Introduction to R

Kevin Johnson

09/10/2014

Getting Started

R is a high level language specifically designed for statistical calculations. You can get it for free at: http://www.cran.r-project.org/. In Unix or Linux, you start R by typing: R. In Windows, click on the R icon. You can now use R interactively, just start typing commands.

Basic Commands

We'll start off with some basic calculator operations in R.

```
10^2 + 36
```

```
## [1] 136
```

You can store values in a variable using the `<-` operator allowing you to access the values later. You can also use the `=` operator to do the same thing, but it's best practice to use `<-`.

```
a <- 4
a
```

```
## [1] 4
```

```
a * 4
```

```
## [1] 16
```

```
a <- a + 10
a
```

```
## [1] 14
```

R organizes numerical data into *scalars* (single numbers), *vectors* (1-dimensional array of numbers), and *matrices* (2-dimensional array of numbers, like a table). The `c()` function is used to create a vector.

```
b <- c(3,4,5)
```

If we want to compute the mean of all numbers stored in the above vector, we could do it manually:

```
(3+4+5)/3
```

```
## [1] 4
```

We will rarely deal with vectors this small, so R provides functions that take in arguments and give an output. We can use the `mean()` function to calculate the mean of a vector.

```r
mean(b)
```

```
## [1] 4
```

The function `rnorm()` is another example of a standard R function. It takes a number as an argument and outputs that number of random samples from a normal distribution. It defaults to a standard normal distribution with mean 0 and standard deviation 1, but you can specify your own if you want.

```r
rnorm(10)
```

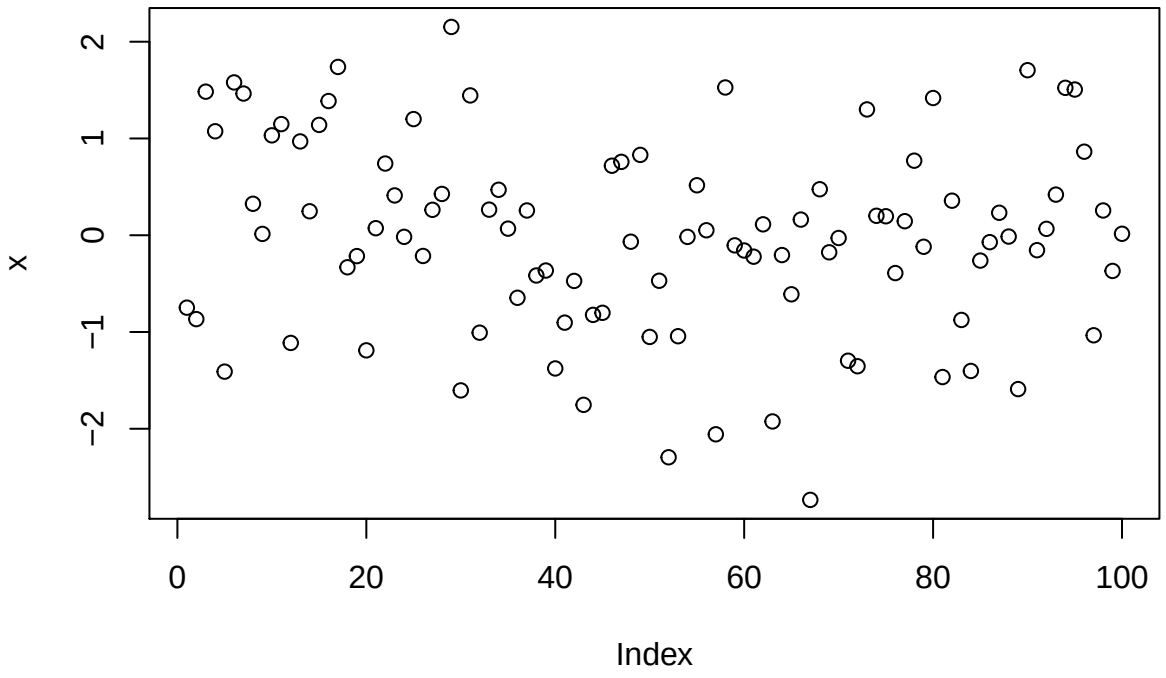
```
##  [1] -0.4438 -1.4914 -1.7560  0.2204  0.4366  0.7945  0.6143  2.1843
##  [9]  0.7242  0.6366
```

```r
rnorm(10, mean = 5, sd = 3)
```

```
##  [1]  3.864  6.830  4.643  5.332  4.874 10.151  4.115  9.854  8.553  6.179
```

R can also natively create plots. Here is a simple example.

```r
x <- rnorm(100)
plot(x)
```



The documentation for R is incredibly rich and will be your best resource when learning how to use R. You can access documentation for any function using the `help()` command or the `?` operator.

Data Structures

Vectors

We've looked at vectors already, but let's go deeper into what we can do with them. Vectors are the building blocks of most other data structures in R. They can be contain numerical, character, or logical data. First, let's create a simple vector.

```
vec1 <- c(1,4,6,8,10)
vec1
```

```
## [1]  1  4  6  8 10
```

What if we want to access the 5th element in the vector?

```
vec1[5]
```

```
## [1] 10
```

Let's assign a new value to the 3rd element in the vector:

```
vec1[3] <- 12
vec1
```

```
## [1]  1  4 12  8 10
```

You can use the `seq()` function to create a vector of numbers without having to type them all out.

```
vec2 <- seq(0,1,0.25)   ### Same as seq(from = 0, to = 1, by = 0.25)
vec2
```

```
## [1] 0.00 0.25 0.50 0.75 1.00
```

Many functions take vectors as inputs. Here are a few examples.

```
sum(vec1)
```

```
## [1] 35
```

```
max(vec1)
```

```
## [1] 12
```

```
min(vec1)
```

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

```r
median(vec1)
```

```
## [1] 8
```

```r
sd(vec1)
```

```
## [1] 4.472
```

You can also add, subtract, multiply, and divide two vectors.

```r
vec1 + vec2
```

```
## [1]  1.00  4.25 12.50  8.75 11.00
```

```r
vec1 - vec2
```

```
## [1]  1.00  3.75 11.50  7.25  9.00
```

```r
vec1 * vec2
```

```
## [1]  0  1  6  6 10
```

```r
vec1 / vec2
```

```
## [1]   Inf 16.00 24.00 10.67 10.00
```

Matrices

Matrices are just 2-dimensional vectors. You can define a matrix using the `matrix()` function. I won't spend much time on this because you won't be using matrices in practice very often unless you are manually implementing algorithms.

```r
mat <- matrix(data = c(9,2,3,4,5,6), ncol = 3)
mat
```

```
##      [,1] [,2] [,3]
## [1,]    9    3    5
## [2,]    2    4    6
```

There are many ways of defining a matrix (see `?matrix`). In this method, you specify the number of columns in the matrix and R will fill in the columns using the provided vector. Accessing and manipulating matrices is very similar to vectors.

```r
mat[1,2]
```

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

```
mat[2,]
```

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

```
mean(mat)
```

```
## [1] 4.833
```

Data Frames

Data frames are the bread and butter of R. In practice, you will spend the vast majority of your time dealing with data frames. Data frames are really nothing more than a matrix with column names, which is more useful than it sounds. You can create a data frame using the `data.frame()` function.

```
d <- data.frame(x = c(11,12,14), y = c(19,20,21), z = c(10,9,7))
d
```

```
##    x  y  z
## 1 11 19 10
## 2 12 20  9
## 3 14 21  7
```

The `$` operator allows you to access columns of a data frame by name. When you access a column you get a vector in return, and thus can perform any of the vector operations we have discussed so far.

```
d$x
```

```
## [1] 11 12 14
```

```
mean(d$x)
```

```
## [1] 12.33
```

```
d$x[1]
```

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

You can access several columns at once, either by name or by column number.

```
d[,c(1,2)]
```

```
##    x  y
## 1 11 19
## 2 12 20
## 3 14 21
```

```
d[,c("x","y")]
```

```
##    x  y
## 1 11 19
## 2 12 20
## 3 14 21
```

You can subset the rows of a data frame based on a set of criteria.

```
d[d$x >= 12,]
```

```
##    x  y z
## 2 12 20 9
## 3 14 21 7
```

```
d[d$y == 21,]
```

```
##    x  y z
## 3 14 21 7
```

```
d[d$x < 12 | d$z == 7,]
```

```
##    x  y  z
## 1 11 19 10
## 3 14 21  7
```

```
d[d$x == 12 & d$z == 7,]
```

```
## [1] x y z
## <0 rows> (or 0-length row.names)
```

The `%in%` operator checks to see if the variable is equal to any of the values in the provided vector.

```
d[d$x %in% c(12,14),]
```

```
##    x  y z
## 2 12 20 9
## 3 14 21 7
```

Here are some miscellaneous useful functions for data frames (most can be applied to other data types as well).

```
class(d)
```

```
## [1] "data.frame"
```

```r
nrow(d)     ### length() is for vectors
```

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

```r
names(d)
```

```
## [1] "x" "y" "z"
```

```r
str(d)
```

```
## 'data.frame':    3 obs. of  3 variables:
##  $ x: num  11 12 14
##  $ y: num  19 20 21
##  $ z: num  10 9 7
```

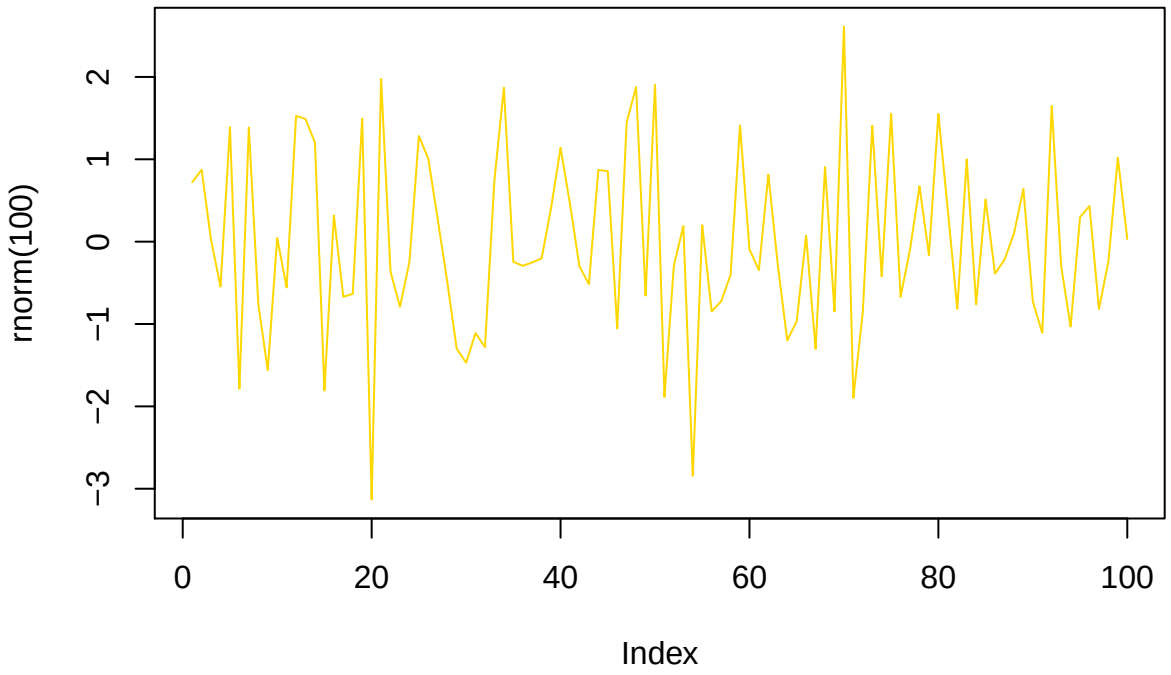
```r
summary(d)
```

```
##        x               y               z        
##  Min.   :11.0   Min.   :19.0   Min.   : 7.00  
##  1st Qu.:11.5   1st Qu.:19.5   1st Qu.: 8.00  
##  Median :12.0   Median :20.0   Median : 9.00  
##  Mean   :12.3   Mean   :20.0   Mean   : 8.67  
##  3rd Qu.:13.0   3rd Qu.:20.5   3rd Qu.: 9.50  
##  Max.   :14.0   Max.   :21.0   Max.   :10.00
```

Plotting

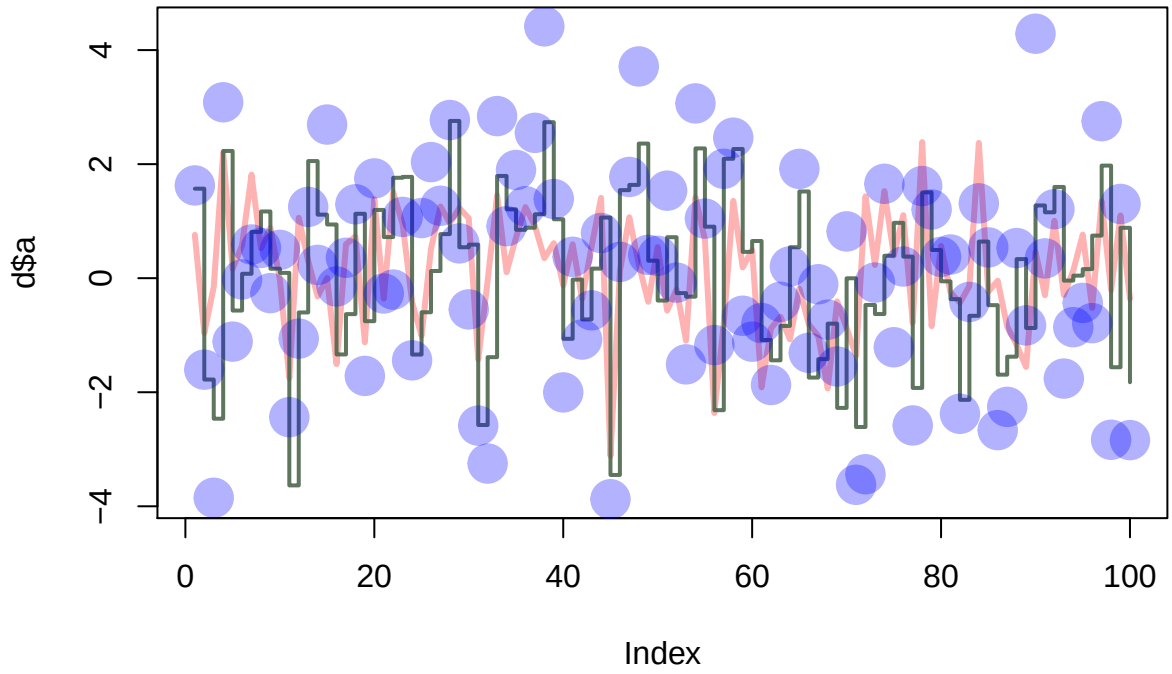
Plotting is an important part of analytics, and R comes with several plotting functions built in. I cannot provide a detailed overview of plotting in R here, but I can give a few examples to get you started. The documentation is very helpful for this. First, a simple line chart.

```r
plot(rnorm(100), type = "l", col = "gold")
```



Next, we'll create a data frame from randomly generated values and show how information can be added to a plot in layers. The `lines()` and `points()` functions add information to the existing plot from the `plot()` function.

```
x1 <- rnorm(100)
x2 <- rnorm(100)
x3 <- rnorm(100)
d <- data.frame(a = x1, b = x1 + x2, c = x1 + x2 + x3)
plot(d$a, type = "l", ylim = range(d), lwd = 3, col = rgb(1,0,0,0.3))
lines(d$b, type = "s", lwd = 2, col = rgb(0.3,0.4,0.3,0.9))
points(d$c, pch = 20, cex = 4, col = rgb(0,0,1,0.3))
```



Packages

R comes with many built-in functions, but the true power of R comes from the vast library of packages available. Use the `install.packages()` function to install packages, and the `library()` function to load them into R. I've compiled a list of some useful packages, but Google is your friend here. Chances are if you want to do it in R, somebody has already made a package for it.

Load Data

- RODBC, RMySQL, RPostgresSQL, RSQLite - used to connect R with a database.
- XLConnect, xlsx - read and write Excel files, though I recommend just saving them as csv files.
- foreign - read data files from programs like SAS and SPSS.

Manipulate Data

- dplyr - subsetting, summarizing, rearranging, and joining data sets, I use this package almost every day.
- reshape2 - change the layout of your data set (i.e. wide to long format).
- stringr - regular expressions and general string manipulation utilities.
- lubridate - dates and times, much better than the base R functions.

Data Visualization

- ggplot2 - the premiere package in R for data visualization, I use this almost every day.
- rgl - interactive 3D visualizations.
- googleVis - create interactive Google Charts in R.

Spatial Data

- sp, maptools, rgdal - tools for loading and using spatial data including shapefiles.
- maps - provides a set of common map polygons for plotting.
- ggmap - download street maps from Google or OpenStreetMap and plot with them.

Time Series or Financial Data

- zoo - the standard package for all things time series, it is the basis for almost every other time series package in R.
- quantmod - download financial data and conduct technical analysis.

High Performance

- Rcpp - call C++ from within R to speed up certain operations.
- data.table - an alternative to data frames that offers a significant performance boost.
- parallel, foreach - parallel processing in R.

Colors and Scales

- colorspace - provides an extensive framework for defining colors and color palettes.
- RColorBrewer - gives access to all palettes defined by the very populr Color Brewer website.
- scales - Lots of useful scale functions.

Example with Real Data

Greene and Shaffer (1992), cited in Fox (1997), analyzed decisions by the Canadian Federal Court of Appeals on cases filed by refugee applicants who had been turned down by the Immigration and Refugee Board. The resulting data is provided in `judges_and_immigration.txt`. We are interested in the fact that there are large differences among judges.

It is possible, of course, that the judges get to hear very different cases. In a later analysis we will control for an expert assessment of whether the case had merit, the city where the original application was filed (Toronto, Montreal or other) and the language in which it was filed (English, French). An additional predictor is the logit of the success rate for all cases from the applicant's country. The country itself is also available. For now, we're just going to explore the data.

First you need to set your working directory using `setwd()` so R knows where to look for files.

```
setwd("/home/kevin/Copy/MS Analytics/Intro to R/")
```

For this data we need to use the `read.table()` function, but there is also a handy function called `read.csv()` for csv files. If your data is in Excel format, I recommend re-saving it as a csv file, but there are packages available that will read Excel files directly. The `sep` argument tells R how columns are separated in your data. In this case the columns are separated by spaces, but typically you'll run into either commas or tabs. I always set the `stringsAsFactors` argument to be false since it can cause some odd behavior. The `header` argument tells R that variable names are included in the first line of the data file.

```
data <- read.table("judges_and_immigration.txt", sep = " ",
    stringsAsFactors = FALSE, header = TRUE)
```

Usually, the first thing you want to do with any dataset is use the `head()` function to look at the first few rows and the `summary()` function to look at some standard statistics. The `str()` function is also useful to look at the structure of the data frame.

```
head(data)
```

```
##   Index  JudgeName        Country GrantedAppeal Merit Language     City
## 1    13      Heald        Lebanon            no    no  English  Toronto
## 2    15      Heald      Sri_Lanka            no    no  English  Toronto
## 3    19      Heald    El_Salvador            no   yes  English  Toronto
## 4    30  MacGuigan Czechoslovakia            no   yes   French Montreal
## 5    36 Desjardins        Lebanon           yes   yes   French Montreal
## 6    42      Stone        Lebanon           yes   yes  English  Toronto
##   SuccessRate
## 1     -1.0986
## 2     -0.7538
## 3     -1.0460
## 4      0.4055
## 5     -1.0986
## 6     -1.0986
```

```
summary(data)
```

```
##      Index       JudgeName            Country          GrantedAppeal
##  Min.   :  13   Length:384         Length:384         Length:384
##  1st Qu.: 539   Class :character   Class :character   Class :character
##  Median :1247   Mode  :character   Mode  :character   Mode  :character
##  Mean   :1210
##  3rd Qu.:1831
##  Max.   :2461
##     Merit              Language             City            SuccessRate
##  Length:384         Length:384         Length:384         Min.   :-2.091
##  Class :character   Class :character   Class :character   1st Qu.:-1.099
##  Mode  :character   Mode  :character   Mode  :character   Median :-0.995
##                                                           Mean   :-1.020
##                                                           3rd Qu.:-0.754
##                                                           Max.   : 0.406
```

```
str(data)
```

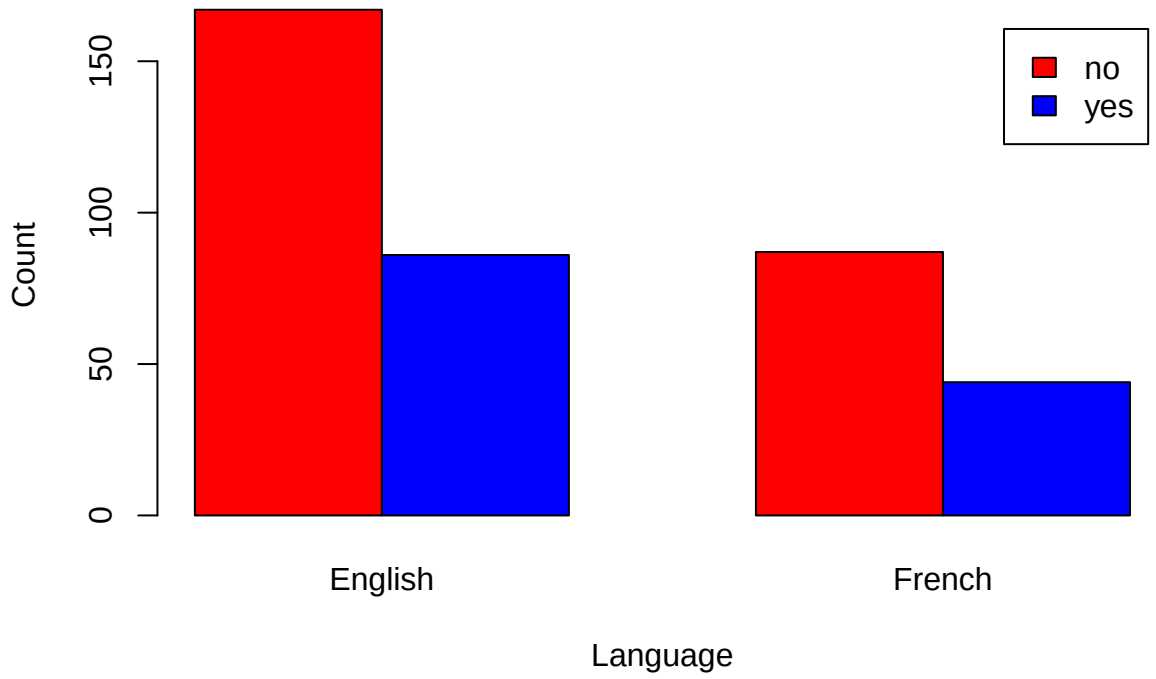
```
## 'data.frame':    384 obs. of  8 variables:
##  $ Index        : int  13 15 19 30 36 42 45 46 51 52 ...
##  $ JudgeName    : chr  "Heald" "Heald" "Heald" "MacGuigan" ...
##  $ Country      : chr  "Lebanon" "Sri_Lanka" "El_Salvador" "Czechoslovakia" ...
##  $ GrantedAppeal: chr  "no" "no" "no" "no" ...
##  $ Merit        : chr  "no" "no" "yes" "yes" ...
##  $ Language     : chr  "English" "English" "English" "French" ...
##  $ City         : chr  "Toronto" "Toronto" "Toronto" "Montreal" ...
##  $ SuccessRate  : num  -1.099 -0.754 -1.046 0.405 -1.099 ...
```

Now, let's examine the relationships between the number of granted appeals and several possible explanatory variables. The `table()` function can give us counts for categorical variables, allowing us to see how many applications were accepted and rejected for each language, city, judge, etc. We can plot the output of `table()` using the `barplot()` function.

```
counts <- table(data[,c("GrantedAppeal", "Language")])
counts
```

```
##              Language
## GrantedAppeal English French
##           no      167     87
##           yes      86     44
```

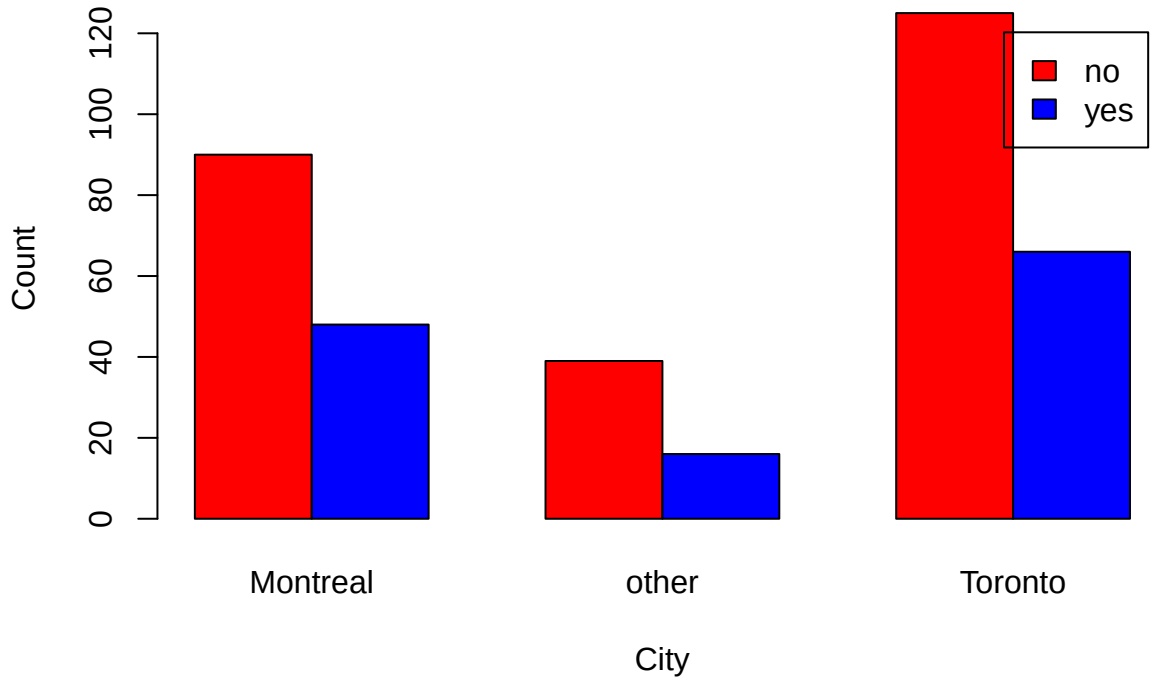
```
barplot(counts, beside = TRUE, xlab = "Language", ylab = "Count",
    legend = rownames(counts), col = c("red", "blue"))
```




```
counts <- table(data[,c("GrantedAppeal", "City")])
counts
```

```
##              City
## GrantedAppeal Montreal other Toronto
##           no         90    39     125
##           yes        48    16      66
```

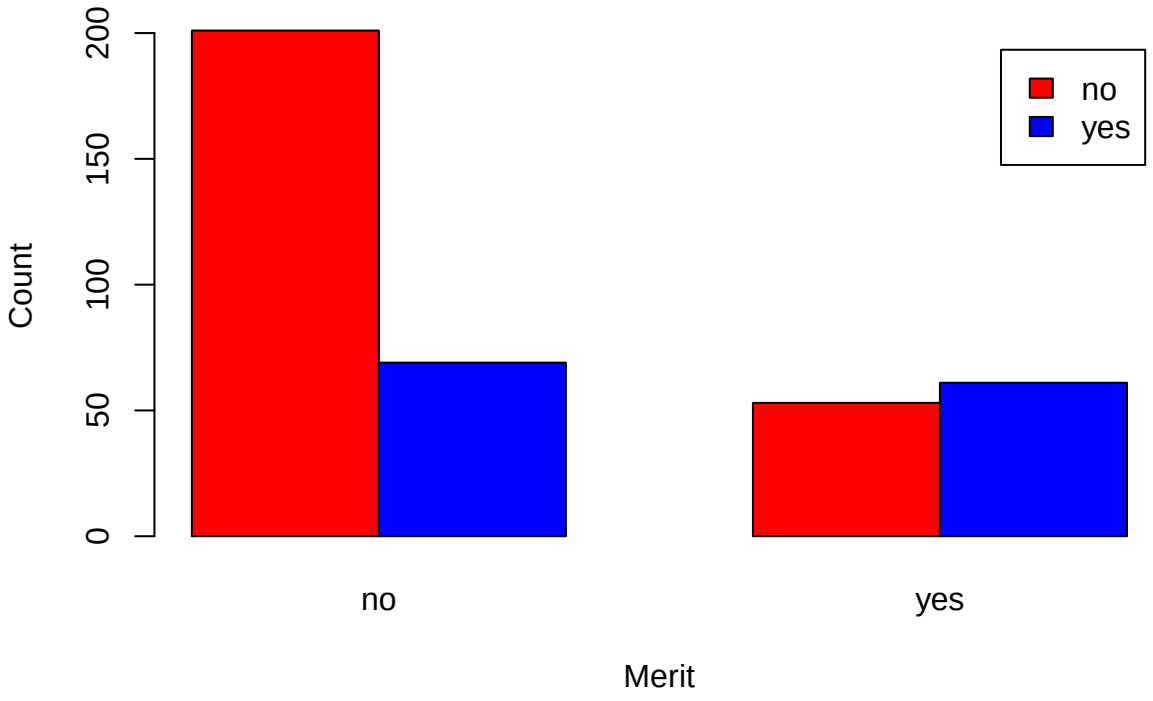
```
barplot(counts, beside = TRUE, xlab = "City", ylab = "Count",
    legend = rownames(counts), col = c("red", "blue"))
```



```
counts <- table(data[,c("GrantedAppeal", "Merit")])
counts
```

```
##              Merit
## GrantedAppeal  no yes
##           no  201  53
##           yes  69  61
```

```
barplot(counts, beside = TRUE, xlab = "Merit", ylab = "Count",
    legend = rownames(counts), col = c("red", "blue"))
```



The last plot shows that there is significant disagreement between the expert opinion on whether or not a case has merit and the ruling of the judge. In total, 122 out of 384 cases showed disagreement between the two variables. Let's see if we can find out which judges disagree with the experts most often.

```
counts <- table(data[,c("GrantedAppeal", "Merit", "JudgeName")])
counts
```

```
## , , JudgeName = Desjardins
##
##              Merit
## GrantedAppeal no yes
##           no  15  12
##           yes  7  12
##
## , , JudgeName = Heald
##
##              Merit
## GrantedAppeal no yes
##           no  20   4
```

```
##           yes  5   7
##
## , , JudgeName = Hugessen
##
##              Merit
## GrantedAppeal no yes
##           no  34   5
##           yes 16   7
##
## , , JudgeName = Iacobucci
##
##              Merit
## GrantedAppeal no yes
##           no  21   0
##           yes  5   3
##
## , , JudgeName = MacGuigan
##
##              Merit
## GrantedAppeal no yes
##           no  40   8
##           yes 13   9
##
## , , JudgeName = Mahoney
##
##              Merit
## GrantedAppeal no yes
##           no  12   4
##           yes  5   9
##
## , , JudgeName = Marceau
##
##              Merit
## GrantedAppeal no yes
##           no   9  11
##           yes  1   4
##
## , , JudgeName = Pratte
##
##              Merit
## GrantedAppeal no yes
##           no  25   3
##           yes 11   3
##
## , , JudgeName = Stone
##
##              Merit
## GrantedAppeal no yes
##           no  19   3
##           yes  6   5
##
## , , JudgeName = Urie
##
##              Merit
```

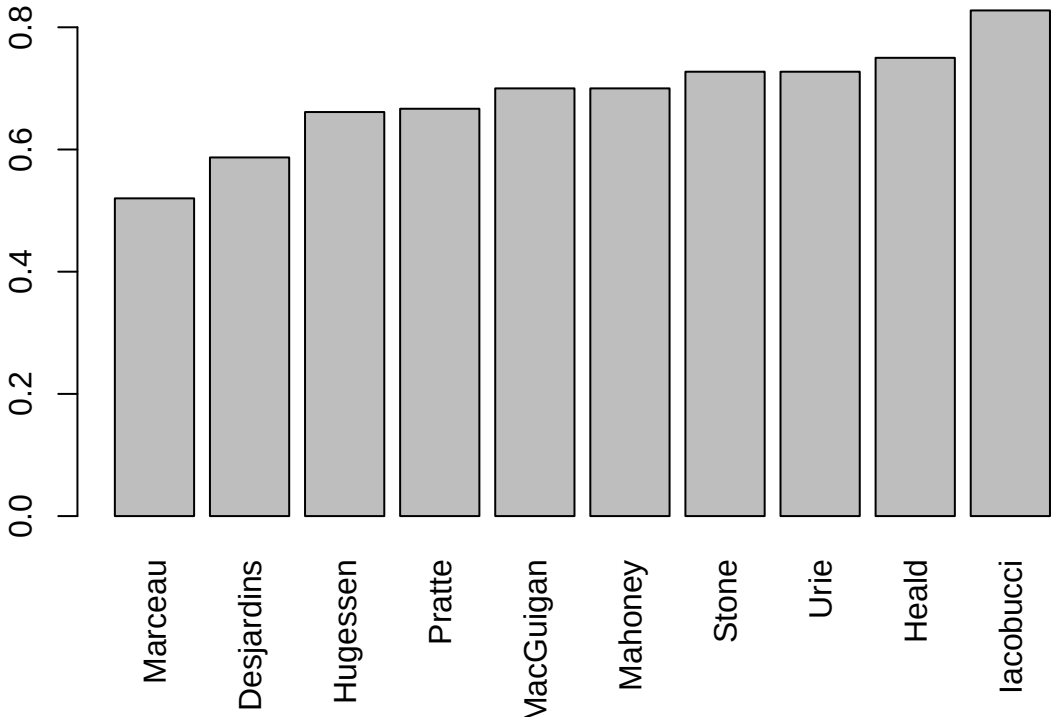
```
## GrantedAppeal no yes
##           no   6   3
##           yes  0   2
```

This is where a package like `dplyr` comes in handy. It provides an extensive framework for grouping and rearranging data. I'll give a short example here, but the capabilities of this package go far beyond what I'm about to show.

```
library(dplyr)
data$agreed <- ifelse(data$Merit == data$GrantedAppeal, 1, 0)
judgeAccuracy <- data %>%
    group_by(JudgeName) %>%
    summarize(correct = sum(agreed),
        total = length(agreed),
        accuracy = correct/total) %>%
    arrange(accuracy)
judgeAccuracy
```

```
## Source: local data frame [10 x 4]
##
##     JudgeName correct total accuracy
## 1     Marceau      13    25   0.5200
## 2  Desjardins      27    46   0.5870
## 3    Hugessen      41    62   0.6613
## 4      Pratte      28    42   0.6667
## 5   MacGuigan      49    70   0.7000
## 6     Mahoney      21    30   0.7000
## 7       Stone      24    33   0.7273
## 8        Urie       8    11   0.7273
## 9       Heald      27    36   0.7500
## 10  Iacobucci      24    29   0.8276
```

```
par(las = 3)
barplot(height = judgeAccuracy$accuracy, names.arg = judgeAccuracy$JudgeName)
```



Much better, we can now clearly see that Judge Marceau has the lowest agreement with the experts and Judge Iacobucci has the highest agreement with the experts. We can test the significance of the difference using the `prop.test()` function. The test fails to reject the null hypothesis that the two proportions are equal, so we conclude that Judge Marceau has significantly less accuracy than Judge Iacobucci.

```
prop.test(x = judgeAccuracy$correct[c(1,10)], n = judgeAccuracy$total[c(1,10)])
```

```
##
##  2-sample test for equality of proportions with continuity
##  correction
##
## data:  judgeAccuracy$correct[c(1, 10)] out of judgeAccuracy$total[c(1, 10)]
## X-squared = 4.549, df = 1, p-value = 0.03294
## alternative hypothesis: two.sided
## 95 percent confidence interval:
##  -0.58411 -0.03107
## sample estimates:
## prop 1 prop 2
## 0.5200 0.8276
```