Intro to R

Gibran Hemani

Basic calculations

What is the probability of winning the lottery? Assume 49 balls, and 6 balls chosen without replacement. This is how many unique combinations there are.

$$\frac{49!}{6!(49-6)!} = \frac{49 \times 48 \times 47 \times 46 \times 45 \times 44}{6 \times 5 \times 4 \times 3 \times 2 \times 1}$$

In R it can be calculated like this:

```
49 * 48 * 47 * 46 * 45 * 44 / (6 * 5 * 4 * 3 * 2 * 1)
```

```
## [1] 13983816
```

$$49 \times 48 \times 47 \times 46 \times 45 \times 44/(6 \times 5 \times 4 \times 3 \times 2 \times 1)$$

Or we can use built in functions:

```
factorial(49) / (factorial(6) * factorial(49-6))
```

```
## [1] 13983816
```

We can find out about the `factorial` function like this:

```
?factorial
```

We want to store the result from our calculation. This is done like so:

```
lottery <- factorial(49) / (factorial(6) * factorial(49-6))
```

What happens now if you type

```
lottery
```

```
## [1] 13983816
```

This retrieves the stored value.

What is the probability of winning the lottery if you buy 1 ticket?

```
1 / lottery
```

```
## [1] 7.151124e-08
```

How about 10 tickets?

```r
10 / lottery
```

```
## [1] 7.151124e-07
```

What is the chance of winning the lottery twice?

```r
(1 / lottery)^2
```

```
## [1] 5.113857e-15
```

In R we can make vectors of numbers instead of dealing in single elements. For example

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

Here the `c()` command tells R to string the numbers 1 to 10 into an array. Another way to do this is:

```r
n <- 1:10
```

Now we can calculate the chances of winning for 1, 2, 3, . . . , 10 tickets in one command:

```r
n / lottery
```

```
##  [1] 7.151124e-08 1.430225e-07 2.145337e-07 2.860450e-07 3.575562e-07
##  [6] 4.290674e-07 5.005787e-07 5.720899e-07 6.436011e-07 7.151124e-07
```

This prints out 10 values, each one is for a different value in `n`.

Notice that we can overwrite a value in a variable, for example

```r
n <- 30:40
```

Now type `n` - it has changed because R has replaced the original value with the new value.

It's possible to only extract a single value from an array, using square brackets:

```r
n[1]
```

```
## [1] 30
```

```r
n[5:8]
```

```
## [1] 34 35 36 37
```

will extract only the first value of `n` for the first command; or the 5th, 6th, 7th and 8th values in the second command.

How many values are in `n`?

```r
length(n)
```

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

What is the sum of all the values in `n`?

```r
sum(n)
```

```
## [1] 385
```

What is the median value of `n`?

```r
median(n)
```

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

If you type `ls()` you can see your **workspace**. This is the list of all the objects that you have created. Notice that there is one object there called `lottery`, and one called `n`. We can remove objects like this:

```r
rm(lottery)
```

Type `ls()` again. Type `lottery`, what happens?

Data

R comes pre-loaded with some example datasets, one of which we will use here as an example of some basic data manipulation. We will be using the US States Facts and Figures dataset, which is stored as the `state.x77` R object. There is a help file available with background information on this dataset.

The dataset itself is quite large: typing `state.x77` into the R console to look at it results in the output running off the screen.

Instead we can use the `head` function in R to look at the first few rows of the dataset.

```r
head(state.x77)
```

```
##            Population Income Illiteracy Life Exp Murder HS Grad Frost
## Alabama          3615   3624        2.1    69.05   15.1    41.3    20
## Alaska            365   6315        1.5    69.31   11.3    66.7   152
## Arizona          2212   4530        1.8    70.55    7.8    58.1    15
## Arkansas         2110   3378        1.9    70.66   10.1    39.9    65
## California      21198   5114        1.1    71.71   10.3    62.6    20
## Colorado         2541   4884        0.7    72.06    6.8    63.9   166
##              Area
## Alabama     50708
## Alaska     566432
## Arizona    113417
## Arkansas    51945
## California 156361
## Colorado   103766
```

or the `tail` function to see the last few rows

```r
tail(state.x77)
```

```
##                Population Income Illiteracy Life Exp Murder HS Grad Frost
## Vermont               472   3907        0.6    71.64    5.5    57.1   168
## Virginia             4981   4701        1.4    70.08    9.5    47.8    85
## Washington           3559   4864        0.6    71.72    4.3    63.5    32
## West Virginia        1799   3617        1.4    69.48    6.7    41.6   100
## Wisconsin            4589   4468        0.7    72.48    3.0    54.5   149
## Wyoming               376   4566        0.6    70.29    6.9    62.9   173
##                 Area
## Vermont         9267
## Virginia       39780
## Washington     66570
## West Virginia  24070
## Wisconsin      54464
## Wyoming        97203
```

Use the `dim` function to see how many rows and columns it has.

```r
dim(state.x77)
```

```
## [1] 50  8
```

Type and run the following portion of R code

```r
Alaska_Life_Exp <- state.x77[2, 4]
ffrc <- state.x77[1:4, 1:4]
Population <- state.x77[ , 1]
```

This portion of R code uses square brackets to extract data from the `state.x77` R object. Being a table (or matrix) the entries of `state.x77` are indexed by two indices that refer to the row and column. So `state.x77[2, 4]` gives the entry in the second row and fourth column (the Alaskan life expectancy, 69.05 years). Also `state.x77[1:4, 1:4]` gives the first four rows and columns of the table. Finally, `state.x77[ , 1]` gives the first column (the population of all the states). Note that the first row displayed in the R console gives the column headings and the first column displayed in the R console gives the row headings.

Can you use this data to calculate:

1. The total area of the US?
2. The total population US?
3. The average illiteracy US?

Plotting

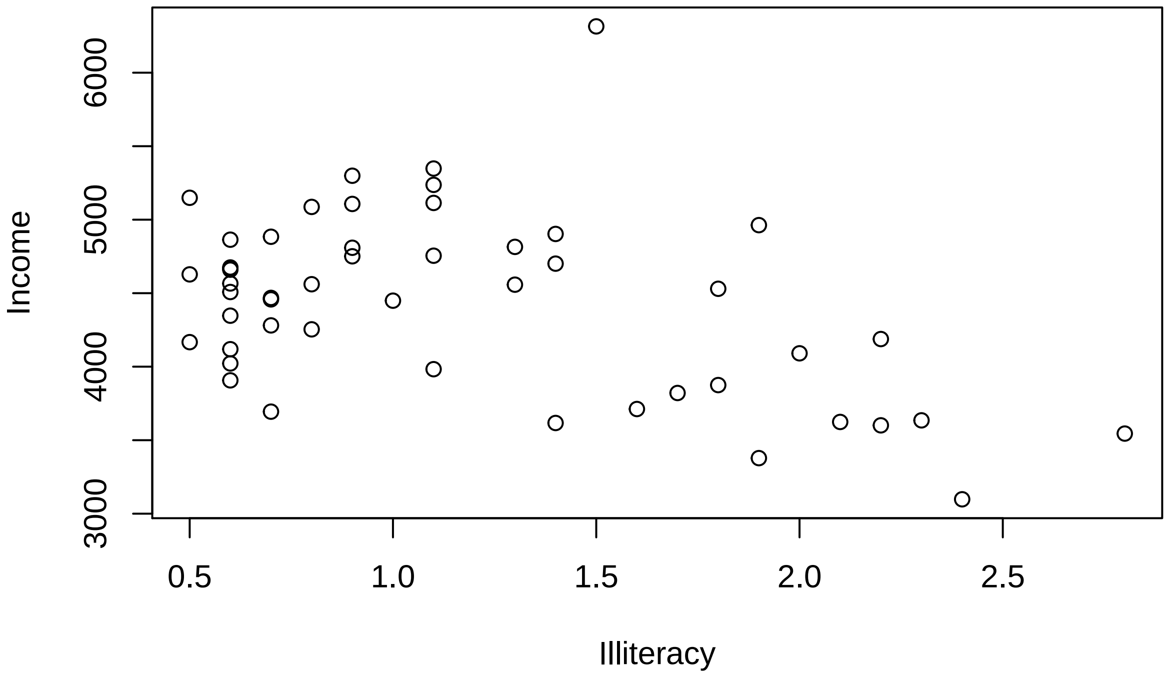
What is the relationship between income and literacy amongst the US states? What is the correlation?

```r
cor(state.x77[, "Income"], state.x77[, "Illiteracy"])
```

```
## [1] -0.4370752
```

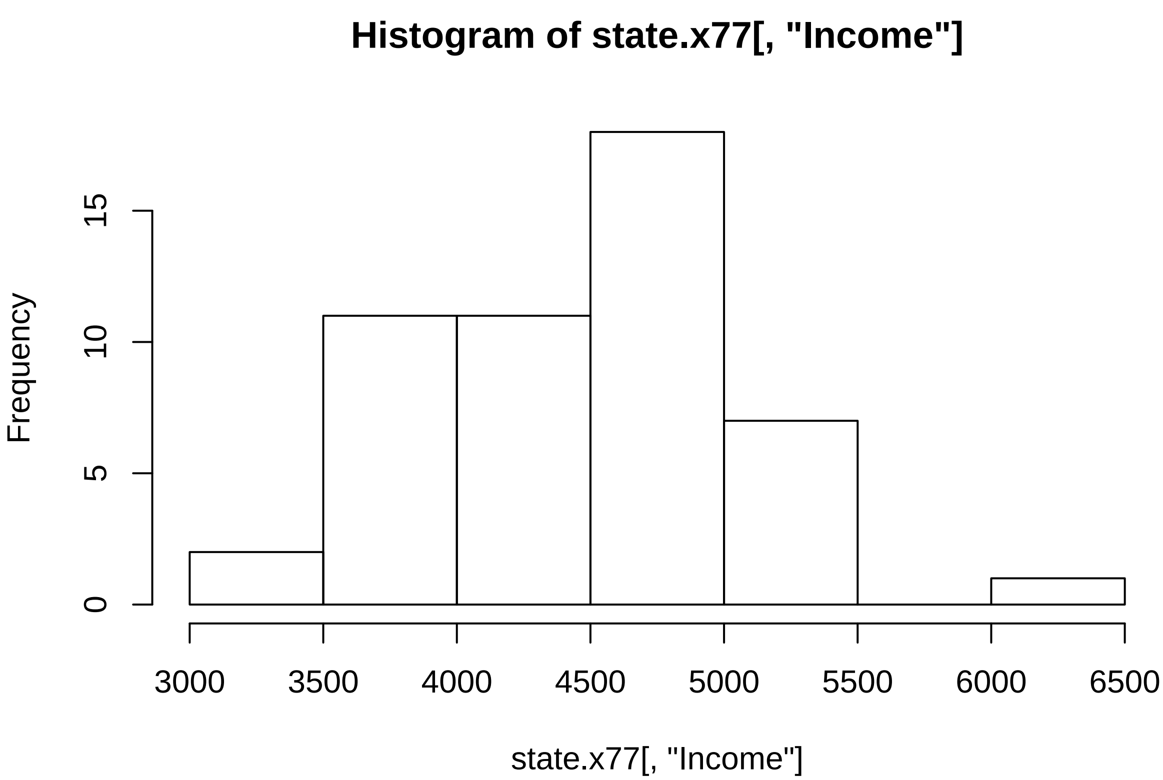
There seems to be an inverse proportional relationship. We can visualise this:

```r
plot(Income ~ Illiteracy, state.x77)
```



What is the distribution of Income?

```r
hist(state.x77[, "Income"])
```



You can save a plot like this:

```r
pdf("test.pdf")
hist(state.x77[, "Income"])
dev.off()
```

Reading and writing data

It is important to be able to get data into R, and back out again. Here we will look at two examples - Excel files, and Stata files.

Excel

In Excel it is possible to save spreadsheet data as `.csv` files - "comma separated values". R can read `.csv` files using the `read.csv()` function. Have a look at the documentation:

```r
?read.csv
```

Notice that there are a lot of options here to be as flexible as possible for reading in data that has been formatted in different ways. The default options for `read.csv` are usually suitable for reading in a file that has just been experted from Excel.

Let's try reading in a csv file…

```r
phen <- read.csv("../data/example_data/phen.csv", stringsAsFactors=FALSE)
```

What does this data look like?

```r
head(phen)
```

```
##   X IID      BMI       DBP      SBP       CRP HT
## 1 1 id1 31.86647  87.23318 152.2902 1.4446800  2
## 2 2 id2 35.42533  86.05953 143.0827 3.7545641  2
## 3 3 id3 29.77809  98.74657 234.1238 0.5148577  2
## 4 4 id4 34.79715 102.91224 198.5010 4.9365146  2
## 5 5 id5 26.92786  77.81723 151.0387 3.3885281  2
## 6 6 id6 27.57659  67.16845 114.8678 1.5661815  1
```

What are the dimensions?

```r
dim(phen)
```

```
## [1] 8237    7
```

Note, the `phen` object and the `state.x77` object are actually different **data types**. Look:

```r
class(phen)
```

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

```r
class(state.x77)
```

```
## [1] "matrix"
```

The difference between a **data.frame** and a **matrix** is that in a matrix every element must be the same type of data. In the examlpe of `state.x77`, every element is a numeric value. A **data.frame** on the other hand allows each column to be a different type of data. You can access a particular column using the `$` operator like this:

```r
class(phen$IID)
```

```
## [1] "character"
```

```r
class(phen$BMI)
```

```
## [1] "numeric"
```

Let's calculate the mean value of DBP (diastolic blood pressure):

```r
mean(phen$DBP)
```

```
## [1] 83.10646
```

```r
mean(phen[,"DBP"])
```

```
## [1] 83.10646
```

```r
mean(phen[,4])
```

```
## [1] 83.10646
```

```r
mean(phen[["DBP"]])
```

```
## [1] 83.10646
```

```r
mean(phen[[4]])
```

```
## [1] 83.10646
```

All the above are different ways of accessing the same thing - data frames are quite methods of storing data.

Can you use this data to:

1. Draw a histogram of BMI values?
2. Plot the relationship between DBP and SBP?

Stata

We can also read in files that are in Stata format. But first we need to install a library that will provide the necessary functions.

```
install.packages("readstata13")
```

Once the library is installed we can load it

```
library(readstata13)
```

And now we can use the functions that are provided by this package. Let's read in a Stata file:

```
phen <- read.dta13("../data/example_data/phen.dta")
```

Monty Hall problem

Suppose you're on a game show, and you're given the choice of three doors: Behind one door is a car; behind the others, goats. You pick a door, say No. 1, and the host, who knows what's behind the doors, opens another door, say No. 3, which has a goat. He then says to you, "Do you want to pick door No. 2?" Is it to your advantage to switch your choice?

Let's simulate this scenario to check!

We will make a function which simulates one game:

```
monty <- function()
{
    doors <- 1:3 # initialize the doors behind one of which is a good prize
    win <- 0 # to keep track of number of wins

    prize <- sample(1:3, 1) # randomize which door has the good prize
    guess <- sample(1:3, 1) # guess a door at random

    ## Reveal one of the doors you didn't pick which has a goat
    if(prize != guess) {
        reveal <- doors[-c(prize,guess)]
    } else {
        reveal <- sample(doors[-c(prize,guess)], 1)
    }

    ## Stay with your initial guess or switch
    switch_guess <- doors[-c(reveal,guess)]
    stay_guess <- guess

    ## Did you win?
    win <- ifelse(switch_guess == prize, "switch", "stay")

    ## return results
    result <- data.frame(
        prize = prize,
        guess = guess,
        win = win,
        stringsAsFactors=FALSE
    )
    return(result)
}
```

This function requires drawing random numbers (using the `sample` function). To make the results reproducible we should set the "random seed". This means that each time you run the result you will get the same answer.

```
set.seed(12345)
```

Here's how it works:

```
monty()
```

```
##   prize guess  win
## 1     3     3 stay
```

Let's see what happens if we do this multiple times...

```
n_simulations <- 10
all_results <- list()
for(i in 1:n_simulations)
{
    message(i)
    all_results[[i]] <- monty()
}
all_results <- do.call(rbind, all_results)
```

Let's see what the results look like! We want to know the proportion of wins for the 'stay' strategy, and the proportion of wins for the 'switch' strategy

```
table(all_results$win)
```

```
##
##   stay switch
##      4      6
```

Perhaps this was just chance? Let's plot what it looks like. We will need to install a new library, `ggplot2`. This library is fantastic for making fairly complex plots very quickly.

```
install.packages("ggplot2")
```

```
library(ggplot2)
```

Here's the plot

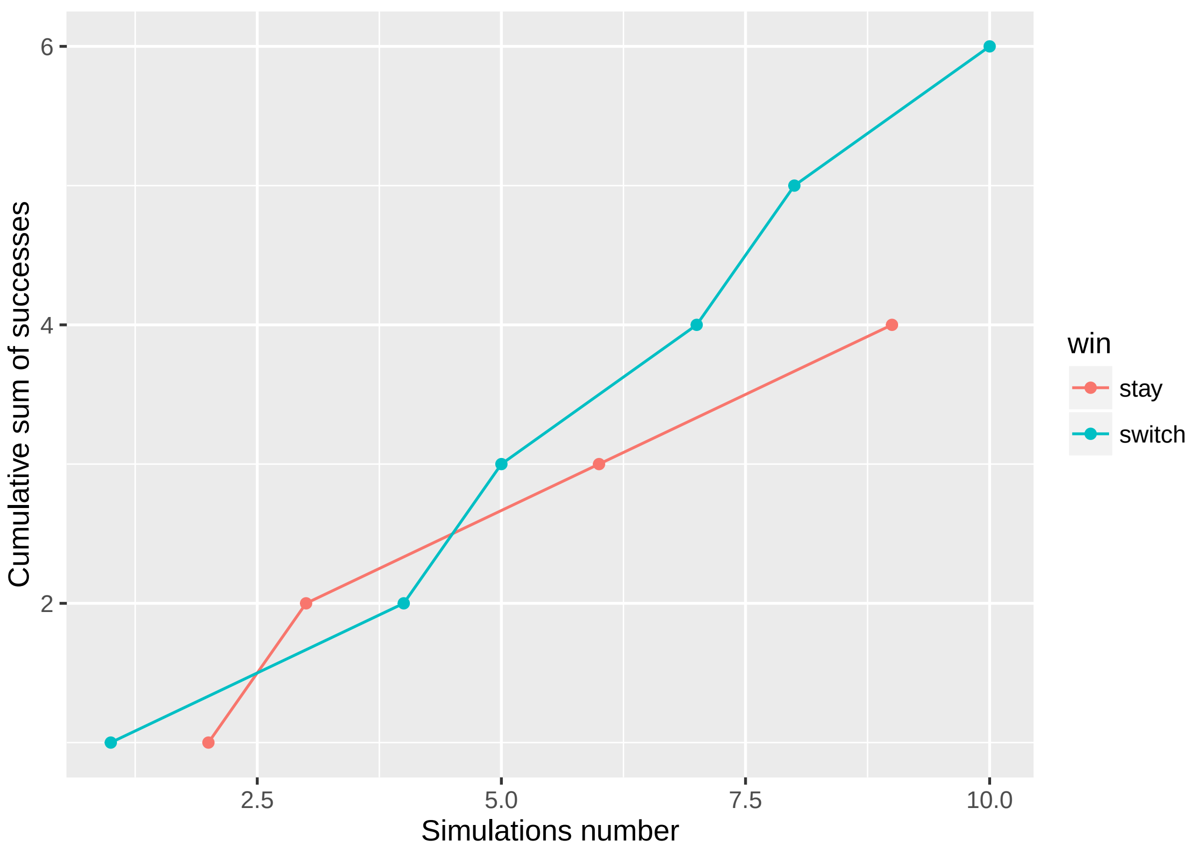
```
# Label the simulations
all_results$simulation <- 1:n_simulations

# Get the cumulative sum of wins
all_results$cumulative <- NA

stay_index <- all_results$win == "stay"
all_results$cumulative[stay_index] <- 1:sum(stay_index)
```

```
switch_index <- all_results$win == "switch"
all_results$cumulative[switch_index] <- 1:sum(switch_index)

# Make the plot
ggplot(all_results, aes(x=simulation, y=cumulative)) +
geom_point(aes(colour=win)) +
geom_line(aes(colour=win)) +
labs(y="Cumulative sum of successes", x="Simulations number")
```



We need more simulations to be sure about this.

1. Run the simulations again, but this time do 1000 simulations instead of just 10.
2. If you were playing the game, would you stick with your initial choise or switch?

Packages

We have already installed two packages. For the remainder of the course we are going to need some more. There are three main sources to get packages

- **CRAN** This is the main R package repository. It has over 8000 packages for a huge variaty of things. https://cran.r-project.org
- **Bioconductor** This is another repository which has packages that are mostly focused on genomic data. http://bioconductor.org
- **GitHub** A lot of people publish packages, or updates to packages, on GitHub before they are released to the CRAN or Bioconductor.

We need to install the following packages from CRAN:

```r
install.packages("CpGassoc")
install.packages("GenABEL")
```

And the following package from Bioconductor:

```r
source("http://bioconductor.org/biocLite.R")
biocLite("minfi")
```