr [1:3] "(Intercept)" "wt" "am"
## $ residuals    : Named num [1:32] -2.274 -0.909 -2.079 1.288 -0.208 ...
## .. attr(*, "names")= chr [1:32] "Mazda RX4" "Mazda RX4 Wag" "Datsun 710" "Hornet 4 Drive" ...
## $ effects      : Named num [1:32] -113.6497 -29.1157 0.0473 1.2698 -0.1748 ...
## .. attr(*, "names")= chr [1:32] "(Intercept)" "wt" "am" "" ...
## $ rank         : int 3
## $ fitted.values: Named num [1:32] 23.3 21.9 24.9 20.1 18.9 ...
## .. attr(*, "names")= chr [1:32] "Mazda RX4" "Mazda RX4 Wag" "Datsun 710" "Hornet 4 Drive" ...
## $ assign       : int [1:3] 0 1 2
## $ qr           :List of 5
## ..$ qr        : num [1:32, 1:3] -5.657 0.177 0.177 0.177 0.177 ...
## .. .. attr(*, "dimnames")=List of 2
## .. .. ..$ : chr [1:32] "Mazda RX4" "Mazda RX4 Wag" "Datsun 710" "Hornet 4 Drive" ...
## .. .. ..$ : chr [1:3] "(Intercept)" "wt" "am"
## .. .. attr(*, "assign")= int [1:3] 0 1 2
## ..$ qraux: num [1:3] 1.18 1.05 1.08
## ..$ pivot: int [1:3] 1 2 3
## ..$ tol  : num 1e-07
## ..$ rank : int 3
## .. attr(*, "class")= chr "qr"
## $ df.residual  : int 29
## $ xlevels      : Named list()
## $ call         : language lm(formula = mpg ~ wt + am, data = mtcars)
## $ terms        :Classes 'terms', 'formula' language mpg ~ wt + am
## .. .. attr(*, "variables")= language list(mpg, wt, am)
## .. .. attr(*, "factors")= int [1:3, 1:2] 0 1 0 0 0 1
## .. .. .. attr(*, "dimnames")=List of 2
## .. .. .. ..$ : chr [1:3] "mpg" "wt" "am"
## .. .. .. ..$ : chr [1:2] "wt" "am"
## .. .. attr(*, "term.labels")= chr [1:2] "wt" "am"
## .. .. attr(*, "order")= int [1:2] 1 1
## .. .. attr(*, "intercept")= int 1
## .. .. attr(*, "response")= int 1
## .. .. attr(*, ".Environment")=<environment: R_GlobalEnv>
## .. .. attr(*, "predvars")= language list(mpg, wt, am)
## .. .. attr(*, "dataClasses")= Named chr [1:3] "numeric" "numeric" "numeric"
## .. .. .. attr(*, "names")= chr [1:3] "mpg" "wt" "am"
## $ model        :'data.frame': 32 obs. of 3 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 ...
## ..$ am : num [1:32] 1 1 1 0 0 0 0 0 0 0 ...
## .. attr(*, "terms")=Classes 'terms', 'formula' language mpg ~ wt + am
## .. .. .. attr(*, "variables")= language list(mpg, wt, am)
## .. .. .. attr(*, "factors")= int [1:3, 1:2] 0 1 0 0 0 1
## .. .. .. .. attr(*, "dimnames")=List of 2
```

```
## .. .. . $ : chr [1:3] "mpg" "wt" "am"
## .. .. . $ : chr [1:2] "wt" "am"
## .. .. .- attr(*, "term.labels")= chr [1:2] "wt" "am"
## .. .. .- attr(*, "order")= int [1:2] 1 1
## .. .. .- attr(*, "intercept")= int 1
## .. .. .- attr(*, "response")= int 1
## .. .. .- attr(*, ".Environment")=<environment: R_GlobalEnv>
## .. .. .- attr(*, "predvars")= language list(mpg, wt, am)
## .. .. .- attr(*, "dataClasses")= Named chr [1:3] "numeric" "numeric" "numeric"
## .. .. .- attr(*, "names")= chr [1:3] "mpg" "wt" "am"
## - attr(*, "class")= chr "lm"
```

```
names(mylm)
```

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

```
mylm$effects
```

```
## (Intercept)      wt      am
## -113.64973741 -29.11572170  0.04733203  1.26976047 -0.17484609
##
## -0.66325556  -3.84950765  4.13027231  2.30709126  0.32515391
##
## -1.07484609   1.04025555  0.04321657 -1.77780711  1.62409672
##
##  2.59493431   6.45415172  7.39582527  2.13180232  6.85929813
##
## -2.81488433  -2.92848397 -3.70274372 -3.34303552  2.58486211
##
##  0.81725077   0.66105369  1.56269063 -3.79203411 -2.12384467
##
## -2.36022355  -0.36804941
```

```
lm(mpg ~ wt, data = mtcars)$coefficients
```

```
## (Intercept)      wt
##  37.285126   -5.344472
```

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]
```

```
## [1] "This"
```

```
mys[[1]][1:2]
```

```
## [1] "This" "is"
```

```
unlist(mys)
```

```
## [1] "This" "is"  "a"   "test"
```

When we create our own functions we can return a list

```
my.summary <- function(x) {  
  return.list <- 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
```

Remember the sapply command ? We use it to apply a function over each element of a list or a vector. This helps us avoid having to write a “for-loop” every time we want to process a list or a vector.

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

```
sapply(family1,class)
```

```
##      husband      wife numofchildren  agesofkids  
## "character" "character"  "numeric"    "numeric"
```

```
sapply(family1,length)
```

```
##      husband      wife numofchildren  agesofkids  
##          1          1          1          3
```

sapply tries to return a “simplified” version of the output (either a vector or matrix) hence the “s” in the “sapply”. If you don’t use something like sapply then the example on the previous slide would look this:

```
family1 <- list(husband="Fred",  
               wife="Wilma",  
               numofchildren=3,  
               agesofkids=c(8,11,14))  
  
for (ii in 1:length(family1)) {  
  cat(names(family1)[ii], " : ", class(family1[[ii]]), "\n")  
}
```

```
## husband : character  
## wife : character  
## numofchildren : numeric  
## agesofkids : numeric
```

Similar to sapply, the lapply function lets you “apply” some function over each element of a list or vector. It will return a list version of the output hence the “l” in the lapply. So deciding between sapply and lapply simply is a question of format. What do you want back ? A vector or list ? Most of the time I use sapply.

```
# lapply( vector_or_list, function_to_apply_to_each_element)
family1 <- list(husband="Fred",
               wife="Wilma",
               numofchildren=3,
               agesofkids=c(8,11,14))
```

```
lapply(family1,class)
```

```
## $husband
## [1] "character"
##
## $wife
## [1] "character"
##
## $numofchildren
## [1] "numeric"
##
## $agesofkids
## [1] "numeric"
```

check out the following:

```
lapply(family1,mean)
```

```
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
```

```
## Warning in mean.default(X[[i]], ...): argument is not numeric or logical:
## returning NA
```

```
## $husband
## [1] NA
##
## $wife
## [1] NA
##
## $numofchildren
## [1] 3
##
## $agesofkids
## [1] 11
```

We can write our own processing function that checks to see if the list element is valid input for the mean function.

```
my.func <- function(x) {
  if(class(x)=="numeric") {
    return(mean(x))
  }
}
lapply(family1, my.func)
```

```
## $husband
## NULL
##
## $wife
## NULL
```

```
##
## $numofchildren
## [1] 3
##
## $agesofkids
## [1] 11
```

See these videos on the lapply function at:

<https://www.youtube.com/playlist?list=PL905DXZOAgwwj16m6C3ioh6aVKDDrEiiO>

See this Blog post on lapply

<https://rollingyours.wordpress.com/2014/10/20/the-lapply-command-101/>

Data Frames

Why should you use Data Frames ?

- 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 but is much more flexible 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

Here we have 2 character vectors and 2 numeric vectors. Let's say we want to do some 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 do some summary on this

```
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"
```

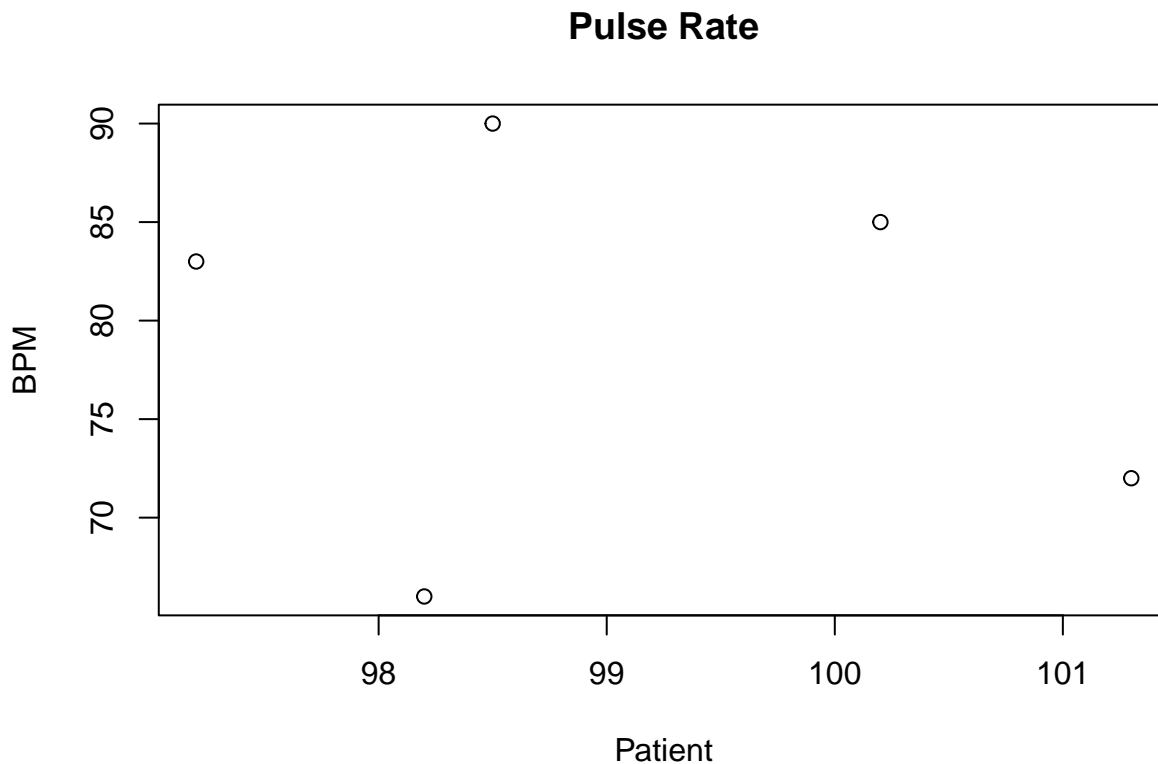
That doesn't generalize at all. Use the dataframe() function to create a data frame. It looks like a matrix but allows for mixed data types

```
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
```

So now what ?

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



```
mean(my_df[,2:3])
```

```
## Warning in mean.default(my_df[, 2:3]): argument is not numeric or logical:
```

```
## returning NA
```

```
## [1] NA
```

Once you have a data frame you could edit it with the Workspace viewer in RStudio although this doesn't generalize. Imagine if your data set had 10,000 lines ?

Builtin Example Data Frames

```
library(help="datasets")
```

```
help(mtcars)
```

```
data(mtcars)
```

```
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 ...
```

Get some info on the data frame

```
nrow(mtcars)
```

```
## [1] 32
```

```
ncol(mtcars)
```

```
## [1] 11
```

```
dim(mtcars)
```

```
## [1] 32 11
```

More information is possible

```
rownames(mtcars)
```

```
## [1] "Mazda RX4"           "Mazda RX4 Wag"       "Datsun 710"
## [4] "Hornet 4 Drive"      "Hornet Sportabout"   "Valiant"
## [7] "Duster 360"         "Merc 240D"           "Merc 230"
## [10] "Merc 280"           "Merc 280C"           "Merc 450SE"
## [13] "Merc 450SL"         "Merc 450SLC"         "Cadillac Fleetwood"
## [16] "Lincoln Continental" "Chrysler Imperial"   "Fiat 128"
## [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"
```

We can actually set the rownames using the same function

```
rownames(mtcars) <- 1:32
```

```
head(mtcars)
```

```
##      mpg cyl disp  hp drat   wt  qsec vs am gear carb
## 1  21.0   6  160 110 3.90 2.620 16.46  0  1   4    4
## 2  21.0   6  160 110 3.90 2.875 17.02  0  1   4    4
## 3  22.8   4  108  93 3.85 2.320 18.61  1  1   4    1
## 4  21.4   6  258 110 3.08 3.215 19.44  1  0   3    1
## 5  18.7   8  360 175 3.15 3.440 17.02  0  0   3    2
## 6  18.1   6  225 105 2.76 3.460 20.22  1  0   3    1
```

```
rownames(mtcars) <- paste("car",1:32,sep="_")
```

```
head(mtcars)
```

```
##      mpg cyl disp  hp drat   wt  qsec vs am gear carb
## car_1 21.0   6  160 110 3.90 2.620 16.46  0  1   4    4
## car_2 21.0   6  160 110 3.90 2.875 17.02  0  1   4    4
## car_3 22.8   4  108  93 3.85 2.320 18.61  1  1   4    1
## car_4 21.4   6  258 110 3.08 3.215 19.44  1  0   3    1
## car_5 18.7   8  360 175 3.15 3.440 17.02  0  0   3    2
## car_6 18.1   6  225 105 2.76 3.460 20.22  1  0   3    1
```

There are many ways to index into or interrogate a data frame. This is where your previous knowledge of the bracket notation will be very useful to you

```
head(mtcars[,-1])      # Get all rows / columns except for column 1
```

```
##      cyl disp  hp drat   wt  qsec vs am gear carb
## car_1   6  160 110 3.90 2.620 16.46 0  1   4   4
## car_2   6  160 110 3.90 2.875 17.02 0  1   4   4
## car_3   4  108  93 3.85 2.320 18.61 1  1   4   1
## car_4   6  258 110 3.08 3.215 19.44 1  0   3   1
## car_5   8  360 175 3.15 3.440 17.02 0  0   3   2
## car_6   6  225 105 2.76 3.460 20.22 1  0   3   1
```

Compare this to:

```
head(mtcars)
```

```
##      mpg cyl disp  hp drat   wt  qsec vs am gear carb
## car_1 21.0   6  160 110 3.90 2.620 16.46 0  1   4   4
## car_2 21.0   6  160 110 3.90 2.875 17.02 0  1   4   4
## car_3 22.8   4  108  93 3.85 2.320 18.61 1  1   4   1
## car_4 21.4   6  258 110 3.08 3.215 19.44 1  0   3   1
## car_5 18.7   8  360 175 3.15 3.440 17.02 0  0   3   2
## car_6 18.1   6  225 105 2.76 3.460 20.22 1  0   3   1
```

What about the following:

```
mtcars[,] # same as
```

```
##      mpg cyl  disp  hp drat   wt  qsec vs am gear carb
## car_1 21.0   6 160.0 110 3.90 2.620 16.46 0  1   4   4
## car_2 21.0   6 160.0 110 3.90 2.875 17.02 0  1   4   4
## car_3 22.8   4 108.0  93 3.85 2.320 18.61 1  1   4   1
## car_4 21.4   6 258.0 110 3.08 3.215 19.44 1  0   3   1
## car_5 18.7   8 360.0 175 3.15 3.440 17.02 0  0   3   2
## car_6 18.1   6 225.0 105 2.76 3.460 20.22 1  0   3   1
## car_7 14.3   8 360.0 245 3.21 3.570 15.84 0  0   3   4
## car_8 24.4   4 146.7  62 3.69 3.190 20.00 1  0   4   2
## car_9 22.8   4 140.8  95 3.92 3.150 22.90 1  0   4   2
## car_10 19.2   6 167.6 123 3.92 3.440 18.30 1  0   4   4
## car_11 17.8   6 167.6 123 3.92 3.440 18.90 1  0   4   4
## car_12 16.4   8 275.8 180 3.07 4.070 17.40 0  0   3   3
## car_13 17.3   8 275.8 180 3.07 3.730 17.60 0  0   3   3
## car_14 15.2   8 275.8 180 3.07 3.780 18.00 0  0   3   3
## car_15 10.4   8 472.0 205 2.93 5.250 17.98 0  0   3   4
## car_16 10.4   8 460.0 215 3.00 5.424 17.82 0  0   3   4
## car_17 14.7   8 440.0 230 3.23 5.345 17.42 0  0   3   4
## car_18 32.4   4  78.7  66 4.08 2.200 19.47 1  1   4   1
## car_19 30.4   4  75.7  52 4.93 1.615 18.52 1  1   4   2
## car_20 33.9   4  71.1  65 4.22 1.835 19.90 1  1   4   1
## car_21 21.5   4 120.1  97 3.70 2.465 20.01 1  0   3   1
## car_22 15.5   8 318.0 150 2.76 3.520 16.87 0  0   3   2
## car_23 15.2   8 304.0 150 3.15 3.435 17.30 0  0   3   2
## car_24 13.3   8 350.0 245 3.73 3.840 15.41 0  0   3   4
## car_25 19.2   8 400.0 175 3.08 3.845 17.05 0  0   3   2
## car_26 27.3   4  79.0  66 4.08 1.935 18.90 1  1   4   1
## car_27 26.0   4 120.3  91 4.43 2.140 16.70 0  1   5   2
## car_28 30.4   4  95.1 113 3.77 1.513 16.90 1  1   5   2
## car_29 15.8   8 351.0 264 4.22 3.170 14.50 0  1   5   4
```

```
## car_30 19.7 6 145.0 175 3.62 2.770 15.50 0 1 5 6
## car_31 15.0 8 301.0 335 3.54 3.570 14.60 0 1 5 8
## car_32 21.4 4 121.0 109 4.11 2.780 18.60 1 1 4 2
```

```
mtcars
```

```
##      mpg  cyl  disp  hp drat    wt  qsec vs  am gear carb
## car_1 21.0   6 160.0 110 3.90 2.620 16.46 0  1   4   4
## car_2 21.0   6 160.0 110 3.90 2.875 17.02 0  1   4   4
## car_3 22.8   4 108.0  93 3.85 2.320 18.61 1  1   4   1
## car_4 21.4   6 258.0 110 3.08 3.215 19.44 1  0   3   1
## car_5 18.7   8 360.0 175 3.15 3.440 17.02 0  0   3   2
## car_6 18.1   6 225.0 105 2.76 3.460 20.22 1  0   3   1
## car_7 14.3   8 360.0 245 3.21 3.570 15.84 0  0   3   4
## car_8 24.4   4 146.7  62 3.69 3.190 20.00 1  0   4   2
## car_9 22.8   4 140.8  95 3.92 3.150 22.90 1  0   4   2
## car_10 19.2   6 167.6 123 3.92 3.440 18.30 1  0   4   4
## car_11 17.8   6 167.6 123 3.92 3.440 18.90 1  0   4   4
## car_12 16.4   8 275.8 180 3.07 4.070 17.40 0  0   3   3
## car_13 17.3   8 275.8 180 3.07 3.730 17.60 0  0   3   3
## car_14 15.2   8 275.8 180 3.07 3.780 18.00 0  0   3   3
## car_15 10.4   8 472.0 205 2.93 5.250 17.98 0  0   3   4
## car_16 10.4   8 460.0 215 3.00 5.424 17.82 0  0   3   4
## car_17 14.7   8 440.0 230 3.23 5.345 17.42 0  0   3   4
## car_18 32.4   4  78.7  66 4.08 2.200 19.47 1  1   4   1
## car_19 30.4   4  75.7  52 4.93 1.615 18.52 1  1   4   2
## car_20 33.9   4  71.1  65 4.22 1.835 19.90 1  1   4   1
## car_21 21.5   4 120.1  97 3.70 2.465 20.01 1  0   3   1
## car_22 15.5   8 318.0 150 2.76 3.520 16.87 0  0   3   2
## car_23 15.2   8 304.0 150 3.15 3.435 17.30 0  0   3   2
## car_24 13.3   8 350.0 245 3.73 3.840 15.41 0  0   3   4
## car_25 19.2   8 400.0 175 3.08 3.845 17.05 0  0   3   2
## car_26 27.3   4  79.0  66 4.08 1.935 18.90 1  1   4   1
## car_27 26.0   4 120.3  91 4.43 2.140 16.70 0  1   5   2
## car_28 30.4   4  95.1 113 3.77 1.513 16.90 1  1   5   2
## car_29 15.8   8 351.0 264 4.22 3.170 14.50 0  1   5   4
## car_30 19.7   6 145.0 175 3.62 2.770 15.50 0  1   5   6
## car_31 15.0   8 301.0 335 3.54 3.570 14.60 0  1   5   8
## car_32 21.4   4 121.0 109 4.11 2.780 18.60 1  1   4   2
```

```
# Get first 5 rows and first 3 columns
```

```
mtcars[1:5,1:3]
```

```
##      mpg  cyl disp
## car_1 21.0   6 160
## car_2 21.0   6 160
## car_3 22.8   4 108
## car_4 21.4   6 258
## car_5 18.7   8 360
```

What about this ?

```
# Get all but the first 5 rows and first 3 columns
```

```
mtcars[-1:-5,-1:-3]
```

```
##      hp drat    wt  qsec vs  am gear carb
## car_6 105 2.76 3.460 20.22 1  0   3   1
```

```
## car_7 245 3.21 3.570 15.84 0 0 3 4
## car_8 62 3.69 3.190 20.00 1 0 4 2
## car_9 95 3.92 3.150 22.90 1 0 4 2
## car_10 123 3.92 3.440 18.30 1 0 4 4
## car_11 123 3.92 3.440 18.90 1 0 4 4
## car_12 180 3.07 4.070 17.40 0 0 3 3
## car_13 180 3.07 3.730 17.60 0 0 3 3
## car_14 180 3.07 3.780 18.00 0 0 3 3
## car_15 205 2.93 5.250 17.98 0 0 3 4
## car_16 215 3.00 5.424 17.82 0 0 3 4
## car_17 230 3.23 5.345 17.42 0 0 3 4
## car_18 66 4.08 2.200 19.47 1 1 4 1
## car_19 52 4.93 1.615 18.52 1 1 4 2
## car_20 65 4.22 1.835 19.90 1 1 4 1
## car_21 97 3.70 2.465 20.01 1 0 3 1
## car_22 150 2.76 3.520 16.87 0 0 3 2
## car_23 150 3.15 3.435 17.30 0 0 3 2
## car_24 245 3.73 3.840 15.41 0 0 3 4
## car_25 175 3.08 3.845 17.05 0 0 3 2
## car_26 66 4.08 1.935 18.90 1 1 4 1
## car_27 91 4.43 2.140 16.70 0 1 5 2
## car_28 113 3.77 1.513 16.90 1 1 5 2
## car_29 264 4.22 3.170 14.50 0 1 5 4
## car_30 175 3.62 2.770 15.50 0 1 5 6
## car_31 335 3.54 3.570 14.60 0 1 5 8
## car_32 109 4.11 2.780 18.60 1 1 4 2
```

```
# Get first 5 rows and the mpg and am columns
mtcars[1:5,c("mpg","am")]
```

```
##      mpg am
## car_1 21.0 1
## car_2 21.0 1
## car_3 22.8 1
## car_4 21.4 0
## car_5 18.7 0
```

The following is the same as above except we index into the columns using numbers instead of names.

```
mtcars[1:5,c(1,9)]
```

```
##      mpg am
## car_1 21.0 1
## car_2 21.0 1
## car_3 22.8 1
## car_4 21.4 0
## car_5 18.7 0
```

To find the names of all the columns

```
names(mtcars)
```

```
## [1] "mpg" "cyl" "disp" "hp" "drat" "wt" "qsec" "vs" "am" "gear"
## [11] "carb"
```

So this is kind of boring because we can interrogate the data frame to find rows and columns that satisfy certain conditions

```
# find all rows where the MPG >= 30
mtcars[mtcars$mpg >= 30,]
```

```
##      mpg cyl disp  hp drat   wt  qsec vs am gear carb
## car_18 32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## car_19 30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## car_20 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## car_28 30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
```

```
# Find all rows where the mpg >= 30 but return only columns 2-6
mtcars[mtcars$mpg >= 30.0,2:6]
```

```
##      cyl disp  hp drat   wt
## car_18   4 78.7  66 4.08 2.200
## car_19   4 75.7  52 4.93 1.615
## car_20   4 71.1  65 4.22 1.835
## car_28   4 95.1 113 3.77 1.513
```

We can have compound statements. Find all rows where the mpg is ≥ 30 and the cylinder variable is less than 6.

```
mtcars[mtcars$mpg >= 30.0 & mtcars$cyl < 6,]
```

```
##      mpg cyl disp  hp drat   wt  qsec vs am gear carb
## car_18 32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## car_19 30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## car_20 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## car_28 30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
```

Find all rows that correspond to Automatic and count them. Frequently you just want to know how many rows satisfy as certain condition

```
nrow(mtcars[mtcars$am == 0,])
```

```
## [1] 19
```

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

```
## [1] 13
```

Of course there are other ways to do this

```
table(mtcars$am)
```

```
##
##  0  1
## 19 13
```

So we can use other R functions as part of our interrogation query. Let's find all rows in the data frame where the MPG is greater than the mean MPG for the entire data set.

```
mtcars[mtcars$mpg > mean(mtcars$mpg),]
```

```
##      mpg cyl  disp  hp drat   wt  qsec vs am gear carb
## car_1  21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## car_2  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## car_3  22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## car_4  21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## car_8  24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## car_9  22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
```

```
## car_18 32.4 4 78.7 66 4.08 2.200 19.47 1 1 4 1
## car_19 30.4 4 75.7 52 4.93 1.615 18.52 1 1 4 2
## car_20 33.9 4 71.1 65 4.22 1.835 19.90 1 1 4 1
## car_21 21.5 4 120.1 97 3.70 2.465 20.01 1 0 3 1
## car_26 27.3 4 79.0 66 4.08 1.935 18.90 1 1 4 1
## car_27 26.0 4 120.3 91 4.43 2.140 16.70 0 1 5 2
## car_28 30.4 4 95.1 113 3.77 1.513 16.90 1 1 5 2
## car_32 21.4 4 121.0 109 4.11 2.780 18.60 1 1 4 2
```

Here is one which is slightly more involved. Let's find all the rows in the data frame where the MPG is greater than the 75th percentile for all MPG values in the data frame. To work this, you could break this down or do it all on one line.

```
# Looks like the 4th element return represents the 75th percentile
quantile(mtcars$mpg)
```

```
##      0%      25%      50%      75%     100%
## 10.400 15.425 19.200 22.800 33.900
```

```
quantile(mtcars$mpg)[4]
```

```
## 75%
## 22.8
```

```
mtcars[mtcars$mpg > quantile(mtcars$mpg)[4],]
```

```
##      mpg cyl  disp  hp drat   wt  qsec vs am gear carb
## car_8 24.4  4 146.7  62 3.69 3.190 20.00 1  0   4   2
## car_18 32.4  4  78.7  66 4.08 2.200 19.47 1  1   4   1
## car_19 30.4  4  75.7  52 4.93 1.615 18.52 1  1   4   2
## car_20 33.9  4  71.1  65 4.22 1.835 19.90 1  1   4   1
## car_26 27.3  4  79.0  66 4.08 1.935 18.90 1  1   4   1
## car_27 26.0  4 120.3  91 4.43 2.140 16.70 0  1   5   2
## car_28 30.4  4  95.1 113 3.77 1.513 16.90 1  1   5   2
```

Dealing with Factors

Factors are a special type of variable that we discussed last week. They can be identified by the fact that they usually assume only a small number of unique values like < 10.

```
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)
```

```
## [1] 1 0
```

This begs the question - how many unique values does each column take on ? Is there a way to get that ?

```
sapply(mtcars, function(x) length(unique(x)))
```

```
## mpg   cyl  disp    hp  drat    wt  qsec    vs  am gear carb
##   25     3   27    22   22    29   30     2   2   3   6
```

If we summarize one of these potential factors right now, R will treat it as being purely numeric which we might not want.

```
summary(mtcars$am)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.0000  0.0000  0.0000  0.4062  1.0000  1.0000
```

So this really isn't helpful since we know that the "am" values are transmission types

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

```
## Auto  Man
##    19   13
```

We can add columns to a data frame. Let's say we want to create a new column called "mpgrate" that, based on the output of the quantile command, will have a rating of the that car's MPG in terms of "horrible", "bad", "good", or "great"

The labels could be more scientific but this is still a good use case. There are a couple of ways to do this:

```
data(mtcars) # Reload a "pure" copy of mtcars
mpgrate <- cut(mtcars$mpg,
              breaks = quantile(mtcars$mpg),
              labels=c("horrible","Bad","Good","Great"),include.lowest=T)
mtcars <- cbind(mtcars,mpgrate)
head(mtcars[,10:12])
```

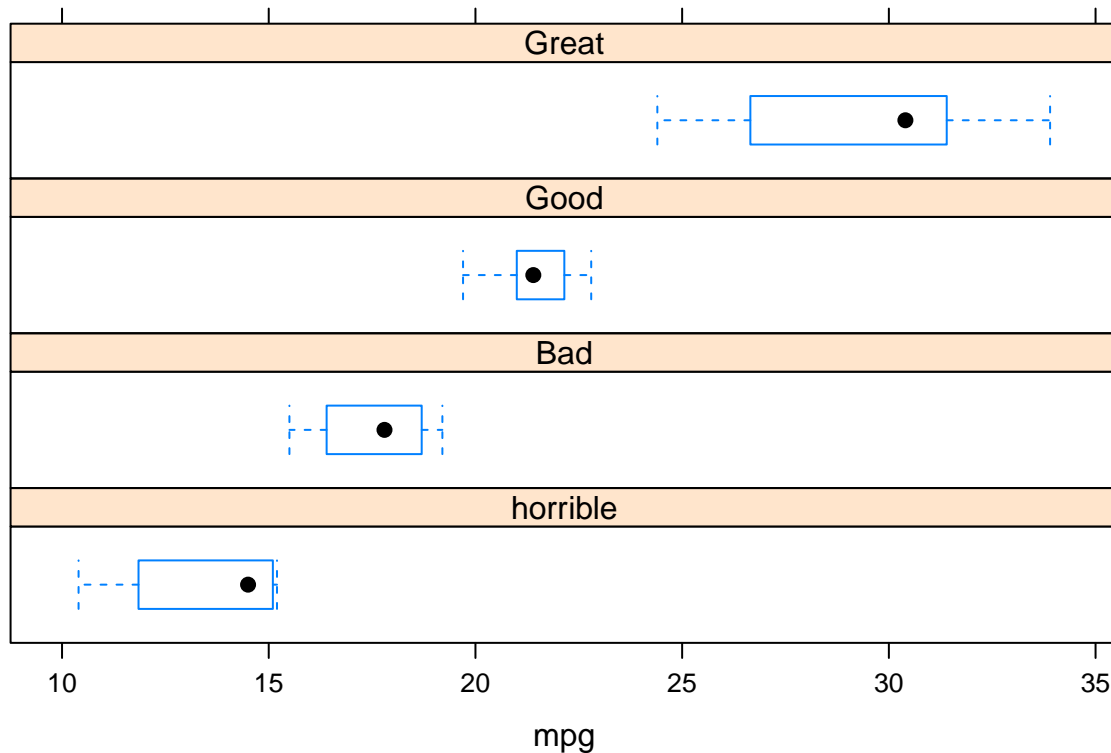
```
##                gear carb mpgrate
## Mazda RX4         4    4    Good
## Mazda RX4 Wag     4    4    Good
## Datsun 710         4    1    Good
## Hornet 4 Drive     3    1    Good
## Hornet Sportabout  3    2    Bad
## Valiant           3    1    Bad
```

We could also add a new column like:

```
mtcars$mpgrate <- mpgrate
```

Why go to all this trouble ? Because then the plotting functions know how to deal with the data more easily.

```
library(lattice)
bwplot(~mpg|mpgrate,data=mtcars,layout=c(1,4))
```



Transforming columns is a common activity.

```
transform(mtcars, wt = (wt*1000),
          qsec = round(qsec),
          am = factor(am, labels=c("A", "M")))
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb	mpg_rate
## Mazda RX4	21.0	6	160.0	110	3.90	2620	16	0	M	4	4	Good
## Mazda RX4 Wag	21.0	6	160.0	110	3.90	2875	17	0	M	4	4	Good
## Datsun 710	22.8	4	108.0	93	3.85	2320	19	1	M	4	1	Good
## Hornet 4 Drive	21.4	6	258.0	110	3.08	3215	19	1	A	3	1	Good
## Hornet Sportabout	18.7	8	360.0	175	3.15	3440	17	0	A	3	2	Bad
## Valiant	18.1	6	225.0	105	2.76	3460	20	1	A	3	1	Bad
## Duster 360	14.3	8	360.0	245	3.21	3570	16	0	A	3	4	horrible
## Merc 240D	24.4	4	146.7	62	3.69	3190	20	1	A	4	2	Great
## Merc 230	22.8	4	140.8	95	3.92	3150	23	1	A	4	2	Good
## Merc 280	19.2	6	167.6	123	3.92	3440	18	1	A	4	4	Bad
## Merc 280C	17.8	6	167.6	123	3.92	3440	19	1	A	4	4	Bad
## Merc 450SE	16.4	8	275.8	180	3.07	4070	17	0	A	3	3	Bad
## Merc 450SL	17.3	8	275.8	180	3.07	3730	18	0	A	3	3	Bad
## Merc 450SLC	15.2	8	275.8	180	3.07	3780	18	0	A	3	3	horrible
## Cadillac Fleetwood	10.4	8	472.0	205	2.93	5250	18	0	A	3	4	horrible
## Lincoln Continental	10.4	8	460.0	215	3.00	5424	18	0	A	3	4	horrible
## Chrysler Imperial	14.7	8	440.0	230	3.23	5345	17	0	A	3	4	horrible
## Fiat 128	32.4	4	78.7	66	4.08	2200	19	1	M	4	1	Great
## Honda Civic	30.4	4	75.7	52	4.93	1615	19	1	M	4	2	Great
## Toyota Corolla	33.9	4	71.1	65	4.22	1835	20	1	M	4	1	Great
## Toyota Corona	21.5	4	120.1	97	3.70	2465	20	1	A	3	1	Good
## Dodge Challenger	15.5	8	318.0	150	2.76	3520	17	0	A	3	2	Bad
## AMC Javelin	15.2	8	304.0	150	3.15	3435	17	0	A	3	2	horrible
## Camaro Z28	13.3	8	350.0	245	3.73	3840	15	0	A	3	4	horrible


```
## Pontiac Firebird      19.2   8 400.0 175 3.08 3845   17 0 A   3   2      Bad
## Fiat X1-9            27.3   4  79.0  66 4.08 1935   19 1 M   4   1     Great
## Porsche 914-2       26.0   4 120.3  91 4.43 2140   17 0 M   5   2     Great
## Lotus Europa        30.4   4  95.1 113 3.77 1513   17 1 M   5   2     Great
## Ford Pantera L      15.8   8 351.0 264 4.22 3170   14 0 M   5   4      Bad
## Ferrari Dino        19.7   6 145.0 175 3.62 2770   16 0 M   5   6     Good
## Maserati Bora       15.0   8 301.0 335 3.54 3570   15 0 M   5   8 horrible
## Volvo 142E         21.4   4 121.0 109 4.11 2780   19 1 M   4   2     Good
```

Missing Values

The NA (datum Not Available) is R's way of dealing with missing data. NAs can give you trouble unless you explicitly tell functions to ignore them. You can also pass the data through `na.omit()`, `na.exclude()`, or `complete.cases()` to insure that R handles data accordingly.

```
data <- data.frame(x=c(1,2,3,4),
                   y=c(5, NA, 8,3),
                   z=c("F", "M", "F", "M"))
```

```
data
```

```
##   x y z
## 1 1 5 F
## 2 2 NA M
## 3 3 8 F
## 4 4 3 M
```

```
na.omit(data)
```

```
##   x y z
## 1 1 5 F
## 3 3 8 F
## 4 4 3 M
```

```
data <- data.frame(x=c(1,2,3,4),
                   y=c(5, NA, 8,3),
                   z=c("F", "M", "F", "M"))
```

```
complete.cases(data)
```

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

```
sum(complete.cases(data)) # total number of complete cases
```

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

```
sum(!complete.cases(data)) # total number of incomplete cases
```

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

```
data[complete.cases(data),] # Same as na.omit(data)
```

```
##   x y z
## 1 1 5 F
## 3 3 8 F
## 4 4 3 M
```

```
url <- "https://raw.githubusercontent.com/stevieep42/bios545_spring_2021/master/DATA.DIR/hs0.csv"
```

```
data1 <- read.table(url,header=F,sep=",")
```

```
names(data1) <- c("gender","id","race","ses","schtyp","prgtype",
                 "read", "write","math","science","socst")
```

```
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 vocational  68   59  53     63     61
## 3      0  86   4  3      1   general   44   33  54     58     31
## 4      0 141   4  3      1 vocational  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
```

```
nrow(data1)
```

```
## [1] 200
```

```
sum(complete.cases(data1))
```

```
## [1] 195
```

```
sum(!complete.cases(data1))
```

```
## [1] 5
```

```
data1[!complete.cases(data1),]
```

```
##   gender  id race ses schtyp   prgtype read write math science socst
## 9      0  84   4  2      1   general   63   57  54     NA     51
## 18     0 195   4  2      2   general   57   57  60     NA     56
## 37     0 200   4  2      2 academic   68   54  75     NA     66
## 55     0 132   4  2      1 academic   73   62  73     NA     66
## 76     0   5   1  1      1 academic   47   40  43     NA     31
```

Many R functions have an argument to exclude missing values

```
data1[!complete.cases(data1),]
```

```
##   gender  id race ses schtyp   prgtype read write math science socst
## 9      0  84   4  2      1   general   63   57  54     NA     51
## 18     0 195   4  2      2   general   57   57  60     NA     56
## 37     0 200   4  2      2 academic   68   54  75     NA     66
## 55     0 132   4  2      1 academic   73   62  73     NA     66
## 76     0   5   1  1      1 academic   47   40  43     NA     31
```

```
mean(data1$science)
```

```
## [1] NA
```

```
mean(data1$science,na.rm=T)
```

```
## [1] 51.66154
```

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 <- "https://raw.githubusercontent.com/stevie42/bios545_spring_2021/master/DATA.DIR/hsb2.csv"
```

```
data1 <- read.table(url,header=T,sep=",")
head(data1)
```

```
##      id female race ses schtyp prog read write math science socst
## 1   70      0    4    1      1    1   57   52   41     47    57
## 2  121      1    4    2      1    3   68   59   53     63    61
## 3   86      0    4    3      1    1   44   33   54     58    31
## 4  141      0    4    3      1    3   63   44   47     53    56
## 5  172      0    4    2      1    2   47   52   57     53    61
## 6  113      0    4    2      1    2   44   52   51     63    61
```

Let's look at a "real" file. I got a file from this site <https://data.cityofchicago.org/> and put it on a server if you want to download it and give it a whirl.

Also, my laptop has 8GB of RAM. I suspect if you have 2GB of RAM on your laptop you will be okay but I cannot be sure. On campus it took about 30 seconds to download and read it into R.

I zipped it to make it more manageable. You can download it and read it into R using the following commands

```
url <- "https://github.com/steviepie42/bios545_spring_2021/blob/master/DATA.DIR/chi_crimes.csv.zip?raw=true"
```

```
download.file(url,destfile="chi_crimes.csv.zip")
```

```
chi <- read.csv(unzip("chi_crimes.csv.zip"),
               header=TRUE,sep="," ,stringsAsFactors = FALSE)
```

```
names(chi)
```

```
## [1] "Case.Number"      "ID"                "Date"
## [4] "Block"            "IUCR"              "Primary.Type"
## [7] "Description"      "Location.Description" "Arrest"
## [10] "Domestic"         "Beat"              "District"
## [13] "Ward"             "FBI.Code"          "X.Coordinate"
## [16] "Community.Area"   "Y.Coordinate"      "Year"
## [19] "Latitude"         "Updated.On"        "Longitude"
## [22] "Location"
```

```
sapply(chi, function(x) length(unique(x)))
```

```
##      Case.Number      ID      Date
##      334114      334139      121484
##      Block      IUCR      Primary.Type
##      28383      358      30
##      Description Location.Description      Arrest
##      296      120      2
##      Domestic      Beat      District
##      2      302      25
##      Ward      FBI.Code      X.Coordinate
##      51      30      60704
##      Community.Area      Y.Coordinate      Year
##      79      89895      1
##      Latitude      Updated.On      Longitude
##      180396      1311      180393
##      Location
##      178534
```

```
# Make the date a "real date"
```

```
chi$Date <- strptime(chi$Date,"%m/%d/%Y %r")
```

```
chi$month <- months(chi$Date)
```

```
chi$month <- factor(chi$month,levels=c("January","February","March",
                                       "April","May","June","July","August", "September",
```

```

      "October", "November", "December"), ordered=TRUE)

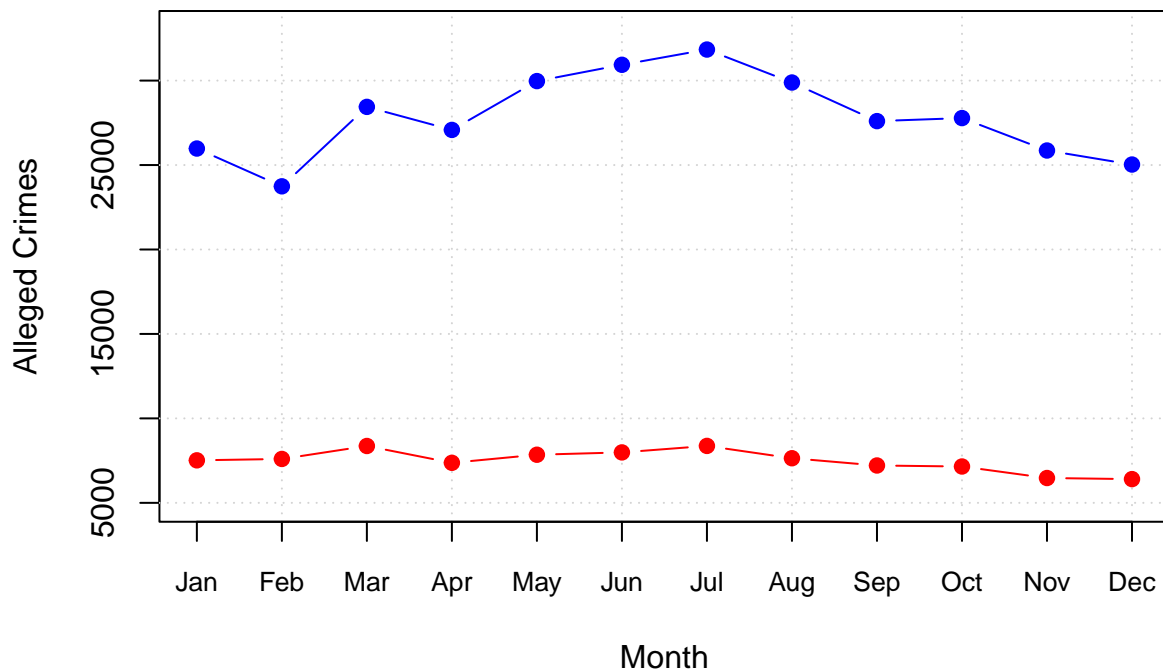
# Okay how many crimes were committed in each Month of the year ?
plot(1:12, as.vector(table(chi$month)),
     type="n", xaxt="n", ylab="Alleged Crimes",
     xlab="Month", main="Chicago Crimes in 2012 by Month", ylim=c(5000, 33000))
grid()

axis(1,
     at=1:12,
     labels=as.character(sapply(levels(chi$month), function(x) substr(x, 1, 3))), cex.axis=0.8)

points(1:12, as.vector(table(chi$month)), type="b", pch=19, col="blue")
points(1:12, as.vector(table(chi$month, chi$Arrest)[, 2]),
     col="red", pch=19, type="b")

```

Chicago Crimes in 2012 by Month



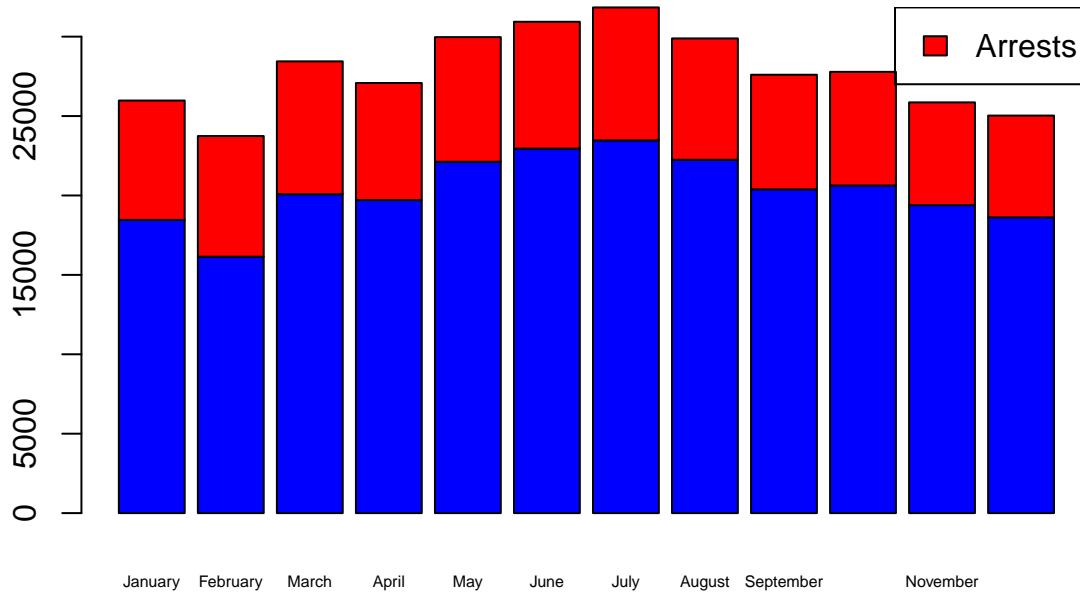
```

barplot(table(chi$Arrest, chi$month),
        col=c("blue", "red"),
        cex.names=0.5,
        main="Chicago: Reported Crimes vs. Actual Arrests")

legend("topright", c("Arrests"), fill="red")

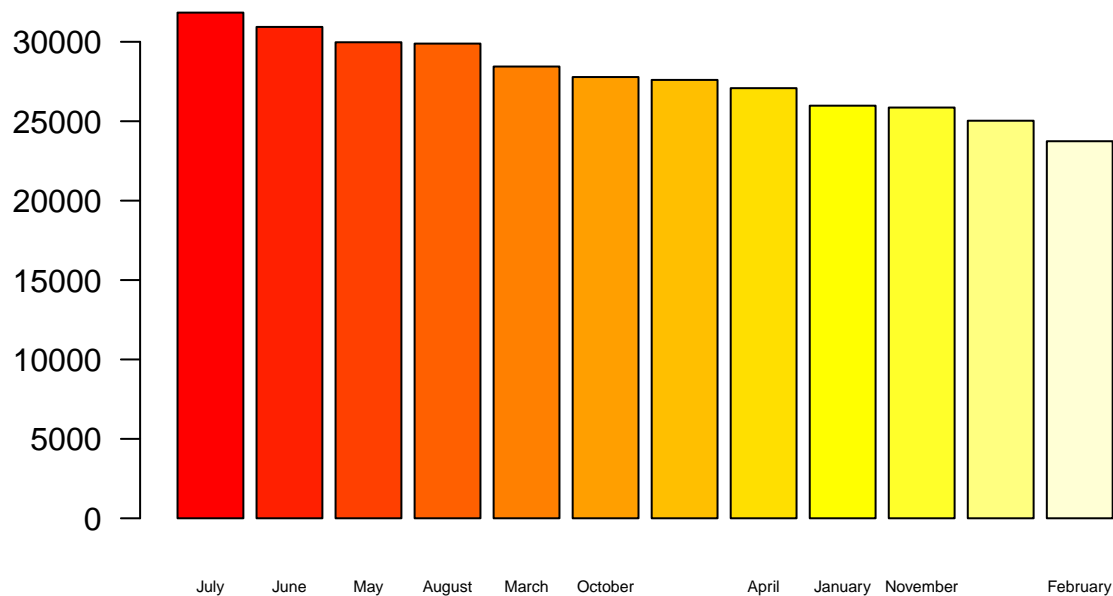
```

Chicago: Reported Crimes vs. Actual Arrests



```
# Even easier to do
hold <- rev(sort(table(chi$month)))
barplot(rev(sort(hold)),horiz=F,
        las=1,cex.names=0.5,col=heat.colors(12),
        main="Chicago: Reported Crimes in 2012 by Month")
```

Chicago: Reported Crimes in 2012 by Month



```
# Find out number of alleged crimes by type
categories <- rev(sort(sapply(unique(as.character(chi$Primary.Type)), function(x) { nrow(chi[chi$Primary.Type == x,]) })))
categories <- rev(sort(table(chi$Primary.Type)))
```

```
barplot(categories,horiz=F,las=1,cex.names=0.6,col=heat.colors(30), las=2, main="Chicago: Types of Crimes Reported")
```

Chicago: Types of Crimes Reported

