Lists - Intro

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

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

* Many functions in R return lists.

Lists - Functions

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

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

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

print(mylm)

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

Coefficients:
(Intercept)           wt
     37.285       -5.344

# But there is a lot more information

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

Lists - Functions

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

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

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

Lists - Functions

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

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

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

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

Lists - Functions

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

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

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

     return(return.list)
}
my.summary(1:10)

$mean
[1] 5.5

$sd
[1] 3.02765

$var
[1] 9.166667

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

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

Lists - Functions

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

```
mystring = "This is a test"

mys = strsplit(mystring, " ")

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

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

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

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

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

Lists - Creating

In C and C++ lists are directly comparable to "struct" types.

```
struct database {
  int id_number;
  int age;
  float salary;
};

int main()
{
  database employee;  //There is now an employee variable that has modifiable
                      // variables inside it.
  employee.age = 22;
  employee.id_number = 1;
  employee.salary = 12000.21;
}
```

The equivalent in R would be:

```
employee = list(id_number = 1, age = 22, salary = 12000.21)

str(employee)
List of 3
 $ id_number: num 1
 $ age      : num 22
 $ salary   : num 12000
```

Lists - Creating

Not to jump ahead but if we have a collection of similar lists like this:

```
employee1 = list(id_number = 1, age = 22, salary = 12000.21)

employee2 = list(id_number = 2, age = 32, salary = 13000.00)

employee3 = list(id_number = 3, age = 40, salary = 90000.00)
```

Then we can easily collect them into a larger "master" list:

```
emp_database = list(employee1, employee2, employee3)
```

Better yet we can collect the master list into a data frame....

```
my.df = do.call(rbind,emp_database)
     id_number age salary
[1,] 1         22  12000.21
[2,] 2         32  13000
[3,] 3         40  90000
```

But we'll get to that later.

Lists - Creating

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

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

family1
$husband
[1] "Fred"

$wife
[1] "Wilma"

$numofchildren
[1] 3

$agesofkids
[1]  8 11 14

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

Lists - Indexing

Note that this list has names. This makes life much easier.

```
names(family1)
[1] "husband"       "wife"          "numofchildren" "agesofkids"

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

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

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

```
family1 = list("Fred", "Wilma", 3, c(8,11,14))

family1
[[1]]
[1] "Fred"

[[2]]
[1] "Wilma"

[[3]]
[1] 3

[[4]]
[1]  8 11 14
```

Lists - Indexing

If you anticipate writing programs that "consume" lists then its better to work with numeric access than it is name access.

```
family1 = list("Fred", "Wilma", 3, c(8,11,14))

family1[1]    # So we get back the list element number as well as the element's value
[[1]]
[1] "Fred"

family1[[1]]  # Oh so the double bracket is more specific - we get just the element value
[1] "Fred"


family1[[4]][1:2]  # With respect to the 4th element show the first two values of the vector
[1]  8 11
```

Lists - Indexing

Even if your list has names for its elements then you can still use numeric access. However, if your list elements have no names then you cannot use name access unless you first apply some names to the list.

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

family1$agesofkids[2:3]
[1] 11 14

family1[[4]][2:3]
[1] 11 14
```

So if you create the list without named elements you have no choice except to use numbers

```
family1 = list("Fred", "Wilma", 3, c(8,11,14))
```

But we can always name the list elements after the fact:

```
names(family1) = c("husband","wife","numofchildren","agesofkids")

family1$husband
[1] "Fred"
```

Lists - Converting to a Vector

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

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


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

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

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

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

Lists - Adding Elements

You can add elements to a list a couple of different ways. 1) By name (the easiest way)

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

family1$numofpets = 2

family1
$husband
[1] "Fred"

$wife
[1] "Wilma"

$numofchildren
[1] 3

$agesofkids
[1]  8 11 14

$numofpets
[1] 2
```

Lists - Adding Elements

You can add elements to a list a couple of different ways. 2) By element number

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

family1[5] = 2

family1
$husband
[1] "Fred"

$wife
[1] "Wilma"

$numofchildren
[1] 3

$agesofkids
[1]  8 11 14

[[5]]
[1] 2

newnames = c(names(family1),"numofpets")
```

Lists - Deleting Elements

You can remove elements from a list by setting that element to a value of **NULL**

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

family1$wife = NULL

$husband
[1] "Fred"

$numofchildren
[1] 3

$agesofkids
[1]  8 11 14

# OR USE ELEMENT NUMBER IF YOU WISH

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

family1[2] = NULL

$husband
[1] "Fred"

$numofchildren
[1] 3

$agesofkids
[1]  8 11 14
```

Lists - Digging Deeper with sapply

```
sapply( vector_or_list, function_to_apply_to_each element)

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
```

Similar to apply, the sapply function let's you "apply" some function over a range of values. In this case each element of the list or vector provided. It tries to return a "simplified" version of the output (either a vector or matrix) hence the "s" in the "sapply". Like apply, it allows you to avoid having to write a for loop. Don't do the following unless you have a very good reason.

```
for (ii in 1:length(family1)) {
   cat(names(family1)[ii]," : ",class(family1[[ii]]),"\n")
}

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

Lists - Digging Deeper with lapply

```
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"
```

Similar to sapply, the lapply function let's you "apply" some function over a range of values. In this case each element of the list or vector provided. 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.

Lists - Digging Deeper with lapply

```
lapply(family1,mean)
$husband
[1] NA

$wife
[1] NA

$numofchildren
[1] 3

$agesofkids
[1] 11

Warning messages:
1: In mean.default(X[[1L]], ...) :
  argument is not numeric or logical: returning NA
2: In mean.default(X[[2L]], ...) :
  argument is not numeric or logical: returning NA
>
```

Lists - Digging Deeper with lapply

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


lapply(Family, my.fun)
$husband
NULL

$wife
NULL

$num.of.children
[1] 3

$child.ages
[1] 6.67
```

Lists - Twitter

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

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

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

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

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

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

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

..
```

Lists - Twitter

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

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

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

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

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

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

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

Dataframes

Dataframes - "Chapter Check-In"

Activity	Solution
Creating	read.table, data.frame, as.data.frame (to convert matrices)
Editing	Workspace viewer in RStudio
Meta Info:	rownames, names, nrow, ncol, sapply
Indexing:	Use bracket notation, subset command, or split command
Transform:	Use transform command, rbind, cbind, or $ notation to create new columns
Missing Values:	Use complete.cases to find only complete cases
Combining:	Use cbind, rbind, or merge
Summarizing:	Use summary, colmeans, rowmeans, (make sure you are dealing with numeric)
Factors:	Use factor command or leave as character until you need the factor
Sort:	Use the order function or rank function

Dataframes - Creating

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

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

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

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

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

Dataframes

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

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

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

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

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

Dataframes

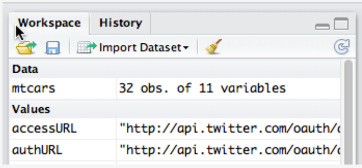
A data frame can be regarded as a matrix with columns possibly of differing modes and attributes. It may be displayed in matrix form, and its rows and columns extracted using matrix indexing conventions. Let's create a data frame:

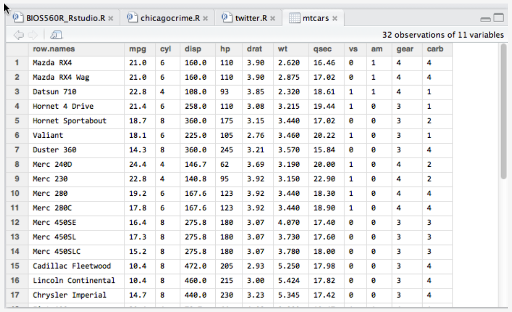
```
names=c("P1","P2","P3","P4","P5")
temp=c(98.2,101.3,97.2,100.2,98.5)
pulse=c(66,72,83,85,90)
gender=c("M","F","M","M","F")

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

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

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

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

Dataframes

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

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





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

If you are using the basic R console then you can type:

```
fix(mtcars)   # or whatever dataframe you are editing
```

Dataframes

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

```
library(help="datasets")

# Gives detailed descriptions on available data sets

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

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

Dataframes

```
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 ...

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

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

sapply(mtcars, class)  # Equivalent to abovestr command
```

Dataframes

There are a number of functions that provide metadata about a data frame.

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


rownames(mtcars) = 1:32

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

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

Dataframes

There are a number of functions that provide info about a data frame.

```
> summary(mtcars)
      mpg             cyl             disp             hp       
 Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0  
 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5  
 Median :19.20   Median :6.000   Median :196.3   Median :123.0  
 Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7  
 3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0  
 Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0  
      drat             wt             qsec             vs        
 Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000  
 1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000  
 Median :3.695   Median :3.325   Median :17.71   Median :0.0000  
 Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375  
 3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000  
 Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000  
       am              gear            carb      
 Min.   :0.0000   Min.   :3.000   Min.   :1.000  
 1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000  
 Median :0.0000   Median :4.000   Median :2.000  
 Mean   :0.4062   Mean   :3.688   Mean   :2.812  
 3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000  
 Max.   :1.0000   Max.   :5.000   Max.   :8.000  
```

Dataframes

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

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

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

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

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

Dataframes

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

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

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

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

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

Dataframes

Find all rows that correspond to Automatic and Count them

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

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

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

Dataframes

Extract all rows whose MPG value exceeds the mean MPG for the entire data frame

```
> mtcars[mtcars$mpg > mean(mtcars$mpg),]
                mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    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
>
```

Dataframes

```
# Find the quartiles for the MPG vector

quantile(mtcars$mpg)
    0%    25%    50%    75%   100% 
10.400 15.425 19.200 22.800 33.900 

# Now find the cars for which the MPG exceeds the 75% value:

mtcars[mtcars$mpg > quantile(mtcars$mpg)[4],]
                mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  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
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
```

Dataframes

There is an alternative to the bracket notation. It is called the subset function.

```
subset(mtcars, mpg >= 30.0)   # Get all records with MPG > 30.0

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

subset(mtcars, mpg >= 30.0, select=c(mpg:drat) )  # Get just columns mpg-drat

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

subset(mtcars, mpg >= 30.0 & cyl < 6 ) # Get all records with MPG >=30 and cyl <6

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

Dataframes

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

```
url = "http://steviep42.bitbucket.org/bios560rs2014/DATA.DIR/hsb2.csv"

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

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

Dataframes

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

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

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

Dataframes

See how many unique values each columns takes on. Potential factors are in red.

```
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, the summary function 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 Manu
  19   13
```

Dataframes

And we can do some aggregation and summary directly on the data frame.

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

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

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

We will investigate some more powerful aggregation functions in a later session

Dataframes

We can also easily add columns to a data frame. Let's say we have a 32 element vector called "myrate" that we want to put into our data frame. "G","B","O" stands for "Good","Bad","Okay". There are a couple of ways to do this:

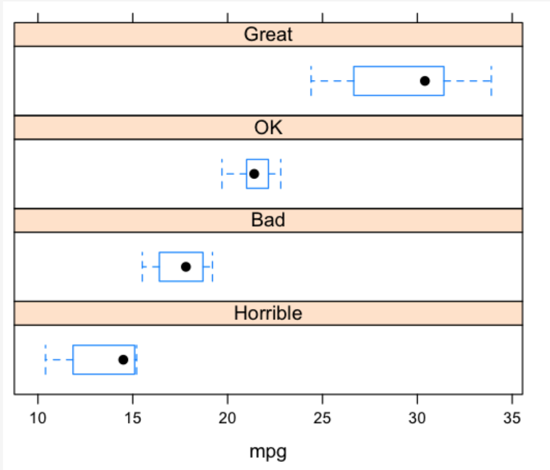
```
myrate = c("B","G","G","G","B","G","G","G","B","O","B","O","B","B","O","G","B","G","G",
           "G","B","G","B","B","G","B","O","B","B","O","B","O")

mtcars = cbind(mtcars,myrate)
                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb myrate
Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4    B
Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4    G
Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1    G
Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1    G

-OR more simply-

mtcars$myrate = myrate    # The column just shows up
```

Dataframes

We can also apply our knowledge of the cut command to assign categories, for example, that rate the MPG of the cars. This can be useful.

```
data(mtcars)   # Reload a "pure" copy of mtcars

mpgrate = cut(mtcars$mpg,
              breaks = quantile(mtcars$mpg),
              labels=c("Horrible","Bad","OK","Great"),include.lowest=T)

mtcars = cbind(mtcars,mpgrate)

head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb mpgrate
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4      OK
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4      OK
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1      OK
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1      OK
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2     Bad
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1     Bad

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

Dataframes



Dataframes

You can also use the **transform()** command to change the types/classes of the columns

```
head(mtcars)

                   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
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1


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
Mazda RX4           21.0   6 160.0 110 3.90 2620   16  0  M    4    4
Mazda RX4 Wag       21.0   6 160.0 110 3.90 2875   17  0  M    4    4
Datsun 710          22.8   4 108.0  93 3.85 2320   19  1  M    4    1
Hornet 4 Drive      21.4   6 258.0 110 3.08 3215   19  1  A    3    1
Hornet Sportabout   18.7   8 360.0 175 3.15 3440   17  0  A    3    2
```

Dataframes - 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, or pass the data through na.omit() (drop all NAs in the data), na.exclude() or complete.cases(). In some cases you may wish to give the NAs a specific value. Use **na.omit()** to eliminate missing data from a data set.

```
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        # Note missing value
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
```

Dataframes - missing values

```
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
```

Dataframes - missing values

```
url = "http://homepages.wmich.edu/~hgv7680/data/SAS/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, n=3)
  gender  id race ses schtyp  prgtype read write math science socst
1      0  70    4   1      1  general   57    52   41      47    57
2      1 121    4   2      1   vocati   68    59   53      63    61
3      0  86    4   3      1  general   44    33   54      58    31

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
```

Dataframes - missing values

Many R functions have a way 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
```

Supplemental Dataframes - more involved

Missing values can be set by using correlations between variables or by using the most frequent value for that column, mean or median, similarity or correlations with other variables. There are other possibilities of course.

```
# Using the median

data1$science = ifelse(is.na(data1$science),
                  median(data$science,na.rm=T),data1$science)
```

Sometimes we can look to see if the variable that has missing values is strongly correlated with another variable, which, in turn, could be used to predict a value.

```
cor(data1[,c(7:11)],use="complete.obs")

             read     write      math   science     socst
read    1.0000000 0.5967765 0.6622801 0.3665406 0.6214843
write   0.5967765 1.0000000 0.6174493 0.4160699 0.6047932
math    0.6622801 0.6174493 1.0000000 0.3635822 0.5444803
science 0.3665406 0.4160699 0.3635822 1.0000000 0.3239351
socst   0.6214843 0.6047932 0.5444803 0.3239351 1.0000000
```

Supplemental Dataframes - more involved

The strongest correlation for science and another variable is 0.41, which corresponds to writing. This isn't so strong actually but it's the best we have here.

```
             read     write      math   science     socst
read    1.0000000 0.5967765 0.6622801 0.3665406 0.6214843
write   0.5967765 1.0000000 0.6174493 0.4160699 0.6047932
math    0.6622801 0.6174493 1.0000000 0.3635822 0.5444803
science 0.3665406 0.4160699 0.3635822 1.0000000 0.3239351
socst   0.6214843 0.6047932 0.5444803 0.3239351 1.0000000

> ( my.lm = lm(science ~ write,data1) )

Call:
lm(formula = science ~ write, data = data1)

Coefficients:
(Intercept)        write
    20.7840       0.5601
```

Assuming this mode is any good (and that is a very big "if") then we could use the equation
missing_science_val = 0.56*write + 20.7840

Supplemental Dataframes - more involved

Assuming this mode is any good (and that is a very big "if") then we could use the equation:

missing_science_val = 0.56*write + 20.7840

```
> summary(my.lm)
Call:
lm(formula = science ~ write, data = data1)

Residuals:
    Min      1Q  Median      3Q     Max 
-56.512  -4.060   0.350   6.698  28.251 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  20.7841     4.6645   4.456 1.40e-05 ***
write         0.5601     0.0870   6.438 8.94e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 11.63 on 198 degrees of freedom
Multiple R-squared: 0.1731,    Adjusted R-squared: 0.1689 
F-statistic: 41.45 on 1 and 198 DF,  p-value: 8.937e-10
```

Supplemental Dataframes - more involved

Assuming this mode is any good (and that is a very big “if”) then we could use the equation:

missing_science_val = 0.56*write + 20.7840

```
> my.write.vals = data1[!complete.cases(data1),"write"]

> my.write.vals
[1] 57 57 54 62 40

> my.fit = predict(my.lm,data.frame(write=my.write.vals),interval="predict")
       fit      lwr      upr
1 52.71156 29.70278 75.72033
2 52.71156 29.70278 75.72033
3 51.03116 28.03285 74.02948
4 55.51222 32.46047 78.56396
5 43.18932 20.08776 66.29087

> my.pred.science = predict(my.lm,data.frame(write=my.write.vals),interval="predict")

> my.pred.science[,1]
       1        2        3        4        5
52.71156 52.71156 51.03116 55.51222 43.18932
```

Supplemental Dataframes - more involved

Assuming this mode is any good (and that is a very big "if") then we could use the equation:
missing_science_val = 0.56*write + 20.7840

```
> ( for.replace = which(!complete.cases(data1)) )
[1]  9 18 37 55 76

> data1[ for.replace ,]
   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


> data1[ for.replace, ]$science = my.fit[,1]


> data1[ for.replace, ]
   gender  id race ses schtyp  prgtype read write math  science socst
9       0  84    4   2      1  general   63    57   54 52.71156    51
18      0 195    4   2      2  general   57    57   60 52.71156    56
37      0 200    4   2      2 academic   68    54   75 51.03116    66
55      0 132    4   2      1 academic   73    62   73 55.51222    66
76      0   5    1   1      1 academic   47    40   43 43.18932    31
```

Supplemental - Chicago Crime

The City of Chicago let's you download lots of different data for analysis.

https://data.cityofchicago.org/

Supplemental - Chicago Crime

I've put this on the class server if you want to download it and give it a whirl. This a about 82 MB so don't try reading it over a home-based connection. Also, my laptop has 4GB 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 1 minute for R to process it.

```
url = "http://steviep42.bitbucket.org/bios560rs2014/DATA.DIR/chi_crimes.csv"

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

I tried reading this file into Excel. While it ultimately loaded the file it took a long time and response was very slow on my laptop. Part of the problem is that Excel loads the whole thing for purposes of display when in reality it might not be necessary to see everything. In fact with 300K records it is impractical to want to see every record.

Supplemental - Chicago Crime

A better approach is to first download the file to your computer using the "download.file" function and then using read.table. This way R won't simultaneously be downloading and reading the file which can sometimes cause trouble.

```
url = "http://steviep42.bitbucket.org/bios560rs2014/DATA.DIR/chi_crimes.csv"

download.file(url,"chi_crimes.csv")
trying URL 'http://steviep42.bitbucket.org/bios560rs2014/DATA.DIR/
chi_crimes.csv'
Content type 'text/csv' length 85753091 bytes (81.8 Mb)
opened URL
==================================================
downloaded 81.8 Mb
```

Supplemental - Chicago Crime

So I downloaded a .CSV file containing data for all reported crimes in the 2012 year.

```
system("ls -lh chi*")
-rw-r--r--@ 1 fender  staff    82M Sep 13 06:20 chi_crimes.csv

system("wc -l chi*")         # 334,142 lines !!
  334142 chi_crimes.csv

# It takes about 25 seconds to read this in on my laptop

system.time(mychi <- read.table("chi_crimes.csv",header=T,sep=","))
   user  system elapsed
 25.026   0.323  25.417

nrow(mychi)
[1] 334141

ncol(mychi)
[1] 22
```

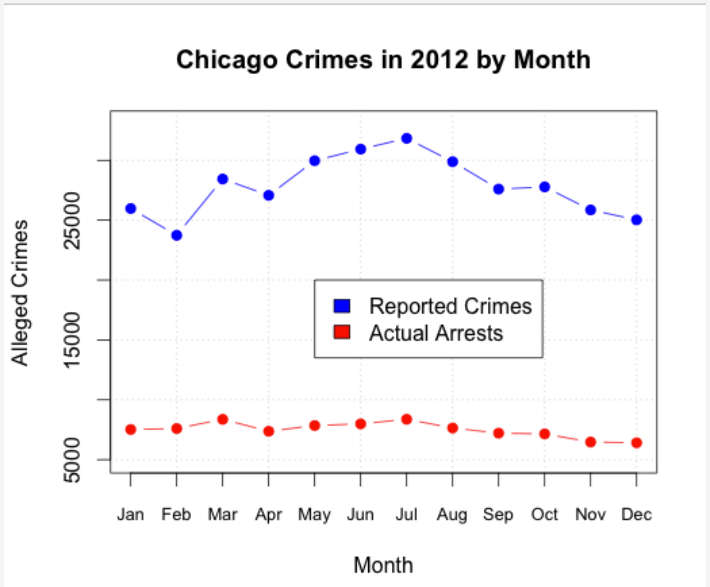
Supplemental - Chicago Crime

```
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"
[23] "month"

sapply(chi, function(x) length(unique(x)))
         Case.Number                   ID                 Date
              334114               334139               121480
               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                month
              178534                   12
```

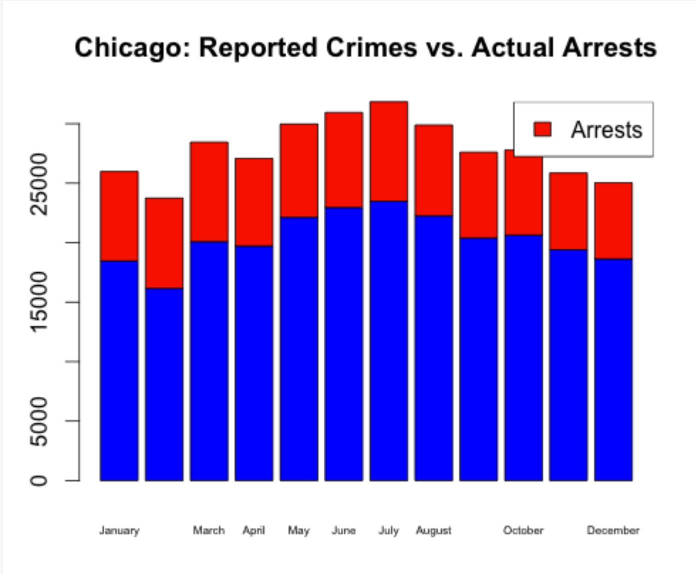
Supplemental - Chicago Crime

```
chi$Date = strptime(chi$Date,"%m/%d/%Y %r") # Change Dates from factor to a "real" Date
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")
legend(5,20000,c("Reported Crimes","Actual Arrests"),fill=c("blue","red"))

# Might look better in a barplot

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")


# Even easier to do

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")

# Looks like the Summer is when more crimes are committed
```

Supplemental - Chicago Crime

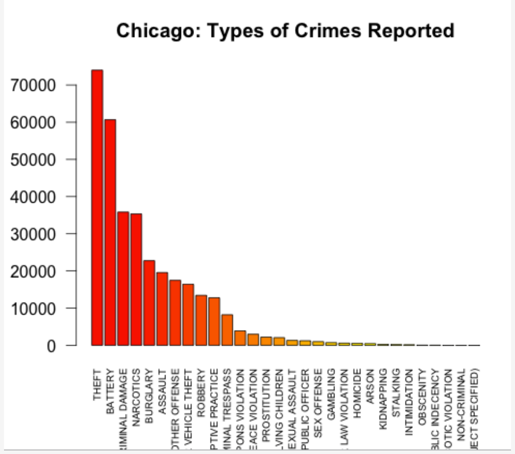


Supplemental - Chicago Crime



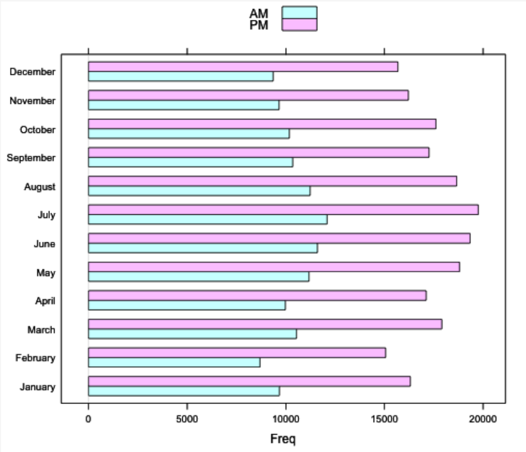
Supplemental - Chicago Crime

```
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")
```



Supplemental - Chicago Crime

```
library(lattice)
barchart(table(chi$month,chi$ampm),stack=FALSE,auto.key=T,freq=F)
```



Supplemental - Chicago Crime

```
Let's map some of these reported crimes

# Let's zone in on the reported gambling offenses
# Most of these are for Dice games. Let's see the ones that are Gambling but not dice
related

hold = chi[chi$Primary.Type == "GAMBLING",]
hold = chi[chi$Primary.Type == "GAMBLING" & chi$Description != "GAME/DICE",]

nrow(hold) # How many non-Dice related gambling offenses were there ?

# About 26 I think
# Let's plot them on a map

library(googleVis)  # This is an addon package you must install

hold$LatLon = paste(hold$Latitude,hold$Longitude,sep=":")
hold$Tip = paste(hold$Description,hold$Locate.Description,hold$Block,"<BR>",sep=" ")
```

To learn more about using googleVis see my blog entries:

http://rollingyours.wordpress.com/2013/03/20/geocoding-r-and-the-rolling-stones-part-1/

http://rollingyours.wordpress.com/2013/03/20/geocodingr-and-the-rolling-stones-part-2/

Supplemental - Chicago Crime