Chapter 1. Introduction to R, and Descriptive Data Analysis

What is *R*: an environment for data analysis and graphics based on *S* language

- a full-featured programming language
- freely available to everyone (with complete source code)
- Easier access to the means of handling BigData such as parallel computation, Hadoop, distributed computation.
- official homepage: http://www.R-project.org

1.1 Installation

Installing *R*: R consists of two major parts: the base system and a collection of (over 8.5K) user contributed add-on packages, all available from the above website.

To install the base system, Windows users may follow the link

`http://CRAN.R-project.org/bin/windows/base/release.htm`

Note. The base distribution comes with some high-priority add-on packages such as graphic systems, linear models etc.

After the installation, one may start R in the PC by going to `Start -> Statistics -> R`, or simply doubt-click the logo 'R' on your desktop. An R-console will pops up with a prompt character like '>'.

R may be used as a calculators. Of course it can do much more. Try out

```
> sqrt(9)/3 -1
```

To quit R, type at the prompt '`q()`'.

It is strongly advised to use RStudio instead of R. You may find it with the link

https://www.rstudio.com/

We assume you use RStudio throughout the course.

To define a vector x consisting of integers $1, 2, \cdots, 100$

```
> x <- 1:100
> x
  [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
 [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
 [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
 [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
 [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
 [91]  91  92  93  94  95  96  97  98  99 100
> sum(x)
> [1] 5050
```

Or we may also try

```
> y <- (1:100)^2
> y
  [1]     1     4     9    16    25    36    49    64    81   100   121   144
 [13]   169   196   225   256   289   324   361   400   441   484   529   576
 [25]   625   676   729   784   841   900   961  1024  1089  1156  1225  1296
 [37]  1369  1444  1521  1600  1681  1764  1849  1936  2025  2116  2209  2304
 [49]  2401  2500  2601  2704  2809  2916  3025  3136  3249  3364  3481  3600
```

```
 [61]  3721  3844  3969  4096  4225  4356  4489  4624  4761  4900  5041  5184
 [73]  5329  5476  5625  5776  5929  6084  6241  6400  6561  6724  6889  7056
 [85]  7225  7396  7569  7744  7921  8100  8281  8464  8649  8836  9025  9216
 [97]  9409  9604  9801 10000
> y[14]     # print out the 14-th element of vector y
[1] 196
```

One may also try `x+y`, `(x+y)/(x+y)`, `help(log)`, `log(x)` etc.

Additional packages can be installed directly from the R prompt. Information on the available packages is available at

`http://cran.r-project.org/web/views/`

`http://cran.r-project.org/web/packages/`

For example, one may install HSAUR2 – *A Handbook of Statistical Analysis Using R (2nd edition)*:

```
> install.packages("HSAUR2")
> library("HSAUR2")  # To load all the objects in the package\\
                     #  into the current session
```

You may start an R help manual using command `help.start()`. By clicking `Packages` in the manual, you will see `HSAUR2` is listed among the installed packages.

1.2 Help and documentation

To start a manual page of R: `help.start()`

Alternatively we may access online manual at

`http://cran.r-project.org/manuals.html`

To access a manual for function 'mean': `help(mean)`, or `?mean`

To access the info on an added-on package: `help(package="HSAUR2")`

To access the info on a data set or a function in the installed package:
`help(package="HSAUR2", men1500m)`

To load all the functions in an added-on package: `library("HSAUR2")`

To load a data set from the installed package into the current session:
`data(men1500m, package="HSAUR2")`

Type `men1500m` to print out all the info in the data set 'men1500m'.

Two other useful sites:

R Newsletter: `http://cran.r-project.org/doc/Rnews/`

R FAQ: `http://cran.r-project.org/faqs.html`

You may also simply follow the links on the main page of the R project

`http://www.R-project.org`

Last but certainly not least, `google` whatever questions often leads to most helpful answers

1.3 Data Import/Export

The easiest form of data to import into R is a simple text file. The primary function to import from a text file is `scan`. You may check out what 'scan' can do: `> ?scan`

Create a plain text file '`simpleData`', in the folder 'statsI' in your Drive D, as follow:

```
This is a simple data file, created for illustration
of importing data in text files into R
1 2 3 4
5 6 7 8
9 10 11 12
```

It has two lines of explanation and 3 lines numbers. The R session below imports it into R as a vector `x` and 3×4 matrix `y`, perform some simple operations. Note the flag `skip=2` instructs R to ignore the first two lines in the file.

Note. R ignores anything after '#' in a command line.

```
> x <- scan("D:/statsI/simpleData.txt", skip=2)
> x                          # print out vector x
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
> length(x)
[1] 12
> mean(x); range(x) # write 2 commands in one line to save space
[1] 6.5
[1]  1 12
> summary(x)             # a very useful command!
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    3.75    6.50    6.50    9.25   12.00
```

```
> y <- matrix(scan("D:/statsI/simpleData.txt", skip=2), byrow=T,
            ncol=4)
> y                              # print out matrix y
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
> dim(y)            # size of matrix y
[1] 3 4
> y[1,]             # 1st row of y
[1] 1 2 3 4
> y[,2]             # 2nd column of y
[1]  2  6 10
> y[2,4]            # the (2,4)-th element of matrix y
[1] 8
```

A business school sent a questionnaire to its graduates in past 5 years and received 253 returns. The data are stored in a plain text file 'Jobs' which has 6 columns:

C1: ID number
C2: Job type, 1 - accounting, 2 - finance, 3 - management, 4 - marketing and sales, 5 -others
C3: Sex, 1 - male, 2 - female
C4: Job satisfaction, 1 - very satisfied, 2 - satisfied, 3 - not satisfied
C5: Salary (in thousand pounds)
C6: No. of jobs after graduation

IDNo.	JobType	Sex	Satisfaction	Salary	Search
1	1	1	3	51	1
2	4	1	3	38	2
3	5	1	3	51	4
4	1	2	2	52	5
...	...				

We import data into R using command `read.table`

```
> jobs <- read.table("D:/statsI/Jobs.txt"); jobs
          V1       V2  V3           V4     V5     V6
1        IDNo. JobType Sex Satisfaction Salary Search
2          1        1   1            3     51      1
3          2        4   1            3     38      2
4          3        5   1            3     51      4
   ... ...
> dim(jobs)
[1] 254   6
> jobs[1,]
     V1       V2  V3           V4     V5     V6
1 IDNo. JobType Sex Satisfaction Salary Search
```

We repeat the above again by taking the 1st row as the names of variables (`header=T`) and the entries in 1st column as the names of the rows (`row.names =1`).

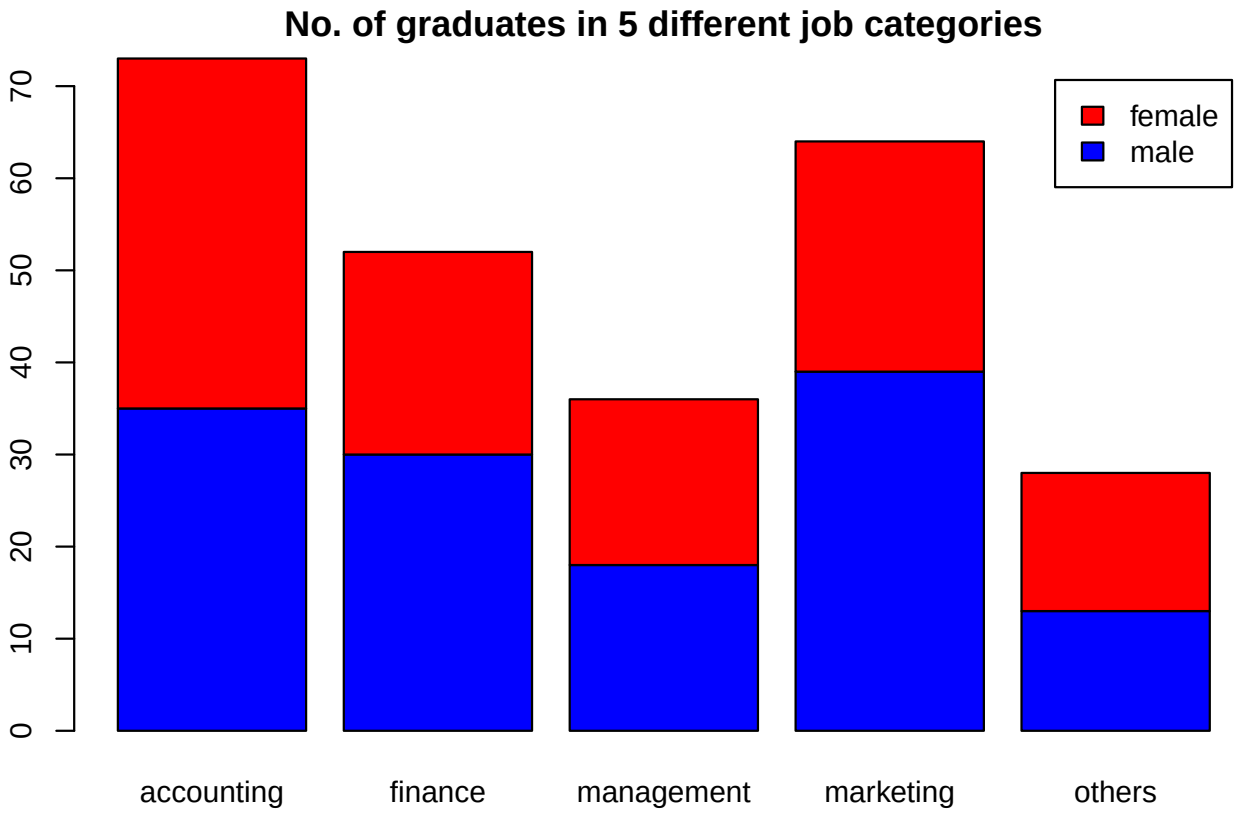
```
> jobs <- read.table("D:/statsI/Jobs.txt", header=T, row.names=1)
> dim(jobs)
[1] 253   5
> names(jobs)
[1] "JobType"  "Sex"   "Satisfaction"   "Salary"  "Search"
> class(jobs)
[1] "data.frame"
> class(jobs[,1]); class(jobs[,2]); class(jobs[,3]);
   class(jobs[,4]); class(jobs[,5])
[1] "integer"
[1] "integer"
[1] "integer"
[1] "integer"
[1] "integer"
```

Since the first three variables are nominal, we may specify them as ‘factor’, while "Salary" can be specified as ‘numeric’:

```
>  jobs <- read.table("D:/statsI/Jobs.txt", header=T, row.names=1,
            colClasses = c("factor", "factor", "factor",
            "numeric", "integer"))
> class(jobs[,1]); class(jobs[,2]); class(jobs[,3]);
       class(jobs[,4]); class(jobs[,5])
[1] "factor"
[1] "factor"
[1] "factor"
[1] "numeric"
[1] "integer"
```

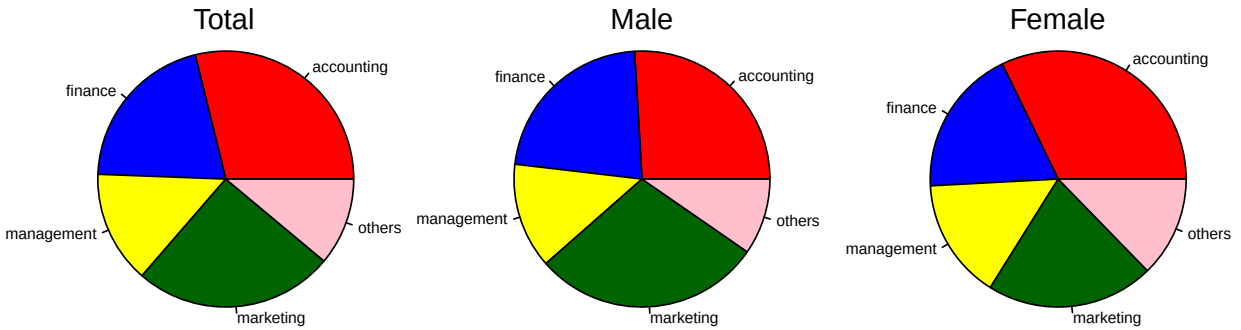
Note. we need to specify the class for the row name variable (i.e. 1st column) as well. Now we do some simple descriptive statistical analysis for this data.

```
> table(jobs[,1])
 1  2  3  4  5
73 52 36 64 28   # No. of graduates with 5 different JobTypes
> t <-table(jobs[,2], jobs[,1], deparse.level=2)  # store table in t
> t
         jobs[, 1]
jobs[, 2]  1  2  3  4  5
        1 35 30 18 39 13          # No. of males with 5 different JobTypes
        2 38 22 18 25 15          # No. of females with 5 different JobTypes
> 100*t[1,]/sum(t[1,])
       1        2        3        4        5
25.92593   22.22222   13.33333   28.88889   9.62963
                        # Percentages of males with 5 different JobTypes
> 100*t[2,]/sum(t[2,])
       1        2        3        4        5
32.20339   18.64407   15.25424   21.18644   12.71186
                        # Percentages of females with 5 different JobTypes
> barplot(t, main="No. of graduates in 5 different job categories",
          legend.text=c("male", "female"), names.arg=c("accounting",
          "finance", "management", "marketing", "others"))  # draw a bar-plot
```


No. of graduates in 5 different job categories
female
male
0
10
20
30
40
50
60
70
accounting
finance
management
marketing
others

The barplot shows the difference in job distribution due to gender. We may also draw pie-plots, which are regarded as less effective.

```
> pie(t[1,]+t[2,],label=c("accounting","finance","management",
      "marketing","others")); text(0,1, "Total", cex=2)
> pie(t[1,],label=c("accounting","finance","management",
                "marketing","others")); text(0,1, "Male", cex=2)
> pie(t[2,],label=c("accounting","finance","management",
                "marketing","others")); text(0,1, "Female", cex=2)
```

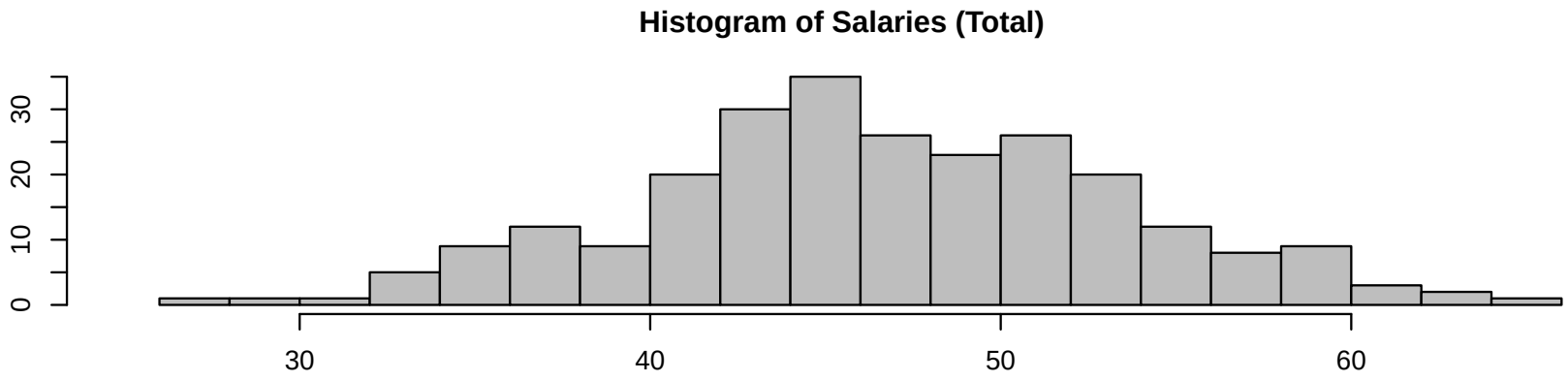


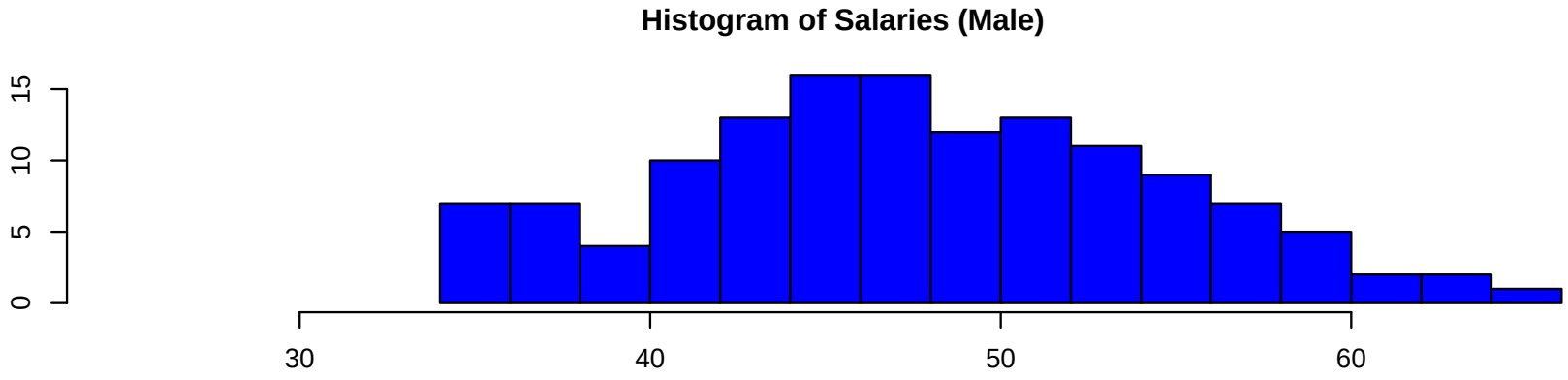
Now let us look at the salary (`jobs[,4]`) distribution, and the impact due to gender.

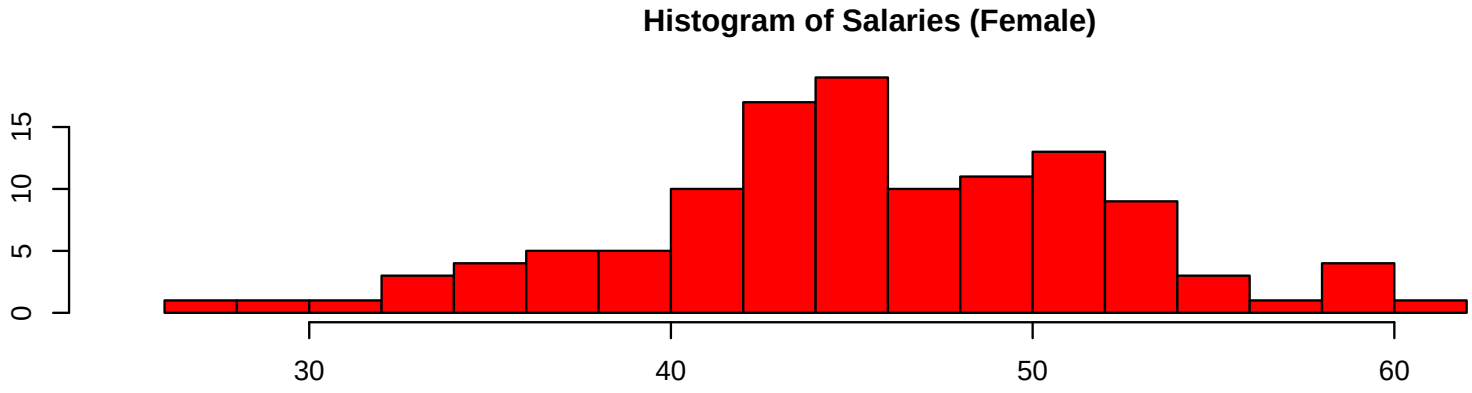
```
> mSalary <- jobs[,4][jobs[,2]==1]
                # extract the salary data from male
> fSalary <- jobs[,4][jobs[,2]==2]
                # extract the salary data from female
> summary(jobs[,4]); summary(mSalary); summary(fSalary)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  26.00   43.00   47.00   47.13   52.00   65.00
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  34.00   44.00   48.00   48.11   53.00   65.00
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  26.00   42.25   46.00   46.00   51.00   61.00
> hist(jobs[,4], col="gray", nclass=15, xlim=c(25,66),
                main="Histogram of Salaries (Total)")
      # plot the histogram of salary data
> hist(mSalary, col="blue", nclass=15, xlim=c(25,66),
                main="Histogram of Salaries (Male)")
```

```
> hist(fSalary, col="red", nclass=15, xlim=c(25,66),
                main="Histogram of Salaries (Female)")
```

You may also try stem-and-leaf plot: `stem(jobs[,4])`


Histogram of Salaries (Total)
0
10
20
30
30
40
50
60


Histogram of Salaries (Male)
0
5
10
15
30
40
50
60


Histogram of Salaries (Female)
0
5
10
15
30
40
50
60

To export data from R, use `write.table` or `write`.

To write `jobs` into a plain text file 'Jobs1.txt':

```
> write.table(jobs, "Jobs1.txt")
```

which retains both the row and column names. Note the different entries in the file are separated by spaces.

We may also use

```
> write.table(jobs, "Jobs2.txt", row.names=F, col.names=F),
> write.table(jobs, "Jobs3.txt", sep=",")
```

Compare the three output files.

Note that the values of factor variables are recorded with “ ". To record all the levels of factor variables as numerical values, we need to define a pure numerical data.frame first:

```
> t <- data.frame(as.numeric(jobs[,1]), as.numeric(jobs[,2]),
          as.numeric(jobs[,3]), jobs[,4], jobs[,5])
> write.table(t, "Jobs4.txt")
```

The file "Jobs4.txt" contains purely numerical values.

Note. (i) Working directory — all exported files are saved in ‘My Documents’ by default. You may change your working directory by clicking

```
File -> Change dir...
```

in the RGui window. For example, I create on my laptop `D:\statsI` as my working directory for this course.

(ii) Saving a session — when you quit an R session `q()`, you will be offered an option to 'save workspace image'. By clicking on "yes", you will save all the objects (including data sets, loaded functions from added-on packages etc) in your R session. You may continue to work on this session by directly double-clicking on the image file in your working directory.

A useful tip: Create a separate working directory for each of your R projects.

1.4 Organising an Analysis

An R analysis typically consists of executing several commands. Instead of typing each of those commands on the R prompt, we may collect them

into a plain text file. For example, the file "jobsAnalysis.r" in my working directory reads like:

```
jobs <- read.table("Jobs.txt", header=T, row.names=1)
        # File "Jobs.txt" is in the working directory now
mSalary <- jobs[,4][jobs[,2]==1]
fSalary <- jobs[,4][jobs[,2]==2]
summary(jobs[,4])
summary(mSalary)
summary(fSalary)
par(mfrow=c(3,1))   # display 3 figures in one column
hist(jobs[,4], col="gray", nclass=15, xlim=c(25,66),
         main="Histogram of Salaries (Total)")
hist(mSalary, col="blue", nclass=15, xlim=c(25,66),
        main="Histogram of Salaries (Male)")
hist(fSalary, col="red", nclass=15, xlim=c(25,66),
        main="Histogram of Salaries (Female)")
```

You may carry out the project by sourcing the file into an R session:

```
> source("jobAnalysis.r", echo=T)
```

Also try `source("jobAnalysis.r")`.

1.5 Writing functions in R

For some repeated task, it is convenient to define a function in R. We illustrate this idea by an example.

Consider the famous 'Birthday Coincidences' problem: *In a class of k students, what is the probability that at least two students have the same birthday?*

Let us make some assumptions to simplify the problems:

(i) only 365 days in every year,
(ii) every day is equally likely to be a birthday,
(iii) students' birthdays are independent with each other.

With k people, the total possibilities is $(365)^k$.

Consider the complementary event: all k birthdays are different. The total such possibility is

$$365 \times 364 \times 363 \times \cdots \times (365 - k + 1) = \frac{365!}{(365 - k)!}$$

So the probability that at least two students have the same birthday is

$$p(k) = 1 - \frac{365!}{(365 - k)!(365)^k}.$$

We may use R to compute $p(k)$. Unfortunately factorials are often too large, e.g. $52! = 8.065525e+67$, and often cause overflow in computer. We adopt the alternative formula

$$p(k) = 1 - \exp\{\log(365!) - \log((365-k)!) - k\log(365)\}.$$

We define a R-function pBirthday to perform this calculation for different k.

```
> pBirthday <- function(k)
+ 1 - exp(lfactorial(365) - lfactorial(365-k) - k*log(365))
                  # lfactorial(n) returns log(n!)
> pBirthday(100)
[1] 0.9999997   # probability with a class of 100 students
> x <- c(20, 30, 40, 50, 60)
> pBirthday(x)
[1] 0.4114384 0.7063162 0.8912318 0.9703736 0.9941227
```

With 20 students in class, the probability of having overlapping birthdays is about 0.41. But with 60 students, the probability is almost 1, i.e., *it is almost always true that at least 2 out of 60 students have the same birthday.*

Note. The expression in a function may have several lines. In this case the expression is enclosed in curly braces { }, and the final line determines the return value.

Another Example — **The capture and recapture problem**

To estimate the number of whitefish in a lake, 50 whitefish are caught, tagged and returned to the lake. Some time later another 50 are caught and only 3 are tagged ones. Find a reasonable estimate for the number of whitefish in the lake.

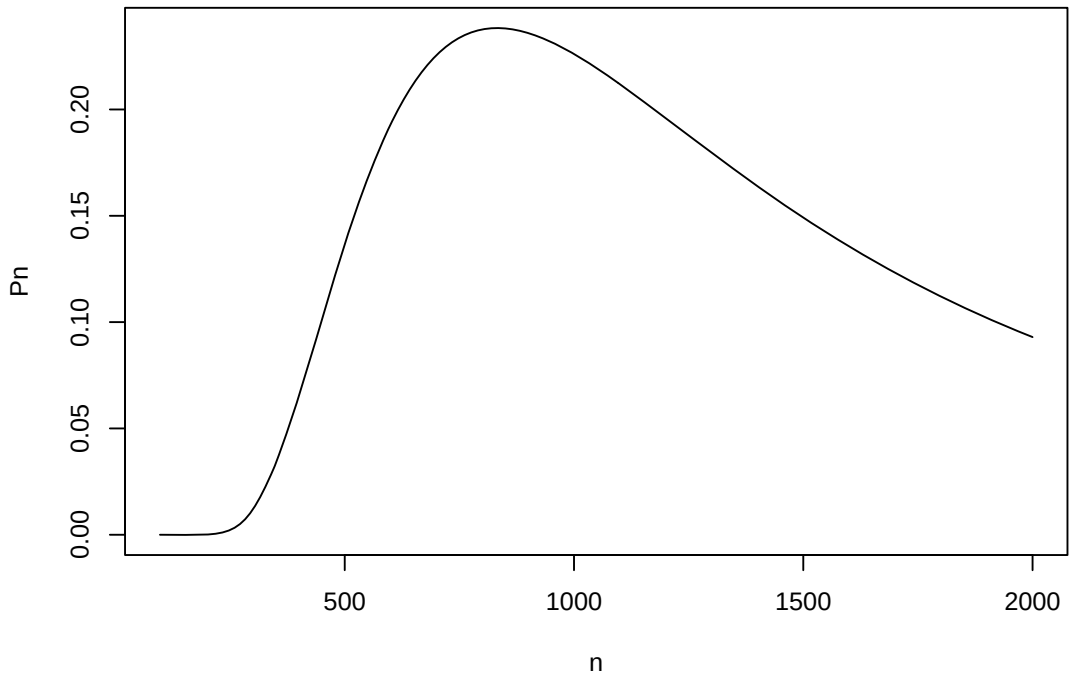
Suppose there are n whitefish in the lake. Catching 50 fish can be done in $\binom{n}{50} = \frac{n!}{50!(n-50)!}$ ways, while catching 3 tagged ones and 47 untagged can be done in $\binom{50}{3}\binom{n-50}{47}$ ways. Therefore the probability for the latter event to occur is

$$P_n = \binom{50}{3}\binom{n-50}{47} \Big/ \binom{n}{50}.$$

Therefore, a reasonable estimate for n should be the value at which P_n obtains its maximum. We use R to compute P_n and to find the estimate.

```
> Pn <- function(n) {
+       tmp <- choose(50,3)*choose(n-50,47)
+       tmp/choose(n,50)
+ }             # Definition for function Pn ends here
> n <- 97:2000    # as there are at least 97 fish in the lake
> plot(n, Pn(n), type='l')
```

It produces the plot of P_n against n:



To find the maximum:

```
> m <- max(Pn(n)); m
[1] 0.2382917
> n[Pn(n)==m]
[1] 833
```

Hence the estimated number of fish in the lake is 833.

1.6 Control structure: loops and conditionals

An `if` statement has the form

```
if (condition) expression1 else expression2
```

It executes ‘expression1’ if ‘condition’ is true, and ‘expression2’ otherwise. When ‘condition’ contains several lines, they should be enclosed in curly braces { }. The same applies to expressions.

The above statement can be compactly written in the form

```
ifelse(condition, expression1, expression2)
```

When the else-part is not present:

```
if (condition) expression
```

It executes ‘expression’ if ‘condition’ is true, and does nothing otherwise.

A `for` loop allows a statement to be iterated as a variables assumes values in a specified sequence. It has the form:

```
for(variable in sequence) statement
```

A `while` loop does not use an explicit loop variable:

```
while (condition) expression
```

It repeats 'expression' as long as 'condition' holds. This makes it differently from the "if-statement" above.

We illustrate those control commands by examining a simple 'doubling' strategy in gambling.

You go to a casino to play a simple 0-1 game: you bet x dollars and flip a coin. You win $2x$ dollars and keep your bet if 'Head', and lose x dollars if 'Tail'. You start 1 dollar in first game, and double your bet in each new games, i.e. you bet 2^{i-1} dollars in the i-th game, $i = 1, 2, \cdots$.

With this strategy, once you win, say, at the $(k+1)$-th game, you will recover all your losses in your previous games plus a profit of $2^k + 1$ dollars, as

$$2 \times 2^k > \sum_{i=1}^{k} 2^{i-1} = 2^k - 1.$$

Hence as long as (i) the probability p of the occurrence of 'Head' is positive (no matter how small), and (ii) you have enough capital to keep you in the games, you may win handsomely at the end — is it really true?

Condition (ii) is not trivial, as the maximum loss in 20 games is $2^{20} - 1 = 1,048,575$ dollars!

Plan A: Suppose you could afford to lose maximum n games and, therefore, decide to play n games. We define the R-function `nGames` below to simulate your final earning/loss (after n games).

```r
nGames <- function(n,p) {
      # n is the No. of games to play
      # p is the prob of winning each game
x <- 0 # earning after each game
for(i in 1:n)  ifelse(runif(1)<p, x <- x+2^i, x <- x-2^(i-1))
      # runif(1) returns a random number from uniform dist on (0, 1)
x     # print out your final earning/loss
}
```

To play $n = 20$ games with $p = 0.1$:

```
> nGames(20, 0.1)
[1] -999411
> nGames(20, 0.1)
[1] -1048575
> nGames(20, 0.1)
[1] 524289
> nGames(20, 0.1)
```

```
[1] -655263
> nGames(20, 0.1)
[1] -1016895
```

We repeated the experience 5 times above, with 5 different results.

One way to assess this gameplan is to repeat a large number of times and look at the average earning/loss:

```
> x = vector(length=5000)
> for(i in 1:5000) x[i] <- nGames(20, 0.1)
> mean(x)
[1] -733915
```

In fact, this mean -733915 is stable measure reflecting the average loss of this gameplan.

Plan B: Play the maximum n, but quit as soon as winning one game. The R-function `winStop` simulates the earning/loss.

```
winStop <- function(n,p) {
      # n -- maximum No. of games, p -- prob of winning each game
i <- 1
ifelse((runif(1)<p), x<- 2, x<- -1)  # play 1st game
while((x<0)&(i<n)){ i <- i+1      # i records the no. of games played
                ifelse(runif(1)<p, x <- x+2^i, x <- x-2^(i-1))
              }
x
}
```

Set $n = 20,\ p = 0.1$, we repeat the experience a few times:

```
>winStop(20, 0.1)
[1] 2
```

```
> winStop(20, 0.1)
[1] 17
> winStop(20, 0.1)
[1] 129
> winStop(20, 0.1)
[1] -1048575
> winStop(20, 0.1)
[1] 16385
```

To assess the gameplan:

```
> x<- 1:5000
> for(i in 1:5000) x[i] <- winStop(20, 0.1)
> mean(x)
[1] -112672.9   # This indicates "Plan B" is better than "Plan A"
> for(i in 1:5000) x[i] <- winStop(80, 0.1)
          # the maximum no. of games is 80 now
> mean(x)
```

```
[1] -7.22886e+20
> for(i in 1:5000) x[i] <- winStop(90, 0.1)
          # the maximum no. of games is 90 now
> mean(x)
3.790896e+18
```

With p as small as 0.1, you need a huge capital in order to play about 90 games to generate the positive returns in average.

The best and the most effective way to learn R: use it!

Hands-on experience is the most illuminating.