Lecture 1: Introduction

Thomas Chadefaux

PO7001: Quantitative Methods I

Learning Outcomes

- Understand data concepts and basic descriptive quantitative analysis tools
- Work with real datasets to perform basic quantitative analyses
- Graph data effectively for presentation and analysis
- Recognize and understand the basics of the linear regression model
- Use the R statistical software package for analyzing and graphing data
- Understand sufficient theoretical and practical material to build on in a second, more advanced quantitative methods course

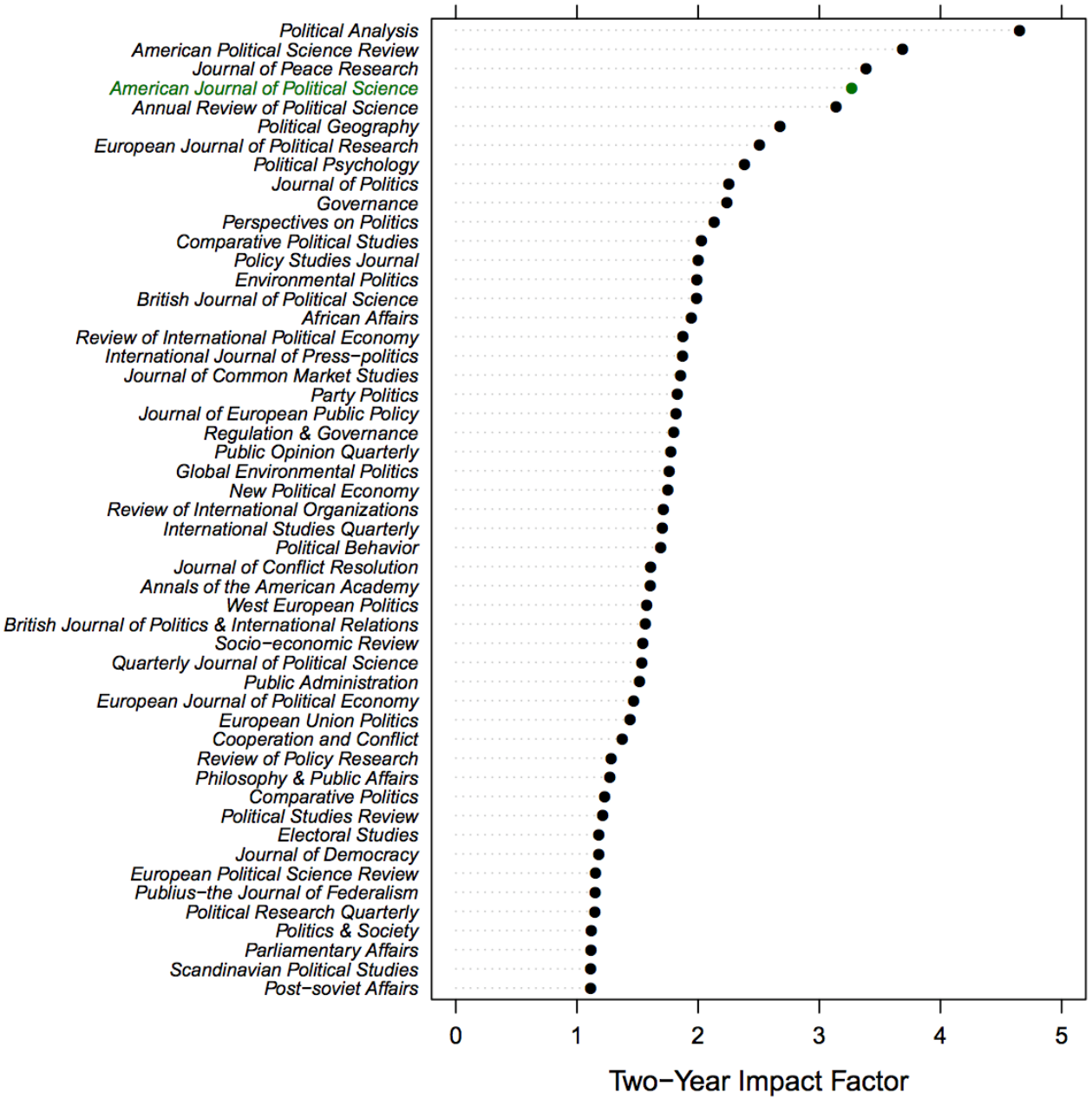
Why should I care?

Table 5b.

Distribution of Papers Accepted by Approach(%)

	APPROACH						
YEAR	FORMAL	QUANTITATIVE	FORMAL AND QUANTITATIVE	SMALL-N	INTERPRETIVE/ CONCEPTUAL	QUALITATIVE AND/OR EMPIRICAL	OTHER
2012–13	8.5	54	4	0	27.5	6	0
2011–12	12	48	14	2	19	5	0
2010–11	11	65	8	0	16	NA	0

Why should I care?



Software

- R. VERY powerful and free!
- Get it from `http://www.r-project.org`

Why R?

- Powerful
- Versatile
- Gives you a lot more freedom

Figure 1c. Analytics job trends for R and SPSS. Note that the legend labels at the top of the graph are truncated due to the very long size of the query.

Grading

- Problem Sets: 50%
  - Handed out Tuesday, due back next Tues. before class
  - 5 problem sets, each worth 10%
  - Submit as a single pdf on turnitin (see syllabus)
  - Scan if needed
- Course paper: 50%. See syllabus

Texts for this course

- Verzani, John. 2005. Using R for introductory statistics. Boca Raton, FL: Chapman & Hall/CRC.
- Moore, David S., George P. McCabe and Bruce A. Craig. 2012. Introduction to the practice of statistics. 7th international edition ed. New York: W.H. Freeman.

R Crash Course: Numbers

Add numbers

```
1 + 2
## [1] 3
```

Assign numbers to a variable

```
x <- 1 + 2
```

Note that x was not printed. To see the contents of x, just type:

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

How would you store the result of $x \times 3$ in a variable called *myprecious*?

```
myprecious <- x*3
```

Vectors

What if we want to create a larger collection of numbers? Use the c() function:

```
z <- c(1, 2, 3, 5, 6)
```

Or, for a sequence of integers, use ':'

```
z <- 1:6
```

Now let's add the number 7 to that vector, without retyping the whole sequence

```
z <- c(z, 7)
```

Notice that I have now overwritten z:

```
z
## [1] 1 2 3 4 5 6 7
```

Functions

You can conduct all kinds of operations on z:

```
z*2
## [1]  2  4  6  8 10 12 14
(z+100)/6
## [1] 16.83333 17.00000 17.16667 17.33333 17.50000 17.66667 17.83333
```

Take the square root of $(z + 1)$, and then log it:

```
z2 <- sqrt(z + 1)
z3 <- log(z2)
z3
## [1] 0.3465736 0.5493061 0.6931472 0.8047190 0.8958797 0.9729551 1.039720
```

Or you can do it in one go:

```
z3 <- log(sqrt(c(1,2,3,4,5,6,7)*2))
z3
## [1] 0.3465736 0.6931472 0.8958797 1.0397208 1.1512925 1.2424533 1.319528
```

Recycling

What happens if you add c(1,2,3,4) to c(1,2)?

```
c(1,2,3,4) + c(1,2)
## [1] 2 4 4 6
```

The shorter vector gets 'recycled'. However, this yields an error:

```
c(1,2,3,4) + c(1,2,3)
## Warning in c(1, 2, 3, 4) + c(1, 2, 3): longer object length is not a
## multiple of shorter object length
## [1] 2 4 6 5
```

Creating data

The workhorse of data analysis in R is the data frame. To create a data frame, for example:

```
localdat <- data.frame(ID = c(1,2),
                       gender = c('male', 'female'),
                       income = c(50000, 60000))
```

You can put complex statements inside that data.frame:

```
localdat <- data.frame(age = round(runif(4,0,100)),
                       income=round(runif(4, 0, 100000)),
                       gender=c('male', 'female'))
```

Importing and exporting data

Instead of creating them, you typically import data frames from your local file system or from the web. head() lets you take a look at the first few rows.

```
localdat <- read.csv('mydata.csv')
head(localdat)
##   ID age income gender
## 1  1   4  49672   male
## 2  2  56  20300 female
## 3  3  56  73388   male
## 4  4  99  33374 female
## 5  5  22  84749   male
## 6  6  22  73580 female
```

Export it to your local file system:

```
write.csv(localdat, file = 'newfile.csv')
```

Extracting data

Localdat is a data frame. Loosely, a table with data. To access its information, we need to ask R for a row and column number, in that order. For example, to ask for row 1 and column 2, we would write:

```
localdat[1, 2]
## [1] 4
```

Or perhaps we want to see all columns associated with row 1, in which case we leave the column indicator empty, and similarly it we want all rows associated with a column:

```
localdat[1, ]
##   ID age income gender
## 1  1   4  49672   male
localdat[, 2]
## [1]  4 56 56 99 22 22 29 14 91
```

Extracting data (Cont'd)

We can also ask for a specific variable by name in three ways, though the first one is the most common

```
localdat$age
## [1]  4 56 56 99 22 22 29 14 91
localdat[, 'age']
## [1]  4 56 56 99 22 22 29 14 91
with(localdat, age)
## [1]  4 56 56 99 22 22 29 14 91
```

Extracting data (Cont'd)

We might have more specific requests. E.g., we want to see all males younger than 50 with income of less than 20000. In this case there is only one, with ID 7:

```
localdat[localdat$age < 50
         & localdat$gender =='male'
         & localdat$income < 20000, ]
##   ID age income gender
## 7  7  29  13124   male
```

How about

Summarizing data

What is the mean income?

```
mean(localdat$income)
## [1] 45678.89
```

What is the maximum age?

```
max(localdat$age)
## [1] 99
```

Also useful:

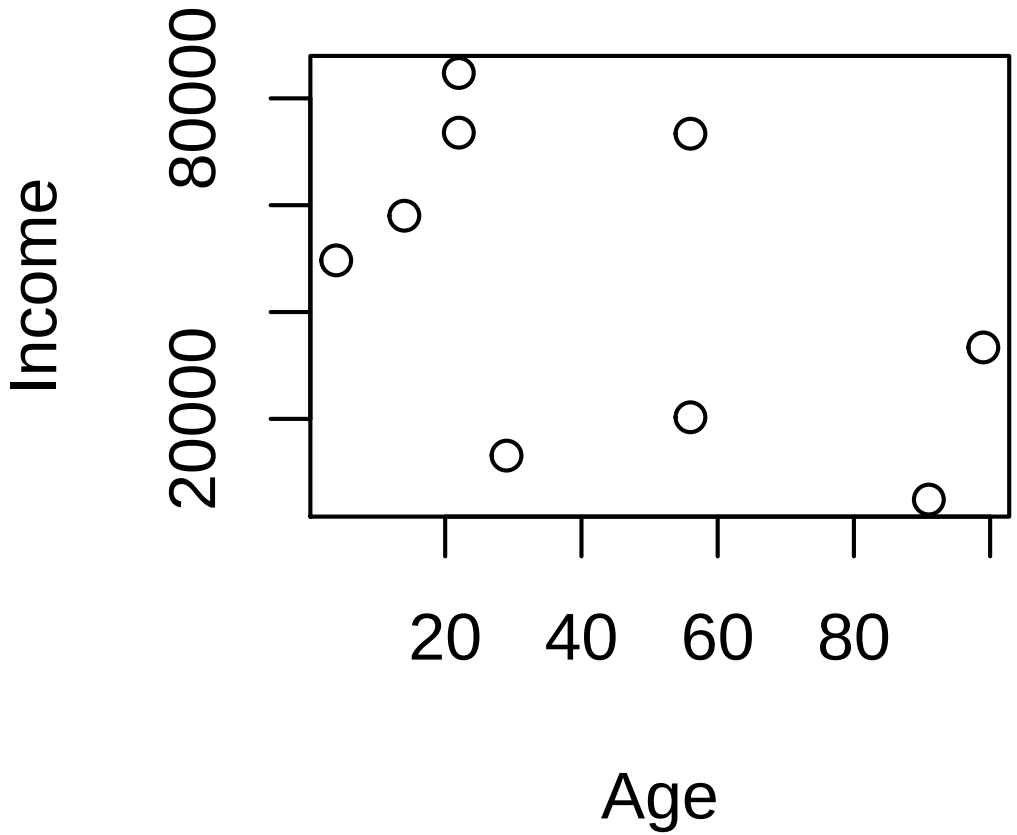
```
summary(localdat$income)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    4893   20300   49670   45680   73390   84750
```

TODO: What is the average income of women under the age of 50?
mean(localdat$income[localdat$gender == 'female' & localdat$age < 50])

XY plots

To plot x, y:

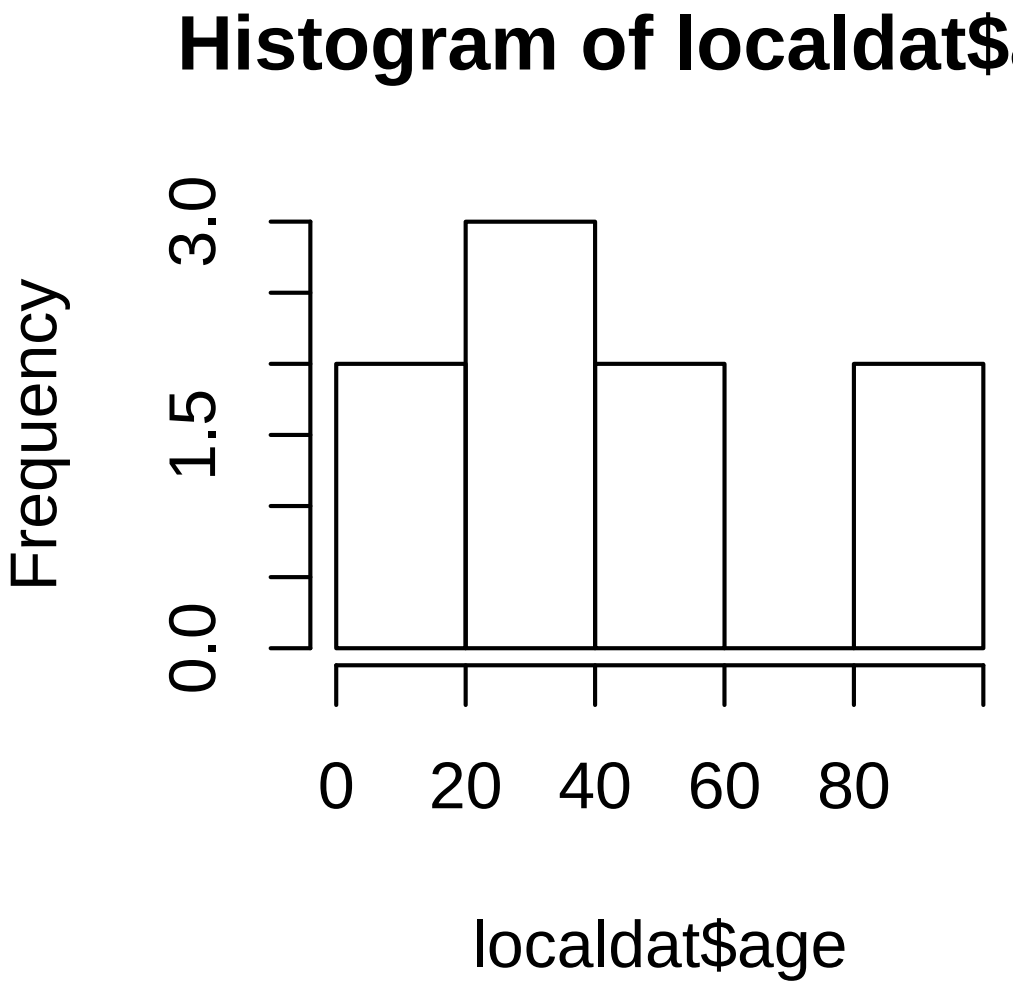
```
x <- localdat$age
y <- localdat$income
plot(x, y, xlab='Age', ylab='Income')
```



Histogram

To plot x, y:

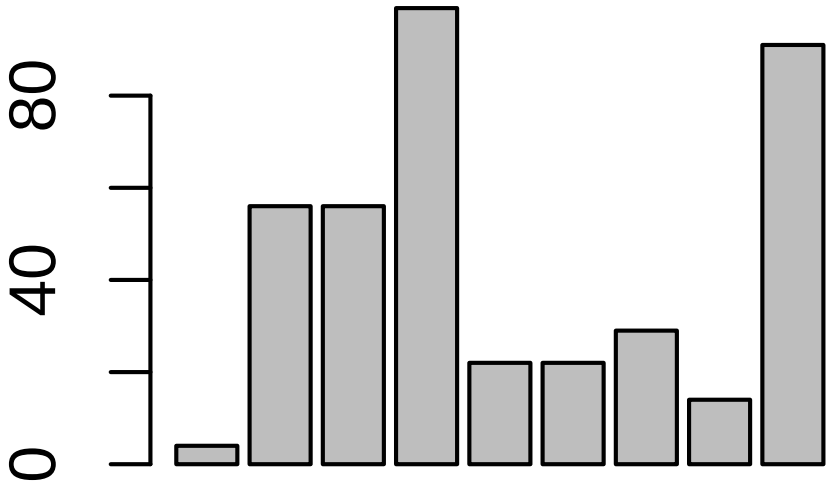
```
hist(localdat$age)
```



Barplot

To plot x, y:

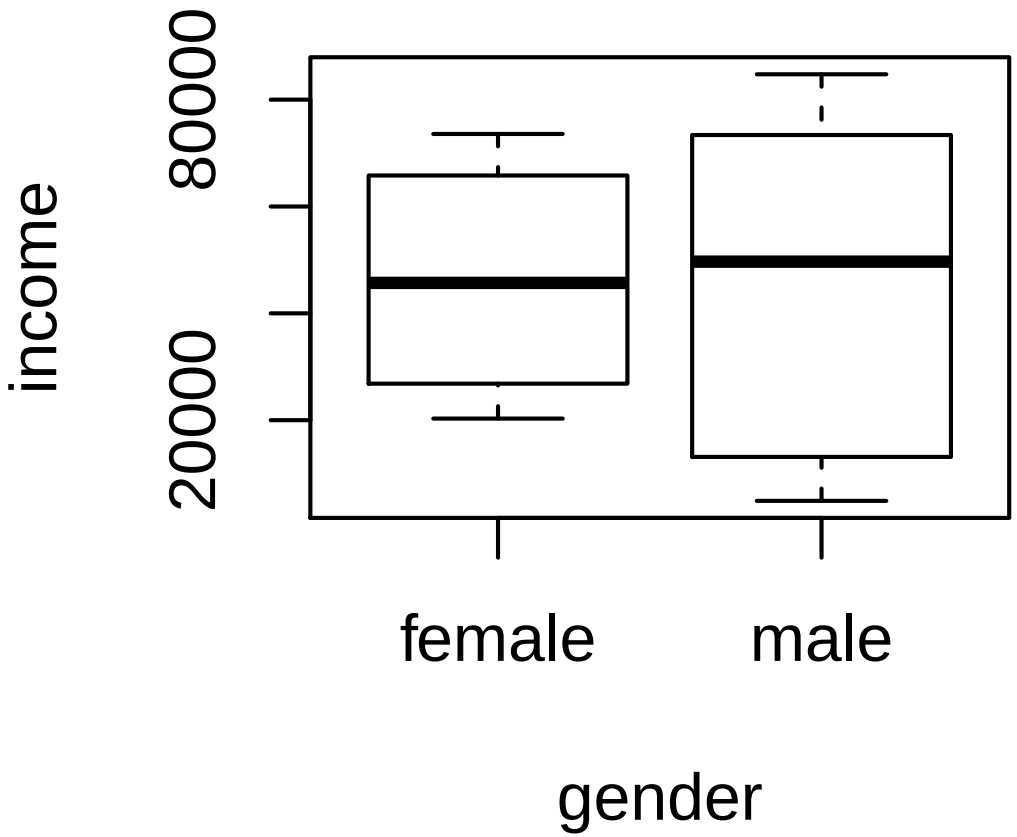
```
barplot(localdat$age)
```



Boxplot

Plot income by gender:

```
with(localdat, plot(income ~ gender))
```



Creating your own Functions

Often you will want to reuse the same routine. It is then useful to create your own function. For example, you may want to 1. generate a sequence of 100 random numbers; 2. calculate its mean; 3. find its maximum

```
myfunction <- function(){
      x <- rnorm(100)
      print(mean(x))
      print(max(x))
      }
myfunction()
## [1] 0.133356
## [1] 3.168311
```

Note that you need 'print', because R does not return the results of calculations done within a function or loop

Using arguments in your functions

Often you will want to pass an argument to your function. For example, instead of generating a random sequence of numbers, you want to ask your function to calculate the mean and max of a given sequence of numbers

```
myfunction <- function(x){
      print(mean(x))
      print(max(x))
      }
myfunction(1:10)
## [1] 5.5
## [1] 10
```

Note that you need 'print', because R does not return the results of calculations done within a function or loop

Ready for the Monte Hall Problem?

Let's see how we can use what we learned to do pretty much anything. For example:

The Monte Hall problem: You have a choice among 3 doors. Behind a random door is a car; behind the other two are goats. You choose one at random. Monte peeks behind the other two doors and opens the one (or one of the two) with the goat and asks if you'd like to switch your door for the other door that hasn't been opened yet. Should you switch?

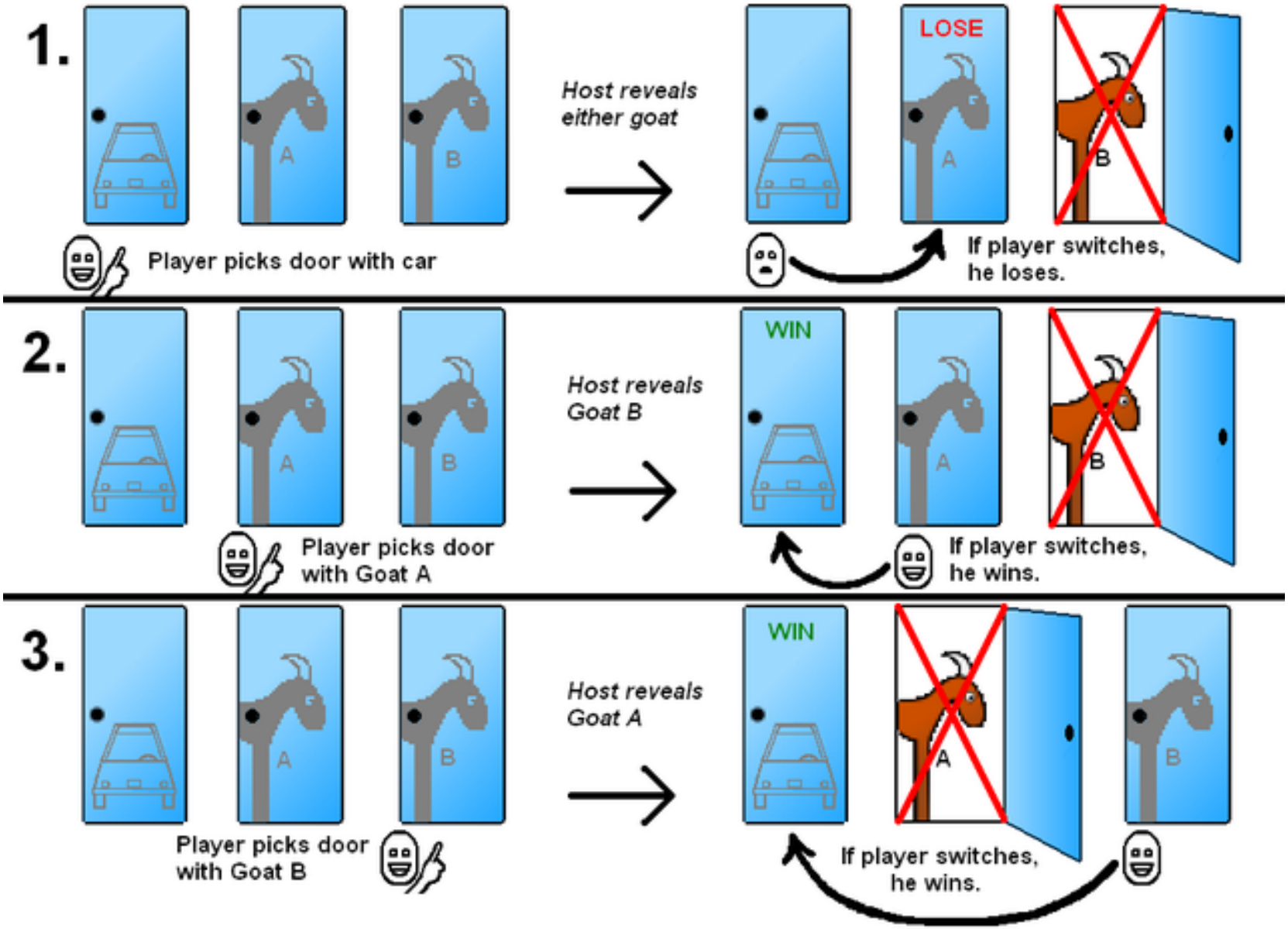
Suppose first that I don't switch:

```
doors <- c(1, 2, 3)
didIWinThisTime <- function(){
     Iwin = 0 # Reset Iwin
     choice <- sample(doors, 1)
     WinDoor <- sample(doors, 1)
     if(WinDoor == choice){Iwin = 1}
     return(Iwin)
     }

historyOfWins <- NULL #Initialize the variable
for(i in 1:100){
     historyOfWins <- c(historyOfWins, didIWinThisTime())
     }
historyOfWins
##   [1] 0 0 1 1 1 1 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 0
##  [36] 1 0 0 0 0 1 0 1 0 1 1 0 1 0 1 1 0 1 0 1 1 0 0 0 0 1 0 1 0 0 0 0 0
##  [71] 0 1 0 0 1 1 0 0 0 0 1 0 1 1 0 0 0 1 1 0 1 1 1 0 1 0 1 0 0 0
mean(historyOfWins)
## [1] 0.36
```

What if I switch?

```
doors <- c(1, 2, 3)
didIWinThisTime <- function(){
     Iwin = 0 # Reset Iwin
     choice <- sample(doors, 1)
     WinDoor <- sample(doors, 1)
     if(WinDoor != choice){Iwin = 1}
     return(Iwin)
     }

historyOfWins <- NULL #Initialize the variable
for(i in 1:100){
     historyOfWins <- c(historyOfWins, didIWinThisTime())
     }
historyOfWins
##   [1] 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0 1 1 1 0
##  [36] 1 1 1 1 0 1 1 1 1 0 1 1 0 0 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 0 1 1 0
##  [71] 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 0 0 1 1 1 1 1 0
mean(historyOfWins)
## [1] 0.77
```

(Intuition for the curious or incredulous)



Getting help

Type ?function. For example:

```
?c
```