Introduction to R

Greg Ridgeway (gridge@upenn.edu)
Ruth Moyer (moyruth@upenn.edu)

September 24, 2017

Introduction

This is the first set of notes for an introduction to R programming from criminology and criminal justice. These notes assume that you have the latest version of R and R Studio installed. We are also assuming that you know how to start a new script file and submit code to the R console. From that basic knowledge about using R, we are going to start with `2+2` and by the end of this set of notes you will load in a small Chicago crime dataset, create a few plots, count some crimes, and be able to subset the data. Our aim is to build a firm foundation on which we will build throughout this set of notes.

R sometimes provides useful help as to how to do something, such as choosing the right function or figuring what the syntax of a line of code should be. Let's say we're stumped as to what the `sqrt()` function does. Just type `?sqrt` at the R prompt to read documentation on `sqrt()`. Most help pages have examples at the bottom that can give you a better idea about how the function works. R has over 7,000 functions and an often seemingly inconsistent syntax. As you do more complex work with R (such as using new packages), the Help tab can be useful.

Basic Math and Functions in R

R, on a very unsophisticated level, is like a calculator.

```
2+2
1*2*3
(1+2+3-4)/(5*7)
sqrt(2)
(1+sqrt(5))/2 # golden ratio
2^3
log(2.718281828)
round(2.718281828,3)
12^2
factorial(4)
abs(-4)
```

```
[1] 4
[1] 6
[1] 0.05714286
[1] 1.414214
[1] 1.618034
[1] 8
```

```
[1] 1
[1] 2.718
[1] 144
[1] 24
[1] 4
```

Combining values together into a collection (or vector)

We will use the `c()` function a lot. `c()` *c*ombines elements, like numbers and text to form a vector or a collection of values. If we wanted to combine the numbers 1 to 5 we could do

```
c(1,2,3,4,5)
```

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

With the `c()` function, it's important to separate all of the items with commas.

Conveniently, if you want to add 1 to each item in this collection, there's no need to add 1 like `c(1+1,2+1,3+1,4+1,5+1)`... that's a lot of typing. Instead R offers the shortcut

```
c(1,2,3,4,5)+1
```

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

In fact, you can apply any mathematical operation to each value in the same way.

```
c(1,2,3,4,5)*2
sqrt(c(1,2,3,4,5))
(c(1,2,3,4,5)-3)^2
abs(c(-1,1,-2,2,-3,3))
```

```
[1]  2  4  6  8 10
[1] 1.000000 1.414214 1.732051 2.000000 2.236068
[1] 4 1 0 1 4
[1] 1 1 2 2 3 3
```

Note in the examples below that you can also have a collection of non-numerical items. When combining text items, remember to use quotes around each item.

```
c("CRIM600","CRIM601","CRIM602","CRIM603")
c("yes","no","no",NA,NA,"yes")
```

```
[1] "CRIM600" "CRIM601" "CRIM602" "CRIM603"
[1] "yes" "no"  "no"  NA    NA    "yes"
```

In R, `NA` means a missing value. We'll do more exercises later using data containing some `NA` values. In any dataset, you're virtually guaranteed to find some NAs. The function `is.na()` helps determine whether there are any missing values (any NAs). In some of the problems below, we'll use `is.na()`.

You can use double quotes or single quotes in R as long as you are consistent. When you have quotes inside the text you need to be particularly careful.

```
"Lou Gehrig's disease"
'The officer shouted "halt!"'
```

```
[1] "Lou Gehrig's disease"
[1] "The officer shouted \"halt!\""
```

The backslashes in the above text "protect" the double quote, communicating to you and to R that the next double quote is not the end of the text, but a character that is actually part of the text the user wants to keep.

The `c()` function isn't the only way to make a collection of values in R. For example, placing a `:` between two numbers can return a collection of numbers in sequence. The functions `rep()` and `seq()` produce repeated values or sequences.

```
1:10
5:-5
c(1,1,1,1,1,1,1,1,1,1)
rep(1,10)
rep(c(1,2),each=5)
seq(1, 5)
seq(1, 5, 2)
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
 [1]  5  4  3  2  1  0 -1 -2 -3 -4 -5
 [1] 1 1 1 1 1 1 1 1 1 1
 [1] 1 1 1 1 1 1 1 1 1 1
 [1] 1 1 1 1 1 2 2 2 2 2
[1] 1 2 3 4 5
[1] 1 3 5
```

R will also do arithmetic with two vectors, doing the calculation pairwise. This following will compute 1+11 and 2+12 up to 10+20.

```
1:10 + 11:20
```

```
 [1] 12 14 16 18 20 22 24 26 28 30
```

Yet, other functions operate on the whole collection of values in a vector. See the following examples:

```
sum(c(1,10,3,6,2,5,8,4,7,9)) # sum
length(c(1,10,3,6,2,5,8,4,7,9)) # how many?
cumsum(c(1,10,3,6,2,5,8,4,7,9)) # cumulative sum
mean(c(1,10,3,6,2,5,8,4,7,9)) # mean of collection of 10 numbers
median(c(1,10,3,6,2,5,8,4,7,9)) # median of same population
```

```
[1] 55
[1] 10
 [1]  1 11 14 20 22 27 35 39 46 55
[1] 5.5
[1] 5.5
```

There are also some functions in R that help us find the biggest and smallest values. For example:

```r
max(c(1,10,3,6,2,5,8,4,7,9)) # what is the biggest value in vector?
which.max(c(1,10,3,6,2,5,8,4,7,9)) # in which "spot" would we find it?
min(c(1,10,3,6,2,5,8,4,7,9)) # what is the smallest value in vector?
which.min(c(1,10,3,6,2,5,8,4,7,9)) # in which "spot" would we find it?
```

```
[1] 10
[1] 2
[1] 1
[1] 1
```

A lot of functions in R are to help you see and understand what's in a dataset. For example, we can rearrange a collection of values in ascending or descending order. Note the `order()` function. How is it similar to the `which.max()` or `which.min()` function? Note the `sort()` function.

```r
sort(c(1,10,3,6,2,5,8,4,7,9))
```

[1] 1 2 3 4 5 6 7 8 9 10

```r
rev(c(1,10,3,6,2,5,8,4,7,9))
```

[1] 9 7 4 8 5 2 6 3 10 1

```r
rev(sort(c(1,10,3,6,2,5,8,4,7,9)))
```

[1] 10 9 8 7 6 5 4 3 2 1

```r
sort(c(1,10,3,6,2,5,8,4,7,9),decreasing=TRUE)
```

[1] 10 9 8 7 6 5 4 3 2 1

```r
order(c(1,10,3,6,2,5,8,4,7,9))#   where is the ith biggest number?
```

[1] 1 5 3 8 6 4 9 7 10 2

```r
rank(c(1,100,3,20)) #how does each value rank compared to others?
```

[1] 1 4 2 3 The above examples have involved mostly numerical values in a vector. Here are some examples involving non-numerical "character" values. Let's create an object called `my.states` (a name I made up) that will contain the postal codes of places in which I've lived or worked.

```r
my.states <- c("WA","DC","CA","PA","MD","VA","OH")
```

Take a look at the arrow `<-`. This is how you tell R to take the result of what is on the right and store it in an object named on the left. We're going to talk more about this arrow soon. Now let's run some new functions on this collection of postal codes.

```r
nchar(my.states)
paste(my.states, ", USA")
paste(my.states, ", USA", sep="")
paste(my.states, collapse=",")
paste0(my.states)
```

```
[1] 2 2 2 2 2 2 2
[1] "WA , USA" "DC , USA" "CA , USA" "PA , USA" "MD , USA" "VA , USA"
```

```
[7] "OH , USA"
[1] "WA, USA" "DC, USA" "CA, USA" "PA, USA" "MD, USA" "VA, USA" "OH, USA"
[1] "WA,DC,CA,PA,MD,VA,OH"
[1] "WA" "DC" "CA" "PA" "MD" "VA" "OH"
```

What does the `nchar()` function do? The `paste()` function? Does it make a difference to use `sep=""` or `collapse=","`? What about `paste0()`?

Exercises

1. Print all even numbers less than 100
2. What is the mean of even numbers less than 100
3. Have R put in alphabetical order `c("WA","DC","CA","PA","MD","VA","OH")`

Assignment of values to variables

The left-facing arrow symbol is an extremely important tool in R. Try the following:

```
a <- 1
```

Now type:

```
a
```

```
[1] 1
```

R has assigned a the value of "1" - here are more examples:

```
b <- 2+2
a <- a+b
a <- 1:10
b <- 2*a
a+b
sd(a)
state.names <- c("WV","OH","OK","NV","CA","IN","MA","MI","IL","IA","SC","NH",
                 "LA","GA","CT","WI","CO","NY","UT","AK","MS","AL","OR","MT",
                 "ND","WY","FL","ME","AZ","TN","PA","MN","NM","SD","MO","RI",
                 "HI","WA","DE","NJ","NE","KY","AR","TX","NC","MD","VA","VT",
                 "KS","ID","DC")
```

```
 [1]  3  6  9 12 15 18 21 24 27 30
[1] 3.02765
```

R programmers typically pronounce the `<-` as "gets". So we would read `a <- 1` as "a gets one".

Indexing

We can extract items from a vector, matrix, or data frame using indexing. In R, we use square brackets to index.

```
state.names[1] # get the first state
state.names[1:3] # get the first three states
state.names[c(1,5,9)] # get states 1, 5, and 9
state.names[2*(1:25)] # get the even states
```

```
[1] "WV"
[1] "WV" "OH" "OK"
[1] "WV" "CA" "IL"
 [1] "OH" "NV" "IN" "MI" "IA" "NH" "GA" "WI" "NY" "AK" "AL" "MT" "WY" "ME"
[15] "TN" "MN" "SD" "RI" "WA" "NJ" "KY" "TX" "MD" "VT" "ID"
```

If you put a negative number inside the `[]`, this will communicate to R to remove that item from the collection. Let's remove DC from `state.names` since it is not one of the 50 states. Since it is the 51st item in `state.names` we can remove like this

```
state.names[-51]
```

```
 [1] "WV" "OH" "OK" "NV" "CA" "IN" "MA" "MI" "IL" "IA" "SC" "NH" "LA" "GA"
[15] "CT" "WI" "CO" "NY" "UT" "AK" "MS" "AL" "OR" "MT" "ND" "WY" "FL" "ME"
[29] "AZ" "TN" "PA" "MN" "NM" "SD" "MO" "RI" "HI" "WA" "DE" "NJ" "NE" "KY"
[43] "AR" "TX" "NC" "MD" "VA" "VT" "KS" "ID"
```

Let's combine the sort and order functions from above (along with variable assignment) with the concept of indexing.

```
sort(state.names)[1] # sort, then give the first value
i <- order(state.names) # index the states in order
i[1:3]                  # which positions are the first three
state.names[i[1:3]]     # show me those three states
```

```
[1] "AK"
[1] 20 22 43
[1] "AK" "AL" "AR"
```

Note that in the last example we used square brackets within square brackets. First, we asked R to give us the indices of the first three states in alphabetical order and that was 20, 22, 43. Then R took those three values and plugged them into the second set of square brackets to show you the state names in those positions in the collection.

Exercises

4. What's the last state in the `state.names`?
5. Pick out states that begin with "M" using their indices
6. Pick out states where you have lived
7. What's the last state in alphabetical order?

8. What are the last three states in alphabetical order?

Logical values and operations

Logical values in R are the two values `TRUE` and `FALSE`, always written in all capital letters in R. You can also combine a bunch of `TRUE` and `FALSE` values into a collection.

```
TRUE
FALSE
c(TRUE,FALSE,TRUE,FALSE)
```

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

We use logical operators to create logical expressions and r can evaluate them as either `TRUE` or `FALSE`. For example, `&` represents the logical "and" and `|` represents the logical "or."

```
TRUE  & TRUE
FALSE & TRUE
FALSE | TRUE
FALSE | FALSE
```

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

We can use R to compare values using greater than or less than symbols. We can also express "greater than or equal to" or "less than or equal to." These will evaluate to `TRUE` or `FALSE` depending, of course, on whether the statement is true or false.

```
6>5
6<5
6>=5
5<=5
```

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

We can combine logical operators into more complicated expressions.

```
(6>5) | (100<3)
(6>5) & (100<3)
```

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

Here are some additional examples. We are going to make `a` be the values 1 to 10 and then use logical operators to ask a question (like "are you equal to?" or "are you smaller than?") of each of those values. Note that the double equal sign == asks the question whether the two values are the same.

```
a <- 1:10
a==5
a!=5  # ! means "not"
a<5
a>=5
a>5 & a<8
a<3 | a>=7
```

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

The %% operator computes the remainder after dividing the left side by the right side.

```
13 %% 5       # = 3, 13/5 = 2 with remainder 3
a %% 2 == 0  # here's a way to ask each number if it's even
```

```
[1] 3
 [1] FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE
```

There are special functions `any()` and `all()` that check whether all/any of the values are true.

```
all(a<11)
all(a>5 & a<8)
any(a>5 & a<8)
```

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

Logical values may be used inside square brackets too. R will show you the values corresponding to `TRUE`s inside the square brackets and will eliminate any values corresponding to `FALSE`s. For example, let's store in `i` `TRUE` for even numbers and `FALSE` for odd numbers. So `i` will consist of ten logical values. Putting `i` inside the square brackets will extract just the values of `a` for which `i` has a `TRUE`.

```
i <- a%%2==0
i
a[i]
```

```
 [1] FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE
[1]  2  4  6  8 10
```

We can use `!`, which means "not," to reverse all the logical values and get the values of `a` that are not even.

```
a[!i]
```

```
[1] 1 3 5 7 9
```

Before we removed DC from the list of states by noticing that it was in position #51. This time, let's have R do the work of locating DC in the collection of states. We'll have R ask each element in `state.names` whether or not it equals "DC".

```
i <- state.names!="DC"
state.names[i]
state.names[state.names!="DC"] # can also put directly inside []
```

```
 [1] "WV" "OH" "OK" "NV" "CA" "IN" "MA" "MI" "IL" "IA" "SC" "NH" "LA" "GA"
[15] "CT" "WI" "CO" "NY" "UT" "AK" "MS" "AL" "OR" "MT" "ND" "WY" "FL" "ME"
[29] "AZ" "TN" "PA" "MN" "NM" "SD" "MO" "RI" "HI" "WA" "DE" "NJ" "NE" "KY"
[43] "AR" "TX" "NC" "MD" "VA" "VT" "KS" "ID"
 [1] "WV" "OH" "OK" "NV" "CA" "IN" "MA" "MI" "IL" "IA" "SC" "NH" "LA" "GA"
[15] "CT" "WI" "CO" "NY" "UT" "AK" "MS" "AL" "OR" "MT" "ND" "WY" "FL" "ME"
[29] "AZ" "TN" "PA" "MN" "NM" "SD" "MO" "RI" "HI" "WA" "DE" "NJ" "NE" "KY"
[43] "AR" "TX" "NC" "MD" "VA" "VT" "KS" "ID"
```

The R operator `%in%` asks each value on the left whether or not it is a member of the set on the right.

```
a %in% c(3,7,10)

my.states <- c("MD","OH","VA","CA","WA","DC")
# do these states touch the Pacific Ocean?
my.states %in% c("CA","OR","WA","AK","HI")
# how many of these states touch the Pacific Ocean?
sum(my.states %in% c("CA","OR","WA","AK","HI"))
```

```
 [1] FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE
[1] FALSE FALSE FALSE  TRUE  TRUE FALSE
[1] 2
```

Note in the last line we used `sum()` to count for how many of `my.states` did `%in%` evaluate to `TRUE`.

Exercises

9. Report `TRUE` or `FALSE` for each state depending on if you have lived there
10. With `a <- 1:100`, pick out odd numbers between 50 and 75
11. Use greater than less than signs to get all state names that begin with M

Sampling

The function `sample()` randomly shuffles a collection of values.

```
sample(1:10) # each time different values will appear
sample(1:10)
```

```r
sample(1:10)
a <- sample(1:1000,size=10) # pick 10 numbers between 1-1000
a <- sample(1:6,size=1000,replace=TRUE) # roll a die 1000 times
```

```
 [1]  8  2  3  5  4  9  1  6  7 10
 [1]  3 10  5  2  9  6  7  8  4  1
 [1] 10  5  3  8  7  6  1  4  9  2
```

Notice that `sample()` has several options including `size=` to indicate how many to select and `replace=` to indicate whether to sample with or without replacement. You can access the help on the `sample()` function by typing `?sample` at the R prompt.

Tabulating

The `table()` function counts how many of each value appear in a collection. We just set `a` to be a random collection of numbers 1 to 6, simulating rolling a die. With `table()` we can see how often each number appeared.

```r
table(a)
max(table(a)) # find out which value appears most frequently
```

```
a
  1   2   3   4   5   6
161 182 180 160 167 150
[1] 182
```

Exercises

12. Use `sample()` to estimate the probability of rolling a 6
13. Use `sample()` to estimate the probability that the sum of two die equal 7
14. Use `sample()` to select randomly five states without replacement
15. Use `sample()` to select randomly 1000 states with replacement
    - Tabulate how often each state was selected
    - Which state was selected the least? Make R do this for you

Lists

So far we have worked with very simple collections of numbers or text or logical values. Eventually we will need to work with more complicated kinds of data, like datasets, maps, and other objects. R stores these more complex objects in a list. A list is essentially a collection of objects, potentially of different types. Let's start with a simple list.

```r
a <- list(1:3,5:1,1:10)
a
```

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

[[2]]
[1] 5 4 3 2 1

[[3]]
 [1]  1  2  3  4  5  6  7  8  9 10
```

The list a has three components, each of which is a collection of values and each has different length. Here's another list consisting of three components, each of which is a collection of different types, numeric, text, and logical values.

```
b <- list(0:9, c("A","B","C"),c(TRUE,FALSE,NA))
b
```

```
[[1]]
 [1] 0 1 2 3 4 5 6 7 8 9

[[2]]
[1] "A" "B" "C"

[[3]]
[1]  TRUE FALSE    NA
```

We use a double set of square brackets to access the components of a list. Let's say we just want the first component of a, just the part with the numbers 1, 2, and 3.

```
a[[1]]
```

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

We can even grab the first element in the first component of the list a.

```
a[[1]][1]
```

```
[1] 1
```

Or we just select the first and third component of the list a. This will return a new list, but just without the second component.

```
a[c(1,3)]
```

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

[[2]]
 [1]  1  2  3  4  5  6  7  8  9 10
```

`lapply()` means "list apply" and lets us apply a given function to every item in a list and obtain a list in return. Let's say we want to sort each of the components in a. It would take too much typing to run `sort(a[[1]])` and `sort(a[[2]])` and `sort(a[[3]])`. Instead, `lapply()` can apply the sort function to each of the three components in a.

```r
lapply(a,sort)
```

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

[[2]]
[1] 1 2 3 4 5

[[3]]
 [1]  1  2  3  4  5  6  7  8  9 10
```

There is also a function `sapply()` that works in a manner quite similar to `lapply()`. The only difference is that `sapply()` will try to simplify the results. Think about the "s" meaning "simplified". Let's compute the number of elements in each component and the average of the numbers in each component.

```r
sapply(a,length)
sapply(a,mean)
```

```
[1]  3  5 10
[1] 2.0 3.0 5.5
```

Since `length()` and `mean()` will return a single number for each component, the result can be simplified into a collection of three values, one for each component of the list.

Let's find the component that has the most values in it.

```r
i <- which.max(sapply(a,length))
a[[i]]
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

If `sapply()` is not able to simplify the result, then the result is just like `lapply()`.

```r
sapply(a,sort)
```

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

[[2]]
[1] 1 2 3 4 5

[[3]]
 [1]  1  2  3  4  5  6  7  8  9 10
```

Let's return to our state example. Before we just had a collection of 51 postal codes. Instead, let's create a list that separates them into three components depending on whether they are in the west, east, or central United States.

```r
state.list <- list(
   west=c("AK","HI","WA","NV","CA","CO","UT","OR","AZ","NM","ID"),
   east=c("KY","RI","PA","DE","DC","NJ","WV","MA","SC","NH","GA","CT","NY","IN",
          "MS","AL","OH","NC","MD","VA","VT","FL","ME","TN"),
```

```r
  central=c("SD","MO","MN","ND","WY","OK","MI","IL","IA","LA","WI","MT","NE",
            "AR","TX","KS"))
```

We can now use `lapply()` to ask R to sort each region, sample three states from each region, and tell us how many states are in each region.

```r
lapply(state.list,sort)
```

```
$west
 [1] "AK" "AZ" "CA" "CO" "HI" "ID" "NM" "NV" "OR" "UT" "WA"

$east
 [1] "AL" "CT" "DC" "DE" "FL" "GA" "IN" "KY" "MA" "MD" "ME" "MS" "NC" "NH"
[15] "NJ" "NY" "OH" "PA" "RI" "SC" "TN" "VA" "VT" "WV"

$central
 [1] "AR" "IA" "IL" "KS" "LA" "MI" "MN" "MO" "MT" "ND" "NE" "OK" "SD" "TX"
[15] "WI" "WY"
```

```r
lapply(state.list,sample,size=3,replace=FALSE)
```

```
$west
[1] "ID" "UT" "AZ"

$east
[1] "IN" "AL" "VA"

$central
[1] "MI" "LA" "TX"
```

```r
sapply(state.list,length)
```

```
   west    east central 
     11      24      16 
```

Notice here that we have given names (west, east, and central) to each of the three components of `state.list`. We can ask R to tell us what the names of the `state.list` components are.

```r
names(state.list)
```

```
[1] "west"    "east"    "central"
```

We can use the double square brackets to extract the western states. Since they are first in the list we use `[[1]]`

```r
state.list[[1]]
```

```
 [1] "AK" "HI" "WA" "NV" "CA" "CO" "UT" "OR" "AZ" "NM" "ID"
```

However, this can be dangerous. Are we sure the first component has the western states? A safer approach is to call it by name inside the square brackets.

```r
state.list[["west"]]
```

```
 [1] "AK" "HI" "WA" "NV" "CA" "CO" "UT" "OR" "AZ" "NM" "ID"
```

We can also use the $ to extract a named component from a list.

```r
state.list$west
```

```
 [1] "AK" "HI" "WA" "NV" "CA" "CO" "UT" "OR" "AZ" "NM" "ID"
```

The dollar sign in R is going to be extremely important. We will be using it a lot to extract variables, map components, and other values from lists.

You can use the $ to add new components to a list. Let's add all the postal codes for all of the United States territories.

```r
state.list$other <- c("AS","GU","MP","PR","VI","UM","FM","MH","PW")
```

What happens if we ran just the following?

```r
other <- c("AS","GU","MP","PR","VI","UM","FM","MH","PW")
```

This creates a separate object called `other`, unconnected to our `state.list`. By using the $ we add our new collection of states (other) to `state.list`.

We have now created a lot of objects. At any time you can run `ls()` to list all the objects that R has in memory.

```r
ls()
```

```
[1] "a"                 "b"                 "counterExercise"  
[4] "exerciseQuestions" "exNum"             "i"                
[7] "my.states"         "state.list"        "state.names"      
```

Assuming you are using R Studio, you can also see the objects stored in memory by clicking on the Environment tab.

Exercises

16. Fix `state.list` so that "DC" is in "other" rather than "east". Here are a few hints
    - access "other" using $
    - combine things using `c()`
    - assign values using <-
    - remove values using `[]` with a negative index or using a logical statement
17. Print out east and central states together sorted

Functions

So far you have seen several built-in functions in R, like `max()`, `sample()`, `is.na()`, and `table()`. These functions help us complete tasks that normally would take several lines of R code. They also make it easy to read R code… it's easy to know what `max(c(1,3,5,7,9))` means. In R you can also write your own functions. Let's say we want to just extract the first and last state from each component of `state.list`. Now this is not a particularly useful function, but we're going to use it just for demonstration.

```
give.first.and.last <- function(x)
{
   i <- c(1,length(x))
   return(x[i])
}
```

As you can see, the basic template of an R function is to give it a new name (here `give.first.and.last()`), followed by the syntax `<- function` (this tells R that what comes next is a function), followed by parentheses containing the names of arguments (you choose what to call them) that will be sent to this function (here we use the not very creative `x`), followed by squiggly braces containing R code to do calculations on `x`, with the last line being `return()` containing whatever final result the function calculates. Our function here creates `i` to contain the number 1 and the length of `x` so that it can figure out where the last value is. Then it simply returns `x[i]`, using the square brackets to pick out the values of `x` indexed by `i`, the first and last values in `x`. Let's try our new function out on the numbers 1 to 100.

```
give.first.and.last(1:100)
```

```
[1]   1 100
```

The primary benefit of writing a function is to simplify the reading of a script. It is much easier to comprehend what a script is doing if you have code that says something like `give.first.and.last()` rather than a bunch of square brackets picking out values. A secondary benefit is that you can use this function again and again to help solve other problems.

Let's combine `give.first.and.last()` with `lapply()` and `sapply()` to extract the first and last state in each component of our list.

```
lapply(state.list, give.first.and.last)
```

```
$west
[1] "AK" "ID"

$east
[1] "KY" "TN"

$central
[1] "SD" "KS"

$other
[1] "AS" "PW"
```

```
sapply(state.list, give.first.and.last)
```

```
     west east central other
[1,] "AK" "KY" "SD"    "AS"
[2,] "ID" "TN" "KS"    "PW"
```

Note how `sapply()` noticed that `give.first.and.last()` produces exactly two values for each component of the list and went ahead and simplified the result into a 2 by 4 table. Let's first sort the states within each region and then extract the first and last states. This will give us the first and last in alphabetical order.

```
sapply(lapply(state.list,sort), give.first.and.last)
```

```
     west east central other
[1,] "AK" "AL" "AR"    "AS"
[2,] "WA" "WV" "WY"    "VI"
```

For many functions built in to R you can see what they do by typing the name of the function. Here's how R computes the interquartile range of a collection of values.

```
IQR
```

```
function (x, na.rm = FALSE, type = 7)
diff(quantile(as.numeric(x), c(0.25, 0.75), na.rm = na.rm, names = FALSE,
    type = type))
<bytecode: 0x0000000019358c38>
<environment: namespace:stats>
```

You can see that it computes the 0.25 quantile and the 0.75 quantile and uses `diff()` to compute their difference.

Exercises

18. Make a function `is.island(x)` returns `TRUE` if `x` is an island. Islands are "HI", "FM", "MH", "PW", "AS", "GU", "MP", "PR", "VI", "UM". Borrow the template I used for `give.first.and.last()`. Then try using the `%in%` operator
19. Count how many islands are within each region. Use an `sapply()` (or two) and your new `is.island()` function
20. Which components of `b` having missing values? Use `is.na()`. `b` was defined earlier

Matrices and apply()

A matrix is a collection of values of the same type (all numbers or all text or all logical values) with one or more rows and one or more columns. Let's create a matrix with some random numbers.

```
a <- matrix(sample(1:5,size=12,replace=TRUE),nrow=4)
a
```

```
     [,1] [,2] [,3]
[1,]    1    4    2
[2,]    3    3    2
[3,]    2    5    2
[4,]    5    2    5
```

This matrix has two dimensions, 4 rows and 3 columns. You can use square brackets to select elements from the matrix.

```
a[1,2]      # element in first row, second column
a[1,]       # the entire first row
a[,2]       # the entire second column
a[-1,-1]    # dropping the first row and first column
a[3:4,2:3] # rows 3 & 4, columns 2 & 3
```

```
[1] 4
[1] 1 4 2
[1] 4 3 5 2
     [,1] [,2]
[1,]    3    2
[2,]    5    2
[3,]    2    5
     [,1] [,2]
[1,]    5    2
[2,]    2    5
```

The numbers to the left of the comma index rows and the numbers to the right of the comma index columns. The `apply()` function, like the `lapply()` and `sapply()` functions, allow you to apply a function to all the rows or all the columns of a matrix. `apply()` needs the name of the matrix, whether you want to apply the function to the first dimension (rows) or the second dimension (columns), and the name of the function to apply.

```
apply(a, 1, sum)     # compute sum of each row
apply(a, 2, sum)     # compute sum of each column
apply(a, 1, mean)    # compute mean of each row
apply(a, 1, summary) # summarize each row
```

```
[1]  7  8  9 12
[1] 11 14 11
[1] 2.333333 2.666667 3.000000 4.000000
            [,1]     [,2] [,3] [,4]
Min.    1.000000 2.000000  2.0  2.0
1st Qu. 1.500000 2.500000  2.0  3.5
Median  2.000000 3.000000  2.0  5.0
Mean    2.333333 2.666667  3.0  4.0
3rd Qu. 3.000000 3.000000  3.5  5.0
Max.    4.000000 3.000000  5.0  5.0
```

We can also create a new function right on the spot to compute something on each row or column. Let's find the minimum and maximum values in each row and find out if all the values are greater than 1.

```
apply(a, 1, function(x) {c(min(x),max(x))}) # there is also a function range()
apply(a, 1, function(x) {all(x>1)})
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    2    2    2
```

```
[2,]    4    3    5    5
[1] FALSE  TRUE  TRUE  TRUE
```

Setting the working directory

Now that we have covered a lot of fundamental R features, it is time to load in a real dataset. However, before we do that, R needs to know where to find the data file. So we first need to talk about "the working directory". When you start R it has a default folder or directory on your computer where it will retrieve or save any files. You can run `getwd()` to get the current working directory. Here's our current working directory, which will not be the same as yours.

```r
getwd()
```

```
[1] "Z:/Penn/CRIM602/notes/R4crim"
```

Almost certainly this default directory is *not* where you plan to have all of your datasets and files stored. Instead, you probably have an "analysis" or "project" or "R4crim" folder somewhere on you computer where you would like to store your data and work.

Use `setwd()` to tell R what folder you want it to use as the working directory. If you do not set the working directory R will not know where to find the data you wish to import and will save your results in a location in which you would probably never look. Make it a habit to have `setwd()` as the first line of every script your write. If know the working directory you want to use, then you can just put inside the `setwd()` function.

```
setwd("C:\Users\gridge\Google Drive\R4crim")
```

Note that for all platforms, Windows, Macs, and Linux, the working directory only uses forward slashes. So Windows users be careful… most Windows applications use backslashes, but in an effort to make R scripts work across all platforms, R requires forward slashes. Backslashes have a different use in R that you will meet later.

If you do not know how to write your working directory, here comes R Studio to the rescue. In R Studio click Session -> Set Working Directory -> Choose Directory. Then click through to navigate to the working directory that you want to use. When you find it click "Select Folder". Then look over at the console. R Studio will construct the right `setwd()` syntax for you. Copy and paste that into your script for use later. No need to have to click through the Session menu again now that you have your `setwd()` set up.

Now you can use R functions to load in any datasets that are in your working folder. If you have done your `setwd()` correctly, you shouldn't get any errors because R will know exactly where to look for the data files. If the working directory that you've given in the `setwd()` isn't right, R will think the file doesn't even exist. For example, if you give the path for, say, your R4econ folder, R won't be able to load data because the file isn't stored in what R thinks is your working directory. With that out of the way, let's load a dataset.

Data frames

A data frame is a special case of a list where all the components of the list have the same number of elements. Think about each component of the list being a "column" in your dataset. R code load in datasets from numerous sources (plain text, Excel files, databases, websites, etc.) including .RData format, R's unique data format. There is an extensive guide to importing and exporting datasets.

To import data in the .RData format use `load()`. A sample of Chicago crime data is available on the R4Crim github site.

```
load("chicago crime 20141124-20141209.RData")
```

List the objects R now has in memory and you will see that there is a new object, `chicagoCrime`.

```
ls()
```

```
 [1] "a"                   "b"                   "chicagoCrime"
 [4] "counterExercise"     "exerciseQuestions"   "exNum"
 [7] "give.first.and.last" "i"                   "my.states"
[10] "state.list"          "state.names"
```

If you did not spell the name of the .RData file exactly correctly, then R will give you an error. A common occurrence when downloading the same file from the web multiple times is for you web browser to add numbers to the multiple versions you've downloaded. So check the file name carefully. Here's what happens when I request a file that doesn't exist.

```
load("chicago crime.RData")
```

```
Warning in readChar(con, 5L, useBytes = TRUE): cannot open compressed file
'chicago crime.RData', probable reason 'No such file or directory'

Error in readChar(con, 5L, useBytes = TRUE): cannot open the connection
```

Let's check that this is indeed a dataset. You can use the `is()` function on any R object to ask it to identify itself.

```
is(chicagoCrime)
```

```
[1] "data.frame" "list"       "oldClass"   "vector"
```

You can see that `chicagoCrime` is of type `data.frame`... and it is also of type `list`. That means that anything that you can do to lists, like `lapply()` and `sapply()`, you can use on `chicagoCrime` too.

What are the names of the variables in the dataset?

```
names(chicagoCrime)
```

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

```
[22] "Location"
```

As expected, the data have information the crime date, crime type, location (including latitude and longitude), whether an arrest occurred, and more.

Let's look at some parts of the dataset.

```r
#   look at the first three rows
chicagoCrime[1:3,]
```

```
       ID Case.Number                   Date                  Block
1 9885391    HX536570 12/09/2014 11:54:00 PM         040XX W 26TH ST
2 9885433    HX536595 12/09/2014 11:45:00 PM 089XX S SOUTH CHICAGO AVE
3 9885375    HX536553 12/09/2014 11:42:00 PM        052XX S HARPER AVE
  IUCR  Primary.Type
1 0560       ASSAULT
2 0498       BATTERY
3 2820 OTHER OFFENSE
                                                  Description
1                                                      SIMPLE
2 AGGRAVATED DOMESTIC BATTERY: HANDS/FIST/FEET SERIOUS INJURY
3                                            TELEPHONE THREAT
  Location.Description Arrest Domestic Beat District Ward Community.Area
1           DRUG STORE   true    false 1031       10   22             30
2          GAS STATION  false     true  423        4    7             46
3            RESIDENCE  false     true  234        2    4             41
  FBI.Code X.Coordinate Y.Coordinate Year             Updated.On Latitude
1      08A      1150052      1886384 2014 12/16/2014 12:53:13 PM 41.84415
2      04B      1195182      1846473 2014 12/16/2014 12:53:13 PM 41.73363
3       26      1187140      1870924 2014 12/16/2014 12:53:13 PM 41.80092
  Longitude                      Location
1 -87.72483 (41.844145133, -87.724831093)
2 -87.56053 (41.733630144, -87.560531076)
3 -87.58922 (41.800920218, -87.589217569)
```

```r
#   look at the first three rows and first three columns
chicagoCrime[1:3,1:3]
```

```
       ID Case.Number                   Date
1 9885391    HX536570 12/09/2014 11:54:00 PM
2 9885433    HX536595 12/09/2014 11:45:00 PM
3 9885375    HX536553 12/09/2014 11:42:00 PM
```

```r
#   look up by the columns by name
chicagoCrime[1:3,c("Latitude","Longitude")]
```

```
  Latitude Longitude
1 41.84415 -87.72483
2 41.73363 -87.56053
3 41.80092 -87.58922
```

Ask R what types of values each of crime features hold.

```r
#   look at the types of each variable
sapply(chicagoCrime, is)
sapply(chicagoCrime, function(x) is(x)[1])
```

```
$ID
[1] "integer"              "numeric"              "vector"
[4] "data.frameRowLabels"

$Case.Number
[1] "character"            "vector"               "data.frameRowLabels"
[4] "SuperClassMethod"

$Date
[1] "character"            "vector"               "data.frameRowLabels"
[4] "SuperClassMethod"

$Block
[1] "character"            "vector"               "data.frameRowLabels"
[4] "SuperClassMethod"

$IUCR
[1] "character"            "vector"               "data.frameRowLabels"
[4] "SuperClassMethod"

$Primary.Type
[1] "character"            "vector"               "data.frameRowLabels"
[4] "SuperClassMethod"

$Description
[1] "character"            "vector"               "data.frameRowLabels"
[4] "SuperClassMethod"

$Location.Description
[1] "character"            "vector"               "data.frameRowLabels"
[4] "SuperClassMethod"

$Arrest
[1] "character"            "vector"               "data.frameRowLabels"
[4] "SuperClassMethod"

$Domestic
[1] "character"            "vector"               "data.frameRowLabels"
[4] "SuperClassMethod"

$Beat
[1] "integer"              "numeric"              "vector"
[4] "data.frameRowLabels"
```

```
$District
[1] "integer"            "numeric"            "vector"
[4] "data.frameRowLabels"

$Ward
[1] "integer"            "numeric"            "vector"
[4] "data.frameRowLabels"

$Community.Area
[1] "integer"            "numeric"            "vector"
[4] "data.frameRowLabels"

$FBI.Code
[1] "character"          "vector"             "data.frameRowLabels"
[4] "SuperClassMethod"

$X.Coordinate
[1] "integer"            "numeric"            "vector"
[4] "data.frameRowLabels"

$Y.Coordinate
[1] "integer"            "numeric"            "vector"
[4] "data.frameRowLabels"

$Year
[1] "integer"            "numeric"            "vector"
[4] "data.frameRowLabels"

$Updated.On
[1] "character"          "vector"             "data.frameRowLabels"
[4] "SuperClassMethod"

$Latitude
[1] "numeric" "vector"

$Longitude
[1] "numeric" "vector"

$Location
[1] "character"          "vector"             "data.frameRowLabels"
[4] "SuperClassMethod"

                 ID          Case.Number                 Date
          "integer"          "character"          "character"
              Block                 IUCR         Primary.Type
        "character"          "character"          "character"
        Description Location.Description               Arrest
```

```
         "character"          "character"          "character" 
            Domestic                 Beat             District 
         "character"            "integer"            "integer" 
                Ward       Community.Area             FBI.Code 
           "integer"            "integer"          "character" 
        X.Coordinate         Y.Coordinate                 Year 
           "integer"            "integer"            "integer" 
          Updated.On             Latitude            Longitude 
         "character"            "numeric"            "numeric" 
            Location 
         "character" 
```

Use `table()` and `sort()` to see what kinds of crimes are in this dataset.

```
#   tabulate crimes
sort(table(chicagoCrime$Primary.Type))
sort(table(chicagoCrime$Description))
```

```
                         GAMBLING                    NON - CRIMINAL 
                                1                                 1 
        OTHER NARCOTIC VIOLATION                      INTIMIDATION 
                                1                                 5 
                        OBSCENITY                        KIDNAPPING 
                                5                                 7 
             LIQUOR LAW VIOLATION                          HOMICIDE 
                                8                                12 
                     NON-CRIMINAL                          STALKING 
                               12                                12 
                            ARSON                       SEX OFFENSE 
                               15                                17 
                     PROSTITUTION                CRIM SEXUAL ASSAULT 
                               36                                54 
INTERFERENCE WITH PUBLIC OFFICER          OFFENSE INVOLVING CHILDREN 
                               56                                78 
            PUBLIC PEACE VIOLATION                 WEAPONS VIOLATION 
                               85                                92 
                 CRIMINAL TRESPASS                DECEPTIVE PRACTICE 
                              271                               389 
                          ROBBERY               MOTOR VEHICLE THEFT 
                              442                               451 
                     OTHER OFFENSE                           ASSAULT 
                              567                               579 
                         BURGLARY                         NARCOTICS 
                              618                               895 
                  CRIMINAL DAMAGE                           BATTERY 
                             1202                              1842 
                            THEFT 
                             2247 
```

ABUSE/NEGLECT: CARE FACILITY
1
AGGRAVATED DOMESTIC BATTERY: OTHER FIREARM
1
AGGRAVATED FINANCIAL IDENTITY THEFT
1
AGGRAVATED PO: HANDGUN
1
AGGRAVATED PO:KNIFE/CUT INSTR
1
ALTER/FORGE PRESCRIPTION
1
ATTEMPT: ARMED-KNIFE/CUT INSTR
1
CANNABIS PLANT
1
CHILD PORNOGRAPHY
1
CRIM SEX ABUSE BY FAM MEMBER
1
CRIMINAL DRUG CONSPIRACY
1
CYBERSTALKING
1
DELIVERY CONTAINER THEFT
1
ESCAPE
1
FINAN EXPLOIT-ELDERLY/DISABLED
1
FOID - REVOCATION
1
FORCIBLE DETENTION
1
GAME/DICE
1
HARBOR RUNAWAY
1
ILLEGAL POSSESSION CASH CARD
1
INDECENT SOLICITATION/CHILD
1
INTERFERENCE JUDICIAL PROCESS
1
INTOXICATING COMPOUNDS
1
KIDNAPPING

1
MANU/DELIVER:COCAINE
1
MANU/DELIVER:LOOK-ALIKE DRUG
1
MANU/DELIVER:PCP
1
MANU/DELIVER:SYNTHETIC DRUGS
1
OBSCENE TELEPHONE CALLS
1
POS: HYPODERMIC NEEDLE
1
POS: PORNOGRAPHIC PRINT
1
POSS: HEROIN(BLACK TAR)
1
PROBATION VIOLATION
1
SALE/DEL DRUG PARAPHERNALIA
1
SEXUAL EXPLOITATION OF A CHILD
1
UNAUTHORIZED VIDEOTAPING
1
UNLAWFUL USE OF RECORDED SOUND
1
UNLAWFUL USE OTHER FIREARM
1
VIO BAIL BOND: DOM VIOLENCE
1
VIOLATION GPS MONITORING DEVICE
1
VIOLATION OF CIVIL NO CONTACT ORDER
1
VIOLENT OFFENDER: ANNUAL REGISTRATION
1
AGG CRIM SEX ABUSE FAM MEMBER
2
AGG PRO EMP HANDS SERIOUS INJ
2
AGG PRO.EMP: OTHER DANG WEAPON
2
AGG PRO.EMP:KNIFE/CUTTING INST
2
AGG: HANDS/FIST/FEET NO/MINOR INJURY
2
ATTEMPT NON-AGGRAVATED

2
ATTEMPT POSSESSION NARCOTICS
2
CHILD ABANDONMENT
2
CHILD ABDUCTION/STRANGER
2
FALSE POLICE REPORT
2
GUN OFFENDER: ANNUAL REGISTRATION
2
GUN OFFENDER: DUTY TO REGISTER
2
ILLEGAL POSSESSION BY MINOR
2
MANU/DELIVER: HEROIN(BRN/TAN)
2
PREDATORY
2
PUBLIC DEMONSTRATION
2
PUBLIC INDECENCY
2
THEFT BY LESSEE,MOTOR VEH
2
TO AIRPORT
2
UNLAWFUL USE HANDGUN
2
VIOLATION OF STALKING NO CONTACT ORDER
2
AGGRAVATED PO: OTHER DANG WEAP
3
AGGRAVATED: KNIFE/CUT INSTR
3
ATT CRIM SEXUAL ABUSE
3
ATTEMPT ARSON
3
ATTEMPT: ARMED-OTHER DANG WEAP
3
CALL OPERATION
3
COMPUTER FRAUD
3
DECEPTIVE COLLECTION PRACTICES
3
FORFEIT PROPERTY

3
OBSTRUCTING JUSTICE
3
POSSESSION OF BURGLARY TOOLS
3
RECKLESS FIREARM DISCHARGE
3
SEX OFFENDER: FAIL TO REGISTER
3
STOLEN PROP: BUY/RECEIVE/POS.
3
UNLAWFUL INTERFERE/VISITATION
3
UNLAWFUL POSS OTHER FIREARM
3
VEHICULAR HIJACKING
3
ARMED: OTHER FIREARM
4
BOMB THREAT
4
COUNTERFEIT CHECK
4
CRIMINAL SEXUAL ABUSE
4
OTHER VIOLATION
4
POSS FIREARM/AMMO:NO FOID CARD
4
INTIMIDATION
5
OBSCENE MATTER
5
OTHER PROSTITUTION OFFENSE
5
SOLICIT FOR BUSINESS
5
TO STATE SUP PROP
5
UNLAWFUL POSS AMMUNITION
5
AGG CRIMINAL SEXUAL ABUSE
6
AGGRAVATED OF A SENIOR CITIZEN
6
ARMED: OTHER DANGEROUS WEAPON
6
CHILD ABDUCTION

6

LIQUOR LICENSE VIOLATION

6

POSS: BARBITUATES

6

AGG: HANDS/FIST/FEET SERIOUS INJURY

7

AGGRAVATED: OTHER FIREARM

7

ARSON THREAT

7

ATTEMPT: AGGRAVATED

7

BOGUS CHECK

7

CYCLE, SCOOTER, BIKE W-VIN

7

OTHER WEAPONS VIOLATION

7

UNLAWFUL USE OTHER DANG WEAPON

7

ENDANGERING LIFE/HEALTH CHILD

8

OTHER CRIME AGAINST PERSON

8

POSS: HEROIN(BRN/TAN)

8

AGGRAVATED DOMESTIC BATTERY: HANDS/FIST/FEET SERIOUS INJURY

9

AGGRAVATED: OTHER

9

MANU/DELIVER:CRACK

9

POSS: HALLUCINOGENS

9

POSS: SYNTHETIC DRUGS

9

ARMED:KNIFE/CUTTING INSTRUMENT

10

ATTEMPT: STRONGARM-NO WEAPON

10

ATTEMPT: ARMED-HANDGUN

11

BY FIRE

11

SEX OFFENDER: FAIL REG NEW ADD

11

FIRST DEGREE MURDER

12

LOST PASSPORT

12

TRUCK, BUS, MOTOR HOME

12

ATTEMPT - FINANCIAL IDENTITY THEFT

13

HOME INVASION

13

OTHER CRIME INVOLVING PROPERTY

13

FORGERY

14

MANU/DEL:CANNABIS OVER 10 GMS

14

POSS: PCP

14

POSSESSION OF DRUG EQUIPMENT

14

VEHICLE TITLE/REG OFFENSE

15

AGGRAVATED VEHICULAR HIJACKING

16

THEFT OF LOST/MISLAID PROP

16

LICENSE VIOLATION

17

ATTEMPT THEFT

19

COUNTERFEITING DOCUMENT

19

FOUND SUSPECT NARCOTICS

19

POSS: CANNABIS MORE THAN 30GMS

20

RESIST/OBSTRUCT/DISARM OFFICER

20

MANU/DEL:CANNABIS 10GM OR LESS

21

FALSE/STOLEN/ALTERED TRP

22

OTHER OFFENSE

22

SOLICIT NARCOTICS ON PUBLICWAY

22

AGGRAVATED DOMESTIC BATTERY: KNIFE/CUTTING INST

23

SOLICIT ON PUBLIC WAY

23
TO CITY OF CHICAGO PROPERTY
23
TO STATE SUP LAND
24
FINANCIAL IDENTITY THEFT $300 AND UNDER
26
AGGRAVATED DOMESTIC BATTERY: OTHER DANG WEAPON
27
ATTEMPT FORCIBLE ENTRY
27
PURSE-SNATCHING
28
ATT: AUTOMOBILE
30
THEFT/RECOVERY: AUTOMOBILE
30
OBSTRUCTING IDENTIFICATION
31
MANU/DELIVER: HEROIN (WHITE)
33
NON-AGGRAVATED
33
CHILD ABUSE
34
OTHER VEHICLE OFFENSE
35
THEFT OF LABOR/SERVICES
36
AGG PO HANDS NO/MIN INJURY
37
AGGRAVATED
37
FRAUD OR CONFIDENCE GAME
39
POSS: COCAINE
39
PAROLE VIOLATION
47
VIOLATE ORDER OF PROTECTION
49
ILLEGAL USE CASH CARD
50
TO RESIDENCE
52
CRIMINAL DEFACEMENT
55
FINANCIAL IDENTITY THEFT OVER $ 300

55
POCKET-PICKING
58
HARASSMENT BY ELECTRONIC MEANS
62
PRO EMP HANDS NO/MIN INJURY
63
RECKLESS CONDUCT
66
UNLAWFUL POSS OF HANDGUN
67
AGGRAVATED:KNIFE/CUTTING INSTR
71
POSS: CRACK
80
CREDIT CARD FRAUD
94
AGGRAVATED: OTHER DANG WEAPON
106
HARASSMENT BY TELEPHONE
113
AGGRAVATED: HANDGUN
119
STRONGARM - NO WEAPON
129
POSS: HEROIN(WHITE)
150
TELEPHONE THREAT
151
FROM BUILDING
165
UNLAWFUL ENTRY
170
TO LAND
175
ARMED: HANDGUN
206
RETAIL THEFT
356
AUTOMOBILE
372
FORCIBLE ENTRY
408
POSS: CANNABIS 30GMS OR LESS
411
OVER $500
548
TO PROPERTY

```
                                     558
                              TO VEHICLE
                                     579
                   DOMESTIC BATTERY SIMPLE
                                     949
                                  SIMPLE
                                     997
                         $500 AND UNDER
                                    1072
```

Note how we can use the $ to extract just the `Primary.Type` and just the `Description` components of the dataset.

What kinds of crimes occur in Chicago's District 10?

```
sort(table(chicagoCrime$Primary.Type[chicagoCrime$District==10]))
```

```
      CRIM SEXUAL ASSAULT                  HOMICIDE
                        1                         1
             NON-CRIMINAL            NON - CRIMINAL
                        1                         1
                OBSCENITY               SEX OFFENSE
                        1                         1
             INTIMIDATION         WEAPONS VIOLATION
                        2                         3
OFFENSE INVOLVING CHILDREN     PUBLIC PEACE VIOLATION
                        6                         7
        CRIMINAL TRESPASS        DECEPTIVE PRACTICE
                        8                         8
                  ASSAULT                   ROBBERY
                       15                        20
      MOTOR VEHICLE THEFT             OTHER OFFENSE
                       26                        29
                 BURGLARY                 NARCOTICS
                       32                        56
          CRIMINAL DAMAGE                     THEFT
                       68                        76
                  BATTERY
                       89
```

All these `chicagoCrime$`s are making our code long and harder to read. But we need to tell R to look inside `chicagoCrime` to find `Primary.Type` and `District`. `with()` can greatly simplify R code. Tell R to sort the table as before, but tell R that it can find all of the variables it is looking for in the `chicagoCrime` data frame.

```
with(chicagoCrime, sort(table(Primary.Type[District==10])))
```

```
      CRIM SEXUAL ASSAULT                  HOMICIDE
                        1                         1
```

```
           NON-CRIMINAL             NON - CRIMINAL
                      1                          1
              OBSCENITY                SEX OFFENSE
                      1                          1
           INTIMIDATION          WEAPONS VIOLATION
                      2                          3
OFFENSE INVOLVING CHILDREN    PUBLIC PEACE VIOLATION
                      6                          7
       CRIMINAL TRESPASS          DECEPTIVE PRACTICE
                      8                          8
                ASSAULT                    ROBBERY
                     15                         20
     MOTOR VEHICLE THEFT              OTHER OFFENSE
                     26                         29
               BURGLARY                  NARCOTICS
                     32                         56
         CRIMINAL DAMAGE                      THEFT
                     68                         76
                BATTERY
                     89
```

Much easier to read and understand!

Exercises

21. Display three randomly selected rows
22. Count `NA`s in each column
23. Look up `Location.Description`, `Block`, `Beat`, and `Ward` for those missing `Latitude`

For loops

Sometimes we need to have R repeat certain tasks multiple times, such as marching through each row of a dataset and modifying values. For loops accomplish this. Later in this course we will be using Google Maps to extract information about addresses. So we might need to iterate through every row in the dataset, check whether the latitude and longitude are missing, and if missing try to retrieve the latitude and longitude from Google Maps. The last crime in the dataset missing coordinates is in row 9954.

```
chicagoCrime[9954,]
```

```
          ID Case.Number                   Date             Block IUCR
9954 9868731    HX518764 11/24/2014 05:45:00 AM 081XX S THROOP ST 031A
     Primary.Type    Description Location.Description Arrest Domestic Beat
9954      ROBBERY ARMED: HANDGUN             SIDEWALK  false    false  613
     District Ward Community.Area FBI.Code X.Coordinate Y.Coordinate Year
9954       NA   21             71       03           NA           NA 2014
                 Updated.On Latitude Longitude Location
```

```
9954 12/01/2014 12:41:33 PM        NA        NA
```

While the coordinates are missing, the street address, 081XX S THROOP ST, is (mostly) there. Chicago PD has masked the last two digits of the address so that we really only know the location down to the nearest block. Let's look up 8150 S Throop St, likely near the middle of the block, to see where this is. The Google Maps URL is https://www.google.com/maps/place/8150+S+Throop+St,+Chicago,+IL. It would be a pain to have type out each of these URLs for every address that we wanted to look up. So let's learn a little bit about for loops to see how this might work.

Here is a basic for loop that runs through the numbers 1 to 10 and prints them out one at a time.

```
for(i in 1:10)
{
   print(i)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

Note the basic structure. There's the keyword `for`. Inside the parentheses is a variable `i` (but you can use any variable name you want), the keyword `in`, and finally a collection of values, in this case the numbers 1 to 10. The for loop will march through this collection of values, assigning `i` each value in turn, and running the code inside the squiggly braces. So first `i` will be set to 1 and the `print()` function will print the value 1 to the screen. When that is done, `i` will take the next value in the collection, a 2, and the for loop will run the `print()` function will print the number 2. This continues until `i` takes the value 10 and `print()` prints that 10 to the screen.

Let's loop through all the states, printing out which number they are in the collection along with the state postal code.

```
for(i.state in 1:length(state.names))
{
   print(c(i.state,state.names[i.state]))
}
```

```
[1] "1"  "WV"
[1] "2"  "OH"
[1] "3"  "OK"
[1] "4"  "NV"
[1] "5"  "CA"
[1] "6"  "IN"
[1] "7"  "MA"
[1] "8"  "MI"
```

```
[1] "9"  "IL"
[1] "10" "IA"
[1] "11" "SC"
[1] "12" "NH"
[1] "13" "LA"
[1] "14" "GA"
[1] "15" "CT"
[1] "16" "WI"
[1] "17" "CO"
[1] "18" "NY"
[1] "19" "UT"
[1] "20" "AK"
[1] "21" "MS"
[1] "22" "AL"
[1] "23" "OR"
[1] "24" "MT"
[1] "25" "ND"
[1] "26" "WY"
[1] "27" "FL"
[1] "28" "ME"
[1] "29" "AZ"
[1] "30" "TN"
[1] "31" "PA"
[1] "32" "MN"
[1] "33" "NM"
[1] "34" "SD"
[1] "35" "MO"
[1] "36" "RI"
[1] "37" "HI"
[1] "38" "WA"
[1] "39" "DE"
[1] "40" "NJ"
[1] "41" "NE"
[1] "42" "KY"
[1] "43" "AR"
[1] "44" "TX"
[1] "45" "NC"
[1] "46" "MD"
[1] "47" "VA"
[1] "48" "VT"
[1] "49" "KS"
[1] "50" "ID"
[1] "51" "DC"
```

Let's loop through all the letters of the alphabet and see if that letter is in the word "CRIME". `cat()` is like `print()`, but just dumps to the screen exactly what you give it[1]. `print()` will do some

[1] Why "cat" you ask? Programmers in the early 1970s created a program called "cat" to concatenate files together, but most uses of "cat" were to just dump file contents to the screen or to some other program.

formatting to try to present the results a little nicer.

```
for(letter in c("A","B","C","D","E","F","G","H","I","J","K","L","M","N","O",
                "P","Q","R","S","T","U","V","W","X","Y","Z"))
{
   print(letter)
   if(letter %in% c("C","R","I","M","E"))
      cat("The letter",letter,"is in the word 'CRIME'\n")
}
```

```
[1] "A"
[1] "B"
[1] "C"
The letter C is in the word 'CRIME'
[1] "D"
[1] "E"
The letter E is in the word 'CRIME'
[1] "F"
[1] "G"
[1] "H"
[1] "I"
The letter I is in the word 'CRIME'
[1] "J"
[1] "K"
[1] "L"
[1] "M"
The letter M is in the word 'CRIME'
[1] "N"
[1] "O"
[1] "P"
[1] "Q"
[1] "R"
The letter R is in the word 'CRIME'
[1] "S"
[1] "T"
[1] "U"
[1] "V"
[1] "W"
[1] "X"
[1] "Y"
[1] "Z"
```

Actually, R has a built in collection, `LETTERS`, that contains all of the capital letters. There really was no need to type them all out. This works too.

```
for(letter in LETTERS)
{
   print(letter)
   if(letter %in% c("C","R","I","M","E"))
```

```
    cat("The letter",letter,"is in the word 'CRIME'\n")
}
```

```
[1] "A"
[1] "B"
[1] "C"
The letter C is in the word 'CRIME'
[1] "D"
[1] "E"
The letter E is in the word 'CRIME'
[1] "F"
[1] "G"
[1] "H"
[1] "I"
The letter I is in the word 'CRIME'
[1] "J"
[1] "K"
[1] "L"
[1] "M"
The letter M is in the word 'CRIME'
[1] "N"
[1] "O"
[1] "P"
[1] "Q"
[1] "R"
The letter R is in the word 'CRIME'
[1] "S"
[1] "T"
[1] "U"
[1] "V"
[1] "W"
[1] "X"
[1] "Y"
[1] "Z"
```

Let's loop through the states and check each one whether or not it is an island.

```
for(nm.state in state.names)
{
  print(nm.state)
  if(is.island(nm.state))
    cat(nm.state," is an island\n")
}
```

```
[1] "WV"
[1] "OH"
[1] "OK"
[1] "NV"
[1] "CA"
```

```
[1] "IN"
[1] "MA"
[1] "MI"
[1] "IL"
[1] "IA"
[1] "SC"
[1] "NH"
[1] "LA"
[1] "GA"
[1] "CT"
[1] "WI"
[1] "CO"
[1] "NY"
[1] "UT"
[1] "AK"
[1] "MS"
[1] "AL"
[1] "OR"
[1] "MT"
[1] "ND"
[1] "WY"
[1] "FL"
[1] "ME"
[1] "AZ"
[1] "TN"
[1] "PA"
[1] "MN"
[1] "NM"
[1] "SD"
[1] "MO"
[1] "RI"
[1] "HI"
HI  is an island
[1] "WA"
[1] "DE"
[1] "NJ"
[1] "NE"
[1] "KY"
[1] "AR"
[1] "TX"
[1] "NC"
[1] "MD"
[1] "VA"
[1] "VT"
[1] "KS"
[1] "ID"
[1] "DC"
```

Let's get back to our original problem of having R construct all the Google Map URLs that we need. First, we will create a new variable in the dataset called `google.maps.url` and fill it with empty text.

```
chicagoCrime$google.maps.url <- ""
```

Now let's loop through all 10,000 rows in the dataset. First, R will use `gsub()` to replace the XX in the house number with 50, so we get the location in the middle of the block. `gsub()` is like a Find-and-Replace function, but way more powerful and flexible. We will use it extensively when covering regular expressions. After fixing the house number, we use `paste()` to assemble a URL suitable for looking up addresses on Google Maps.

```
time4ForLoop <- system.time(  # system.time() is like a stop watch
for(i in 1:nrow(chicagoCrime))
{
   a <- gsub("XX", "50", chicagoCrime$Block[i])
   chicagoCrime$google.maps.url[i] <- paste("https://www.google.com/maps/place/",
                                            a,
                                            ",+Chicago,+IL",sep="")
}
)
```

Note that we've wrapped the for loop with a call to `system.time()`. This will keep the time on how long this for loop takes. When creating these notes on a laptop it took 0.8 seconds. Not bad. Much faster than having to type out these 10,000 URLs. However, if we had one million addresses, then this code is going to take much more time.

In fact, in R for loops are *very* slow. They are so slow that R programmers attempt to avoid them whenever possible. We can actually accomplish the same task without using a for loop. `gsub()` will accept a whole collection of addresses and modify them all at once. `paste()` also will accept a collection of text values and paste them together with the other parts.

```
timeWithoutForLoop <- system.time(
{
a <- gsub("XX","50",chicagoCrime$Block)
chicagoCrime$google.maps.url <- paste("https://www.google.com/maps/place/",
                                      a,
                                      ",+Chicago,+IL",sep="")
}
)
```

This took 0.02 seconds. That's 40 times faster than the for loop.

Exercises

24. Use a for loop to create a variable `Coordinates` that looks like "(X.Coordinate,Y.Coordinate)"
    - Use `paste()` with the `X.Coordinate` and `Y.Coordinate` variables
    - Remember the `sep=` option in `paste()`
    - You might find using the `with()` function to simplify your code and avoid having a lot of `chicagoCrime$s`

25. Redo the previous exercise without using a for loop and compare computation time

More tabulating, aggregating, and breaking statistics down by group

The variable `Arrest` indicates whether someone was arrested for the crime. Here are the first 10 values.

```
chicagoCrime$Arrest[1:10]
```

```
 [1] "true"  "false" "false" "false" "false" "true"  "true"  "true" 
 [9] "true"  "true" 
```

We can compute the percentage of crimes with an arrest by calculating how often on average `Arrest=="true"`.

```
mean(chicagoCrime$Arrest=="true")
```

```
[1] 0.2398
```

The `aggregate()` function will do this same calculation, but has options for breaking it down by some other crime feature. Let's use `aggregate()` to compute the percentage of crimes with an arrest by ward. We store the result in `a`.

```
a <- aggregate((Arrest=="true")~Ward, data=chicagoCrime, mean)
a
```

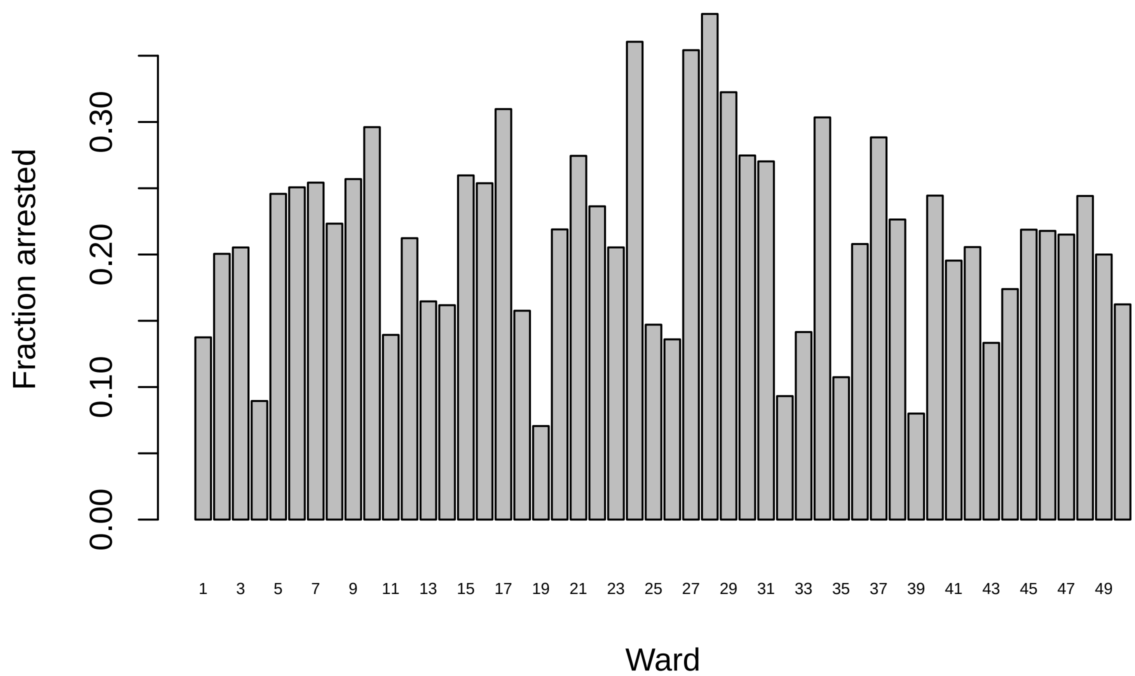
```
   Ward (Arrest == "true")
1     1         0.13750000
2     2         0.20050761
3     3         0.20532319
4     4         0.08947368
5     5         0.24576271
6     6         0.25069638
7     7         0.25423729
8     8         0.22330097
9     9         0.25691700
10   10         0.29611650
11   11         0.13934426
12   12         0.21232877
13   13         0.16463415
14   14         0.16176471
15   15         0.25968992
16   16         0.25378788
17   17         0.30973451
18   18         0.15757576
19   19         0.07058824
20   20         0.21895425
21   21         0.27444795
22   22         0.23636364
23   23         0.20535714
```

```
24   24          0.36051502
25   25          0.14705882
26   26          0.13600000
27   27          0.35416667
28   28          0.38152610
29   29          0.32246377
30   30          0.27472527
31   31          0.27027027
32   32          0.09316770
33   33          0.14150943
34   34          0.30344828
35   35          0.10743802
36   36          0.20792079
37   37          0.28838951
38   38          0.22641509
39   39          0.08000000
40   40          0.24444444
41   41          0.19540230
42   42          0.20560748
43   43          0.13333333
44   44          0.17391304
45   45          0.21875000
46   46          0.21782178
47   47          0.21505376
48   48          0.24418605
49   49          0.20000000
50   50          0.16239316
```

The first part of `aggregate()` gives an R formula for how we want the data broken up. On the left of the ~ is the outcome or feature that we want to study. Here it is whether or not `Arrest` has value true. To the right of the ~ is the feature by which we want to break down the arrests, ward in this case. Then we need to tell `aggregate()` in which data frame it can find `Arrest` and `Ward`. Lastly, we need to tell `aggregate()` what to do with the outcome we are studying. Here we are asking `aggregate()` to compute the mean so that we get an arrest percentage.

We can use `barplot()` to compare arrest percentages by ward.

```
barplot(a$`(Arrest == "true")`,
        names.arg = a$Ward,
        cex.names = 0.5,
        ylab      = "Fraction arrested",
        xlab      = "Ward")
```



Note that the column in a contain the arrest fraction has a complicated name with several special symbols like == and ". R will get very confused unless we "protect" this variable name with the backquotes (also called backticks). You can visit the help for `barplot()` with `?barplot` to learn what all the arguments do.

Frequently we will not to focus on just a subset of the data. For example, we might just want to study assaults rather than all crimes. The `subset()` function does this for us like `subset(data, Primary.Type=="ASSAULT")`. This is particularly useful to use in combination with `with()`. Let's create a table of the number of arrests by ward, but only for assaults.

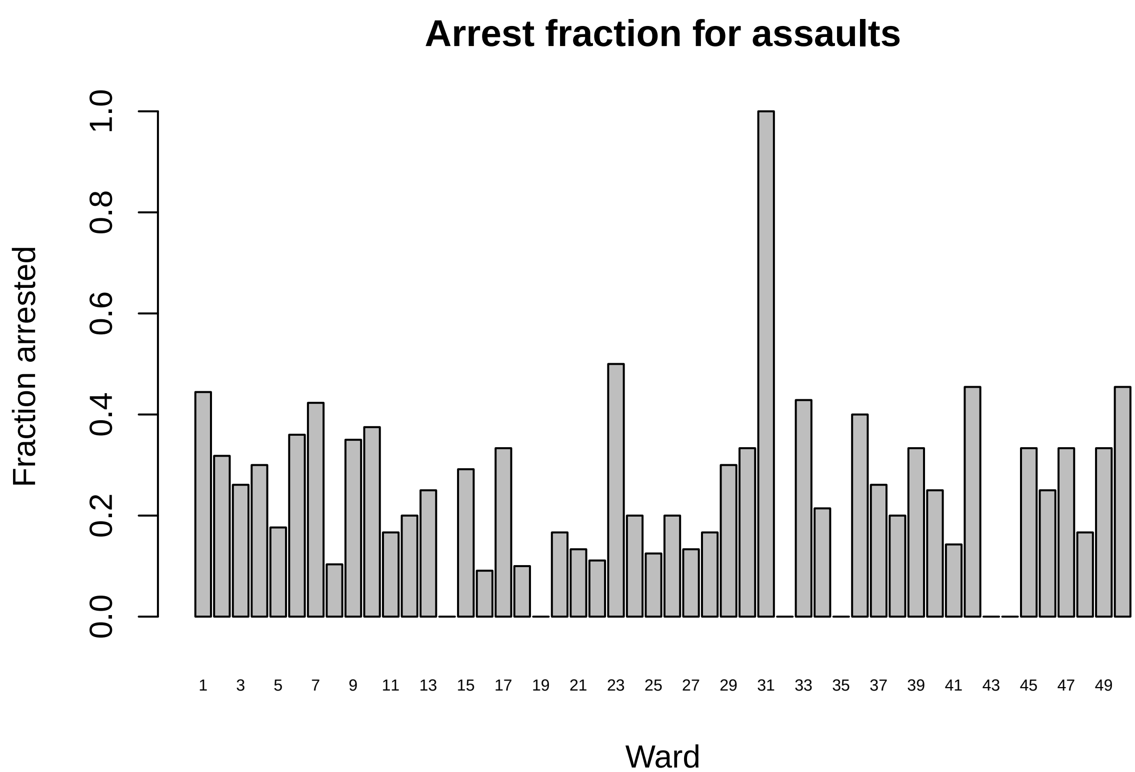
```
with(subset(chicagoCrime,Primary.Type=="ASSAULT"),
     table(Arrest,Ward))
```

```
       Ward
Arrest   1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
  false  5 15 17  7 14 16 15 26 13 10  5  4  6  5 17 20 12  9  6 15 13  8
  true   4  7  6  3  3  9 11  3  7  6  1  1  2  0  7  2  6  1  0  3  2  1
       Ward
Arrest  23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
  false  2 12  7  8 13 25  7  4  0  2  4 11  7  3 17  4  2  3  6  6  6  1
  true   2  3  1  2  2  5  3  2  3  0  3  3  0  2  6  1  1  1  1  5  0  0
       Ward
Arrest  45 46 47 48 49 50
  false  4  6  2  5  4  6
```

```
  true    2  2  1  1  2  5
```

Let's recreate our barplot, but now just using assaults.

```r
a <- aggregate((Arrest=="true")~Ward,
               data=subset(chicagoCrime,Primary.Type=="ASSAULT"),
               mean)
barplot(a$`(Arrest == "true")`,
        names.arg = a$Ward,
        cex.names = 0.5,
        ylab      = "Fraction arrested",
        xlab      = "Ward",
        main      = "Arrest fraction for assaults")
```



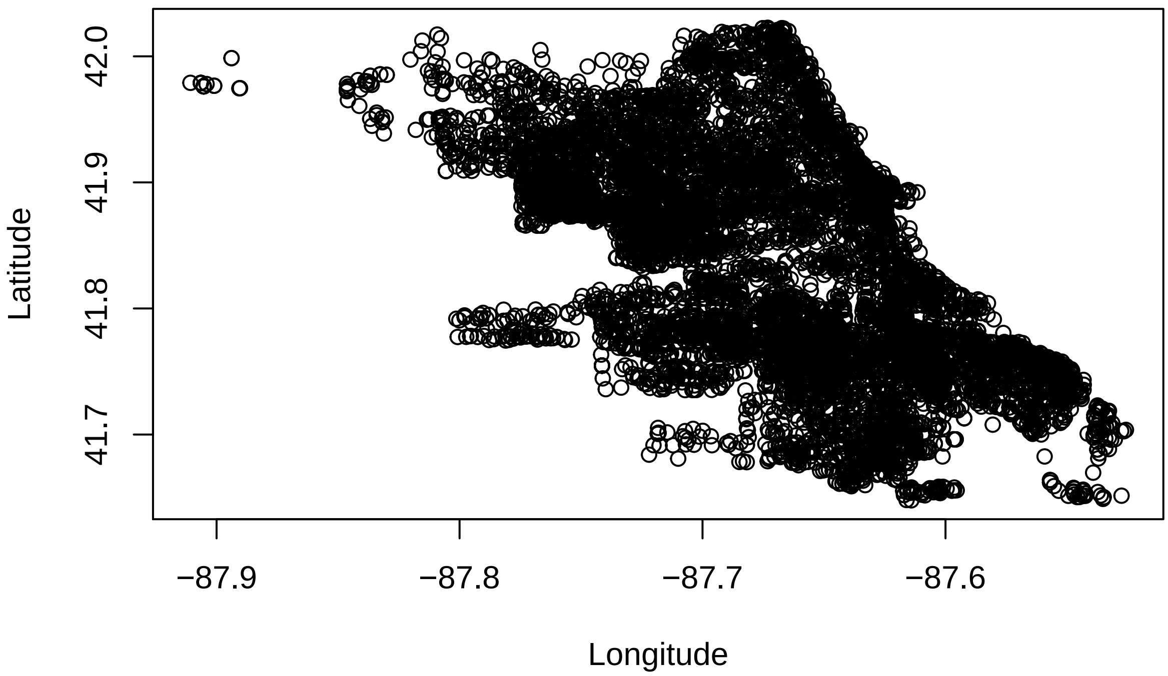
Exercises

26. How many assaults occurred in the street? (`Location.Description=="STREET"`). Try using `subset()` even though there are other ways
27. What percentage of assaults occurred in the street by Ward?

Plotting Data

R enables us to plot points. The points we plotted form the shape of Chicago... which makes total sense because we're using Chicago crime data.

```
plot(Latitude~Longitude, data=chicagoCrime)
```

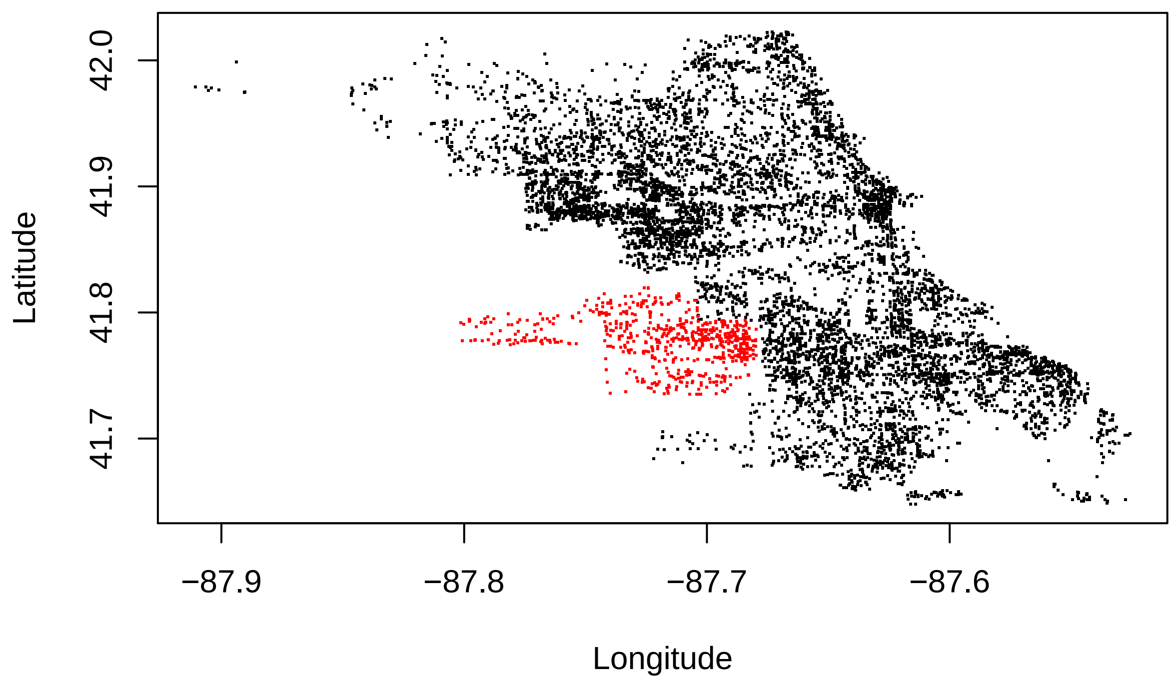


The `plot()` function here uses the same R formula syntax as the `aggregate()` function. The variable on the left of ~ is the outcome, plotted on the y-axis, and the variable on the right appears on the x-axis. And, of course, we need to tell `plot()` that it can find these variables inside the `chicagoCrime` data frame.

Let's plot the district with the most crime. The first line here tabulates how many crimes occurred in each district, sorts those counts, reverse the sorted list so that the largest one comes first, extracts the first one in the collection using `[1]` and then uses `names()` to extract the name of the district (rather than how many crimes occurred in that district). You can see all of District 8's crimes (that's the district with the most crimes) appearing as red points in the plot.

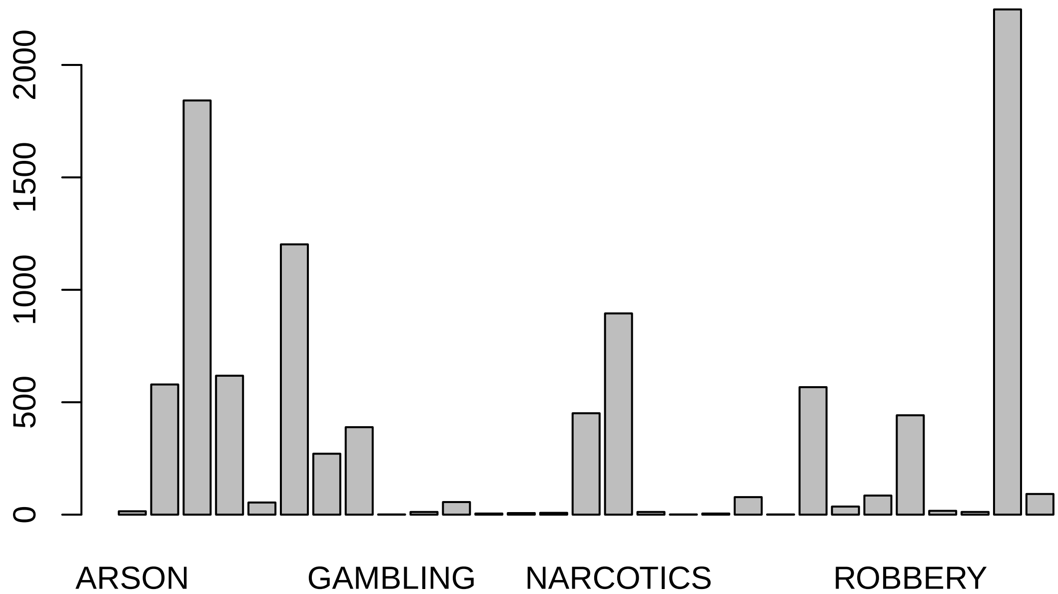
```
# selects district 8, with 713 crimes
max.district <- names(rev(sort(table(chicagoCrime$District)))[1])
plot(Latitude~Longitude,
     data=subset(chicagoCrime, District!=max.district),   # not in District 8
     pch=".",                                              # plot with tiny dot
     xlab="Longitude",ylab="Latitude")
```

```
points(Latitude~Longitude,
       data=subset(chicagoCrime, District==max.district), # in District 8
       pch=".",
       col="red")
```



R tries to set up default graphics settings so that most plots look okay, but sometimes it takes a little more work to adjust them. The good thing is that R lets you adjust everything. So let's make a barplot of the number of crimes of each type.

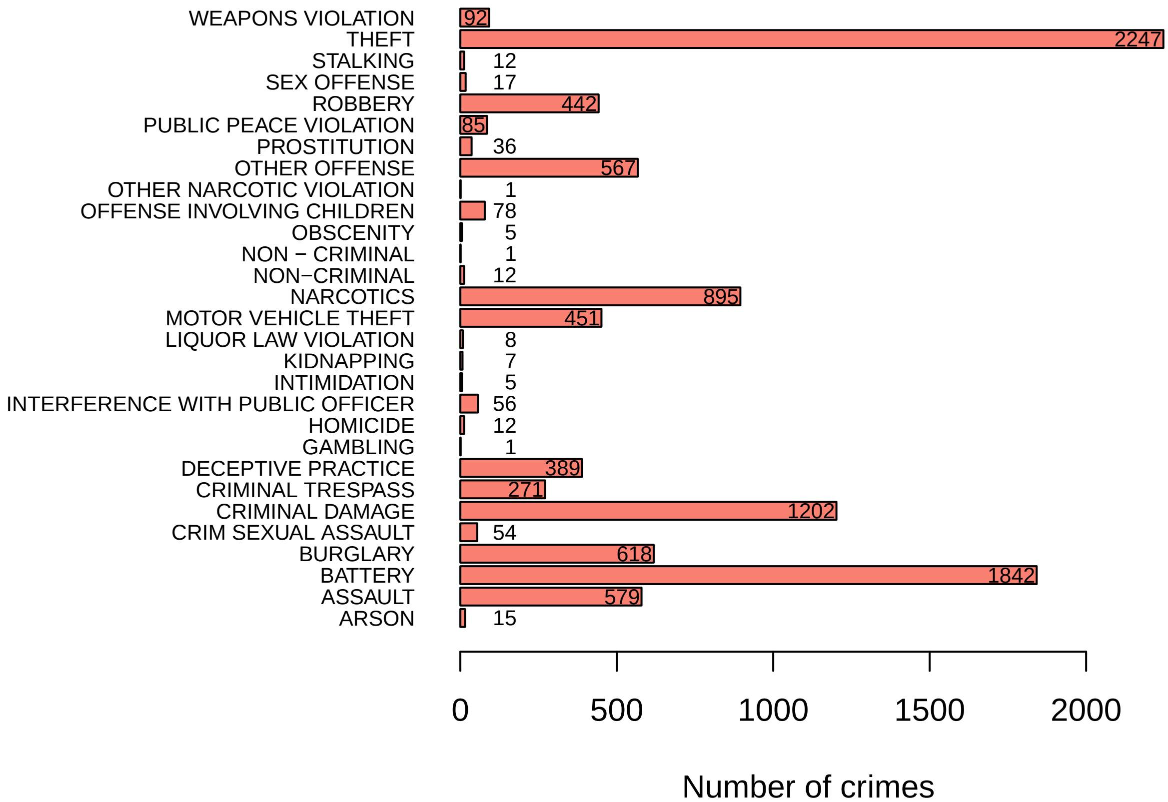
```
barplot(table(chicagoCrime$Primary.Type))
```



The labels on the bars are so long that only a few of them appear. So let's spend a little more time, write a few more lines of R code, and make this plot look right.

```
tab <- table(chicagoCrime$Primary.Type)   # tabulate crime counts
# give 2.5in on the left margin to give lots of space for the crime type labels
par(pin=c(6.5,6),                         # set plot dimensions (inches)
    mai=c(1.02, 2.5, 0, 0.3))             # set plot margins
a <- barplot(tab,
             col="salmon",                # change the bars' color
             horiz=TRUE,                  # make the bars horizontal
             names.arg=rep("",nrow(tab)), # put no labels on the bars
             xlab="Number of crimes")
# add the bar labels on the y-axis
axis(2,                                   # set up the y-axis label (axis #2)
     at=a[,1],                            # midpoints of bars stored in a[,1]
     cex.axis=0.7,                        # shrink the axis text size by 30%
     labels=names(tab),                   # the bar labels
     las=1,                               # make labels horizonal (see ?par)
     tick=FALSE)                          # no tick marks on the axis
# add the actual number on the bars
text(ifelse(tab<80, 180, tab-5),          # x-coord of text,
                                          #   if bar too small, put text to right
     a[,1],                               # y-coord of text,  midpoint of bars
```

```
    tab,                                 # text to add to the plot
    cex=0.7,                             # shrink text (cex=character expansion)
    adj=1)                               # right justify text
```



Exercises

28. Make a barplot indicating how many states are in each region. Use `state.list`
29. Identify the beat with the most crimes
30. Identify the beat with the most domestic violence incidents
31. Part 1 crimes are homicide, robbery, assault, arson, burglary, theft, rape, motor vehicle theft. Calculate the number of Part 1 crimes in Chicago

Solutions to the exercises

1. Print all even numbers less than 100

```
(1:49)*2
```

```
 [1]  2  4  6  8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46
[24] 48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80 82 84 86 88 90 92
```

```
[47] 94 96 98
```

or

```
seq(2,98,by=2)
```

```
 [1]  2  4  6  8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46
[24] 48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80 82 84 86 88 90 92
[47] 94 96 98
```

2. What is the mean of even numbers less than 100

```
mean((1:49)*2)
```

```
[1] 50
```

3. Have R put in alphabetical order `c("WA","DC","CA","PA","MD","VA","OH")`

```
sort(c("WA","DC","CA","PA","MD","VA","OH"))
```

```
[1] "CA" "DC" "MD" "OH" "PA" "VA" "WA"
```

4. What's the last state in the `state.names`?

```
state.names[51]
```

```
[1] "DC"
```

5. Pick out states that begin with "M" using their indices

```
state.names[c(7,8,21,24,28,32,35,46)]
```

```
[1] "MA" "MI" "MS" "MT" "ME" "MN" "MO" "MD"
```

or sort first so that all the M states are together

```
sort(state.names)[20:27]
```

```
[1] "MA" "MD" "ME" "MI" "MN" "MO" "MS" "MT"
```

Here's another possible answer that uses `substring` (which we haven't covered yet):

```
state.names[substring(state.names, 1, 1)=="M"]
```

```
[1] "MA" "MI" "MS" "MT" "ME" "MN" "MO" "MD"
```

6. Pick out states where you have lived Of course, these may vary depending on where you have lived.

```
state.names[c(1, 4, 10, 26)]
```

```
[1] "WV" "NV" "IA" "WY"
```

7. What's the last state in alphabetical order?

```
sort(state.names)[51]
```

```
[1] "WY"
```

or

```r
rev(sort(state.names))[1]
```

```
[1] "WY"
```

8. What are the last three states in alphabetical order?

```r
rev(sort(state.names))[1:3]
```

```
[1] "WY" "WV" "WI"
```

9. Report TRUE or FALSE for each state depending on if you have lived there

```r
my.states <- c("PA", "NJ", "NY", "MD", "DE", "MA", "RI", "CT", "ME", "LA", "IN")
state.names %in% my.states
```

```
 [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
[12] FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
[23] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE
[34] FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE
[45] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
```

10. With `a <- 1:100`, pick out odd numbers between 50 and 75

```r
a <- 1:100
a[a %% 2==1 & a>50 & a<75]
```

```
 [1] 51 53 55 57 59 61 63 65 67 69 71 73
```

11. Use greater than less than signs to get all state names that begin with M

```r
state.names[state.names>"LZ" & state.names<"N"]
```

```
[1] "MA" "MI" "MS" "MT" "ME" "MN" "MO" "MD"
```

12. Use `sample()` to estimate the probability of rolling a 6

```r
a <- sample(1:6, size=100000, replace=TRUE)
table(a)[6]/length(a)
```

```
      6
0.16469
```

Or

```r
sum(a==6)/length(a)
```

```
[1] 0.16469
```

Or

```r
mean(a==6)
```

```
[1] 0.16469
```

13. Use `sample()` to estimate the probability that the sum of two die equal 7

```r
dice1 <- sample(1:6, size=1000, replace=TRUE)
dice2 <- sample(1:6, size=1000, replace=TRUE)
```

```
doubleroll <- dice1 + dice2
mean(doubleroll==7)    # should be close to 1/6 or 0.1666...
```

```
[1] 0.187
```

14. Use `sample()` to select randomly five states without replacement (Answers will vary)

```
sample(state.names, size=5, replace=FALSE)
```

```
[1] "NE" "RI" "OR" "MI" "TN"
```

15. Use `sample()` to select randomly 1000 states with replacement

- Tabulate how often each state was selected (Answers will vary)

```
a <- sample(state.names, size=1000, replace=TRUE)
table(a)
```

```
a
AK AL AR AZ CA CO CT DC DE FL GA HI IA ID IL IN KS KY LA MA MD ME MI MN MO
19 12 26 20 20 20 18 23 17 16 11 14 34 18 19 13 22 14 20 17 24 16 24 26 26
MS MT NC ND NE NH NJ NM NV NY OH OK OR PA RI SC SD TN TX UT VA VT WA WI WV
25 21 14 17 15 17 22 18 26 24 14 12 29 18 21 18 28 17 13 26 14 25 22 17 20
WY
18
```

- Which state was selected the least? (Answers will vary)

```
sort(table(a))[1]
```

```
GA
11
```

16. Fix `state.list` so that “DC” is in “other” rather than “east”

```
state.list$east <- state.list$east[state.list$east!="DC"]
state.list$other <- c(state.list$other, "DC")
state.list
```

```
$west
 [1] "AK" "HI" "WA" "NV" "CA" "CO" "UT" "OR" "AZ" "NM" "ID"

$east
 [1] "KY" "RI" "PA" "DE" "NJ" "WV" "MA" "SC" "NH" "GA" "CT" "NY" "IN" "MS"
[15] "AL" "OH" "NC" "MD" "VA" "VT" "FL" "ME" "TN"

$central
 [1] "SD" "MO" "MN" "ND" "WY" "OK" "MI" "IL" "IA" "LA" "WI" "MT" "NE" "AR"
[15] "TX" "KS"

$other
 [1] "AS" "GU" "MP" "PR" "VI" "UM" "FM" "MH" "PW" "DC"
```

Or

```
state.list$east <- setdiff(state.list$east, "DC")
state.list$other <- c(state.list$other, "DC")
state.list
```

```
$west
 [1] "AK" "HI" "WA" "NV" "CA" "CO" "UT" "OR" "AZ" "NM" "ID"

$east
 [1] "KY" "RI" "PA" "DE" "NJ" "WV" "MA" "SC" "NH" "GA" "CT" "NY" "IN" "MS"
[15] "AL" "OH" "NC" "MD" "VA" "VT" "FL" "ME" "TN"

$central
 [1] "SD" "MO" "MN" "ND" "WY" "OK" "MI" "IL" "IA" "LA" "WI" "MT" "NE" "AR"
[15] "TX" "KS"

$other
 [1] "AS" "GU" "MP" "PR" "VI" "UM" "FM" "MH" "PW" "DC" "DC"
```

17. Print out east and central states together sorted

```
sort(c(state.list$east, state.list$central))
```

```
 [1] "AL" "AR" "CT" "DE" "FL" "GA" "IA" "IL" "IN" "KS" "KY" "LA" "MA" "MD"
[15] "ME" "MI" "MN" "MO" "MS" "MT" "NC" "ND" "NE" "NH" "NJ" "NY" "OH" "OK"
[29] "PA" "RI" "SC" "SD" "TN" "TX" "VA" "VT" "WI" "WV" "WY"
```

Or

```
with(state.list, sort(c(east, central)))
```

```
 [1] "AL" "AR" "CT" "DE" "FL" "GA" "IA" "IL" "IN" "KS" "KY" "LA" "MA" "MD"
[15] "ME" "MI" "MN" "MO" "MS" "MT" "NC" "ND" "NE" "NH" "NJ" "NY" "OH" "OK"
[29] "PA" "RI" "SC" "SD" "TN" "TX" "VA" "VT" "WI" "WV" "WY"
```

18. Make a function `is.island(x)` returns `TRUE` if `x` is an island

```
is.island <- function(x)
{
   return(x %in% c("HI", "FM", "MH", "PW", "AS", "GU", "MP", "PR", "VI", "UM"))
}
```

19. Count how many islands are within each region. Use an `sapply()` (or two) and your new `is.island()` function

First, this `lapply()` asks each state if they are an island.

```
lapply(state.list, is.island)
```

```
$west
 [1] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE

$east
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
[12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[23] FALSE

$central
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[12] FALSE FALSE FALSE FALSE FALSE

$other
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE
```

Now we want to count up how many `TRUE`s there are in each component, so wrap this `lapply()` with an `sapply()`

```
sapply(lapply(state.list, is.island), sum)
```

```
   west    east central   other 
      1       0       0       9 
```

20. Which components of `b` having missing values? Use `is.na()`

```
sapply(lapply(b, is.na), any)
```

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

Or

```
b <- list(0:9, c("A","B","C"), c(TRUE,FALSE,NA))
sapply(b, function(x) any(is.na(x)))
```

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

21. Display three randomly selected rows

```
chicagoCrime[sample(1:nrow(chicagoCrime), size=3),]
```

```
          ID Case.Number                   Date                 Block IUCR
921  9883456    HX534710 12/08/2014 03:00:00 PM 085XX S INDIANA AVE 1320
8756 9871306    HX521688 11/26/2014 07:00:00 AM  006XX N PULASKI RD 0820
1373 9882515    HX533843 12/07/2014 07:45:00 PM  021XX N KEDZIE AVE 0460
        Primary.Type    Description           Location.Description Arrest
921  CRIMINAL DAMAGE     TO VEHICLE                         STREET  false
8756           THEFT $500 AND UNDER PARKING LOT/GARAGE(NON.RESID.)  false
1373         BATTERY         SIMPLE                         STREET  false
     Domestic Beat District Ward Community.Area FBI.Code X.Coordinate
921     false  632        6    6             44       14      1179129
8756    false 1111       11   28             23       06      1149565
1373    false 1413       14   26             22      08B      1154649
     Y.Coordinate Year             Updated.On Latitude Longitude
921       1848198 2014 12/15/2014 12:58:02 PM 41.73874 -87.61929
8756      1904170 2014 12/03/2014 12:41:59 PM 41.89296 -87.72616
1373      1914208 2014 12/14/2014 12:39:06 PM 41.92041 -87.70722
                          Location
921   (41.73874424, -87.619288253)
```

```
8756 (41.892961454, -87.726156636)
1373  (41.920406377, -87.70721585)
                                                      google.maps.url
921  https://www.google.com/maps/place/08550 S INDIANA AVE,+Chicago,+IL
8756  https://www.google.com/maps/place/00650 N PULASKI RD,+Chicago,+IL
1373  https://www.google.com/maps/place/02150 N KEDZIE AVE,+Chicago,+IL
```

22. Count `NA`s in each column

```r
sapply(lapply(chicagoCrime, is.na), sum)
```

```
                  ID          Case.Number                 Date
                   0                    0                    0
               Block                 IUCR         Primary.Type
                   0                    0                    0
         Description Location.Description               Arrest
                   0                    0                    0
            Domestic                 Beat             District
                   0                    0                  191
                Ward       Community.Area             FBI.Code
                   0                    0                    0
        X.Coordinate         Y.Coordinate                 Year
                 191                  191                    0
          Updated.On             Latitude            Longitude
                   0                  191                  191
            Location      google.maps.url
                   0                    0
```

Or

```r
sapply(chicagoCrime, function(x) sum(is.na(x)))
```

```
                  ID          Case.Number                 Date
                   0                    0                    0
               Block                 IUCR         Primary.Type
                   0                    0                    0
         Description Location.Description               Arrest
                   0                    0                    0
            Domestic                 Beat             District
                   0                    0                  191
                Ward       Community.Area             FBI.Code
                   0                    0                    0
        X.Coordinate         Y.Coordinate                 Year
                 191                  191                    0
          Updated.On             Latitude            Longitude
                   0                  191                  191
            Location      google.maps.url
                   0                    0
```

23. Look up `Location.Description`, `Block`, `Beat`, and `Ward` for those missing `Latitude`

```r
i <- is.na(chicagoCrime$Latitude)
# Let's just show the first 5 rows
i <- which(i)[1:5]
chicagoCrime[i,c("Location.Description","Block","Beat","Ward")]
```

```
    Location.Description                     Block Beat Ward
185                      010XX W HOLLYWOOD AVE 2022   48
313            APARTMENT      0000X W CERMAK RD  131    3
463                OTHER    013XX W MADISON ST 1224   27
530               STREET      033XX N KNOX AVE 1731   30
551          GAS STATION       001XX E 71ST ST  322    6
```

Or

```r
subset(chicagoCrime, is.na(chicagoCrime$Latitude),
       select=c("Location.Description","Block","Beat","Ward"))[1:5,]
```

```
    Location.Description                     Block Beat Ward
185                      010XX W HOLLYWOOD AVE 2022   48
313            APARTMENT      0000X W CERMAK RD  131    3
463                OTHER    013XX W MADISON ST 1224   27
530               STREET      033XX N KNOX AVE 1731   30
551          GAS STATION       001XX E 71ST ST  322    6
```

24. Use a for loop to create a variable `Coordinates` that looks like "(X.Coordinate,Y.Coordinate)"

```r
system.time(
for (i in 1:nrow(chicagoCrime))
{
   chicagoCrime$coords[i] <- paste0(chicagoCrime$X.Coordinate[i], ", " ,
                                    chicagoCrime$Y.Coordinate[i])
}
)
```

```
   user  system elapsed
   0.96    0.00    0.95
```

Or

```r
system.time(
for (i in 1:nrow(chicagoCrime))
{
   chicagoCrime$coords2[i] <- with(chicagoCrime,
                                   paste("(",X.Coordinate[i], ",",
                                         Y.Coordinate[i],")",sep=""))
}
)
```

```
   user  system elapsed
   0.95    0.00    0.95
```

25. Redo the previous exercise without using a for loop and compare computation time

```
system.time(
chicagoCrime$coords3 <- with(chicagoCrime,
                             paste0("(", X.Coordinate, ",",Y.Coordinate,")"))
)
```

```
   user  system elapsed 
   0.01    0.00    0.01 
```

26. How many assaults occurred in the street? (`Location.Description=="STREET"`)

```
with(subset(chicagoCrime, Primary.Type=="ASSAULT"),
     sum(chicagoCrime$Location.Description=="STREET"))
```

```
[1] 2353
```

27. What percentage of assaults occurred in the street by Ward?

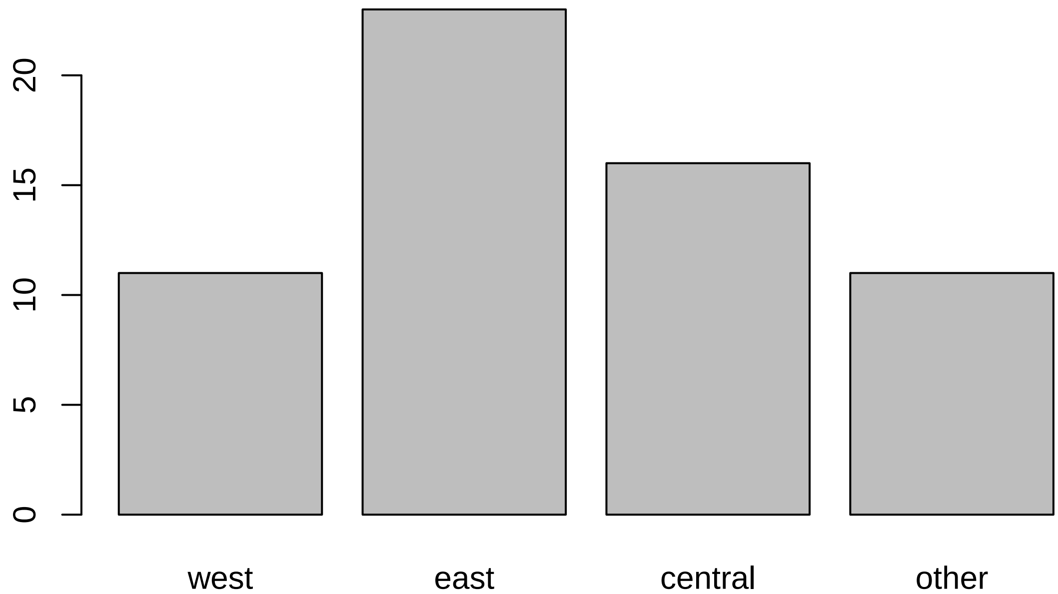
```
aggregate((Location.Description=="STREET")~Ward,
          data=subset(chicagoCrime, Primary.Type=="ASSAULT"),
          mean)
```

```
   Ward (Location.Description == "STREET")
1     1                         0.22222222
2     2                         0.13636364
3     3                         0.17391304
4     4                         0.10000000
5     5                         0.35294118
6     6                         0.24000000
7     7                         0.19230769
8     8                         0.06896552
9     9                         0.15000000
10   10                         0.25000000
11   11                         0.16666667
12   12                         0.20000000
13   13                         0.00000000
14   14                         0.00000000
15   15                         0.20833333
16   16                         0.13636364
17   17                         0.11111111
18   18                         0.00000000
19   19                         0.33333333
20   20                         0.11111111
21   21                         0.13333333
22   22                         0.22222222
23   23                         0.50000000
24   24                         0.13333333
25   25                         0.00000000
26   26                         0.10000000
27   27                         0.00000000
28   28                         0.20000000
```

```
29   29                          0.30000000
30   30                          0.33333333
31   31                          0.66666667
32   32                          0.50000000
33   33                          0.00000000
34   34                          0.21428571
35   35                          0.42857143
36   36                          0.60000000
37   37                          0.13043478
38   38                          0.20000000
39   39                          0.33333333
40   40                          0.25000000
41   41                          0.28571429
42   42                          0.09090909
43   43                          0.00000000
44   44                          0.00000000
45   45                          0.16666667
46   46                          0.25000000
47   47                          0.00000000
48   48                          0.16666667
49   49                          0.00000000
50   50                          0.18181818
```

28. Make a barplot indicating how many states are in each region. Use `state.list`

```r
barplot(sapply(state.list, length))
```



29. Identify the beat with the most crimes

```
names(rev(sort(table(chicagoCrime$Beat)))[1])
```

```
[1] "1533"
```

Or

```
names(which.max(table(chicagoCrime$Beat)))
```

```
[1] "1533"
```

30. Identify the beat with the most domestic violence incidents

```
with(subset(chicagoCrime, Description=="DOMESTIC BATTERY SIMPLE"),
     names(which.max(table(Beat))))
```

```
[1] "421"
```

31. Part 1 crimes are homicide, robbery, assault, arson, burglary, theft, rape, motor vehicle theft. Calculate the number of Part 1 crimes in Chicago

```
sum(chicagoCrime$Primary.Type %in% c("HOMICIDE", "ROBBERY", "ASSAULT", "ARSON",
                                     "BURGLARY", "THEFT", "SEX OFFENSE",
                                     "MOTOR VEHICLE THEFT"))
```

```
[1] 4381
```