

BIOS 545 Week 3, Lecture 1

Department of Biostatistics and Bioinformatics

Steve Pittard wsp@emory.edu

January 25, 2016

Lists



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

Lists

```
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 - Creating

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
```

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

Lists - Creating

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
```

```
str(family2)
```

```
List of 4
```

```
$ : chr "Barney"
```

```
$ : chr "Betty"
```

```
$ : num 2
```

```
$ : num [1:2] 4 6
```

Lists - Creating

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

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

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

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

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

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

Lists - Uses

As newcomers to R we usually create lists in two cases:

- As a precursor to creating a data frame, which represents a hybrid data structure with characteristics of a list, matrix, and vectors.
- 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.

R does this all of the time by returning list structures from statistical modeling functions.

Lists - Functions

R has lots of statistical functions that return lists of information.

```
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

```
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

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 - Functions

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
```

```
names(my.summary(1:10))
```

```
[1] "mean" "sd"   "var"
```

```
my.summary(1:10)$var
```

```
[1] 9.166667
```

Lists - sapply/lapply

As with the `apply` command for matrices, there is a command(s) that will allow us to process each element of a list. This helps us avoid having to write a “for-loop” every time we want to process a list.

```
# 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

Lists - `sapply/lapply`

sapply tries to return a "simplified" version of the output (either a vector, list, or a 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:

```
# sapply( vector_or_list, function_to_apply_to_each element)

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")
}
```

More involved than just doing

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

Lists - sapply/lapply

- Similar to **sapply**, the **lapply** function let's you “apply” some function over each element of a list or vector. (In reality the sapply is a “wrapper” for the lapply command).
- 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 - sapply/lapply

```
sapply(family1,mean)  
$husband  
NULL
```

```
$wife  
NULL
```

```
$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 - sapply/lapply

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

```
sapply(family1, my.fun)  
$husband  
NULL
```

```
$wife  
NULL
```

```
$num.of.children  
[1] 3
```

```
$child.ages  
[1] 11
```

See these videos on the lapply function at:

<https://www.youtube.com/playlist?list=PL905DXZ0Agwwj16m6C3ioh6aVKDDrEii0>

See this Blog post on lapply

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

Data Frames



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

Why 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

Why Use Data Frames ?

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 write a for loop to get information for each patient

```
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"
```

Why Use Data Frames ?

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
```

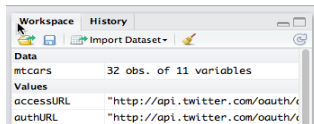
	names	temp	pulse	gender
1	P1	98.2	66	M
2	P2	101.3	72	F
3	P3	97.2	83	M
4	P4	100.2	85	M
5	P5	98.5	90	F

```
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
```


Why Use Data Frames ?

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 ?

```
data(mtcars) # Load the builtin mtcars dataframe
```

The screenshot shows the RStudio Environment pane. At the top, there are tabs for 'BIOS545RStudio.R', 'chicagoCrime.R', 'twitter.R', and 'mtcars'. The main area of the pane shows a table with 32 observations of 11 variables. The table has columns for row.names, mpg, cyl, disp, hp, drat, wt, qsec, vs, am, gear, and carb. The data is sorted by row number from 1 to 17.

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	17.82	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	3.68	3.215	19.44	1	0	3	1
5 Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.82	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	18.90	0	0	3	3
15 Cadillac Fleetwood	16.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
16 Lincoln Continental	16.4	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

Data Frames - Builtin

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
```

Data Frames - Builtin

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.

```
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
```

Data Frames - Getting/Setting Info

```
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

Data Frames - Accessing

There are various ways to select, remove, or exclude rows and columns

```
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

Data Frames - Accessing

There are various ways to select, remove, or exclude rows and columns

```
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 columns 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

Data Frames - Accessing

There are various ways to select, remove, or exclude rows and columns

```
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

Data Frames - Interrogating

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
```


Data Frames - Interrogating

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

Data Frames - Interrogating

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

```
# 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

Data Frames - Interrogating

What columns appear to be factors ? Variables with only a “few” different unique values perhaps ?

```
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
```

Data Frames - Factors

See how many unique values each column takes on

```
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	Manu
19	13

Data Frames - Factors

We can add columns to a data frame. Let's say we want to create a new column called "mpggrade" 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", "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

mpggrade <- cut(mtcars$mpg,
               breaks = quantile(mtcars$mpg),
               labels=c("horrible", "Bad", "Good", "Great"), include.lowest=T)

mtcars <- cbind(mtcars, mpggrade)

-OR-

mtcars$mpggrade <- mpggrade    # The column just magically appears !
```

Data Frames - Factors

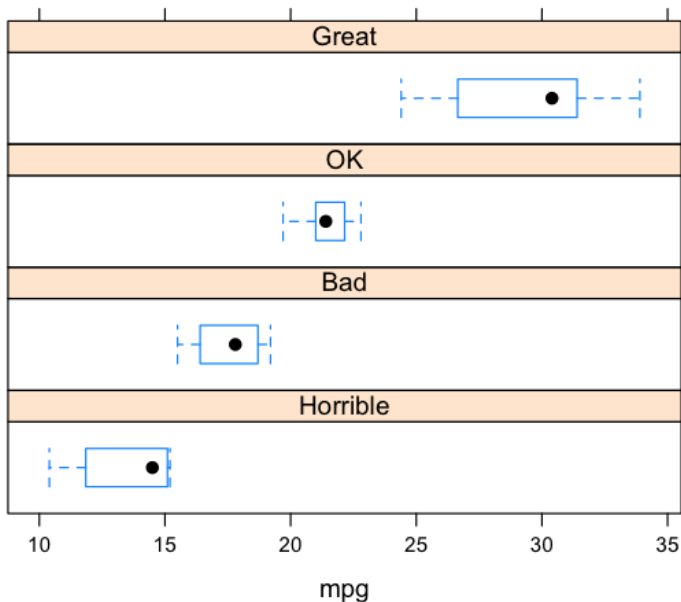
```
head(mtcars)
```

	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	Good
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4	Good
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1	Good
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1	Good
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))
```

Data Frames - Factors



Data Frames - transform()

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

Data Frames - 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      # 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
```

Data Frames - Missing values

```
.  
  
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
```

Data Frames - Missing values

```
.  
  
url <- "http://steviep42.bitbucket.org/bios545r_2016/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, n=1)
```

	gender	id	race	ses	schtyp	prgtype	read	write	math	science	socst
1	0	70	4	1	1	general	57	52	41	47	57

```
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

Data Frames - Missing values

- 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
```

Data Frames - Reading CSV

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://stevie42.bitbucket.org/bios545r_2016/DATA.DIR/hsb2.csv"
```

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

```
head(data1)
```

	gender	id	race	ses	schttyp	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

Data Frames - Reading CSV

A Recent Bug in the XML Package means the following will not work at this point Look at the follow info on World Populations

http://en.wikipedia.org/wiki/World_population

We want to get the table that looks like the following, which seems to be the 4th table on the page

The 10 countries with the largest total population:

Rank	Country / Territory	Population	Date	% of world population	Source
1	 China ^[note 2]	1,353,430,000	October 1, 2012	19.2%	^[71]
2	 India	1,210,193,422	March 2011	17%	^[72]
3	 United States	314,490,000	October 1, 2012	4.47%	^[73]
4	 Indonesia	238,400,000	May 2010	3.33%	^[74]
5	 Brazil	197,067,000	October 1, 2012	2.8%	^[75]
6	 Pakistan	180,819,000	October 1, 2012	2.57%	^[76]
7	 Nigeria	170,123,740	July 2012	2.42%	^[77]
8	 Bangladesh	161,083,804	July 2012	2.29%	^[78]
9	 Russia	141,927,297	January 1, 2010	2.015%	^[79]
10	 Japan	127,610,000	May 1, 2012	1.81%	^[80]

Data Frames - Reading CSV

We want to get the table that looks like the following, which seems to be the 4th table on the page

```
library(XML)
url <- "http://en.wikipedia.org/wiki/World_population"
table.four <- readHTMLTable(url, which=4)
```

```
head(table.four,4)
```

Rank	Country / Territory	Population	Date	% of world population	Source
1	China[note 2]	1,353,430,000	October 1, 2012	19.2%	[71]
2	India	1,210,193,422	March 2011	17%	[72]
3	United States	314,490,000	October 1, 2012	4.47%	[73]
4	Indonesia	238,400,000	May 2010	3.33%	[74]

But there are problems here. Everything is a factor including the population, Date, and percent population. We can fix these of course

Data Frames - Reading CSV

We want to get the table that looks like the following, which seems to be the 4th table on the page

```
table.four <- table.four[,-4:-6] # Eliminate the 4th-6th column
```

```
table.four
```

	Rank	Country / Territory	Population
1	1	China[73]	1,352,190,000
2	2	India	1,210,193,422

```
# Get rid of the commas in the numbers
```

```
table.four$Population <- as.numeric(gsub(",", "", table.four$Population))/100000
```

```
# Give the columns new names
```

```
names(table.four) <- c("Rank", "Country", "Population")
```

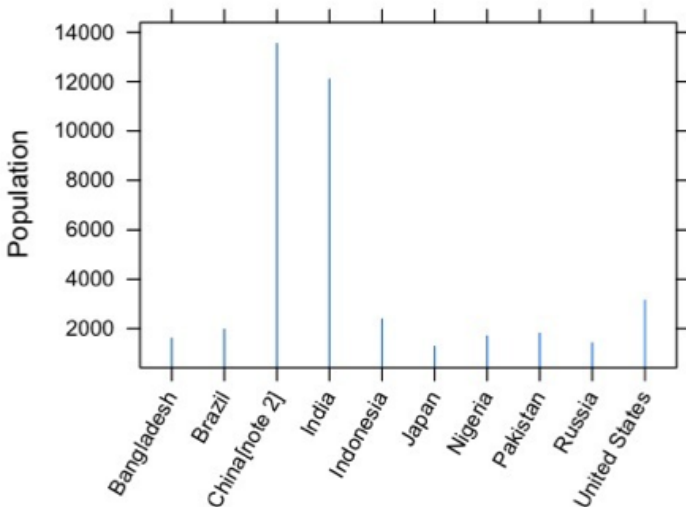
```
# Plot the data
```

```
library(lattice)
```

```
xyplot(Population ~ Country, table.four, scales = list(x = c(rot=60)),  
       type="h", main="Most Densely Populated Countries")
```


Data Frames - Reading CSV

Most Populated Countries



Data Frames - Reading CSV

There is a way to convert columns to the desired format as we are reading it. We use the **colClasses** argument in the **read.table()** function. It works for most formats except perhaps for dates given the wide variety of date formats

```
classes <- c("integer","character","FormattedNumber","character",  
            "Percent","character")
```

```
poptable <- readHTMLTable(url,which=4,colClasses=classes,  
                          stringsAsFactors=F)
```

```
str(poptable)
```

```
'data.frame':  10 obs. of  6 variables:
```

```
$ Rank          : int   1 2 3 4 5 6 7 8 9 10
```

```
$ Country / Territory : chr  "China[note 2]" "India" "United States" "Indonesia" ...
```

```
$ Population      : num  1.37e+09 1.26e+09 3.19e+08 2.52e+08 2.04e+08 ...
```


```
$ Date            : chr  "December 15, 2014" "December 15, 2014" "December 15, ...
```

```
$ Approx. % of world population: num  19 17.5 4.43 3.5 2.82 2.61 2.48 2.18 2.03 1....
```

```
$ Source          : chr  "[83]" "[84]" "[85]" "[86]" ...
```

Data Frames - Chicago Crime - Supplemental

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



The screenshot shows the City of Chicago Data Portal. At the top is the City of Chicago logo and the text "City of Chicago Data Portal". To the right are navigation links: Home, About, Help, Developers, and Terms. Below the header are three featured data categories, each with a thumbnail image and a description:

- Energy Usage**: An unprecedented dataset containing electrical and gas energy usage throughout Chicago in 2010. Data is available by Census block and month. The thumbnail shows a heatmap of Chicago's census tracts.
- Alternative Fuel Locations**: List of locations in Chicago where alternative vehicle fuels are available. The thumbnail shows a map of Chicago with blue dots indicating fuel locations.
- Crimes - 2001 to present**: Review reported incidents of crime that occurred in Chicago from 2001 to present. The thumbnail shows a map of Chicago with red circles indicating crime hotspots.

<https://data.cityofchicago.org/>

Data Frames - Chicago Crime - Supplemental

- I got a file from this site and put it on a server if you want to download it and give it a whirl. This is 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 read and process it.

```
url <- "http://steviep42.bitbucket.org/bios545r_2016/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

Data Frames - Chicago Crime - Supplemental

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/bios545r_2016/DATA.DIR/chi_crimes.csv"

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

Data Frames - Chicago Crime - Supplemental

The file relates to all calls to the Chicago Police Department in 2012

```
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
```

Data Frames - Chicago Crime - Supplemental

```
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"
```

Data Frames - Chicago Crime - Supplemental

```
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	

Data Frames - Chicago Crime - Supplemental

```
# Make the date a "real date"
hi$Date <- strptime(chi$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")
```

Data Frames - Chicago Crime - Supplemental

```
# 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")
```

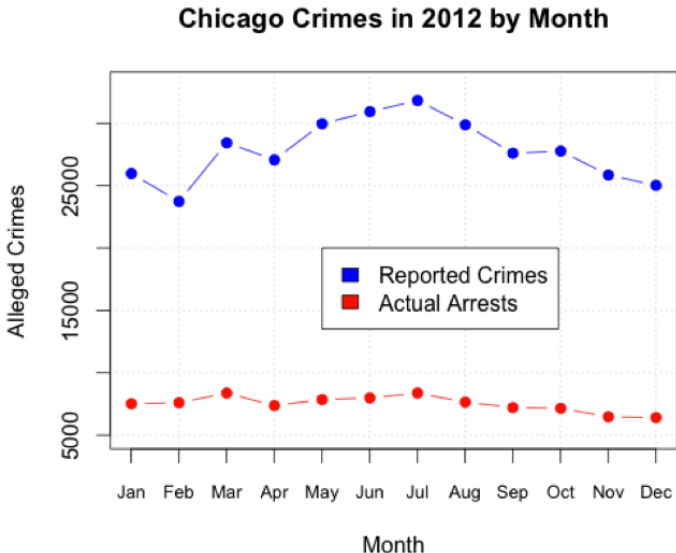
```
# Even easier to do
```

```
rev(sort(table(chi$month)))
```

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

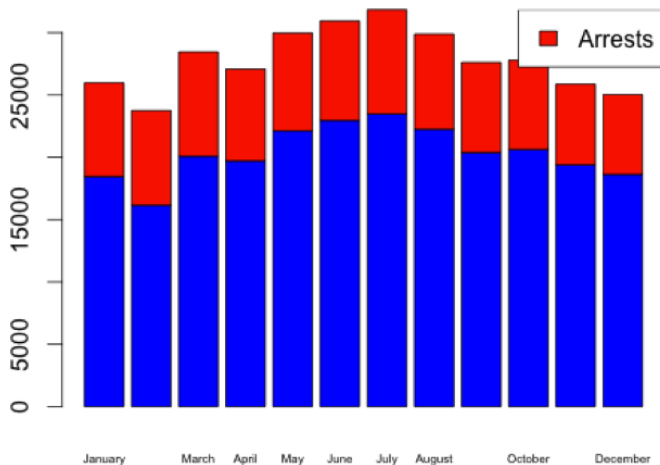
```
# Looks like the Summer is when more crimes are committed
```

Data Frames - Chicago Crime - Supplemental



Data Frames - Chicago Crime - Supplemental

Chicago: Reported Crimes vs. Actual Arrests



Data Frames - Chicago Crime - Supplemental

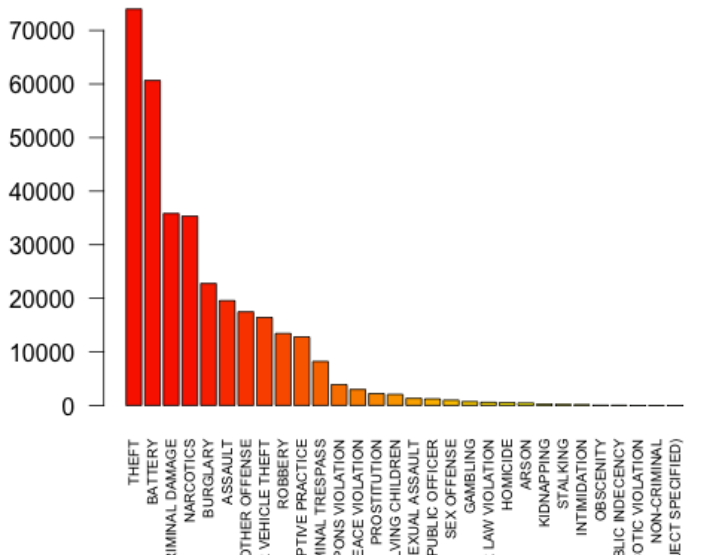
```
# Find out number of alleged crimes by type

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)))

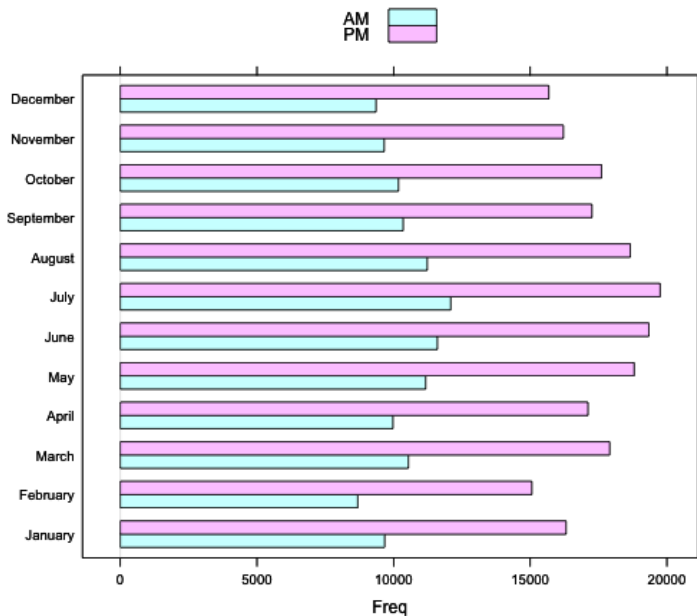
barplot(categories,horiz=F,las=1,cex.names=0.6,col=heat.colors(30),
        las=2, main="Chicago: Types of Crimes Reported")
```

Chicago: Types of Crimes Reported



```
library(lattice)
```

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



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=" ")  
chi.plot <- gvisMap(hold,"LatLon","Tip")  
  
plot(chi.plot)
```