Lecture 14:

Last time

We examined different ways to bring data into R; we discussed several convenience functions that were designed to handle certain kinds of files.

Many of these functions returned objects of class `data.frame`, the class in R that most closely resembles a data table in the style of Excel.

We then went through an easy-to-state but difficult-to-implement process of merging two data frames; we illustrated the process using the operator `%in%` and the function `match()`.

An aside

Recall from last time that `aggregate()` allowed us to combine data that referred to the same building; it is possible to implement this with lower-level tools in R.

Similarly, our dance with `match()` and `%in%` could be accomplished a little easier with the function `merge()` (although it is good to know about `match()` and how it works!).
Today

So far, we have invoked a large number of functions in R without really discussing what’s at work under the hood; today we see how R responds when you enter a command

In the spirit of re-examining and finishing analyses from the beginning of the quarter, we will have a look at the word distributions you computed from the Roberts hearings

Along the way, we will introduce some basic “control-of-flow” operations; we will also get a new take on subsetting and what’s really happening when you type x[1:10]

So far...

So far, we have mainly interacted with R by typing expressions at the prompt; they are then parsed and evaluated

We have printed or exhibited objects,

Identified subsets of objects like vectors, matrices, arrays, data frames and lists,

Performed arithmetic and evaluated special operators,

Created new objects by assignment, and

Invoked functions
Functions

As you might expect by now, functions are also objects, and their class is simply "function"

Most of the functions we have seen so far are written in R, meaning they are built from components in the language itself; consider, for example, the relationship between `scan`, `read.table` and `read.csv`

Combining functions and other constructions in R, you can, in effect, extend the language; John Chambers would cast this in terms of your role, that is, you move from user to programmer

The purpose of statistical software is to help in the process of learning from data. For many situations, the software's crucial contribution to the process is to allow the user to express ideas about the data, ideas that imply some desired view or summary. Expressing the idea to the software amounts to programming, though the user will not initially think of it as programming. The software should help in the deception, by making the expression of simple things simple. Simplicity makes demands on both the form of expression (the language and user interface) and the range of expression (the available tools). In other words, there must be some tool available that implements the desired view or summary (close enough to get started). The user must be able to identify the tool and to express the particular idea, connecting the tool to the data in a simple way. Otherwise the idea will remain unrealized and the underlying process of learning from data will suffer...

A new idea is usually only vaguely formed in the user’s mind and the software usually implements only an approximation to this idea. Some ideas never go beyond this stage: either the idea turned out not to be useful or (less often) the initial rough expression was all that was needed. Most useful ideas, however, continue through a process of gradual refinement. Perhaps the original idea was not quite what we meant, or there were additional requirements that only became obvious with experience. Perhaps we simply need to apply the idea, or a variation of it, to different data.

The software must support this gradual refinement by suitable programming facilities. Adding or changing details and re-using the idea in different contexts must be easy. If the train of thought proves useful, chances are that the programming aspect will gradually become more serious. The "idea" will gradually become itself a re-usable part of the user’s environment. The statistical software should help, by supporting each step from user to programmer, with as few intrusive barriers as possible...

Suggestion 1: The user interface should be integrated into the language and environment, so that the transition from user to programmer is nearly painless.

From Users, Programmers and Statistical Software, JCGS, 9:3, 404-422
Functions

The general form or syntax of a function is

```
function ( arglist ) body
```

The formal arguments of a function are combined into a comma-separated list; it can consist of a symbol or name, a statement of the form ‘name=expression’, or a special formal catch-all argument ‘...’

Within the body of a function we describe a set of computations of some kind; these can be expressed in terms of valid R expressions, and are often contained in curly braces { and }
Calling a function

When we call a function, we can supply values to one or more of the variables in the function’s argument list.

R employs call-by-value semantics meaning that the supplied arguments behave like local variables; changing the value of a supplied argument will not change its value elsewhere.

The arguments you supply are matched to elements in the formal argument list using three simple rules:

Argument matching

**Exact matching:** For each named argument we supply in the function call, R will search the list of formal arguments for one with exactly the same name.

**Partial matching:** Next, matches are made if the name of a supplied argument agrees with the beginning of one of the (previously unmatched) formal arguments.

**Positional matching:** Finally, formal arguments are matched in order (left to right) with the unnamed arguments we supplied in the function call.
Calling a function

R employs lazy evaluation of function arguments; that is, the arguments are not actually evaluated until they are needed.

For example, default expressions of the form ‘name=expression’ are evaluated when name is used in the body of the function.

In the next example, the $x$ in `levels` refers to the value of $x$ inside the function, and not one you have in your workspace (where else does this occur in the example?)
Functions

R has facilities for programming “on” and not just “in” the language; it is possible to grab details about the components of a function and change them.

You can also retrieve information about how a function was called or specifically invoke `eval()` on an expression object.

More on this later, but for now...

Example: Kullback-Leibler divergence

The Kullback-Leibler divergence between two probability distributions \( P \) and \( Q \) is defined to be

\[
D(P\|Q) = \sum_{x \in \mathcal{X}} P(x) \log \frac{P(x)}{Q(x)}
\]

where \( 0 \log 0/q = 0 \) and \( p \log p/0 = \infty \).

In many respects it acts as a measure of dissimilarity or “distance” between distributions.
Properties of KL divergence

Non-negativity (Gibbs inequality)

\[ D(P\|Q) \geq 0 \text{ with equality if and only if } \]
\[ P(x) = Q(x), \ x \in \mathcal{X} \]

Asymmetry

For any distributions \( P \) and \( Q \), \( D(P\|Q) \) and \( D(Q\|P) \) are not necessarily the same.
Adding some protection

Suppose we feed this function two vectors that represent probability functions over different sets of numbers; right now, we may or may not get an error; why?

We can add some simple checks to make sure the data are in the form we expect; why do this?

Are we really getting what we want?

When defining the Kullback-Leibler divergence, we used some simple conventions to deal with zeroes in either of the two distributions

Is R doing the right thing?
A little more interesting

The asymmetry of KL divergence can be problematic when we want to compare two distributions; which is \( P \) and which is \( Q \)?

It is also troubling for numerical work (and maybe also for theoretical applications) to have the divergence become infinite when there is a mismatch in supports.

Over the decades since its introduction, people have introduced variants that are symmetric and better behaved when two distributions have different supports.

A fix

If for some \( x \), \( P(x) > 0, Q(x) = 0 \) the KL divergence is infinite, here is one fix.

Lin (1991) proposed the Jensen-Shannon divergence:

\[
JS(P, Q) = \frac{1}{2} D \left( P \parallel \frac{P + Q}{2} \right) + \frac{1}{2} D \left( Q \parallel \frac{P + Q}{2} \right)
\]

What properties does this divergence have? As a practical matter, we get to reuse some code!
Putting it to work (i)

Let’s see if this measure behaves as we expect; let’s evaluate a number of different binomials and see how the divergence behaves.

We will have two ways to do this; one with a loop (our first!) and another with a variant of apply().

Putting it to work (ii)

Before we stop and talk about loops (and even if-statements, for that matter), let’s press on to an application.

At the risk of bringing back bad memories, let’s reconsider the Roberts confirmation data again and compute “distances” between the senators based on their word distributions.
Back to merging

OK, so Hatch used about 300 fewer unique words than Kennedy did so our word distributions can’t have the same support.

To register these we will want to combine the two data frames into one; as we saw at the start of the lecture, we can do this in one of two ways.
Finally, places you don’t expect to see functions

As we have seen, most of the computation we do in R is orchestrated through functions.

This is true even for operations that don’t immediately appear to be functions.

Some examples...
To be clear...

When we invoke an assignment like

```r
x <- dbinom(0:20, size=20, p=0.2)
x[1:5] <- 0
```

We are really implementing something like

```r
x = "[<-"

```

```
(x,1:5,value=0)
```

The same is true for some other expressions that don't feel very function-like on first blush like...

```r
y = c(2,3,4)
names(x) <- c("count","word")
x+3
```

Next time

We will consider a few of the finer points of functions; we'll at least mention what's technically meant by an environment (alluded to in the first couple slides today)

We will also examine how to connect R to a database (!)