Statistics Illustrated towards data analysis

Farzali Izadi

Preface

As the title suggests, the book will be focusing on statistics with plots, graphs, codes and examples along with the precise definitions and explanations of the concepts. The coding has two purposes. The first purpose is to learn how to code statistics concepts using R. Secondly, it helps to do plots, graphs, charts, tables by which one can visualize and consequently grasp the concepts in an effective fashion. Besides, the plots can throw a culture in the subject which make it more fun and enjoyable. With the growing number of new data scientists coming from different backgrounds without necessary knowledge of statistics as an indispensable tool and limited time, the book can be served as an appropriate source for learning and practicing the statistics at a leisurely pace. The book is based upon a set of notes developed for the course in an elementary statistics for data science students in the Metro college of technology in Toronto, ON Canada with different backgrounds from math, physics, computer science, medicine, life and social sciences. However, the material has been substantially revised so that it will be useful for a larger audience. Although it introduces statistics on examples and coding bases, it doesn't sacrifice the basic tools and concepts. For this reason, it can be appropriate for not only general audience coming from the different backgrounds mentioned above, it can serve a first text for any college and university students of any major requiring the first course in statistics. This book is not a tutorial for R. But it is intended to introduce all necessary codes for the concepts and techniques of practical statistics. To make the codes more accessible, we first preview the basics and general R codes in the first chapter to cover just enough to give the audience they need to work through the concepts and examples in the following chapters where the main topics are presented.

For majors where the basic statistics is taught mainly as a tool, the book goes somewhat further than what is commonly taught at the undergraduate level. Multiple regressions, general linear regressions as well as general additive regressions are rarely taught at that level but very crucial to data science students and may also quickly become essential for practical search. The mathematical background of the students may not be extensive, so we try to keep a good balance between explaining statistical concepts comprehensively and showing their application and interpretation using R without complicated mathematics by lots of problem solving specially simulations, plots, and challenges that the computer coding offers. Several features of our textbook set it apart from the others. First, it emphasizes on data simulation and data plotting. We find that the use of simulation and plotting, are invaluable tool for teaching statistics concepts and tools. In addition to data simulations and plots, another emphasis of the book is on applications. We try to motivate the use of statistics throughout the sciences and give examples from biology, medicine, natural and life sciences, economics as a general discipline and business as a special case. It is divided into two completely separate parts: descriptive statistics and inferential statistics or from the language of the famous John Tukey: exploration analysis and confirmation analysis who wrote a book in 1977 titled "Exploratory Data Analysis" and made the exploratory data analysis or equivalently exploration analysis a formal subject of statistics. He has a quote "exploratory data analysis can never be the whole story, but nothing else can serve as the foundation stone.“ He emphasized the importance of designing simple tools for practical data analysis. In his mind, the exploratory steps in data analysis involve using summaries, tables and basic visualizations to search for hidden patterns in data. Then try to make insightful results. This is the first part of the book. In the second part, we use the patterns and results from the first part to generalize them to whole population which form the confirmation analysis including hypothesis tests, computing confidence intervals, parameter estimations, and regression analysis.

The rest of the book is outlined as follows.

Chapter 1 starts with introduction to statistics, preview the basics of R coding and continues to the definitions and examples of populations, samples, parameters and statistics. Then a little history, and finally ends with different types of sampling.

Chapter 2 deals with variables, constants and their various types, quantitative variables and qualitative variables or levels of basic measurements as well as discrete and continuous quantitative variables along with some related examples.

In chapter 3, we introduce two types of statistics namely descriptive and inferential by contrasting their differences and domain of applications. Then we move to the concept of measures of central tendency and central of measures of variability, five number theory and the definitions and examples of skewness and kurtosis. We close the chapter by generalizing the concept of the mean namely the expectation and using it to define covariance and correlation with some examples.

Chapter 4 contains extensive material on probabilities, random variables, discrete and continuous distribution, including binomial, Poisson, and negative-binomial, uniform, exponential, normal, t- distribution, Chisq-distribution, F-distribution as well as their applications to some interesting problems. Newcomb-Benford's law is also discussed.

In chapter 5, we present the concept of statistical significance, effect size and confidence intervals and parameter estimations.

Chapter 6 deals with hypothesis tests, independent samples t-tests, paired and dependent samples t-tests.

Chapter 7 is devoted to the analysis of variance, one - way analysis of variance and two - way analysis of variance.
The Chi-Square tests and independence of Chi-square tests are topics of final sections.

We close the book with chapter 8 consisting the various topics of regression models including linear regressions, linear multiple Regression, generalized linear regression and generalized additive regression models.

Farzali Izadi September 1, 2020 Toronto ON Canada

Chapter 1 Introduction to statistics
1.1 Preview the basics of R coding
1.2 History
1.3 Populations, samples, parameters and statistics
1.4 Types of sampling

Chapter 2 Variables, constants and their different types
2.1 Variables and constants
2.2 Qualitative and quantitative variables
2.3 Qualitative variables or levels of basic measurements
2.4 Continuous and discrete quantitative variables
2.5 Tables

Chapter 3 Two types of statistics – Descriptive and inferential
3.1 Descriptive statistics
3.2 Measures of central tendency
3.3 Central of measures of variability
3.4 Five number theory
3.5 Skewness
3.6 Kurtosis
3.7 Correlations

Chapter 4 Probabilities and distributions
4.1 Probability
4.2 Random variables
4.3 Conditional probability
4.4 Newcomb - Benford law
4.5 Statistical distributions

Chapter 5 Statistical significance, effect size, and confidence intervals
5.1 Statistical Significance and confidence intervals
5.2 estimating parameter values and effect size

Chapter 6 Hypothesis tests
6.1 Independent samples t-tests
6.2 Paired or dependent samples t-tests

Chapter 7 Analysis of variance
7.1 One - way analysis of variance
7.2 Two - way analysis of variance
7.3 The Chi-Square test
7.4 Independence of Chi-Square

Chapter 8 Regressions

8.1 Linear regressions

8.2 Linear multiple Regression

8.3 Generalized linear Regression

8.4 Generalized additive Regression

Chapter 1
Introduction to statistics

Statistics is a branch of mathematics that deals with the collecting, organizing, summarizing, analyzing the datasets and then interpreting and communicating of the results. These include demographic data, census, experiments, internet, satellite and etc. Although statistics has many theoretical subareas and a lot of work are carried out in universities and institutions around the world to accomplish new insights and ideas, its applications to real world problems are enormous and it is a crucial fact to find appropriate and optimized solutions for these problems. They arise from disciplines throughout the natural and social sciences, especially biology, physics, healthcare, economics and business. Governments may be interested in the life expectancy of their population. Business sectors try to identify and evaluate customer satisfaction and then segment their customers based on their income and favorite products in order to increase their revenues. Financial institutions wish to invest their assets on profitable projects. Health networks analyze their Databases to improve public health. Another important examples are recommendation systems, fraud detections and risk analysis. As we discussed earlier that tables and plots (visualizations) are very effective for both learning and communicating of the results, our first step is to preview R code to not only accomplish these tasks but also gives a quick introduction to R objects and functions which are necessary for the rest of the book.

I close this introduction by a quote from Larry Wasserman -UPMC Professor of Statistics and Data Science: "Statistics, data mining and machine learning are all concerned with collecting and analyzing data. For some time, statistics research was conducted in statistics departments while data mining and machine learning research was conducted in computer science departments. Statisticians thought that computer scientists were reinventing the wheel. Computer scientists thought that statistical theory didn’t apply to their problems. Things are changing. Statisticians now recognize that computer scientists are making novel contributions while computer scientists now recognize the generality of statistical theory and methodology.“

Arthur Conan Doyle
A British writer May 22, 1859 - July 7, 1930

“Data! Data! Data!” he cried impatiently. “I can’t make bricks without clay.”

Arthur Conan Doyle

1.1 Preview the basics of R coding

(i) Simple calculation, data formats, and built-in functions

R is an object - oriented programming The advantages of this kind of language can be explained by example. Let us to do regression analysis. To perform a regression analysis with other statistical packages, such as SAS or SPSS, one gets tons of output on the screen. By contrast, if we use the lm() regression in R, the function returns an *object* containing all the results - the estimated coefficients, their standard errors, residuals, and p-values. We then choose, programmatically, which parts of that object to extract. We start with arithmetic operations on real numbers. Remember that the items marked with # are comments. They are ignored by the R interpreter. They serve as notes to remind us what the code is doing. We start with some simple mathematical calculations.

```
#Math
14+9                    # Sum of two numbers
[1] 23
15-12                   # The difference of two numbers
[1] 3
7*8                     # The product of two numbers
[1] 56
100/3                   # The division of two numbers
[1] 33.33333
```

```
 3^4                  # 3 to the power of 4
[1] 81
5+3*6^4              # more operations
[1] 3893

R uses parentheses for grouping, as is done in math texts.

((12+13) * 15 )^4
[1] 19775390625

# Use %% for modulus i.e., the reminder of the division of a by b.
100 %% 30
[1] 10
log()                # Takes the natural logarithm of its argument.
log(2.7182)
[1] 0.9999699
                     # The Euler number e
exp(1)
[1] 2.718282
exp()                # Raises e the to to the power of its argument.
exp(10)
[1] 22026.47
```

```
log10()                 # Takes the log base 10 of a number.
log10(100)
[1] 2
log2()                  # Takes the log base 2 of a number.
log2(64)
[1] 6
sqrt()                  # Returns the square root of its argument.
sqrt(64)
[1] 8
round()                 # Rounds its argument to the nearest whole number.
round(233.234)
[1] 233
round(233.234, digits=1)
[1] 233.2
round(233.234, digits=-1)
[1] 230
round(2.5)              # Rounds down to the even digit 2.
[1] 2
round(3.5)              # Rounds up to the even digit 4.
[1] 4
```

```
floor(2.8)                  # floor() always rounds down.
[1] 2
ceiling(2.2)                # ceiling() always rounds up.
[1] 3
trunc(2.6)                  # trunc() rounds in the direction of zero.
[1] 2
trunc(-2.6)
[1] -2
abs(-10)                    # abs() returns the absolute value of the argument.
[1] 10
pi                          # pi returns the constant pi
[1] 3.141593
3.14159265358979
[1] 3.141593
cos(0)                      # Cosine function
[1] 1
sin(pi/2)                   # Sine function
[1] 1
tan(pi/4)                   # Tangent function
[1] 1
```

Every R object contains a number of attributes to describe the nature of the information that object may have. Two of the most important attributes of data in R are the mode and the class functions. In addition to the mode and class , the typeof function can sometimes return additional information about the type of an object, although it is not generally as useful as that provided by mode or class. However, in big collection of data such as a matrix or a dataset, the other modes can be encountered.
Some objects like matrices or other arrays demand that all the data contained in them be of the same mode; others like lists and data frames allow for multiple modes within a single object. When working with data, the most commonly encountered modes of individual objects are numeric, character, and logical. To store and represent the categorical data, R uses the factor object in which by examining its mode one always gets numeric. Finally, one of the most frequently encountered modes of data is the list. Lists are the most flexible way of storing data in R, since they can accommodate objects of different modes and lengths. Many functions in R use lists to hold their results. The result of typeof() function for all types of numbers are double, for a complex number is complex, for a character string is a character, for matrices and arrays is a array and for a data frame is a list. Let us examine the typeof, mode, and class functions for various objects in R. Notice that by semicolons ; one can execute more than one commands on a single line.

```
typeof(5); mode(5); class(5)
[1] "double"
[1] "numeric"
[1] "numeric"
> typeof(-17.8); mode(-17.8); class(-17.8)
[1] "double"
[1] "numeric"
[1] "numeric"
> typeof(-Inf) ; mode(-Inf) ; class(-Inf)
[1] "double"
[1] "numeric"
[1] "numeric"
> x = 17.8                    # Assigns a decimal value x.
> print(x)                    # Prints the value of x.
[1] 17.8
> z = 1 + 2i                  # Creates a complex number.
> z                           # Prints the value of z.
[1] 1+2i
> typeof(z); mode(z); class(z)
[1] "complex"
[1] "complex"
[1] "complex"
```

While the mode or class of an object can easily be examined through the mode and class functions, in many cases R provides a simpler way to verify whether an object has a particular mode, or is a member of a particular class. A large number of functions in R, beginning with the string “is.”, can be used to test if an object is of a particular type. Among the many such predicate functions available in R are is.list(), is.numeric(), is.character(), is.data.frame(), is.factor(), is.logical().

```
> m <- 10
> is.integer(m)
[1] FALSE
> is.double(m)
[1] TRUE
> is.integer(7)
[1] FALSE
> is.double(7)
[1] TRUE
> is.infinite(-Inf)
[1] TRUE
> is.logical(TRUE)
[1] TRUE
> is.character(4)
[1] FALSE
> is.character('4')
[1] TRUE
> is.character('R')
[1] TRUE
```

Similarly the string “as.”, can be used to get an object of required type. Among the many such predicate functions available in R are as.list(), as.numeric(), as.character(), as.data.frame(), as.factor(), and so on.

```
> m <- 10; n <- as.integer(m)
> is.integer(n)
[1] TRUE
> as.double(m)
[1] 10
> as.integer('7')
[1] 7
> as.logical(1)
[1] TRUE
> as.logical(6)
[1] TRUE
> as.character(4)
[1] "4"
> is.character('4')
[1] TRUE
> as.integer('R')
[1] NA
Warning message: NAs introduced by coercion
> is.logical(0)
[1] FALSE
> as.logical(0)
[1] FALSE
```

```
#Integer
y = as.integer(3)
y                          # Prints the value of y.
[1] 3
class(y)                   # Print the class name of y.
[1] "integer"
is.integer(y)              # Is y an integer?
[1] TRUE
as.integer(3.14)           # Coerces a numeric value.
x = as.character(3.14)     # as character
x                          # Prints the character string.
[1] "3.14"
class(x)                   # Prints the class name of x.
[1] 3
as.integer("5.27")         # Coerce a decimal string.
[1] 5
as.integer("Joe")          # Coerce an non-decimal string.
[1] NA Warning message: NAs introduced by coercion
as.integer(TRUE)           # The numeric value of TRUE.
[1] 1
```

More examples

```
[1] "logical“
sqrt(-1)                        # Square root of -1
[1] NaN Warning message: In sqrt(-1) : NaNs produced.
```

Instead, we have to use the complex value -1 + 0i.

```
sqrt(-1+0i)                    # Square root of -1+0i
[1] 0+1i
```

An alternative is to coerce -1 into a complex value.

```
sqrt(as.complex(-1))
[1] 0+1i
```

```
x = 1; y = 2; z = x > y   # Logical: Is x larger than y?
z                         # Prints the logical value.
[1] FALSE
class(z)                  # Prints the class name of z.
[1] "logical“
u = TRUE; v = FALSE
u & v                     # u AND v
[1] FALSE
u | v                     # u OR v
[1] TRUE
!u                        # Negation of u
[1] FALSE
[1] "character“
```

Two characters can be concatenated with the paste function.

```
fname = "Joe"; lname ="Smith“; paste(fname, lname)
[1] "Joe Smith“
```

Having introduced the arithmetic operations and some useful functions, let us make a simple data set in R, a vector consisting of the numbers -5, -2, 0, 3,4,7, and name it v:

```
#Vectors
v <- c(-5, -2, 0, 3, 4, 7)
v [1] -5 -2 0 3 4 7
mode(v)
[1] "numeric“
typeof(v)
[1] "double“
class(v)
[1] "numeric”
```

The standard assignment operator in R <- is a left diamond bracket (<) followed by a minus sign (-). It means “is assigned to”. One can also use =, but this is discouraged, as it does not work in some special situations. The function c stands for concatenate. Here, we are concatenating the numbers -5, -2, 0, 3, 4, and 7. If we have two vectors u and v, we can concatenate to get vector, say, w.

```
u <- c(7.8, 4, -6.76)
v <- c(3, 6, 0)
w <- c(u, v)
w
[1]  7.80  4.00 -6.76  3.00  6.00  0.00
```

To access to an element of a vector, we use brackets like v[1] which means the first element. The last element is v[length(v)], where length is the number of the elements in the vector v. The number inside the bracket is called the index of the corresponding element.

length(v)

[1] 11

v[1:length(v)]

[1] -4 -3 -2 -1 0 1 2 3 4 5 6

To get the first and sixth element we use

v[c(1,6)]

[1] -4 1

To remove some elements, we use negative indices.

v[-4]

[1] -4 -3 -2 0 1 2 3 4 5 6

v[-(2:4)]

[1] -4 0 1 2 3 4 5 6

v[-c(1, 3, 5)]

[1] -3 -1 1 2 3 4 5 6

Vectors with logical and character elements

```
> l <- c(TRUE, FALSE, TRUE, FALSE, FALSE); l; typeof(l); mode(l); class(l)
[1]  TRUE FALSE  TRUE FALSE FALSE
[1] "logical"
[1] "logical"
[1] "logical“
```

```
> ch <- c("aa", "bb", "cc", "dd", "ee"); ch; typeof(ch); mode(ch); class(ch)
[1] "aa" "bb" "cc" "dd" "ee"
[1] "character"
[1] "character"
[1] "character"
[1] "aa" "bb" "cc" "dd" "ee“
```

We may insert some other numbers between u and v too.

```
w <- c(u, 88,-43,100, v)
w
[1]   7.80   4.00  -6.76  88.00 -43.00 100.00   3.00   6.00   0.00
```

If we want to make a vector from consecutive numbers, we can use

```
u <- c(-4:6)
u
[1] -4 -3 -2 -1  0  1  2  3  4  5  6
```

Even without c() function, we can get the same vector.

```
v <- -4:6
v
[1] -4 -3 -2 -1  0  1  2  3  4  5  6
```

As we see to print the vector to the screen, simply type its name, so there is no need for the print() function for these type of objects. Also without naming the vector in the first place, we can get the vector as well.

```
-4:6
[1] -4 -3 -2 -1  0  1  2  3  4  5  6
c(-4:6)
[1] -4 -3 -2 -1  0  1  2  3  4  5  6
```

Start from empty vector and add the elements.

```
v <- vector()
v[1] <- -6
v[2] <- 7
v[3] <- 4
v
[1] -6  7  4
```

Instead of indices, one can use names of the elements.

```
u <- vector()
u['a'] <- 5
u['b'] <- -24
u['c'] <- 7
v[0] <- 7
```

To get subset of vector we use

```
v[1]
[1] -4
v[3:5]
[1] -2 -1  0
w<- c("aa", "bb", "cc", "dd", "ee")
length(c("aa", "bb", "cc", "dd", "ee"))
[1] 5
length(w)
[1] 5
```

```
u
 a  b  c
 5 -24  7
names(u)
[1] "a" "b" "c“
```

The vector type is really the heart of R. The elements of a vector must all have the same mode, or data type. One can have a vector consisting of three characters (of mode character) or three integer elements (of mode integer), but not a vector with one integer element and two characters elements.

```
u <- c(7, 8, 9.8)
u
[1] 7.0 8.0 9.8
v <- c('M', 'F')
v
[1] "M" "F“
c(u, v)
[1] "7"   "8"   "9.8" "M"   "F"
```

Two different sub-setting of a vector

```
x <- c(6,1:3, NA, 12)
x
[1]  6  1  2  3 NA 12
x[x > 5]
[1]  6 NA 12
subset(x, x > 5)                    # Subset function are very useful in data frames.
[1]  6 12
```

Vectors with a pre-defined length

```
x <- vector(length = 10)
# with a length and type
vector("character", length = 10)
[1] "" "" "" "" "" "" "" "" "" ""

vector("numeric", length = 10)
[1] 0 0 0 0 0 0 0 0 0 0
vector("integer", length = 10)
[1] 0 0 0 0 0 0 0 0 0 0
vector("logical", length = 10)
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

str() function is very fruitful giving all attributes of the data specially the structure of a data frame , an important type of data format in many data science coding languages.

```
z <- c("Bob", "Don", "Robin", "Sam")
```

Examine your vector

```
str(z)
chr [1:4] "Bob" "Don" "Robin" "Sam“
```

Vector arithmetic

```
a = c(1, 3, 5, 7)
b = c(1, 2, 4, 8)
5 * a
[1]  5 15 25 35
a + b
[1]  2  5  9 15
a - b
[1]  0  1  1 -1
a * b
[1]  1  6 20 56
a / b
[1] 1.000 1.500 1.250 0.87
```

Principal of recycling

When applying an arithmetic operation to two vectors that requires them to be the same length, R automatically recycles, or repeats, the shorter one, until it is long enough to match the longer one. Here is an example:

```
u = c(10, 20, 30)
v = c(1, 2, 3, 4, 5, 6, 7, 8, 9)
u + v
[1] 11 22 33 14 25 36 17 28 39
u - v
[1]  9 18 27  6 15 24  3 12 21
u * v
[1]  10  40  90  40 100 180  70 160 270
u / v
[1] 10.000000 10.000000 10.000000  2.500000  4.000000  5.000000  1.428571  2.500000
[9]  3.333333
v+u
[1] 11 22 33 14 25 36 17 28 39
v -u
[1]  -9 -18 -27  -6 -15 -24  -3 -12 -21
v *u
[1]  10  40  90  40 100 180  70 160 270
v /u
[1] 0.10 0.10 0.10 0.40 0.25 0.20 0.70 0.40 0.3
```

Other functions to generate vectors.

```
y <- seq(from = 1, to = 30, by = 4)
y
[1]  1  5  9 13 17 21 25 2z <- seq(0, 100, 10)     # One can omit the argument names.
z
[1]   0  10  20  30  40  50  60  70  80  90 100
r <- rep(x=1, times=20)                            #  It repeats 1 twenty times.
r
[1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
s <- rep(2:5, 3)                                   #  It repeats 2:5  twenty 3 times.
s
[1] 2 3 4 5 2 3 4 5 2 3 4 5
x <- runif(n=8, min=0, max=100)                     #  Uniform random numbers
```

One more function which we use more often to do random sampling is sample()

```
s <- sample(1:10, 10)
s
[1]  4 10  1  8  3  6  7  5  2  9
```

If we want more from the same numbers, we use replace = TRUE, otherwise it throws an error.

```
s <- sample(1:10, 20, replace = TRUE)
s
[1]  7  4 10  6  9  5  4  2  6  5  7  8  1  1  8  7  2  8  6  5
```

Sum of rolling a pair of dice 1 times

```
sample(1:6, size=1) + sample(1:6, size=1)
[1] 8
```

```
sample(1:6,size=5) + sample(1:6,size=5)                    # Sum of rolling a pair of dice 5 times.
[1]  9  6 10  7  6
```

Combinatorics

Let us return to the case of sampling without replacement, specifically sample(1:10, 4). The probability of obtaining a given number as the first one of the sample should be 1/10, the next one 1/9, and so on. The probability of a given sample should then be $1/(10 \times 9 \times 8 \times 7)$. In R, use the prod() function to calculates the product of a vector of numbers

```
1/prod(10:7)
[1] 0.0001984127
```

However, if we would rather be interested in the probability of guessing a given set of five numbers in a Lotto-like game, we shouldn't consider the order of the numbers been picked. There are $4 \times 3 \times 2 \times 1$ different possibilities for this to happen.

```
 prod(4:1)/prod(10:7)
[1] 0.004761905
```

Matrix in R

An R matrix corresponds to the mathematical concept of the same name: a rectangular array of numbers. Technically, a matrix is a vector, but with two additional attributes: the number of rows and the number of columns.
Elements are arranged column-wise by default taking the numbers sequentially. We can also check the typeof, mode, and class of a matrix.

```
> M <- matrix(c(1:4), nrow = 2); M; typeof(M); mode(M); class(M)
     [,1] [,2]
[1,]    1    3
[2,]    2    4
[1] "integer"
[1] "numeric"
[1] "matrix" "array"
```

To get the matrix by row we need to add byrow = TRUE

```
M <- matrix(c(1:4), nrow = 2, byrow = TRUE)
M
     [,1] [,2]
[1,]    1    2
[2,]    3    4
```

Assigning only one of the nrow or ncol is enough.

```
N <- matrix(1:4, ncol =2, byrow = FALSE)
N
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

Define the column and row names.

```
rownames = c("row1", "row2")

colnames = c("col1", "col2","col3", "col4","col5", "col6","col7", "col8" )

M  <- matrix(c(1:16), nrow = 2, byrow = TRUE, dimnames = list(rownames, colnames))
M
     col1 col2 col3 col4 col5 col6 col7 col8
row1    1    2    3    4    5    6    7    8
row2    9   10   11   12   13   14   15   16
```

Access the element at 3rd column and 1st row.

```
M[1,3]
[1] 3
```

Access only the 2nd row.

```
M[2,]
col1 col2 col3 col4 col5 col6 col7 col8
   9   10   11   12   13   14   15   16
```

Access only the 8th column.

```
M[,8]
row1 row2
   8   16
```

Matrix Addition & Subtraction

Create two 2x3 matrices.

```
matrix1 <- matrix(c(3, 9, -1, 4, 2, 6), nrow = 2)
matrix1
     [,1] [,2] [,3]
[1,]    3   -1    2
[2,]    9    4    6
```

```
matrix2 <- matrix(c(5, 2, 0, 9, 3, 4), nrow = 2)
matrix2
     [,1] [,2] [,3]
[1,]    5    0    3
[2,]    2    9    4
Matrix addition
result <- matrix1 + matrix2
cat("Result of addition","\n")
Result of addition
result
     [,1] [,2] [,3]
[1,]    8   -1    5
[2,]   11   13   10
Matrix Multiplication & Division
# Create two 2x3 matrices.
matrix1 <- matrix(c(3, 9, -1, 4, 2, 6,7,9,7), ncol = 3)
matrix2 <- matrix(c(5, 2, 0, 9, 3, 43,5,3,1), nrow = 3)
# Multiply the matrices.
result <- matrix1 * matrix2           # element by element multiplication
result2 <- matrix1%*% matrix2     # linear algbra multiplication
result
     [,1] [,2] [,3]
[1,]   15   36   35
[2,]   18    6   27
[3,]    0  258    7
```

```
result2
     [,1] [,2] [,3]
[1,]   23  340   34
[2,]   49  474   60
[3,]    7  310   20
Matrix division
result3 <- matrix1 / matrix2                    # element by element
result4 <- matrix1%/% matrix2                   # linear algebra multiplication
result3
     [,1]      [,2] [,3]
[1,]  0.6 0.4444444  1.4
[2,]  4.5 0.6666667  3.0
[3,] -Inf 0.1395349  7.0
result4
     [,1] [,2] [,3]
[1,]    0    0    1
[2,]    4    0    3
[3,] -Inf    0    7
diag(matrix1)                                   # Get elements on the main diagonal of a matrix
[1] 3 2 7
solve(matrix1 )                                 # Get the inverse of a square matrix
     [,1]   [,2]   [,3]
[1,]  2.5 -0.875 -1.375
[2,]  4.5 -1.750 -2.250
[3,] -3.5  1.375  1.878
```

```
eigen(matrix1)                                  # Get the eigenvectors and eigenvalues of a matrix
eigen() decomposition
$values
[1] 14.7702189 -3.1176766  0.3474577

$vectors
         [,1]       [,2]       [,3]
[1,] 0.5237053 -0.01910711 -0.4368839
[2,] 0.7055546 -0.86080006 -0.6996506
[3,] 0.4774154  0.50858448  0.5653508
t(matrix1)                                      # Get the transpose of a matrix
     [,1] [,2] [,3]
[1,]    3    9   -1
[2,]    4    2    6
[3,]    7    9    7
# rbind() and cbind() functions

A <- matrix(1:4, nrow = 2)
B <- matrix(11:14, nrow = 2)
cbind(A, B)
     [,1] [,2] [,3] [,4]
[1,]    1    3   11   13
[2,]    2    4   12   14
```

```
rbind(A, B)
     [,1] [,2]
[1,]    1    3
[2,]    2    4
[3,]   11   13
[4,]   12   10
```

Notice that the matrices are the most convenient data types to deal with. However, when you wish to store the Rows or columns of a matrix with different modes, R automatically returns an error except for the cases when we We have real and complex numbers as well as logical entries.

```
M <- matrix(1:2, c(TRUE, FALSE), c(2+6i, 4+2i), nrow=2)
> typeof(M); mode(M);class(M)
[1] "integer"
[1] "numeric"
[1] "matrix" "array“
```

But for the following we get errors.

```
f = factor('M', 'M', 'F')
M <- matrix(f, 1:3, nrow=2)
M <- matrix(1:2, c('T', TRUE), nrow=2)
```

.

Arrays

Arrays are made of many layers of different matrices.
Create two vectors of different lengths.

```
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)
# Take these vectors as input to the array.
result <- array(c(vector1,vector2),dim=c(3,3,2))
class(result)                                          # To see the class of an array
[1] "array"
result
, , 1

     [,1] [,2] [,3]
[1,]    5   10   13
[2,]    9   11   14
[3,]    3   12   15

, , 2

     [,1] [,2] [,3]
[1,]    5   10   13
[2,]    9   11   14
[3,]    3   12   15
```

```
# Naming columns and rows
column.names <- c("COL1","COL2","COL3")
row.names <- c("ROW1","ROW2","ROW3")
matrix.names <- c("Matrix1","Matrix2")

# Take these vectors as input to the array.
result <- array(c(vector1,vector2),dim=c(3,3,2),dimnames =
            list(row.names,column.names,matrix.names))

result
, , Matrix1

     COL1 COL2 COL3
ROW1    5   10   13
ROW2    9   11   14
ROW3    3   12   15

, , Matrix2

     COL1 COL2 COL3
ROW1    5   10   13
ROW2    9   11   14
ROW3    3   12   15
```

The data frame is the most flexible type of R object to use in data analysis, statistics, and machine learning.
It is a list with the restriction that each element of the list (the variables) must be of the same length as every other element of the list. Thus, the mode of a data frame is a list, and its class is data.frame. Data frames are fundamental to the use of the R modelling and graphical functions. A data frame is a generalization of a matrix, in which different columns may have different modes. However, all elements of any column must have the same mode, i.e. all numeric or all factor, or all character. A factor of a data frame is an important part of R objects. Factors provide a compact way to store character strings. They are crucial in the representation of categorical effects in model and graphics formulae. A factor is stored internally as a numeric vector with values 1, 2, 3, k, where k is the number of levels. Let us create a factor.

```
> # Create a vector as input.
> data <- c("East","West","East","North","North","East","West","West","West","East","North")
> data
 [1] "East"  "West"  "East"  "North" "North" "East"  "West"  "West"  "West"  "East"  "North"
> print(is.factor(data))
[1] FALSE

> # Apply the factor function.
> factor_data <- factor(data)
> factor_data
 [1] East  West  East  North North East  West  West  West  East  North
Levels: East North West
> is.factor(factor_data)
[1] TRUE
```

```
> fdo <- factor(data, ordered = TRUE)
> fdo
 [1] East  West  East  North North East  West  West  West  East  North
Levels: East < North < West
```

We can construct a data frame from scratch, though, using the data.frame() function. Once a data frame is created, we can add observations to a data frame. We make a little data frame with the names, salaries, gender, and starting dates of a few imaginary co-workers. First, you create three vectors that contain the necessary information like this:

```
> employee <- c('John','Peter','Jolie', 'Sam')
> salary <- c(21000, 23400, 26800, 34000)
> startdate <- as.Date(c('2010-11-1','2008-3-25','2007-3-14', '2009-5-23'))
> gender <- c('F', 'M', 'M', 'F')
> df <- data.frame(employee, salary, startdate, gender)
> df
  employee     salary       s  tartdate       gender
1     John       21000       2010-11-01         F
2    Peter       23400       2008-03-25         M
3    Jolie        26800       2007-03-14         M
4     Sam        34000       2009-05-23          F
```

More examples

Create the vectors for data frame.

```
name <- c('john', 'marry', 'Kevin', 'David', 'Bob', 'Sam', ' Marko')
height <- c(132,151,162,139,166,147,122)
weight <- c(48,49,66,53,67,52,40)
gender <- c("male","male","female","female","male","female","male")

# Create the data frame.
dataframe <- data.frame(name, height, weight,gender)

dataframe
   name   height  weight  gender
1   john      132       48       male
2  marry    151       49       male
3  Kevin     162       66       female
4  David     139       53       female
5    Bob      166       67       male
6    Sam     147       52       female
7    Marko   122        40       male
```

Notice that the function head(dataframe, n=k) and head(dataframe , n=k) can be used to return k top and k bttom rows of Data frame. The default without the second argument returns only 6 rows.

The function str() returns the typeof, mode, and class of the data frame.

```
str(dataframe)
'data.frame':          7 obs. of  4 variables:
 $ name  : chr  "john" "marry" "Kevin" "David" ...
 $ height: num  132 151 162 139 166 147 122
 $ weight: num  48 49 66 53 67 52 40
 $ gender: chr  "male" "male" "female" "female" ...
class(dataframe[, "name"])
[1] "character"
class(dataframe[, "gender"])
[1] "character“
```

We see that the class of both name and gender variables is character. Objects of class factor and character basically differ in the way their values are stored internally. Each element of a vector of class character is stored as a character variable whereas an integer variable indicating the level of a factor is saved for factor objects. Let us explicitly change the gender variable to a factor.

```
gender <- as.factor(gender)
dataframe <- data.frame(name, height, weight,gender)
str(dataframe)
'data.frame':          7 obs. of  4 variables:
 $ name  : chr  "john" "marry" "Kevin" "David" ...
 $ height: num  132 151 162 139 166 147 122
 $ weight: num  48 49 66 53 67 52 40
 $ gender: Factor w/ 2 levels "female","male": 2 2 1 1 2 1 2
```

To access the rows of data frame we use the first indexing and to access to the columns, we use the second indexing.

```
dataframe[2:4,]
   name height weight gender
2 marry    151     49   male
3 Kevin    162     66 female
4 David    139     53 female
dataframe [, 1:3]
    name height weight
1   john    132     48
2  marry    151     49
3  Kevin    162     66
4  David    139     53
5    Bob    166     67
6    Sam    147     52
7  Marko    122     40
dataframe[1:2, 3:4]                    # Data frame with rows of 1 and 2 and with columns 3 and 4.
  weight gender
1     48   male
2     49   male
```

Similarly for non-consecutive rows and columns we use c() function as follows: rows 1 and 7 with columns 2 and 4.

```
dataframe[c(1, 7), c(2, 4)]
  height gender
1    132   male
7    122   male
```

Filtering the people with heights of more than 145 are

dataframe[dataframe$height > 147,]
Notice that the $ sign is used to return the specific column like the height even though one can use dataframe[,2]
i.e., dataframe[dataframe[, 2] > 147,]

```
  name height weight gender
2 marry    151     49   male
3 Kevin    162     66 female
5    Bob    166     67   male
```

Filtering with two statements

```
> dataframe[dataframe$height > 147 & dataframe$weight <60 ,]
  name height weight gender
2 marry    151     49   male
> mean(dataframe$height); var(dataframe$weight)
[1] 146
[1] 95.6
```

For categorical data, it makes no sense to compute means and variances, instead one needs a table indicating the frequencies with which the categories occur.

```
# For factor variable gender we use
table(dataframe$gender)
female   male
     3      4
```

If relative frequencies are desired, there exists the function prop.table()

```
prop.table(table(dataframe$gender))
female           male
0.4285714   0.5714286
```

We can use table for two variables as cross-tabulation function.

```
table(dataframe$gender, dataframe$name)
        Marko  Bob  David  Kevin  Sam  john  marry
 female     0      0       1        1       1      0       0
 male       1      1       0        0       0      1       1
```

Notice that we will see tabulation of more than two variables in later chapters.
To get only the records of male, we use the subset() function which takes data frame and the conditions as arguments.

```
subset(dataframe, gender =='male')
   name   height   weight   gender
1  john      132      48         male
2  marry     151      49         male
3  Bob       166      67         male
4  Marko     122      40         male
```

Finally, one of the most frequently used modes of data is the list. Lists are the most comprehensive way of storing R objects because of accommodating objects of different modes and lengths. Also many functions in R use lists to hold their results. Firstly, we start with a simple type of a list.

```
> list_data <- list('My code', c(201, 302, 151), TRUE, 7.8, 9.43, c('M', 'M', 'F', 'F'))
> list_data
[[1]]
[1] "My code"

[[2]]
[1] 201 302 151

[[3]]
[1] TRUE

[[4]]
[1] 7.8

[[5]]
[1] 9.43

[[6]]
[1] "M" "M" "F" "F"
```

As we see the first element is a character of length one, the second is a numeric of vector of length 3, the third is a logical of length one, the fourth and fifth are numeric numbers of length one, and finally the last is a character vector of length 4. To access of element one can use one bracket or double bracket equivalently.

```
> list_data[1]
[[1]]
[1] "My code"

> list_data[[1]]
[1] "My code“

> list_data[[2]]
[1] 201 302 151
> list_data[[6]]
[1] "M" "M" "F" "F"
```

To access the elements of a vector inside the list we use consecutive brackets as follows with a first double bracket.

```
> list_data[[2]][3]
[1] 151
> list_data[[6]][4]
[1] "F«
```

Naming list elements

Create a list containing a vector, a matrix and a list.

```
list_dat <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2), list("green",12.3))
```

Give names to the elements in the list.

```
names(list_dat) <- c("1st Quarter", "A_Matrix", "A_Inner_list")
```

Show the 3 different elements of the list.

```
list_dat[1]
$`1st Quarter`                  # The name of the first element of the list
[1] "Jan" "Feb" "Mar“           # The contents of the first element of the list

list_dat[2]
$A_Matrix                       #  The name  of the 2nd  element of the list
     [,1] [,2] [,3]             #  The rows of the 2nd element
[1,]    3    5   -2
[2,]    9    1    8

List_dat[3]
$`A_Inner_list`                 # The name of the 3rd element

$`A_Inner_list`[[1]]            # The 1st data in the 3rd element
[1] "green"
```

For last example of a list we create one which has list, matrix, and data frame among other things.

```
> L <- list(a= 3, b = 8.8, c=c('M', 'F', 'M'), d = matrix(1:9, ncol=3),
+         e = list(TRUE, 7.8, 'M', 1+3i), f= factor('M', 'M', 'M', 'F', 'F'),
+         dataframe)



> is.list(L)
[1] TRUE
> is.double(L[[1]])
[1] TRUE
> is.numeric(L[[2]])
[1] TRUE
> is.character(L[[3]])
[1] TRUE
> is.matrix(L[[4]])
[1] TRUE
> is.list(L[[5]])
[1] TRUE
> is.factor(L[[6]])
[1] TRUE
> is.data.frame(L[[7]])
[1] TRUE
```

Repeat loop

Here we explain the repeat loop with an example namely the generating of the Fibonacii numbers.

```
> f <- c(1, 1)
> i <- 1
> repeat {f[i+2] <- f[i+1] + f[i]          #Fibonacii
+ i <- i+1
+
+ if (i > 23){
+   break
+ }
+ }
```

First of all, we define the vector f with the first and the second Fibonacci number. Secondly, the index i. These two are out of the loop. Inside the loop we have main recurence function which takes the ith and (i+1)th elements of the vector f and calculat the (i+2)th element by sumning up the first tow. From i = 1 we get f[3] <- f[2]+f[1]. Then we increament i by 1 to get 2. This process will be repeated untill the condition i > 23 satify where the process breaks and ends the loop. To get the generated numbers we type f as

```
> f
 [1]    1    1    2    3    5    8   13   21   34   55   89  144  233
[14]  377  610  987 1597 2584 4181 6765 10946 17711 28657 46368 75025
```

For any arbitrary n one can replace 23 by n.

For loop

Let us explain for loop to sum up numbers from 1 to 100.

```
> sum <- 0
> for(i in 1:100) {
+   sum = sum + i
+
+ }
```

We first define sum <- 0. Then the simplest for loop has for (the range that loop should take) and the expression we want to calculate sum <- sum + i . Then type sum to get the results.

```
> sum
[1] 5050
```

Remark: Gauss trick. When Gauss was in the first grade, one day the teacher asked the class to sum up the same numbers to keep them busy for an hour or so. Gauss paired the numbes as (1,100), (2, 98), ..., (50, 51). Each pair add up to 101 and there are 50 pairs, so 50*101=5050

Exercise. Similarly by changing i to i^2, i^3, etc. One gets sum of squares, sum of cubes, etc.

While loop

While loop is very simple. While some contion (i < 100) is still true, then do the expression here print(i).

```
> i <- 1
> while (i < 100) {
+   print(i)
+   i <- i+23
+ }
[1] 1
[1] 24
[1] 47
[1] 70
[1] 93
```

One can do non-numeric looping with while loop.

```
> i <- 1
> v <- c('a', 'b', 'c', 'g' )
> while (v[i]!='g') {
+   print(v[i])
+   i <- i+1
+ }
[1] "a"
[1] "b"
[1] "c"
```

(ii) Histogram and density

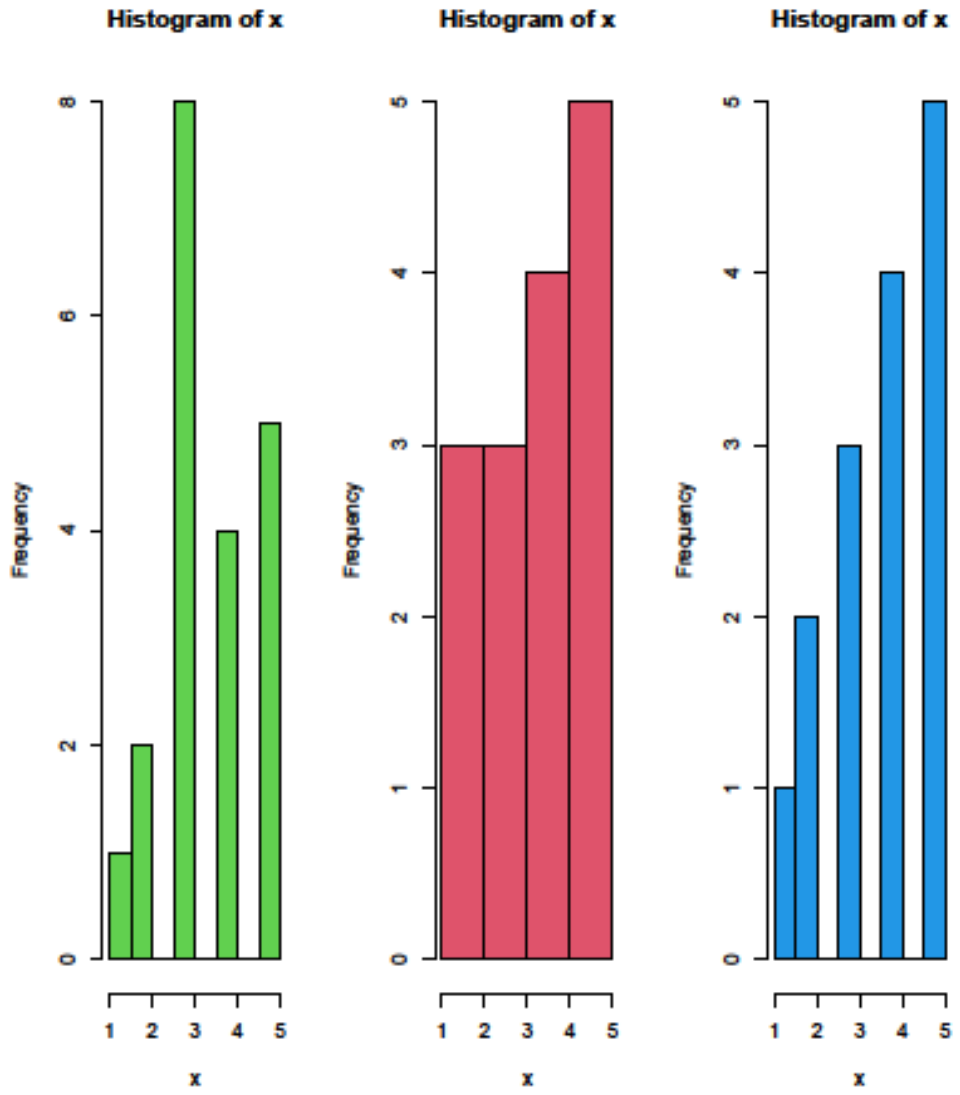
One of the best ways to describe a variable is to report the values that appear in the dataset and how many times each value appears. This description is called the distribution of the variable. The most common representation of a distribution is a histogram, which is a graph that shows the frequency of each value. In this context, “frequency” means the number of times the value appears. Another way to represent a distribution is a density or probability mass function (pmf), in the case of discrete variable and the probability density function (pdf) in the continuous variable. This function maps from each value to its probability. A probability is a frequency expressed as a fraction of the sample size, n. To get from frequencies to probabilities, we divide through by n, which is called normalization. These problems can be reduced by breaking the data; namely, dividing the range of values into non-overlapping sub-intervals and counting the number of values in each subinterval.

Histograms

Every numerical variable including the vectors, rows and columns of a matrix , as well as the columns of data frames can be plotted by a histogram plot , a simple discrete distribution of the variable. In this plot, the values on the x axis are the unique values of the corresponding variable and the values on the y axis is the frequency of each value, or count of how many observations fall within specified divisions of the x-axis .

Example 1.

```
x <- c(1,2,2,3,3,3,3,3,3,3,3,4,4,4,4,5,5,5,5,5)
hist(x)
```



As we see in the right figure, the height of the rectangles are exactly equal to the count of the numbers in the vector x. However, this can't happen for every vector. For an example, let

```
x <-c(1,2,2,3,3,3,4,4,4,4,5,5,5,5,5)
hist(x)
```

To get the correct histogram we use the breaks = 6.

```
hist(x, breaks =6)
```

Notice that in all three histogram the y-axis is frequency.

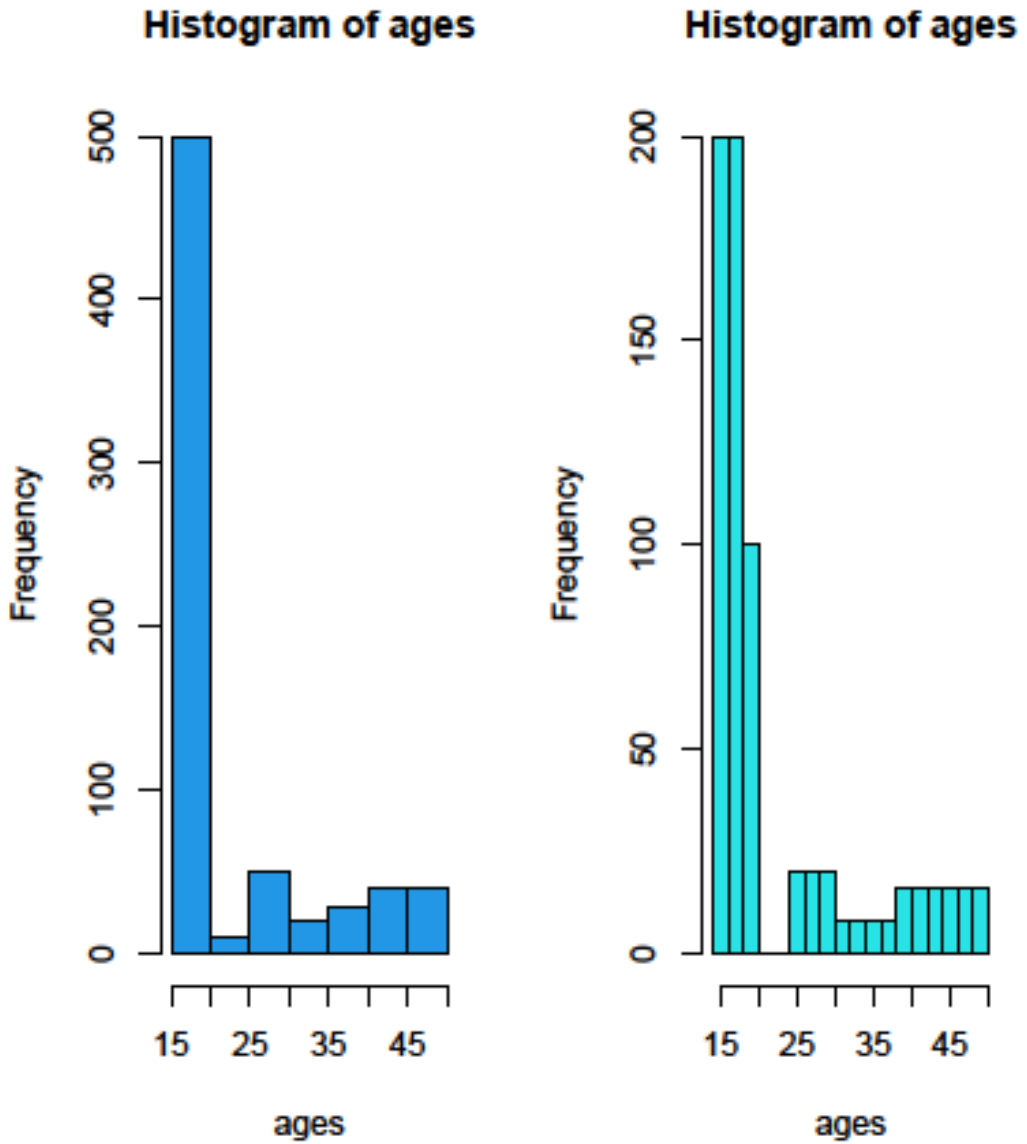
Example2

Suppose that the ages vector shows the ages of students, staff, and teachers in a high school.

```
ages <- c(rep(15:19, 100), rep(25:30,10), rep(31:40, 4), rep(40:50, 8))
hist(ages, col = 5)
```

By choosing breaks=n in the hist call, one gets approximately n bars in the histogram.

```
hist(ages, breaks=15, col = 6)
```

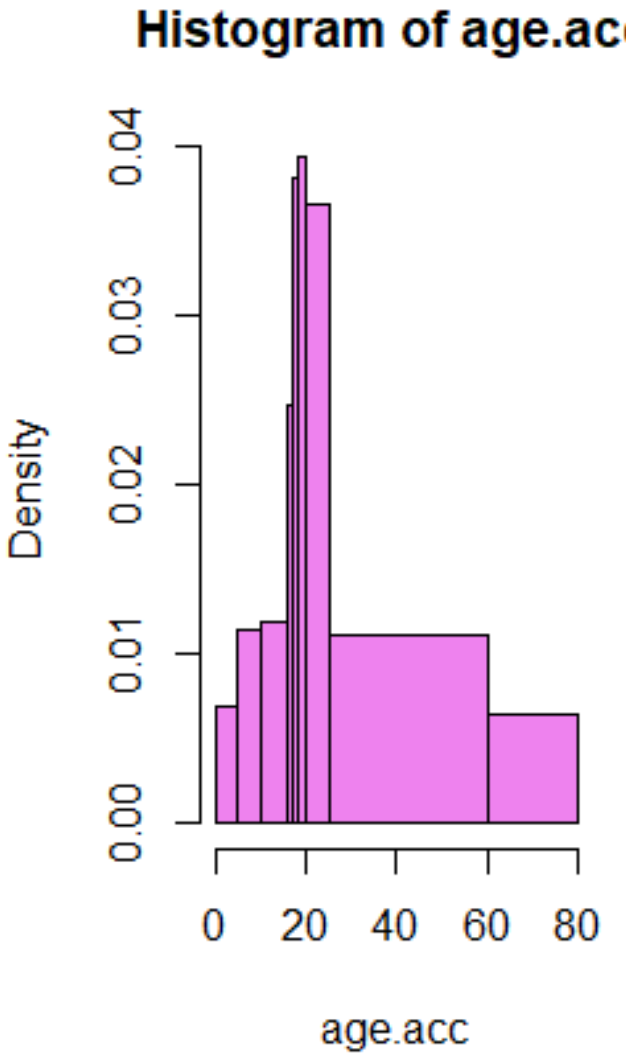


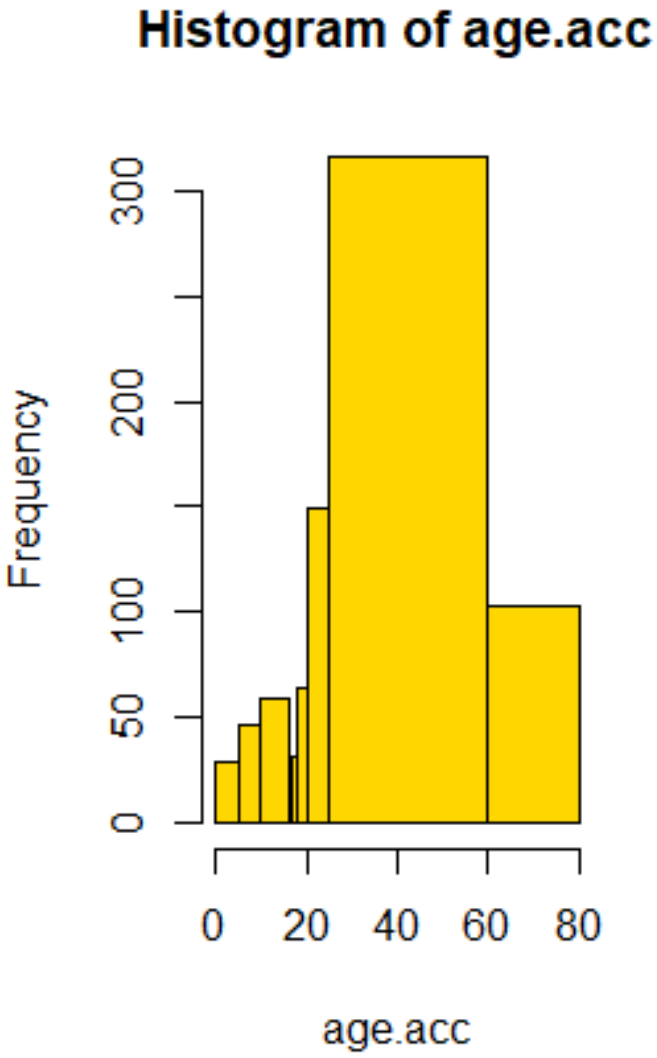
We can also choose the breaks for each unique observation where in this case the breaks is given by a vector.
There is an example of accident rates by age group, see Altman (1991, pp. 25–26),
0–4, 5–9, 10–15, 16, 17, 18–19, 20–24, 25–59, and 60–79 years of age. The data is as follows:

```
mid.Age <- c(2.5,7.5,13,16.5,17.5,19,22.5,44.5,70.5)
acc.count <- c(28,46,58,20,31,64,149,316,103)
age.acc <- rep(mid.age,acc.count)
br <- c(0,5,10,16,17,18,20,25,60,80)
br <- c(0,5,10,16,17,18,20,25,60,80)
hist(age.acc,breaks=br, col= ‘violet’)
```

As we see the y axis got the label of density units instead of frequency. These are the area of columns proportional to the frequency numbers. The total area of the histogram equals to 1. The density units for a breaks vector is a default option. To get the frequency, one can add the option freq = TRUE as follows:

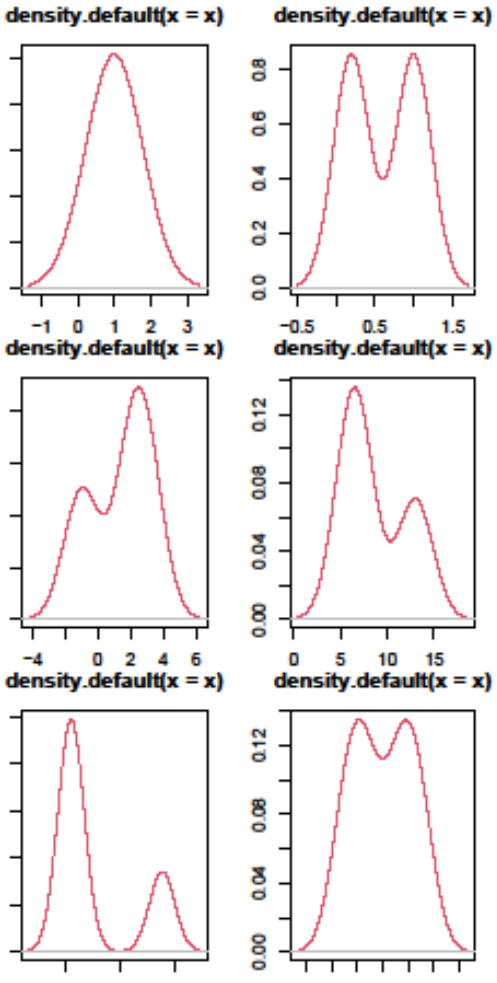
```
hist(age.acc,breaks=br, freq = T, col=‘gold’)
```





Density function

The density is proportional to the chance that any value in data is approximately equal to that value. In fact, for a histogram, the density is calculated from the counts, so the only difference between a histogram with frequencies and one with densities, is the scale of the y-axis. The density object is plotted as a line, with the actual values of data on the x-axis and the density on the y-axis. For discrete distributions, the density is used for the point probability - the probability of getting exactly the value x. It is a density with respect to the counting measure. On the other hand, the density for a continuous distribution is a measure of the relative probability of getting a value close to x. The probability of getting a value in a particular interval is the area under the corresponding part of the curve. Par() function with mfrow = c(m, n) splits the graphical platform into m rows and n columns, so here we have m = 2 and n = 3. Instead of mfrow one can use mfcol = c(1,2).

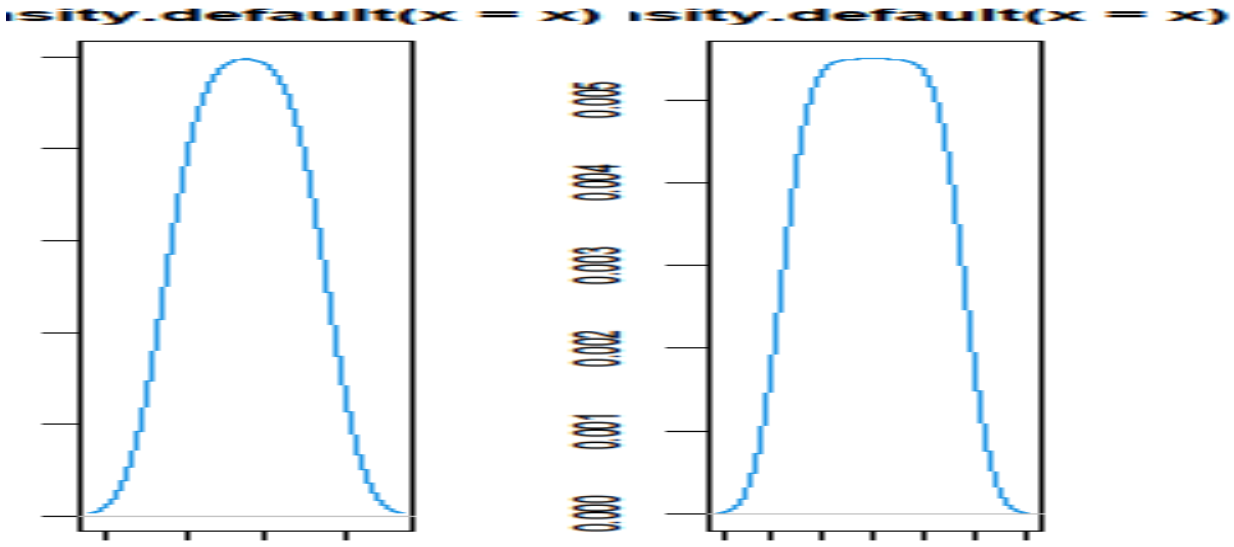


Example 1: discrete cases

```
par(mfrow=c(2,3))
x <-  c(1, 1)
plot(density(x), col=2)
x <-  c(1, 0.2)
plot(density(x), col=2)
x <-  c(-1, 2, 3)
plot(density(x),col=2)
x <-  c(13,6,7)
plot(density(x), col=2)
x <- c(0,6,7,89)
plot(density(x), col=2)
x <- c(3,5,7,9,8,4,3,5,7,4,8,9)
plot(density(x), col=2)
```

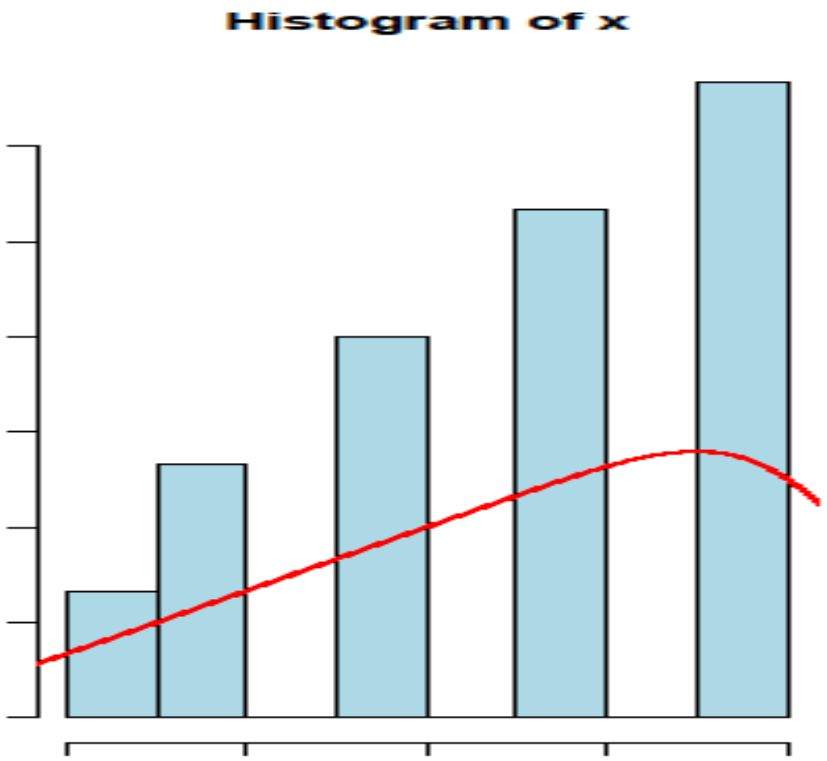
Examples for continuous data

```
par(mfrow=c(1,2))
x <-  seq(-2, 1,by = 0001)
plot(density(x), col=4)
x <-  seq(-90, 90, by = pi)
plot(density(x), col=4)
```



For plotting the histogram and density together, we use the argument freq= FALSE in the hist call and lines in the density plot.

```
par(mfrow=c(1,1))
x <-  c(1,2,2,3,3,3,4,4,4,4,5,5,5,5,5)
hist(x, freq = FALSE, breaks = 6, col='lightblue')
lines(density(x), col='red', lwd=2)
```

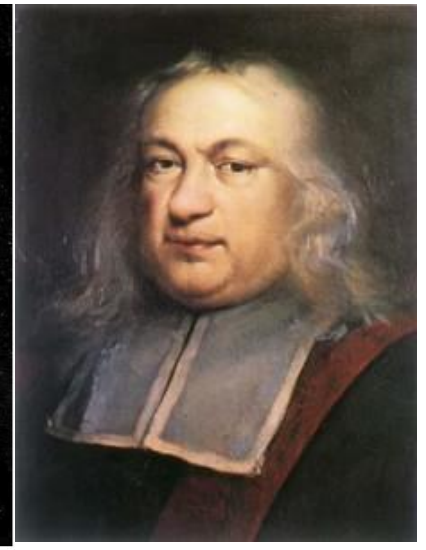


1.2 History

Statistics started in 5th century BC and used for demographic and financial data.

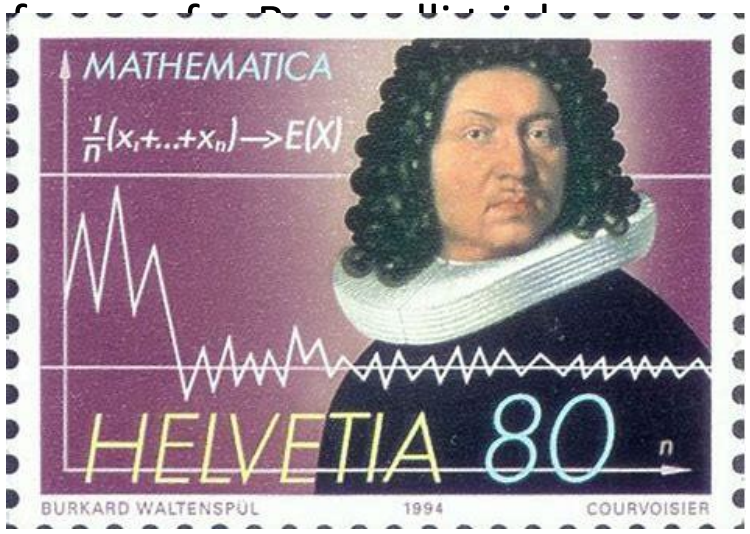
Pascal and Fermat, the Fathers of Probability Theory invented gambling problem related to the expected outcomes.

Blaise Pascal



Bernoulli, a 17th-century

Swiss mathematician is also worked on probability theory.



viii CONTENTS

5 The Foundation of Probability Theory by Pascal and Fermat in 1654 **42**

5.1 Pascal and Fermat, 42
5.2 Pascal's Arithmetic Triangle and Some of Its Uses, 45
5.3 The Correspondence of Pascal and Fermat and Pascal's Treatise on the Problem of Points, 54
5.4 Pascal's Wager, 63

In 19th century the data collection and analysis were applied in general.

1.3 Populations, samples, parameters and statistics

This section, deals with the some of the basic principles and definitions including the distinction between populations and samples as well as the distinction between parameters and statistics. Population is the complete set of elements under study which shares some common characteristics, while sample is some selected subset of the population and a random sample is selected so that every element in the population has an equal chance of being included. When we select the values of a samples randomly, then the corresponding variable is called random variables. Hypothesis of random sampling may not apply to real data. For example, during the summer, hot days are usually followed by hot days. So daily temperature not directly representable by random drawings. Suppose that we wish to study the age, height, and weight of the people living in the city of Mississauga ON Canada. Each of these three characteristics of the people individually can be considered as a population. That is, ages of all people, heights of all people and wights of all people. So we have three different populations. On the other hand, suppose we ask the ages of all people present in a city park on a sunny day. The set of age of these people is a random sample of the all ages. The same are true for both heights and weights. Next suppose that we wish to know the average or mean value of the ages in the park. Then we simply sum up all ages and divide by the number of people to get the mean. This number which is the characteristic of the sample is called a statistic. To get the mean value of the ages of all people in the city, we may collect the data from the census office. If this is possible, we can sum up all the ages again and divide by the number of people to get the mean. This number is called the parameter of the age population. Most often, the parameters of a population is not applicable or difficult to collect . As an example, let us suppose that we wish to measure the diameters of all trees in Amazon jungle. Clearly it is almost impossible or very difficult task both in terms of necessary logistics and the time. However, to do this on an easily accessible and restricted region may be a practical task.

Statisticians and computer scientists often use different language for the same thing. Here is a dictionary that the reader may want to return to throughout the course.

Statistics	Computer Science	Meaning
estimation	learning	using data to estimate an unknown quantity
classification	supervised learning	predicting a discrete Y from $X \in \mathcal{X}$
clustering	unsupervised learning	putting data into groups
data	training sample	$(X_1, Y_1), \ldots, (X_n, Y_n)$
covariates	features	the X_i's
classifier	hypothesis	a map from covariates to outcomes
hypothesis	—	subset of a parameter space Θ
confidence interval	—	interval that contains unknown quantity with a prescribed frequency
directed acyclic graph	Bayes net	multivariate distribution with specified conditional independence relations
Bayesian inference	Bayesian inference	statistical methods for using data to update subjective beliefs
frequentist inference	—	statistical methods for producing point estimates and confidence intervals with guarantees on frequency behavior
large deviation bounds	PAC learning	uniform bounds on probability of errors

1.4 Types of sampling

1. Convenient sampling: Use the results that are easy to get – not random. In the example of the people of the city of the Mississauga, we select all people in a shopping mall on Saturday and ask the height of everyone.

2. Systematic sampling: Put population in some order and select every 13-th member. In a busy street in the city, we start from a person we see, then count till to select 13^{th} person and continue in the same fashion until get the necessary number of people that we wish and record their weights.

3. Stratified sampling: Break population into subgroups based on a character, then sample each subgroup. In this case we divide the population in a city park into kids under 13 years old, teenagers, mothers and fathers, then we pick up samples from each group and the record the ages of the all groups.

4. Cluster sampling: Divide population into clusters regardless of characteristics and randomly select a certain number of clusters, then collect data from the entire cluster. We can consider schools, staffs in big companies, staffs in government buildings, peoples in shopping malls, people in parks. Then we select some of these clusters and pick up the ages of all the people in selected clusters.

In July 30, 2020 the population of the city of Mississauga was 801 877. Using this number we want to simulate ages, weights, and heights of all people function. We sample from this population three variables age, weight, and height. To do this we use the R coding. We have previously introduced the sample() function.
The age can take values from 1 to 120 years and replace=TRUE means that these ages may repeat. The same is true for weight from 10 to 100 kg as well as height from 50 to 200 cm. Even the variables can take slightly different values, but the number of observations for all three variables are the same as 801 877.

```
age  <- sample(1:120, 801877,  replace = T)
weight <- sample(10:100, 801877, replace=T)
height <- sample(50:200, 801877, replace=T)
df  <- data.frame(age, weight, height)
```

Let see a small subset of data frame df.

```
> df[30:37, ]
   age weight height
30  37     61    152
31  24     75     50
32 103     48    145
33  55     58    150
34 117     44    119
35  10     44    164
36  21     91    154
37  55     58     64
```

Next, we want to take four different samples from this population. First the convenient sampling. To get this, we select the first 1000 people.

```
df[1:1000, ]
```

Returns the required sample as the rows varies from 1 to 1000 with all columns.

Secondly, for the systematic sampling we select the first and every 14th person in the street and ask the three statistics of all and combined the values.

```
b <- c()
x <- 1:1000
count <- 1
for (val in x) {
  if (count <= 1000*13 )
    b <- c(b, df[count, ])
    count = count+13 }
print(b)
```

For this code, we first define an empty vector b and the vector x consisting 1000 consecutive natural numbers. Secondly, starting from count=1, we add df[1,] that is the first row with count = 1 which satisfy the inequality condition. Thirdly, we add df[1,] to the vector b with the c() function. In the next step we add 13 to the count and then repeat the process again until the count become less than or equal to 13000. The print(b) returns all the 1000 people.

By examining typeof, mode, and class of b, we see that all are lists. We can also get the first 9 and last 9 elements by providing their corresponding indices. Notice that list elements are returned in a vertical format. To get the elements in a horizontal format, we use the transpose function t() to get them in a readable format.

```
> typeof(b); mode(b); class(b)
[1] "list"
[1] "list"
[1] "list“

> t(b[1:9]); t(b[992:1000])
     age weight height age weight height age weight height
[1,] 53  44     101    15  95     76     16  96     73
     weight height age weight height age weight height age
[1,] 51     161    40  40     161    70  98     196    88
```

Notice that instead of vectors and lists , we can work with data frame. Then the code is as follows:

```
> dg <- data.frame()
> x <- 1:1000
> count <- 1
> for (val in x) {
+   if (count <= 1000*13 )
+     dg <- rbind(dg, df[count, ])
+   count = count+13}
```

In this case we create an empty data frame and then add the corresponding rows by using rbind() function which we have seen it for matrices.

Thirdly, for the stratified sampling, we break population into subgroups based on ages, then sample each subgroup. and take the 250 first elements from each group. To this end, we first use the subset() function which we have previously introduced for sub-setting a vector,

```
df1 <- subset(df, df$age <=12)
df2 <- subset(df, 13<=df$age & df$age<20)
df3 <- subset(df, 20<=df$age & df$age<60)
df4 <- subset(df, 60<=df$age & df$age<=120)
```

We can check the number of rows by dim() function which in fact returns both of the number of rows and columns.

```
> dim(df1); dim(df2);dim(df3);dim(df4)
[1] 81016       3
[1] 47056       3
[1] 267153      3
[1] 406652      3
```

Then we take sample from rows as

```
shuffle1 <- df1[sample(nrow(df1)),  ]
shuffle2 <- df2[sample(nrow(df2)),  ]
shuffle3 <- df3[sample(nrow(df3)),  ]
shuffle4 <- df4[sample(nrow(df4)),  ]
```

where nrow(df) means the number of rows of the data frame df. In fact, the sample() function do nothing here but mixes up the rows only. The last step is to combine 250 elements from each sample to get the 1000 stratified sampling.
c(shuffle1[1:250,], shuffle2[1:250,], shuffle3[1:250,], shuffle4[1:250,])

Finally, for the cluster sampling: we divide population into clusters regardless of characteristic and randomly select a certain number of clusters, then collect data from the entire cluster.
We sample 250 indices from each 100000 consecutive rows up to 800000 and 250 from the 1876 remaining.

```
> s1 <- sample(1:100000, size = 250);          s2 <- sample(100001:200000, size = 250)
> s3 <- sample(200001:300000, size = 250);     s4 <- sample(300001:400000, size = 250)
> s5 <- sample(400001:500000, size = 250);     s6 <- sample(500001:600000, size = 250)
> s7 <- sample(600001:700000, size = 250);     s8 <- sample(700001:800000, size = 250)
> s9 <- sample(800001:801877, size = 250)
```

Then from these indices we obtain 9 data frames as

```
> dh1 <- df[s1, ];   dh2 <- df[s2, ]
> dh3 <- df[s3, ];   dh4 <- df[s4, ]
> dh5 <- df[s5, ];   dh6 <- df[s6, ]
> dh7 <- df[s7, ];   dh8 <- df[s8, ]
> dh9 <- df[s9, ]
```

From these data frames we randomly select 4 data frames and rbind them to get a data frame with 1000 rows

```
> sample(c('dh1', 'dh2','dh3','dh4','dh5','dh6','dh7','dh8', 'dh9'), 4)
[1] "dh2" "dh5" "dh8" "dh9"
> s <- rbind(dh2,dh5,dh8, dh9); dim(s)
[1] 1000    3
```

Finally the head() and tail() functions return the first 6 and the last 6 rows respectively. However we can get any rows by providing head(s, n=10) and tail(s, n=20) to get any number of rows.

```
> head(s)
       age weight height
165835  37     18     95
124975   8     94    192
181102  25     81    113
179797 119     68    198
103325  75     75     61
116225   7     23    169
> tail(s)
       age weight height
801542  95     89     69
801817  29     12    113
800462 117     32     98
801356  89     35     88
801105  95     55    124
800803  39     77    173
```

Reading different data files

To read files such as txt, csv, and in general data frames stored as different files there are appropriate commands or functions to use . These functions take, optionally various parameters additional to the file name that holds the data. Specify Header = TRUE if there is an initial row of header names. The default is header = FALSE. In addition one can specify the separator character or characters. Command alternatives to the default use of a space are sep="," and sep="\t". This last choice makes tabs separators. Similarly, one can control over the choice of missing value character or characters, which by default is NA. If the missing value character is a period (“.”), specify na.strings=".". There are several variants of read.table() that differ only in having different default parameter settings.

Note in particular read.csv(), which has settings that are suitable for comma delimited (csv) files that have been generated from Excel spreadsheets. read_csv() and data.table::fread() are useful for reading big data frames. There is another function read.csv2 which is appropriate for files having semicolon as separator, sep= ‘;’. Read.delim and read.delim2 are two other functions for reading files. One useful argument is stringersAsFactor = TRUE which make the character variables as factors. To see the other arguments of either of the functions type ?read.csv in the R console.

Chapter 2
Variables, constants and their different types
2.1 Variables and constants

As it was stated earlier, statistics deals with the data around us and the data in turn comes from real world problems that we wish to find reasonable answers. The data is pretty much anything that we encounter in our daily lives. It ranges from age, height, weight, income, rank on a race which are numbers to gender, attitudes about the work and scores of happiness or depression. If we pick up a person, his/her age, weight and height are all specific numbers which are simply called constants, while these values for a group of people take different values are called variables. For example, the age variable can be ranged from 1 to 120 years. But the gender has only two values, male and female. If one talk about only the male population, then its value is a constant. The other examples are the geometric number pi, the ratio of a circle to its diameter, Newton's gravitational constant g, the base of natural logarithm e. Variables are in two types, dependent and independent variables. For example, the income of a person is an independent variable which depends on the level of the education and the years of the experience of a person. Then the experience and the level of education are independent variables. Variables can take on number of different forms, determined by the nature of the values they may take. In the next sections, we study these forms and give many example to differentiate among them.

2.2 Qualitative and quantitative variables

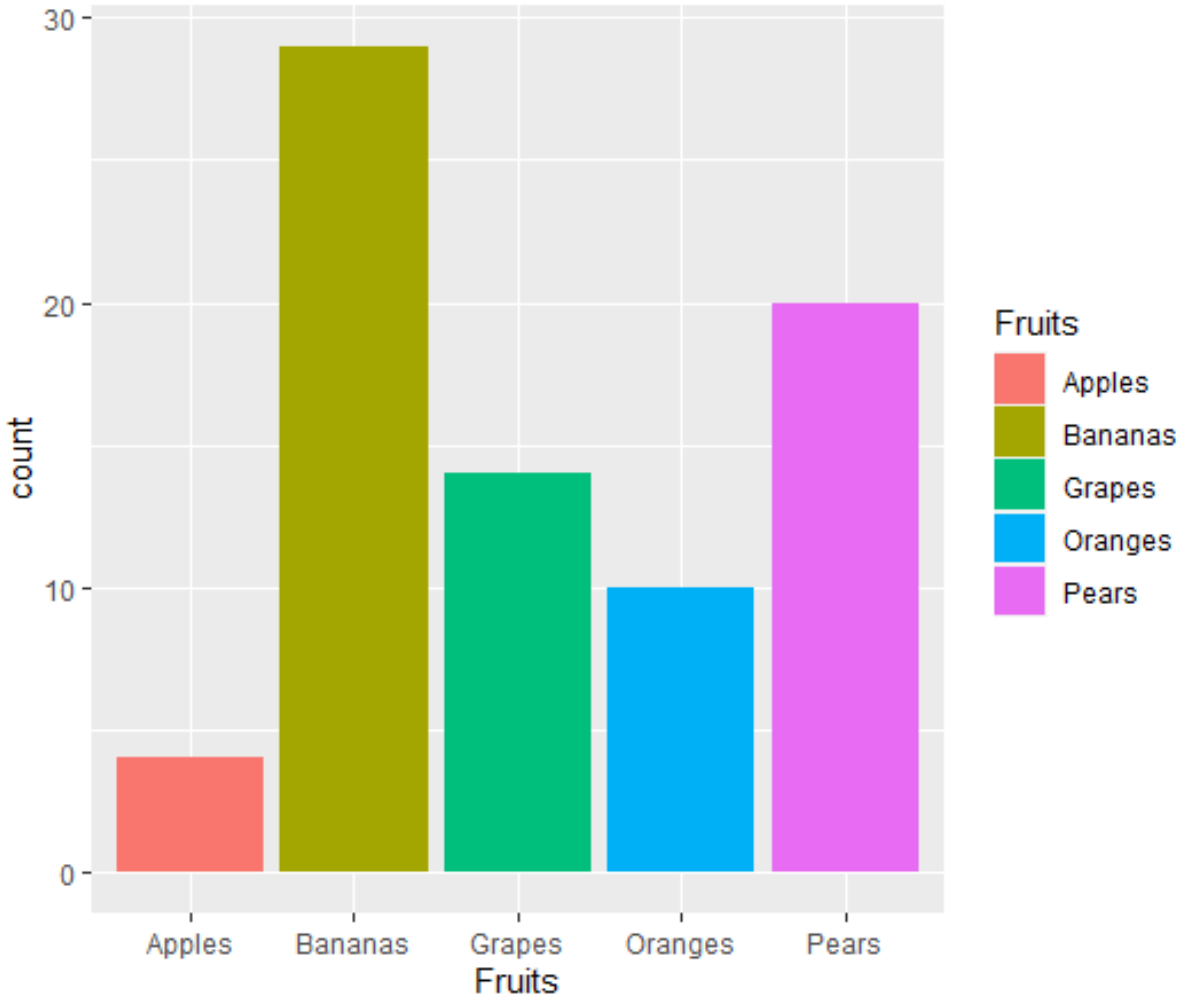
In general, Both variables and constants are two types, quantitative and qualitative (or categorical).
Qualitative variables or categorical which are non-numeric such as gender, color, ranks in a running race, names, religion and zip codes where the mathematical operations are meaningless. All qualitative variables are discrete.
Quantitative variables are numeric such as the number of atoms, wages, heights, weights, temperature, time, speed, distance where the mathematical operations are meaningful.

2.3 Level of measurments

There are five levels of basic measurement scales from which two of them are qualitative. They are:

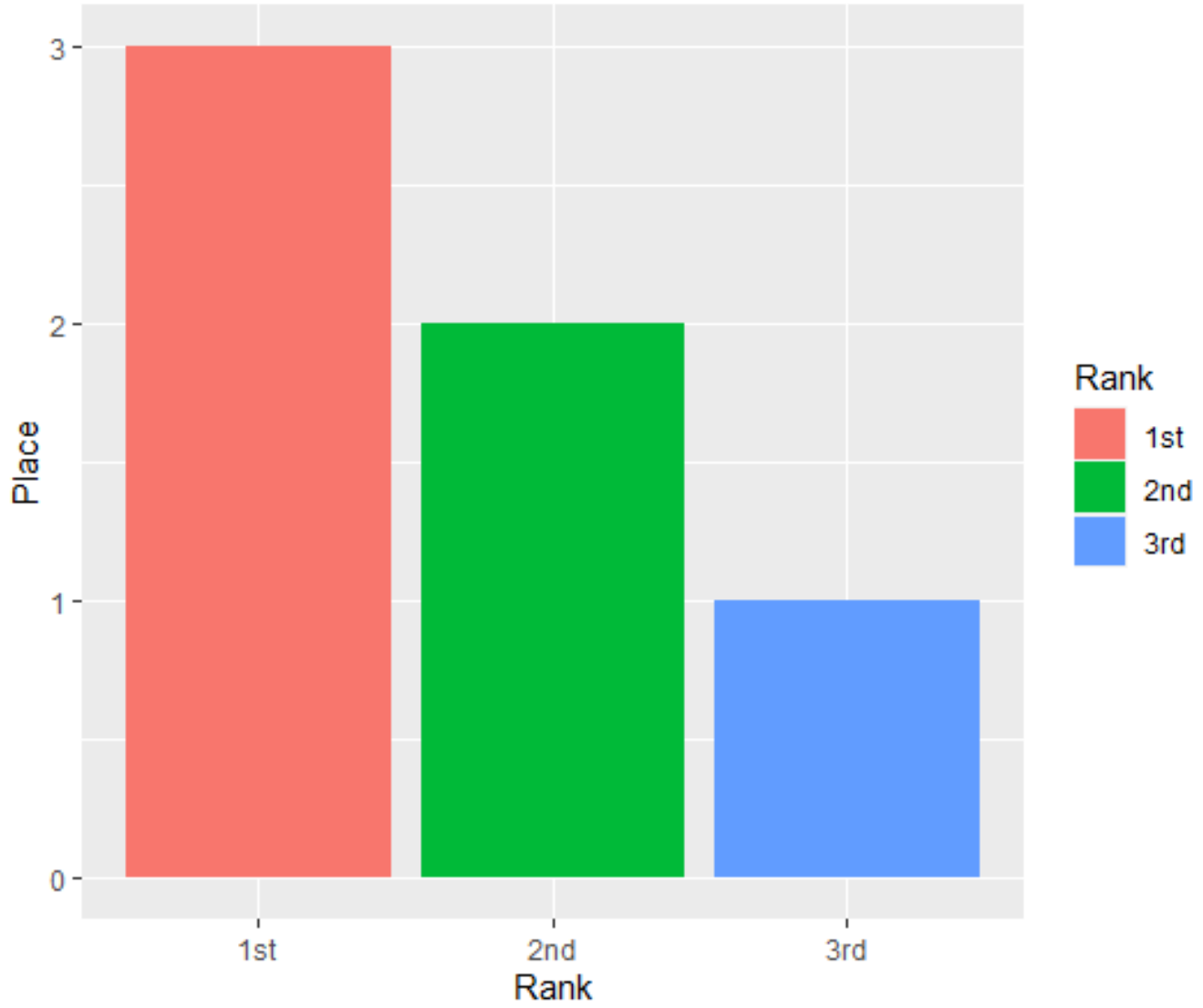
a. Nominal - categorical variables which are not ordered such as gender and religion. One can assign numbers like male = 1 and female = 2 which has no real meaning other than differentiating between objects. In other words, they are just a names or labels and we need them to do computations and conduct statistics

```
> library(ggplot2)
> df <- data.frame(
+   Fruits=c('Apples', 'Oranges','Bananas','Grapes','Pears'),
+            count=c(4, 10, 29, 14, 20))
> df
   Fruits count
1  Apples     4
2 Oranges    10
3 Bananas    29
4  Grapes    14
5   Pears    20
> p<-ggplot(data=df, aes(x=Fruits, y=count, fill=Fruits)) +
+   geom_bar(stat="identity")
> p
```



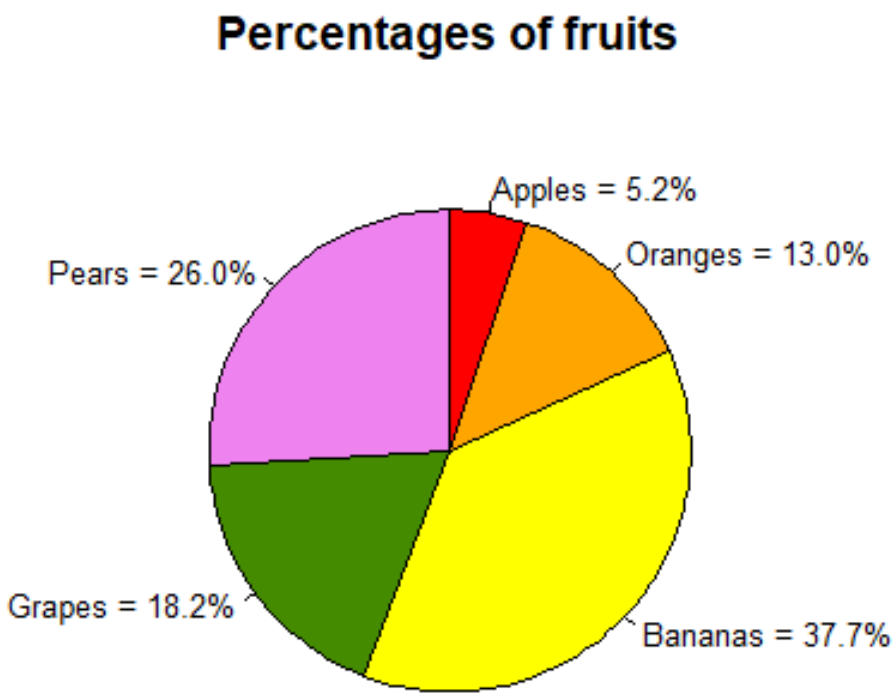
b. Ordinal - variables which can be ordered but the differences are meaningless such as ranks in a running race. 1st, 2nd, 3rd. These three ranks are different, but they don't tell us how much distance there is between each rank.

```
library(ggplot2)
df <- data.frame(Rank=c("1st","2nd", "3rd"),
Place=c(3,2,1), fill=Rank)
> df
  Rank Place
1  1st     3
2  2nd     2
3  3rd     1
> p <-ggplot(data=df, aes(x=Rank, y=Place)) +
+   geom_bar(stat="identity")
> p
```



Pie chart for nominal data with percentage

```
> df <- data.frame(
+   Fruits=c('Apples', 'Oranges',
'Bananas','Grapes','Pears'),
+   count=c(4, 10, 29, 14, 20))
> dj <- df
> labels <- sprintf("%s = %3.1f%s", dj[,1],
+                   100*dj[,2]/sum(dj[,2]), "%")
> pie(dj[,2],
+     labels=labels,
+     clockwise=TRUE,
+     radius=0.8,
+     col=c('red','orange','yellow','chartreuse4','violet'),
+     border="black",
+     cex=0.8,
+     main="Percentages of fruits")
```



The other level of measurements are quantitative which are called interval, ratio. and absolute.

c. Interval – variables which they can be ordered and the differences are meaningful such as temperature in degrees Fahrenheit: differences between 65 and 64 is the same as 43 and 42 but no natural zero, or the difference between −20 and 20 is 40 but the ratio of 40/−20 = −2 does not mean that −40 is twice as cold as 20. (zero does not denote absence: zero degrees- no true zero point.) As a second example, let a person's weight is 70 lb, another is 76 lb, and a third is 80 lb, we know who is heaviest and how much heavier each person is in relation to the others. That is 70 < 76 < 80.

d. Ratio - it is like an interval and there is also a true natural zero such as height: 100/4 = 25 in meter, i.e.,
100 is 4 times of 25. Zero height like ground zero, zero growth - it does denote absence.
There is often one more level of measurements that differ from the above four cases.

e. Absolute - the absolute scale is the same as the ratio scale, with the exception
that the values are measured in “natural” units. An example is number of seasons of a year. There is no unit scale like meters or degrees, we have absolute natural numbers 1, 2, 3, and 4.

2.4 Discrete and continuous quantitative variables

Quantitative variables are numerical variables where the mathematical operations are meaningful and they appear in two forms: continuous and discrete variables. Continuous variables such as wages, heights, weights, temperature, time, speed, distance can have infinitely many decimal digits besides the integer part. In contrast, the number of people in a city, number of countries as well as number of universities on the globe are all discrete quantitative variables. Informally, one can say that discrete variable is a variable which is related to counting something rather than measuring while continuous variable is infinite and impossible to count. Discrete quantitative variable also arise into two different formats: finite discrete and infinitely many discrete variables

1 . Discrete quantitative variables

(a) Finite discrete variables: which can be finite like the faces of any regular 3-dimensional solids except the sphere.

(b) Infinitely countable discrete variables:

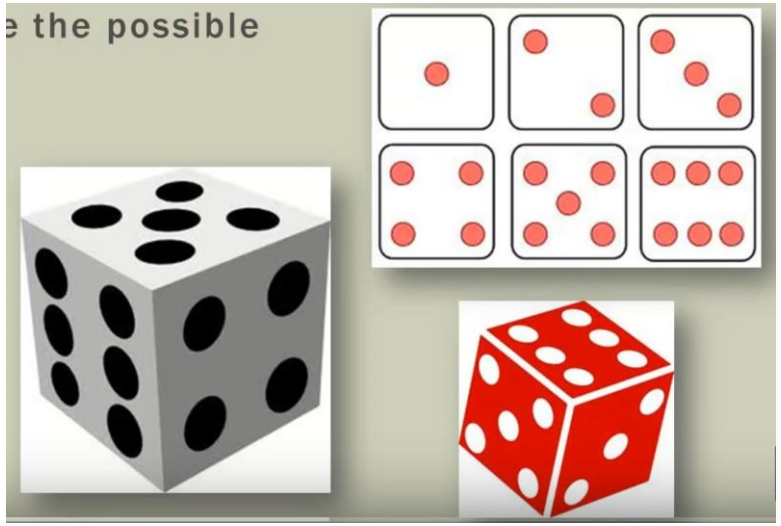
The counting numbers N = {1, 2,3, …} or integers

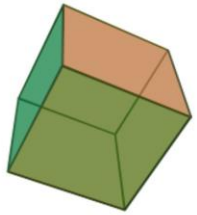
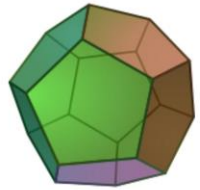
Z = {…, -3, -2, -1, 0, 1, 2, 3, …} are two mathematical set having infinitely many discrete numbers.

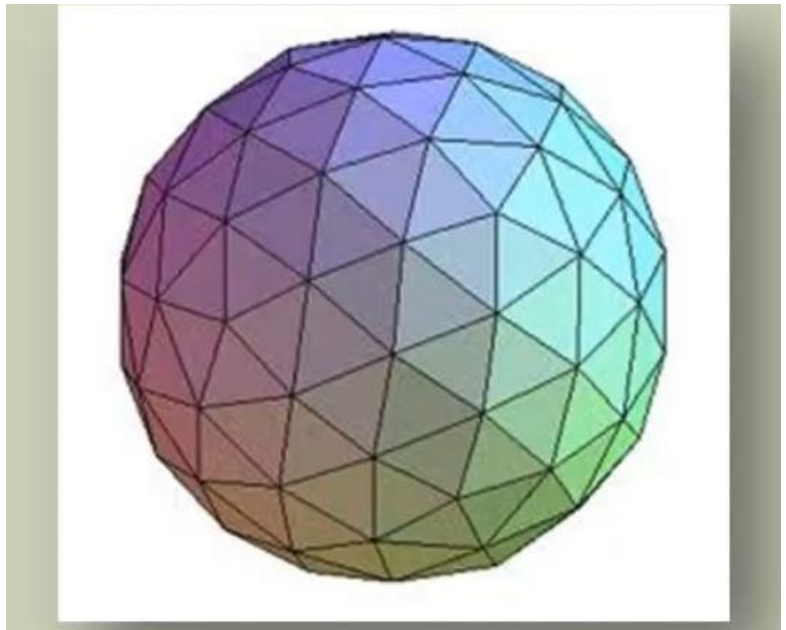
Question? Is there anything in the real world as a real subject that is infinitely countable? How about the particles in whole univers?

Two types of quantitative data:

1. (a) Discrete data which can be finite like the faces of any regular 3-dimensional solids except the sphere.



Tetrahedron	Cube	Octahedron	Dodecahedron	Icosahedron
Four faces	Six faces	Eight faces	Twelve faces	Twenty faces
				
(Animation) (3D model)	(Animation) (3D model)	(Animation) (3D model)	(Animation) (3D model)	(Animation) (3D model)



<---- 100 faces

10000 faces ---->

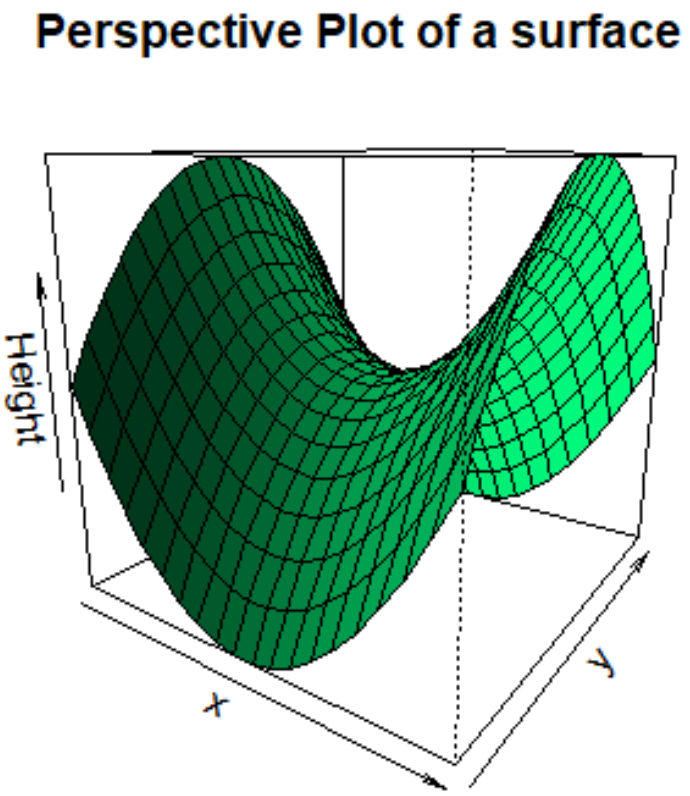
The commonly accepted answer for the number of particles in the observable universe is 10^80. This number would include the total of the number of protons, neutrons, neutrinos and electrons. So the particles is also countably finite.

2. Continuous quantitative variables. Continuous variables can have infinitely many decimal digits in addition to their integer parts. In other words, they are uncountably infinite.

The number of the digits in pi and e are infinitely many. Other examples are 40.78 lb weight, 1.56 ft height and $6500.65 income.

Notice that it is impossible to write down the infinite digits of a continuous variable into a paper or a computer. The values can be any real number in the set R = (-infinity, +infinity). The whole points of a surface may be another example for continuous data.

```
> h <- function(x, y){
+   (x^2-y^2)
+ }
> x <- y <- seq(-1, 1, length= 20)
> persp(x, y, z,
+       main="Perspective Plot of a surface",
+       zlab = "Height",
+       theta = 35, phi = 15,
+       col = "springgreen", shade = 0.5)
```



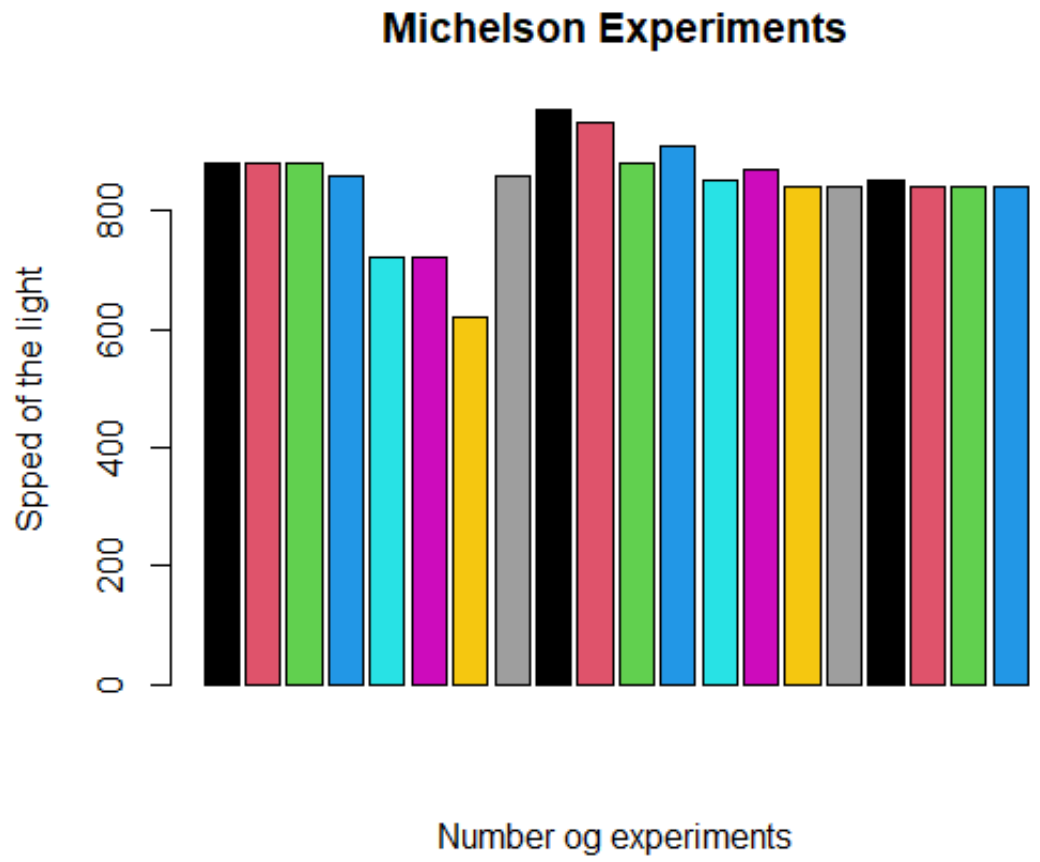
How about the numbers of the golden stair arcs?

Michelson speeds of the light via bar plot

The data frame Michelson gives Albert Michelson's measurements of the velocity of light in air, made from June 5 to July 2, 1879, reported in Michelson (1882). The given values + 299,000 are Michelson's measurements in km/sec. The number of cases is 100 and the "true" value on this scale is 734.5. The data is a part of the package HistData.

Type ??Michelson for more information in the R console.

```
> library(HistData)
> ??michelson
> data(michelson)
> head(michelson, n=3)
  Speed Run Expt
1   850   1    1
2   740   2    1
3   900   3    1
> unique(michelson$Expt)   # Expt for Experiment
[1] 1 2 3 4 5
Levels: 1 2 3 4 5

> # Run variable = The number of measurement in each Experiment
> Expt3 <-   michelson$Speed[michelson$Expt == 3]
> barplot(Expt3, col=1:20, xlab = "Number og experiments",
+        ylab ="Spped of the light", main= "Michelson Experiments")
```



Continuous numbers

Michelson's speed of light experiment

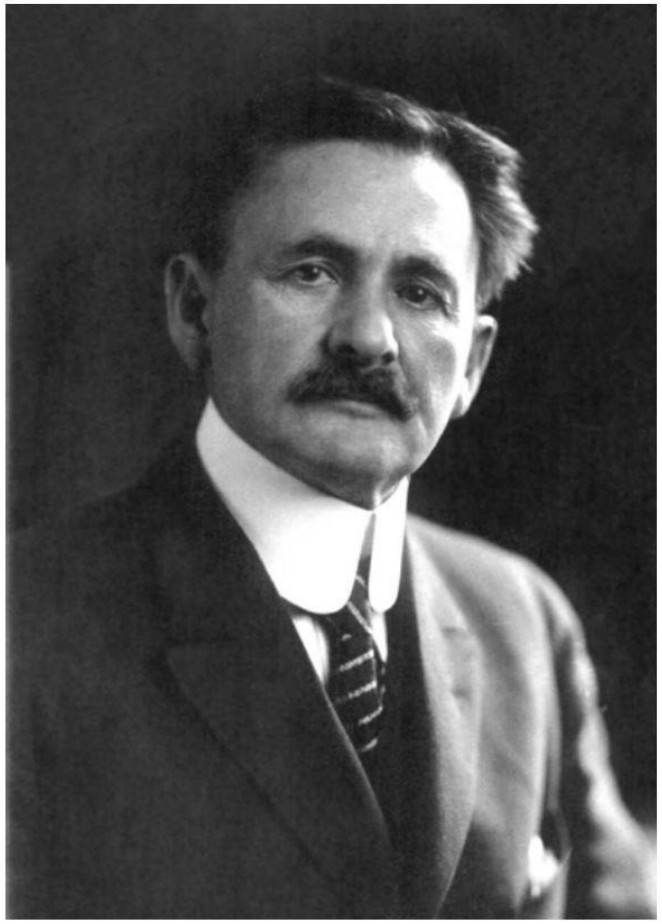


Image: public domain, Smithsonian

Data: Michelson, 1880

measured speed of light (1000 km/s)

299.85	299.74	299.90	300.07	299.93
299.85	299.95	299.98	299.98	299.88
300.00	299.98	299.93	299.65	299.76
299.81	300.00	300.00	299.96	299.96
299.96	299.94	299.96	299.94	299.88
299.80	299.85	299.88	299.90	299.84
299.83	299.79	299.81	299.88	299.88
299.83	299.80	299.79	299.76	299.80
299.88	299.88	299.88	299.86	299.72
299.72	299.62	299.86	299.97	299.95
299.88	299.91	299.85	299.87	299.84
299.84	299.85	299.84	299.84	299.84
299.89	299.81	299.81	299.82	299.80
299.77	299.76	299.74	299.75	299.76
299.91	299.92	299.89	299.86	299.88
299.72	299.84	299.85	299.85	299.78
299.89	299.84	299.78	299.81	299.76
299.81	299.79	299.81	299.82	299.85
299.87	299.87	299.81	299.74	299.81
299.94	299.95	299.80	299.81	299.87

2.5 Tables

In practice, the more frequent cases we have a dataset of variables for each person in a data set. In this case, we can do the tabulation with table, xtabs, or ftable. These functions will generally work for tabulating numeric vectors as well as factor variables, but the latter will have their levels used for row and column names automatically. Hence, it is a good practice to convert numerically coded categorical data into factors. The table is the most basic of the three and is the oldest. The other two offer formula-based interfaces and better printing of multiway tables. Ftable creates 'flat' contingency tables. Similar to the usual contingency tables, these contain the counts of each combination of the levels of the variables (factors) involved.
Relative frequencies in a table are generally expressed as proportions of the row or column totals. Tables of relative frequencies can be constructed using proportion () with the second argument as the variable name which is the same as using prop.table() function for matrices.
In summary, two-way tables were formed with the table command and higher order ones are no exception. If x, y, z, and w are 4 variables, then the function table(x, y) creates a two-way table. The function table(x, y, z) creates two-way tables x versus y for each value of z. Finally x, y, z, w will do the same for each combination of values of z and w. If the variables are stored in a data frame, say DF then the function table(DF) will behave as above with each variable corresponding to a column in the given order.

Categorical data are usually described in the form of tables. This section outlines how one can creates tables from the data and calculate relative frequencies.

1. One-way table

In MASS library, there is a data called painters.

```
> library(MASS)
> data(painters)
> head(painters, n=3)
           Composition  Drawing  Colour  Expression  School
Da Udine            10        8      16           3       A
Da Vinci            15       16       4          14       A
Del Piombo           8       13      16           7       A

> str(painters)
'data.frame':          54 obs. of  5 variables:
 $ Composition: int  10 15 8 12 0 15 8 15 4 17 ...
 $ Drawing    : int  8 16 13 16 15 16 17 16 12 18 ...
 $ Colour     : int  16 4 16 9 8 4 4 7 10 12 ...
 $ Expression : int  3 14 7 8 0 14 8 6 4 18 ...
 $ School     : Factor w/ 8 levels "A","B","C","D",..: 1 1 1 1 1 1 1 1 1 1 ...
```

We see that the data has 54 observations with 5 variables from which the school variable is a factor variable with 8 levels. With table() function of the school variable we ger the count of each factor level. Then we plot this counts a a bar plot.

```
> school = painters$School
> oneway <-  table(school)
> oneway
school
 A  B  C  D  E  F  G  H
10  6  6 10  7  4  7  4

> sum(oneway)
[1] 54

barplot(oneway,ylab='Count',col=1:8,
main='One way table plot’)
```

2. Two-way table

We deal mainly with two-way tables. In the first example we enter a table directly, as is required for tables taken from a book or a journal article. A two-way table needs to be in a matrix object.

Example 1

We take the data from

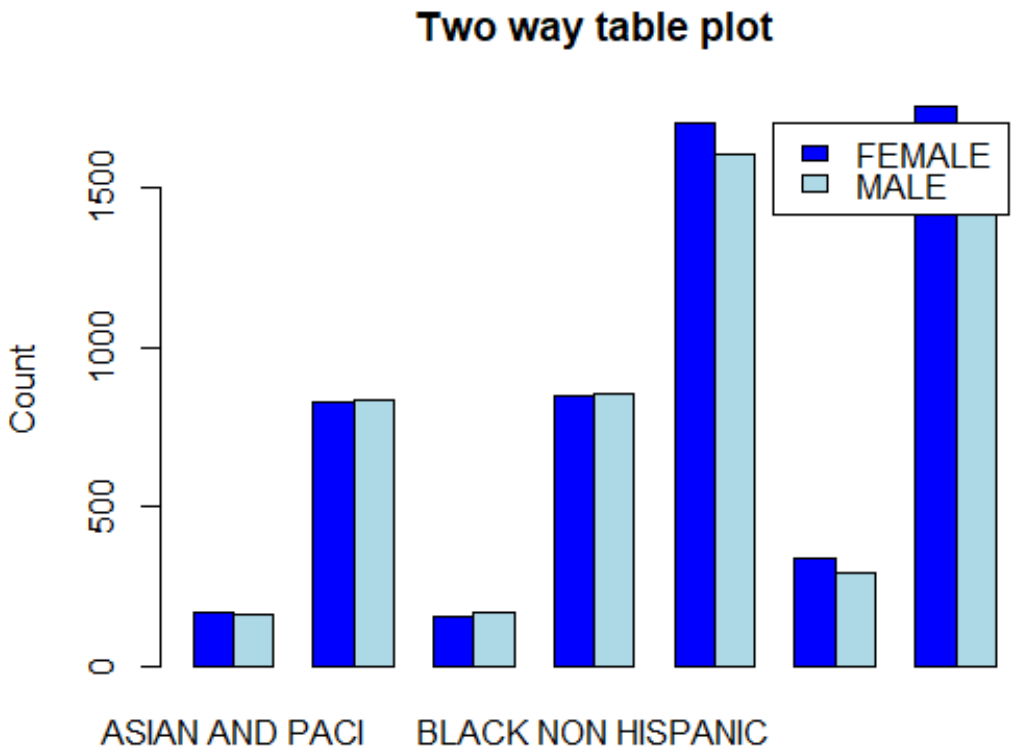
https://github.com/farzaliizadi/data-sets

```
> df <-  read.csv('Popular_Baby_Names.csv', stringsAsFactors = T)
> str(df)
'data.frame':          11345 obs. of  6 variables:
 $ Year.of.Birth     : int  2011 2011 2011 2011 2011 2011 2011 2011 2011 2011 ...
 $ Gender            : Factor w/ 2 levels "FEMALE","MALE": 1 1 1 1 1 1 1 1 1 1 ...
 $ Ethnicity         : Factor w/ 7 levels "ASIAN AND PACI",..: 2 2 2 2 2 2 2 2 2 2 ...
 $ Child.s.First.Name: Factor w/ 3016 levels "Aahil","Aaliyah",..: 2698 671 950 2283 952 1284 2781 377 1057 272 ...
 $ Count             : int  119 106 93 89 75 67 54 52 48 47 ...
 $ Rank              : int  1 2 3 4 5 6 7 8 9 10 ...
```

We read the data by read.csv() function in which the second argument stringsAsFactors = TRUE change the character variables into factors. Using str() function we see that the Gender and Ethnicity are factor variables with labels 2 and 7 respectively. Then One can use table() function with two arguments in this case namely the two factor variables and gets the counts for each pair of factors. Finally, we can plot the counts by par plot.

```
> tb <- table(df$Gender, df$Ethnicity)
> tb

       ASIAN AND PACI ASIAN AND PACIFIC ISLANDER BLACK NON HISP
  FEMALE         171                       827          159
  MALE           164                       835          169

       BLACK NON HISPANIC HISPANIC WHITE NON HISP WHITE NON HISPANIC
  FEMALE             848     1704           341              1756
  MALE               856     1610           296              1609
```

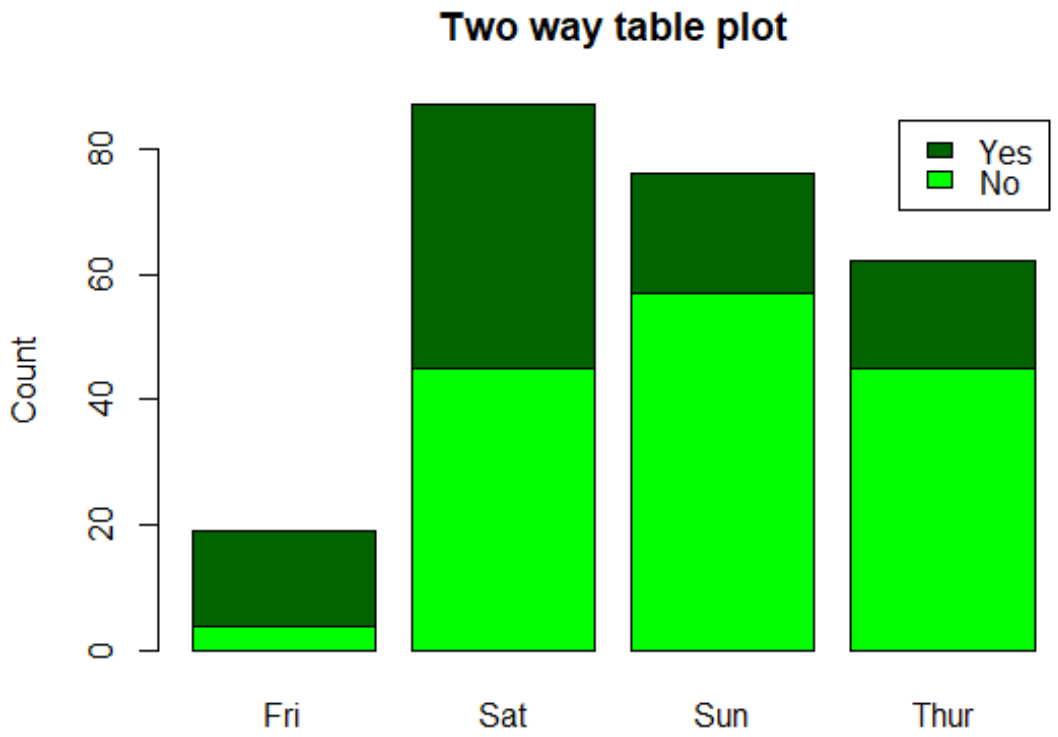
```
> barplot(tb, ylab='Count',
+         main='Two way table plot', beside=T,
+         legend.text= c('FEMALE','MALE'),
+         col=c("blue","lightblue"))
```



Example 2

In the following data frame, we the smoker with two levels and the day with 4 levels.
https://github.com/farzaliizadi/data-sets

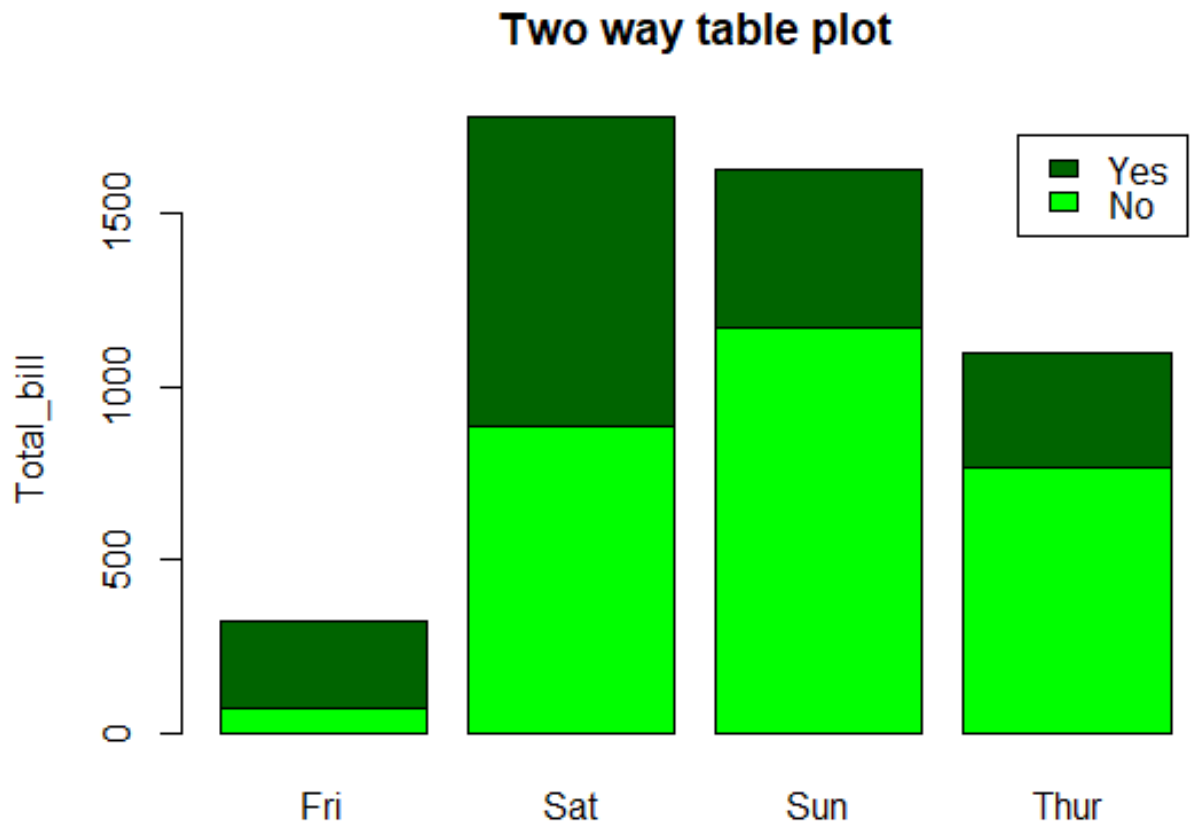
```
> str(dat)
'data.frame':          244 obs. of  7 variables:
 $ total_bill: num  17 10.3 21 23.7 24.6 ...
 $ tip       : num  1.01 1.66 3.5 3.31 3.61 4.71 2 3.12 1.96 3.23 ...
 $ sex       : Factor w/ 2 levels "Female","Male": 1 2 2 2 1 2 2 2 2 2 ...
 $ smoker    : Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
 $ day       : Factor w/ 4 levels "Fri","Sat","Sun",..: 3 3 3 3 3 3 3 3 3 3 ...
 $ time      : Factor w/ 2 levels "Dinner","Lunch": 1 1 1 1 1 1 1 1 1 1 ...
 $ size      : int  2 3 3 2 4 4 2 4 2 2 ...

> tab2 <- table(dat$smoker, dat$day)
> tab2
        Fri Sat Sun Thur
  No     4   45  57   45
  Yes   15   42  19   17

> barplot(tab2, ylab='Count',
+         main='Two way table plot',
+         legend.text= c('No','Yes'),
+         col=c("green","darkgreen"))
```



Example 3. Two-way table with xtabs

```
> xtb <-  xtabs(total_bill ~ smoker + day, dat)
> Xtb

     day
smoker     Fri     Sat     Sun    Thur
  No    73.68  884.78 1168.88  770.09
  Yes  252.20  893.62  458.28  326.24
```

```
> barplot(xtb, ylab='Size',
+         main='Two way table plot',
+         legend.text= c('No','Yes'),
+         col=c("green","darkgreen"))
```



Exercise. Do the xtabs with tip variable.

3. Multiway table

```
> xtb1 <-  xtabs(total_bill ~ smoker + day + sex, dat)
> xtb1
, , sex = Female

     day
smoker    Fri    Sat    Sun   Thur
  No   38.73 247.05 291.54 400.36
  Yes  88.58 304.00  66.16 134.53

, , sex = Male

     day
smoker    Fri    Sat    Sun   Thur
  No   34.95 637.73 877.34 369.73
  Yes 163.62 589.62 392.12 191.71
```

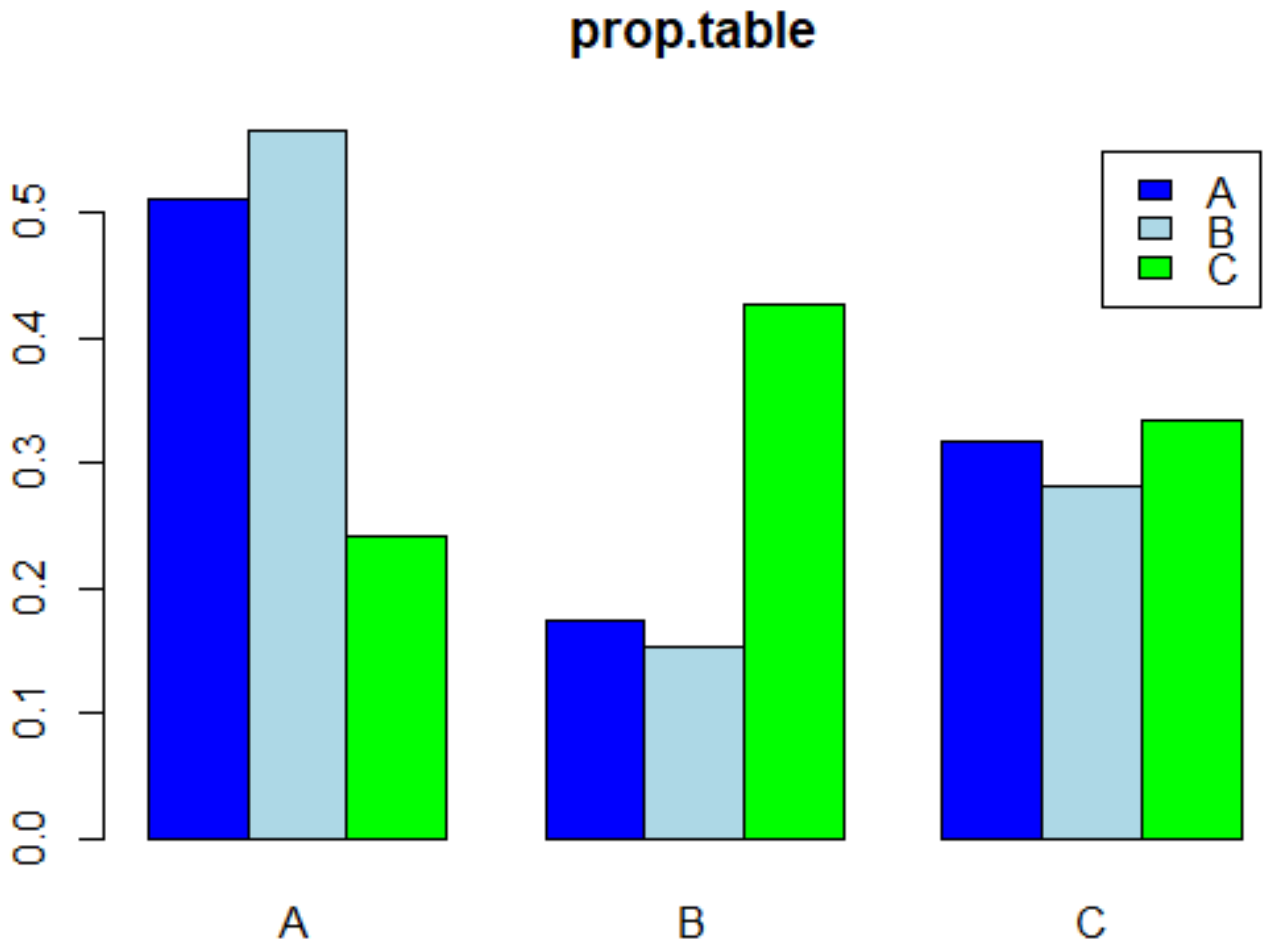
Exercise. Do the same for tip variable.

More example on multiway table

```
xtabs(total_bill ~ smoker + day +sex + time, dat)
, , sex = Female, time = Dinner
     day
smoker    Fri    Sat    Sun   Thur
  No    22.75 247.05 291.54  18.78
  Yes   48.80 304.00  66.16   0.00
, , sex = Male, time = Dinner
     day
smoker    Fri    Sat    Sun   Thur
  No    34.95 637.73 877.34   0.00
  Yes 129.46 589.62 392.12   0.00

, , sex = Female, time = Lunch
     day
smoker    Fri    Sat    Sun   Thur
  No    15.98   0.00   0.00 381.58
  Yes   39.78   0.00   0.00 134.53

, , sex = Male, time = Lunch
     day
smoker    Fri    Sat    Sun   Thur
  No     0.00   0.00   0.00 369.73
  Yes   34.16   0.00   0.00 191.71
```

One can also get the proportions of the table with proportions(xtb, 'smoker').
In the case of a matrix this is the same as prob.table as we see in the following matrix.

```
> mat <- matrix(c(50, 70, 13, 17, 19, 23, 31, 35, 18), nrow = 3)
> row.names(mat) <- (c('u', 'v', 'w'))
> colnames(mat) <-  (c('A', 'B', 'C'))
> proportions(mat, 1) # row-wise
          A         B         C
u 0.5102041 0.1734694 0.3163265
v 0.5645161 0.1532258 0.2822581
w 0.2407407 0.4259259 0.3333333
> proportions(mat, 2) # column-wise
          A         B         C
u 0.37593985 0.2881356 0.3690476
v 0.52631579 0.3220339 0.4166667
w 0.09774436 0.3898305 0.2142857
> prop.table(mat, 1)
          A         B         C
u 0.5102041 0.1734694 0.3163265
v 0.5645161 0.1532258 0.2822581
w 0.2407407 0.4259259 0.3333333

> barplot(prop.table(mat,1),beside=T,
+         legend.text=colnames(mat),
+         col=c("blue","lightblue", "green"))
```

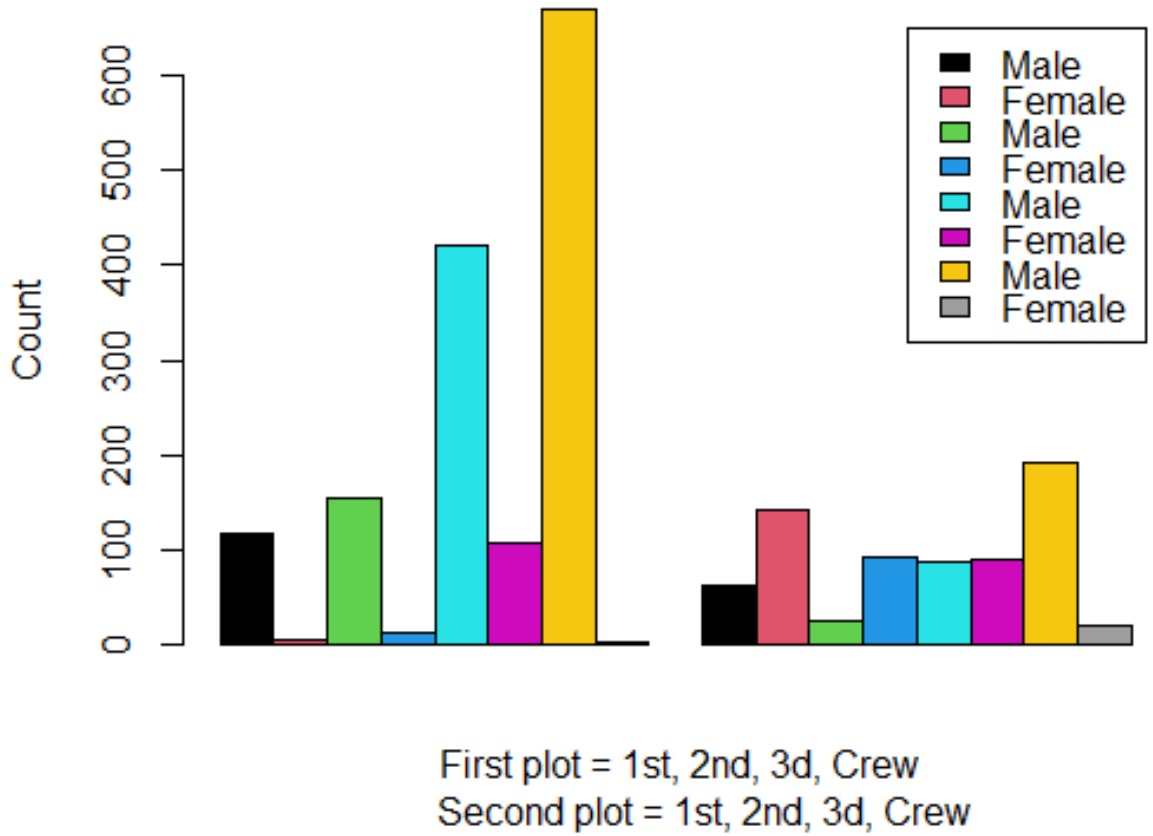


4. Ftables

Ftable creates ‘flat’ contingency tables. Similar to the usual contingency tables, these contain the counts of each combination of the levels of the variables (factors) involved. This information is then re-arranged as a matrix whose rows and columns correspond to unique combinations of the levels of the row and column variables (as specified by row.vars and col.vars, respectively). ftable is a generic function. Its default method, ftable.default, first creates a contingency table in array form from all arguments except row.vars and col.vars.

```
> g <- ftable(Titanic, row.vars = 1:2, col.vars = "Survived")
> g
             Survived  No Yes
Class Sex
1st   Male           118  62
      Female           4 141
2nd   Male           154  25
      Female          13  93
3rd   Male           422  88
      Female         106  90
Crew  Male           670 192
      Female           3  20
```

```
> barplot(g, xlab=' First plot = 1st, 2nd, 3d, Crew
+       Second plot = 1st, 2nd, 3d, Crew', ylab='Count',beside = T,
+        legend.text=c('Male','Female','Male','Female',
+                      'Male','Female','Male','Female'), col=1:8)
```



Ftables on mtcars

```
x <- ftable(mtcars[c("cyl", "vs", "am", "gear")])
ftable(x, row.vars = c(2, 4))
```

Check the plots for yourself.

```
barplot(ftable(x, row.vars = c(1,2)), beside = T, col=1:4)
barplot(ftable(x, row.vars = c(1,3)), beside = T, col=1:4)
barplot(ftable(x, row.vars = c(1,2,3)), beside = T, col=1:4)
barplot(ftable(x, row.vars = c(1,2,3,4)), beside = T, col=1:4)

## Start with expressions
ftable(mtcars$cyl, mtcars$vs, mtcars$am, mtcars$gear, row.vars = c(2, 4),
       dnn = c("Cylinders", "V/S", "Transmission", "Gears"))
```

Proportions table for UCBAdmissions dataset

Check the codes for yourself.

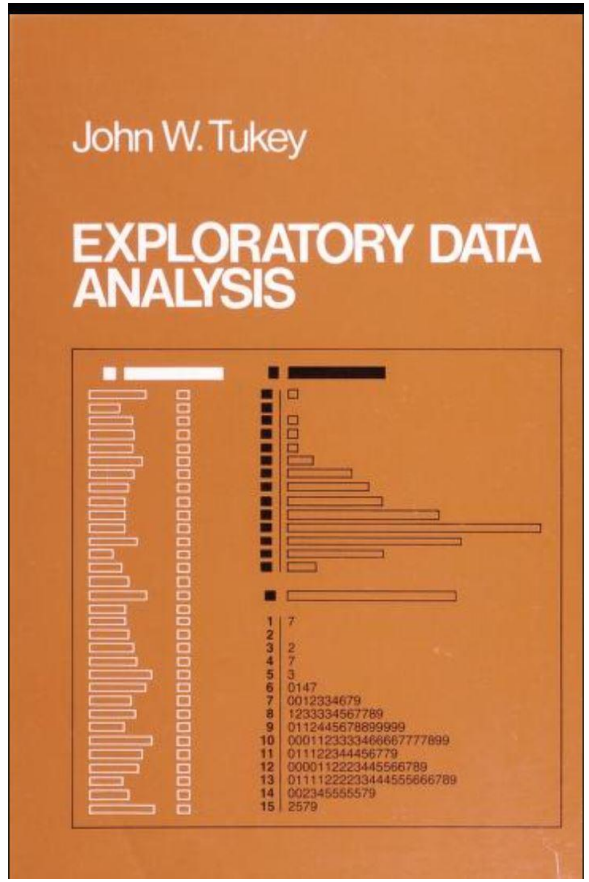
```
DF <- as.data.frame(UCBAdmissions)
str(UCBAdmissions)
tbl <- xtabs(Freq ~ Gender + Admit, DF)
proportions(tbl, "Gender")
proportions(tbl, "Admit")
```

Chapter 3
Two types of statistics – Descriptive and inferential

In general, statistics at least in the applied form divided into two groups: descriptive statistics and inferential statistics. Each of these segments is important, offering different techniques that accomplish different objectives and have some important differences.

Descriptive statistics: Describe what is going on in a sample of whole population or data set drawn from the population. Descriptive statistical procedures are used to organize and summarize the measurements in samples and can only be used to describe the data set under study.
Inferential statistics: By contrast, to take findings from a sample group and generalize them to a larger population.
To this end, one need to confirm the findings from the sample to the population by doing hypothesis tests, parameter estimation, and computing confidence intervals. These can be done by first studying many distributions including normal distribution, t-distribution, chi squared, F-distributions which are topics of chapter 4.

Tukey, John Wilder (1977). *Exploratory Data Analysis*. Addison-Wesley. ISBN 978-0-201-07616-5



Known as the great statistician of his times

Who made the Exploratory data analysis

A formal subject of statistics.

"Exploratory data analysis can never be the whole story, but nothing else can serve as the foundation stone."

—John Tukey

3.1 Descriptive statistics The goal is to describe numerical measures that are used to tell about features of a variable. There are a number of items that belong in this portion of statistics.

3.2 Measures of central tendency: mean, median and mode. Suppose that we randomly select some people in a beach on a hot summer day and recorded their ages.

```
x <- sample(5:60, 40)          # From 5 to 60 sample 40 numbers.
x
58 45 56 39 32 41 28 34 9 27 55 51 22 37 54 30 21 35 52 26 33 57 29 59 31 20
8 60 13 36 24 5 38 48 43 15 49 23 44 12

Next, we sort them from the smallest to the largest.

y <- sort(x, decreasing = FALSE)           #decreasing = TRUE is default

y 5 8 9 12 13 15 20 21 22 23 24 26 27 28 29 30 31 32 33 34 35 36 37 38 39 41
 43 44 45 48 49 51 52 54 55 56 57 58 59 60
```

The average of this numbers is called the mean and calculated by summing all them up and dividing by
The number of element i.e., 40 and get

```
mean(y)
34.975
```

```
The numbers 34 and 35 lie in the middle of the elements. They average i.e.,
(34+35)/2 = 34.5 is called the median of ages. If the number of element is odd,
the middle one is the median value.
median(y)
34.5
```

Finally, the most common value is called the mode of the elements where for this example there isn't any. However, if we have a data like z = c(4,6,8,9,6,5,12,13,5), the mode is 6.

For a N elements in a population and n elements in a sample, their means are

$$\mu = (\sum_{1}^{N} x_i)/N \tag{3.1}$$

$$\bar{x} = (\sum_{1}^{n} x_i)/n, \tag{3.2}$$

3.3 Central of measures of variability

Having figured out the mean, median and the mode, a natural question arises: how much information one can get from these three statistics. Can we know what percentage of the people are teenagers? How the ages are close together or spread out? The answer to this questions is the amount of variability or dispersion in the data. There are two measures of dispersion that statisticians typically examine: the variance, and the standard deviation. Of course, the standard deviation is the square root of the variance and is more informative than the variance. To calculate the variance we simply sum up the square of the differences of the mean from the sample and divide by (n-1), where n is the number of the sample elements.

var(y)
243.4609

sd(y)
15.60323

For a N elements in a population and n elements in a sample, their standard deviation are

$$\sigma = \sqrt{\sum_{1}^{N}(x_i - \mu)^2/N} \tag{3.3}$$

$$s = \sqrt{\sum_{1}^{n}(x_i - \bar{x})^2/(n-1)}, \tag{3.4}$$

Standard error: an estimate of the standard deviation of the sampling distribution - the set of all samples of size n that can be taken from a population reflects the extent to which a statistics changes from sample to sample. For a mean it equals to

$$s/\sqrt{n}.$$

For the difference between two means, it equals to

$$\sqrt{s^2(1/n_1 + 1/n_2)}$$

where the variances are equal and

$$\sqrt{s_1^2/n_1 + s_2^2/n_2}$$

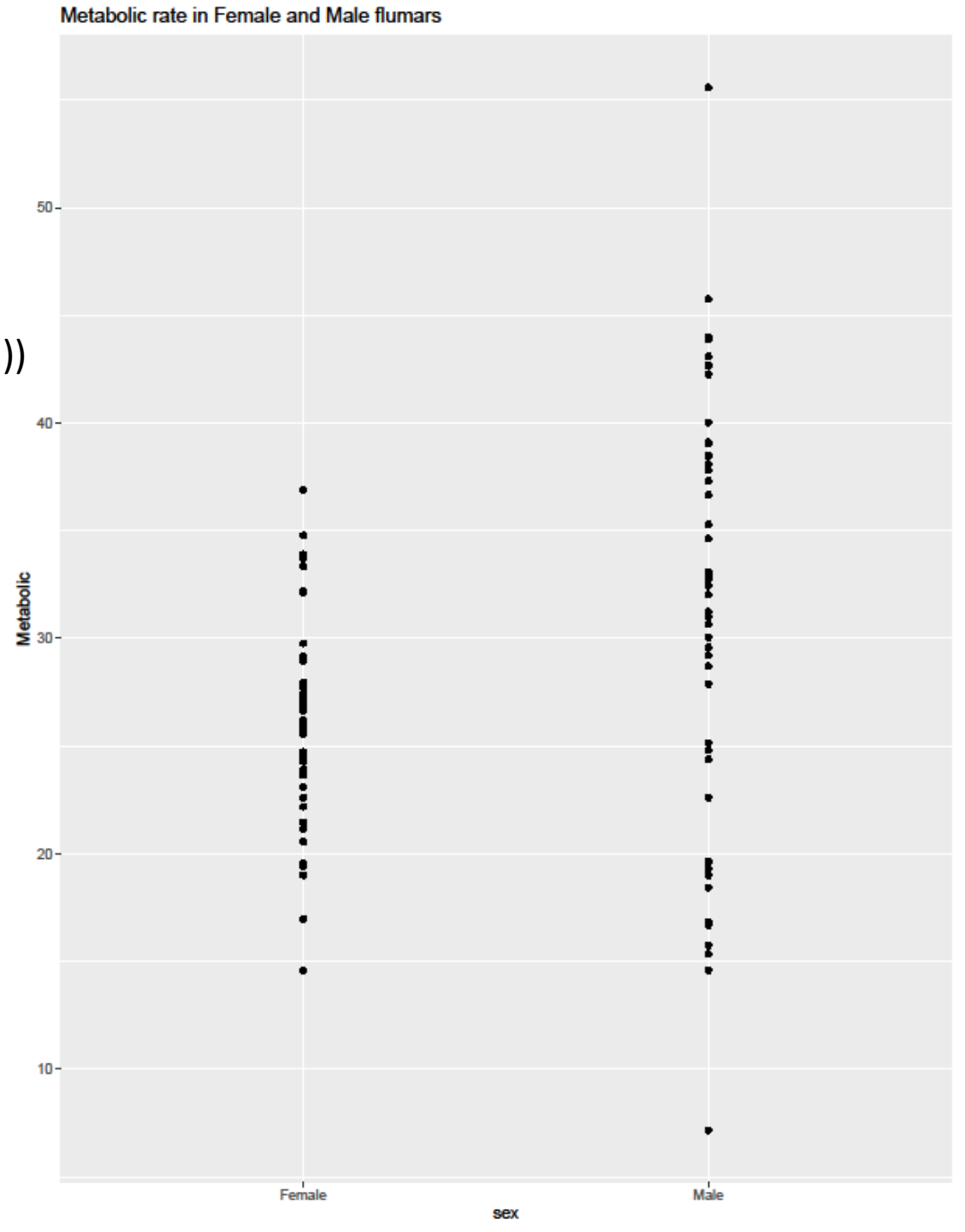
where the variances are different.

Example 1. Comparison of means and standard deviations

```
data = data.frame(
  sex = factor(rep(c("Female", "Male"), each=50)),
  Metabolic = c(rnorm(50, mean=25, sd=5), rnorm(50, mean=30, sd=10)))

head(data)
```

We studied factor() , rep() and rnorm() functions. The latter generates normal random numbers with the given mean and standard deviation. Also we learned how to make a data frame. Ggplot2 is the power package to get awesome and informative plots. Aes sets the axes, geom_point draw the Points and the ggtitle print the title on top of the plot.

```
Library(ggplot2)
ggplot(data=data, aes(x=sex, y=Metabolic))+
  geom_point(stat='identity', fill='red')+
  ggtitle ("Metabolic rate in Female and Male flumars")
```



Example 2. City of the Mississauga means and standard deviation for three parameters of age, weight and height.

```
age <-sample(1:120, 801877, replace = T)
weight <- sample(10:100, 801877, replace=T)
height <- sample(50:200, 801877, replace=T)
df <- data.frame(age, weight, height)
```

Apply() function is a function with three arguments. The first can be a data frame, a matrix, or an array, the second is the row or the column, and the last is the statistics that we want to calculate. We calculate the mean and standard deviation in the following examples.

```
apply(df, 2,mean)
  age          weight       height
60.41765     55.01543    125.00066

apply(df, 2,sd)
   age          weight       height
34.61622     26.27846    43.56767
```

Example 3. Convenient sampling: we simply pick up the first 1000 people from whole population.

```
cs <- df[1:1000,]

apply(cs, 2,mean)

age          weight    height
60.029     53.978    121.622

apply(cs, 2,sd)

age          weight    height
34.60141   26.20279  43.51858
```

Example 4.Systematic sampling: in this case we selected every 14th person. The following code was explained in the code section.

```
> dh <- data.frame()
> x <- 1:1000
> count <- 1
> for (val in x){
+   if(count <= 1000*13)
+     dh <- rbind(dh, df[count,])
+   count = count+13}
> head(dh, n=3)
   age weight height
1   85     47    195
14  71     60     72
27  32     58     81
> apply(dh, 2, mean)
   age weight height
  61.5   54.2  126.1
> apply(dh, 2,sd)
   age weight height
  34.0   26.2   43.2
```

Example 5. Stratified Sampling: we divide the population into different ages, randomly sample them and select 250 people from each random samples.

```
df1 <- subset(df, df$age <=12)
df2 <- subset(df, 13<=df$age & df$age<20)
df3 <- subset(df, 20<=df$age & df$age<60)
Df4 <- subset(df, 60<=df$age & df$age<=120) apply(dh, 2,mean)

x1 <- df1[sample(nrow(df1)),]
x2 <- df2[sample(nrow(df2)),]
x3 <- df3[sample(nrow(df3)),]
x4 <- df4[sample(nrow(df4)),]
```

```
rownames(x1) <- 1:nrow(x1)
rownames(x2) <- 1:nrow(x2)
rownames(x3) <- 1:nrow(x3)
rownames(x4) <- 1:nrow(x4)

y1 <- x1[1:250,] > y2 <- x2[1:250,]
y3 <- x3[1:250,] > y4 <- x4[1:250,]
dh <- rbind(y1,y2,y3,y4)
 age     weight     height
38.690    55.344    126.574

apply(dh, 2,sd)

  age     weight     height
34.65387  26.75256  43.76480
```

Example 6.Cluster sampling: we divide the population into eight clusters, then randomly select 4 clusters and combine these 4 and sample them and finally select the first 1000 from the sample one.

```
dh1 <- df[1:100000,]
dh2 <- df[100001:200000,]
dh3 <- df[200001:300000,]
dh4 <- df[300001:400000,]
dh5 <- df[400001:500000,]
dh6 <- df[500001:600000,]
dh7 <- df[600001:700000,]
dh8 <- df[700001:800000,]
sample(c('dh1','dh2','dh3','dh4','dh5','dh6','dh7', 'dh8') 4)
 f <- rbind(dh2, dh5, dh7, dh8)
g <- f[sample(nrow(f)),]
g <- g[1:1000,]
apply(g,2,mean)
age      weight   height
59.989   54.405   125.851

apply(g,2,sd)
 age      weight   height
34.75732  26.25826 44.26954
```

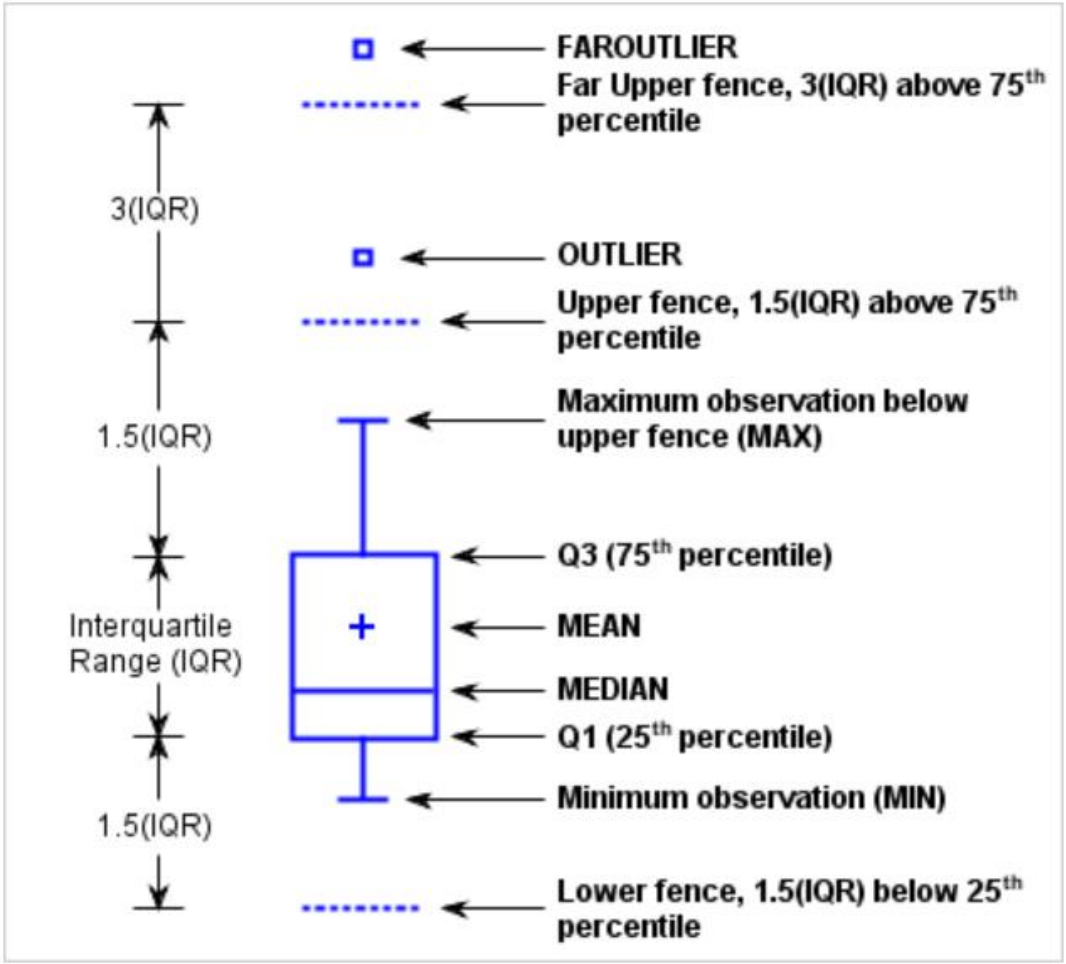
3.4 Five number theory

In previous section, we have seen that the variance or equivalently the standard deviation was an important measure of dispersion of a data. In fact, it is one of the statistics or parameter in the case of the population that can tell how the data is spread out from the mean and median. There are other important statistics that reveal much more information about the data. They are usually called: five number summaries and are the minimum, the 25th percentile (Q1), the median (50th percentile), the 75th percentile (Q3), the maximum. By sorting the data from the smallest to the largest, the minimum is the smallest and the maximum is the largest. We have already defined the median. The first quartile (Q1) is defined as the middle number between the minimum and the median of the data set. It is also known as the lower quartile or the 25th empirical quartile and it marks where 25% of the data is below or to the left of it. The third quartile (Q3) is defined as the middle number between the median and the maximum of the data set. It is also known as the upper quartile or the 75th empirical quartile and 75% of the data lies below this position. The lower quartile is the same as the 25th percentile, the median is the same as the 50th percentile, and the upper quartile is the same as the 75th percentile. The difference between the first
and third quartiles is called the *interquartile range* (IQR) and is sometimes used as a robust alternative to the standard deviation. To find the K-th Percentile, we compute the position L = (K/100) * n, where n is the total number of observations.

Five number summary example

The minimum
The 25th percentile (Q1)
The median (50th percentile)
The 75th percentile (Q3)
The maximum

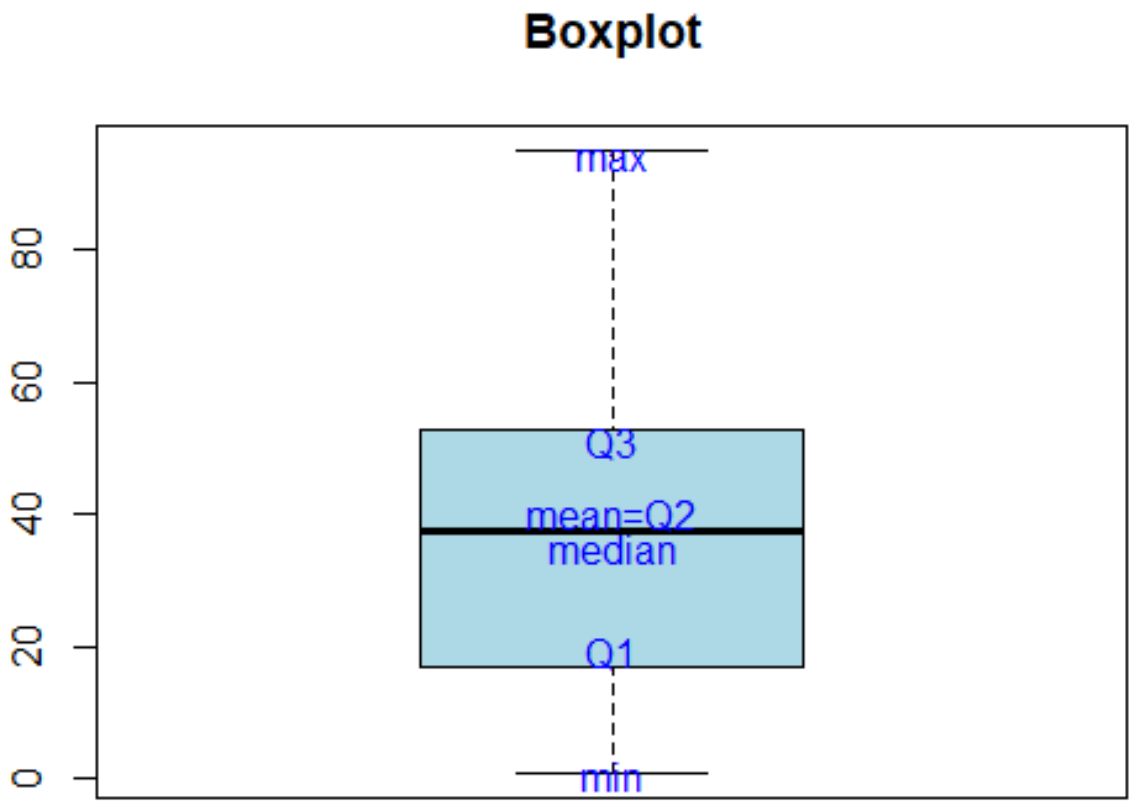
```
x = c(3,1,10,22,45,28,53,36,95,39,83,91,42,17)
min(x)       1
median(x)  37.5
max(x)      95

quantile(x)
          0%    25%    50%    75%    100%
        1.00  18.25  37.50  51.00   95.00

IQR(x) 32.75
quantile(x, c(.32, .57, .98))
       32%   57%   98%
     22.96 40.23 93.96
```



Boxplots

The boxplots are used to summarize data succinctly, quickly displaying if the data is symmetric or has suspected outliers. It is based on the 5-number summary. As the previous page shows, the boxplot has a box with lines at the lower hinge (basically Q1), the Median, the upper hinge (basically Q3) and whiskers which extend to the min and max.
To showcase possible outliers, a convention is adopted to shorten the whiskers to a length of 1.5 times the box length. Any points beyond that are plotted with points.

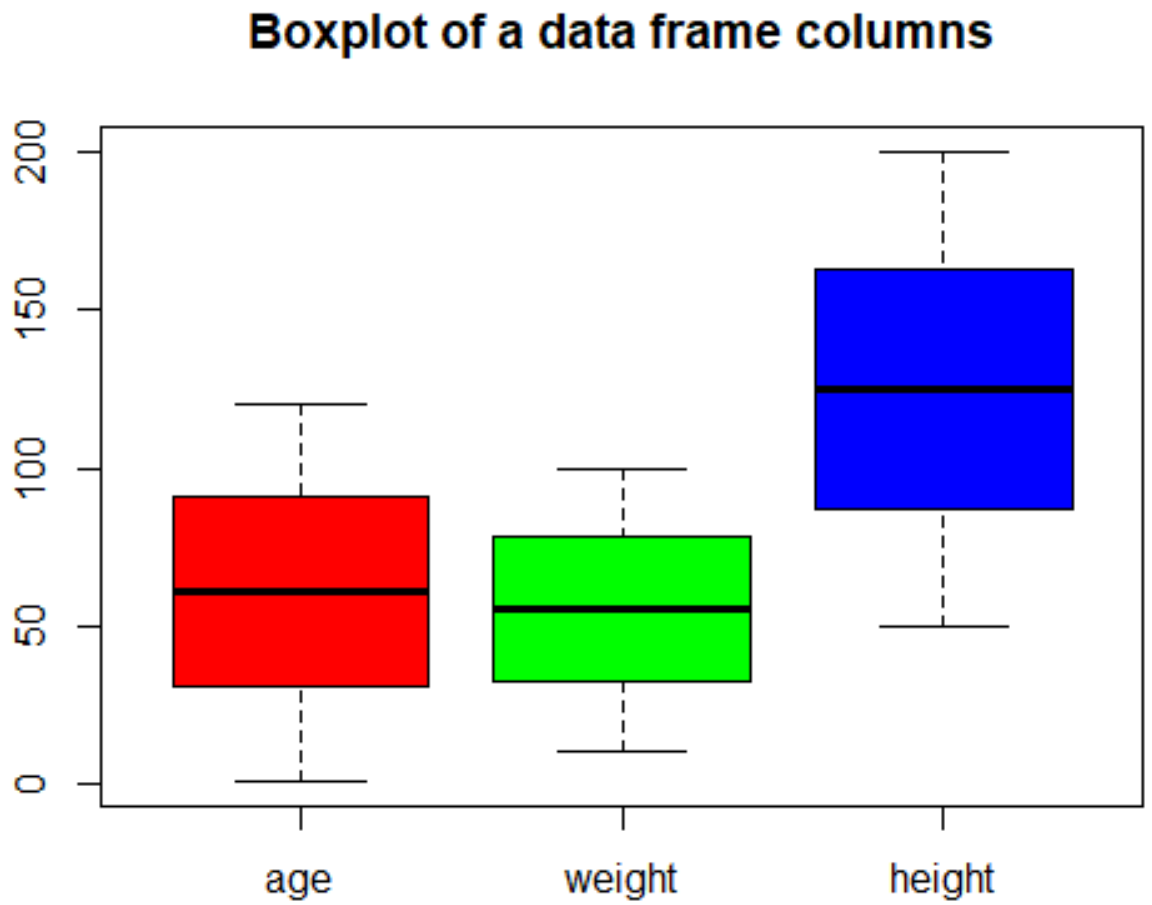
```
x = c(3,1,10,22,45,28,53,36,95,39,83,
91,42,17)

boxplot(x, col='lightblue', main='Boxplot')
text(0.6, 'min', col='blue')
text(19, 'Q1', col='blue')
text(35, 'median', col='blue')
text(40, 'mean=Q2', col='blue')
text(51, 'Q3', col='blue')
text(94, 'max', col='blue')
```



Example 2

```
apply(df, 2, min)
age    weight  height
 1        10      50
apply(df, 2, median)
age    weight  height
 60       55     125
apply(df, 2, max)
age    weight  height
120      100     200
apply(df, 2, quantile)
     age  weight  height
  0%   1      10      50
 25%  30      32      87
 50%  60      55     125
 75%  90      78     163
100% 120     100     200
apply(df, 2, IQR)
     age  weight  height
      60      46      76
```

boxplot(df, main = 'Boxplot of a data frame columns',
col=c('red','green','blue'))



In R we can use summary function to any numerical vector and data frame to get five number summary.

Example 3

```
x <- c(17.16,130.62,150.54,64.46,97.98,118.56,110.94,173.82,81.12,133.44,168.12)
summary(x)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
17.2	89.5	118.6	113.3	142.0	173.8

Example 4

Data is available from
https://github.com/farzaliizadi/Model-selection-for-advertising-dataset
Later in final chapter we will apply EDA as well as linear regression to analyze this dataset Advertising.

```
Advertising <- read.csv('Advertizing.csv')

summary(Advertising[-1])
```

Internet	Email	Blog	WebBanner	Promotional	SmartPhone	Sales
Min. : 0.07	Min. : 0.00	Min. : 0.03	Min. : 0.1	Min. : 0.00	Min. : 0.01	Min. : 0.16
1st Qu.: 22.80	1st Qu.: 3.14	1st Qu.: 5.92	1st Qu.: 29.4	1st Qu.: 3.89	1st Qu.: 3.16	1st Qu.: 3.51
Median : 59.63	Median : 8.46	Median :15.28	Median : 67.8	Median : 9.70	Median :11.66	Median : 6.45
Mean : 74.54	Mean :11.22	Mean :17.85	Mean : 84.3	Mean :12.77	Mean :15.12	Mean : 7.08
3rd Qu.:114.19	3rd Qu.:17.31	3rd Qu.:27.02	3rd Qu.:129.4	3rd Qu.:19.91	3rd Qu.:23.81	3rd Qu.:10.00
Max. :249.03	Max. :37.80	Max. :57.84	Max. :278.9	Max. :42.84	Max. :54.75	Max. :19.62

3.5 Skewness

We call the variable is skewed to the right if mode(x) < mean(x) < median(x).

Likewise it is called normal if mode(x) = mean(x) = median(x).

Finally, it is skewed to the left if mode(x) > mean(x) > median(x).

Example 1.

```
x = c(3,1,10,22,45,4,28,54,65,
      36,95,39,67,83,84,70,42,
       17,109,120,120,165,156,127)
```

mode(x) 50.4

mean(x) 65.1

median(x) 59.5

```
y = c(3,1, 2, -3, -1,-2)
nean(x) =
meadian(x) =
mode(x) = 0
```

```
z = c(3,1,10,22,45,4,28,-54,-50,
      36,95,39,67,83,-84,-70,-42,
       -17,-110,-165,-156,-127)
```

mean(x) -20.09091

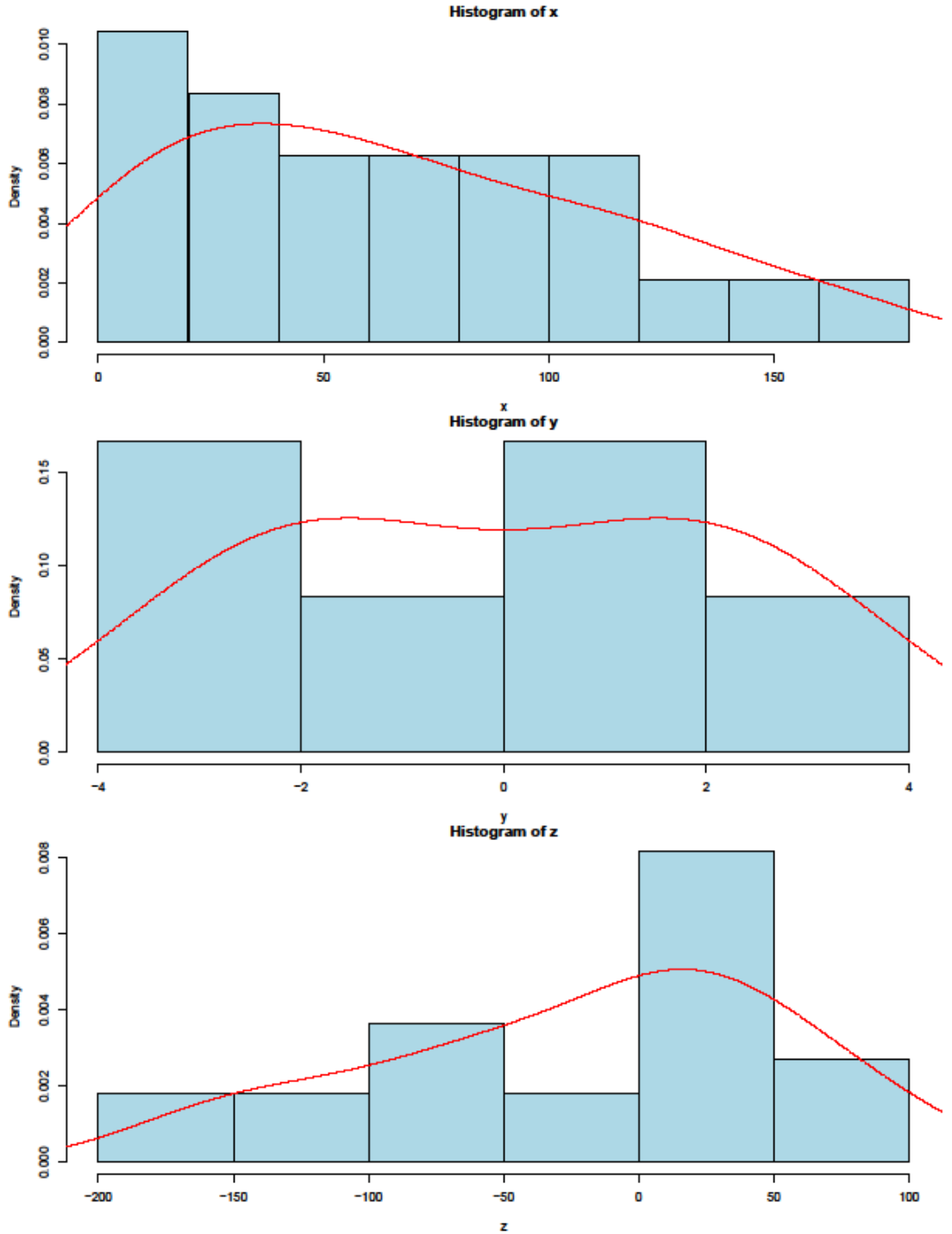
median(x) 2

mode(x) 10

The histograms and the density plots of the three vectors x, y, and z.

```
par(mfrow=c(1,3))
hist(x, freq = F, col='lightblue')
lines(density(x), col='red')
hist(y, freq = F, col='lightblue')
lines(density(y), col='red')
hist(z, freq = F, col='lightblue')
lines(density(z), col='red’)
```

The top plot skewed to the right.
The middle one is almost symmetric.
The bottom one is skewed to the left.



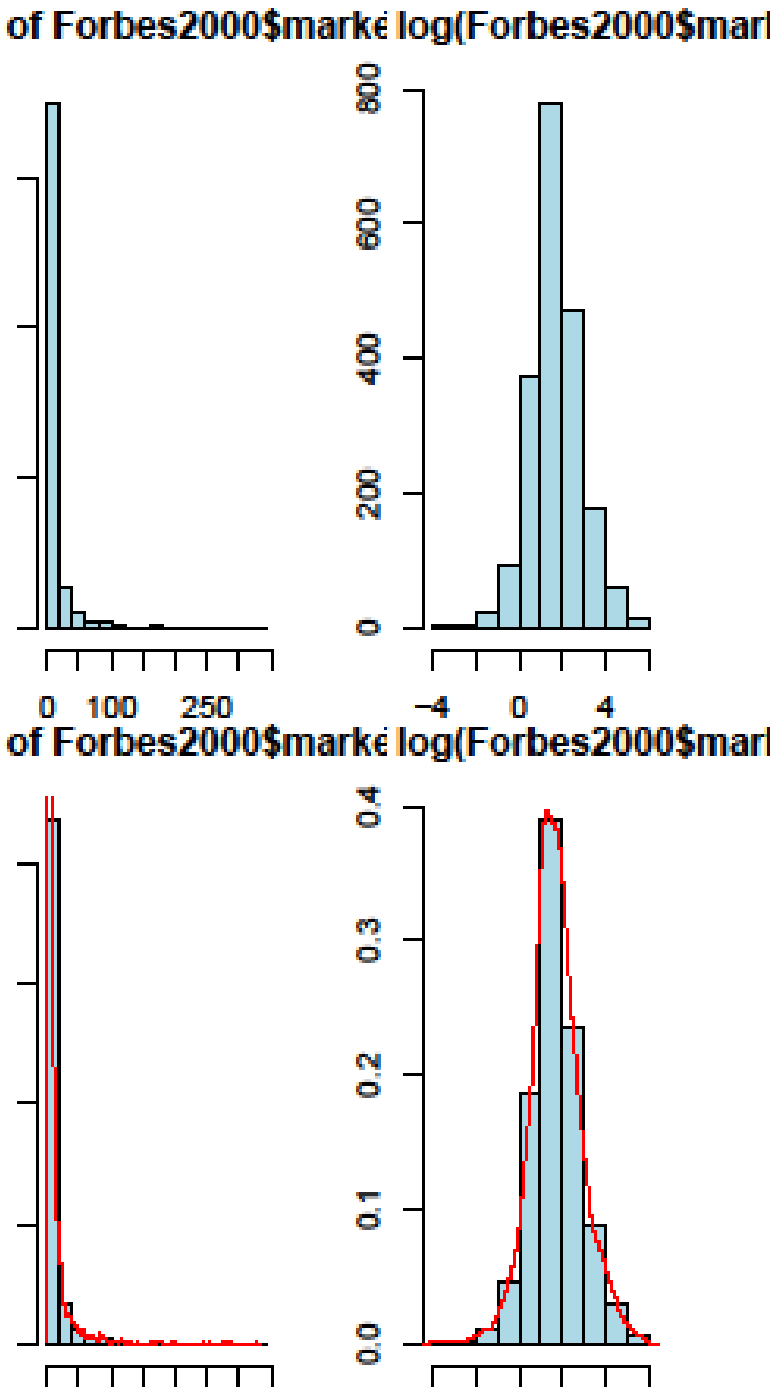
An example of data which is right skewed with symmetric log transformation

The dataset Forbes2000 is in package HSAUR. By using the str() function, we see that the data has 2000 observations of 8 variables. Let us look at the histogram plots of the marketvalue column along with the its log transformation.

```
> str(Forbes2000)

'data.frame':          2000 obs. of  8 variables:
 $ rank       : int  1 2 3 4 5 6 7 8 9 10 ...
 $ name       : chr  "Citigroup" "General Electric" "American Intl Group" "ExxonMobil" ...
 $ country    : Factor w/ 61 levels "Africa","Australia",..: 60 60 60 60 56 60 56 28 60 60 ...
 $ category   : Factor w/ 27 levels "Aerospace & defense",..: 2 6 16 19 19 2 2 8 9 20 ...
 $ sales      : num  94.7 134.2 76.7 222.9 232.6 ...
 $ profits    : num  17.85 15.59 6.46 20.96 10.27 ...
 $ assets     : num  1264 627 648 167 178 ...
 $ marketvalue: num  255 329 195 277 174 ...
```

We have seen par() function before which splits the graphical platform Into 2 rows and 2 columns here with mfrow=c(2,2). The second argument Mar=c(1,1,3,3) sets the four margins of the platform.

```
par(mfrow=c(2,2), mar=c(1,1,3,3))
hist(Forbes2000$marketvalue, col='lightblue')                    # skewed
hist(log(Forbes2000$marketvalue), col='lightblue')               # symmetric
hist(Forbes2000$marketvalue, freq = F, col='lightblue')
lines(density(Forbes2000$marketvalue), col='red')                # skewed
hist(log(Forbes2000$marketvalue), freq = F, col='lightblue')     # symmetric
lines(density(log(Forbes2000$marketvalue)), col='red')
```



3.6 Kurtosis

Kurtosis occurs when the curve of the distribution has longer or shorter tail than normal curve.

To show the kurtosis, we need to generate some data with dnorm() function. This function hast the following arguments.

dnorm(v, mean = 0, sd = 1, log = FALSE), where v is vector of numbers. For example let define v as

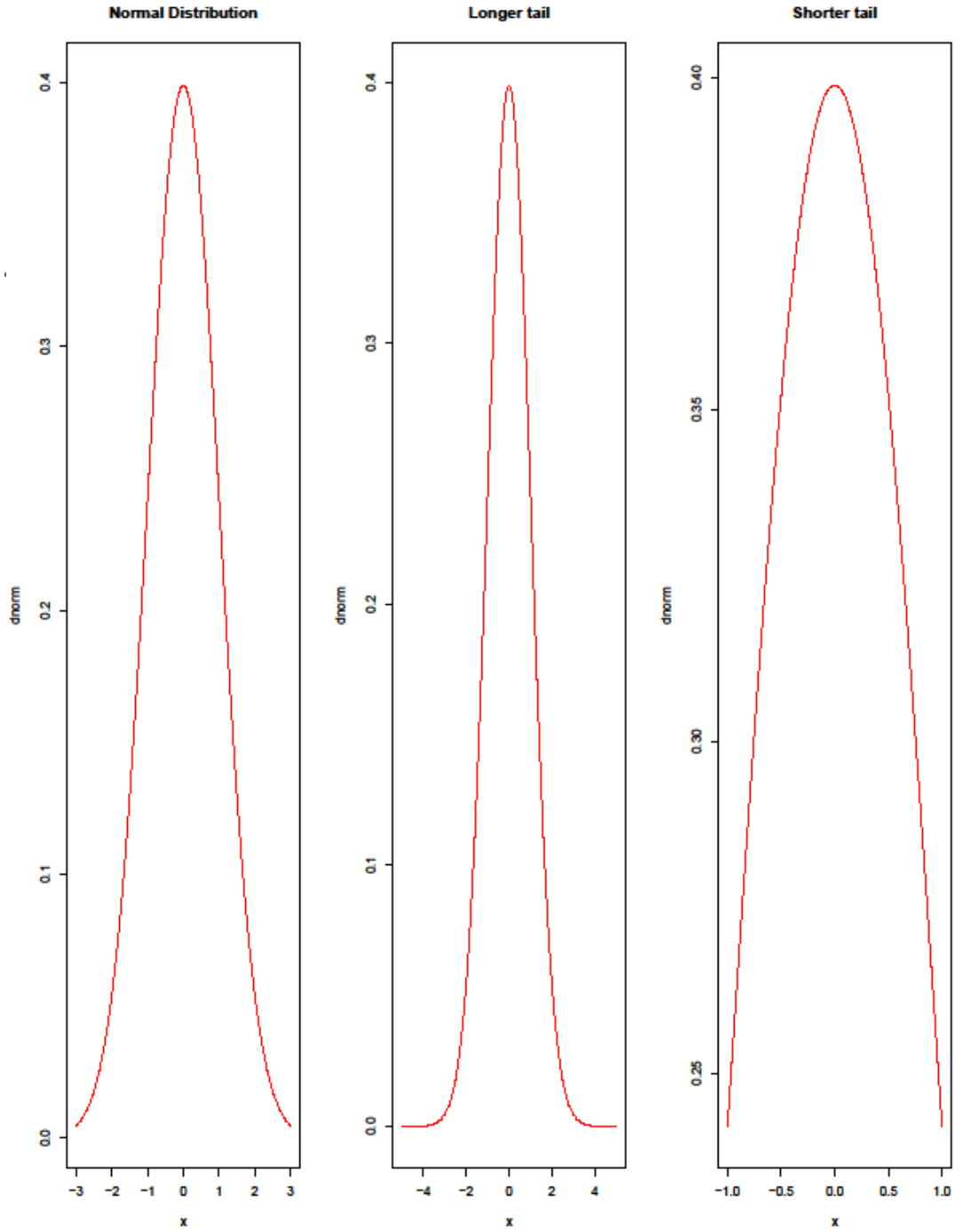
v <- c(9,6,7,8,5,8,6,8,6,5,7). Then

```
> dnorm(v, mean = 0, sd = 1, log = FALSE)
[1] 1.027977e-18 6.075883e-09 9.134720e-12 5.052271e-15
[5] 1.486720e-06 5.052271e-15 6.075883e-09 5.052271e-15
```

We will see more about the dnorm() in chapter 4.

Finally, here are the plots for three different distributions.

```
plot(dnorm, -3, 3, main = "Normal Distribution", col='red')
plot(dnorm, -5, 5, main = "Longer tail", col='red')
plot(dnorm, -1, 1, main = "Shorter tail", col='red')
```



Remark.

Precise definition of Skewness. The kewness of a variable x with mean μ and standard deviation σ is give by:

$$Skew(x) = E(\frac{x-\mu}{\sigma})^3.$$

Precise definition of Kurtosis. The kurtosis of a variable with mean μ and σ is given by:

$$Kurt(x) = E(\frac{x-\mu}{\sigma})^4 - 3$$

In R we have

```
> skewness(x)
[1] 0.5063439
attr(,"method")
[1] "moment"
> kurtosis(x)
[1] -1.103034
attr(,"method")
[1] "excess"
```

3.7 Correlations

So far, we studied the mean, standard deviation, median, mode, min, max, quantiles of a parameter or a statistic of single variables such as age, weight and height of a person from a population a sample. The statistics of these single variables are important in their own rights. However, to make sense of data in the case of multi-variables, statisticians are interested in examining the relations among two or more variables. For example, in the example of age, weight and height they want to see if there is any relationship between any two of these variables. If one of these is increasing or decreasing, what happens for the others. One of the most basic measures of the association among multi variables is the correlation coefficient. Although there are number of different types of correlation coefficients, the most commonly used is the Pearson correlation coefficient. There are other correlation coefficients such as Spearman and Kendell that we may discuss briefly. To define the correlation coefficients, we need to define the mean of a variable in a slightly different way. Let us start with an example.

Suppose the variable x takes the numbers

$$2, 2, 2, 6, 6, 6, 6, 7, 7, 7, 7, 7.$$

It is clear that

$$\frac{2+2+2+6+6+6+6+7+7+7+7+7}{12} = 5.416667.$$

Then one can write this as

$$\frac{1}{12}.(2+2+2+6+6+6+6+7+7+7+7+7) = \frac{3}{12}.2 + \frac{4}{12}.6 + \frac{5}{12}.7$$

Then the numbers

$$\frac{3}{12}, \frac{4}{12}, \frac{5}{12}$$

are called the wights of the numbers 2, 6, and 7 respectively. In general, if the variable x takes the n numbers

$$x_1, x_2, \cdots, x_n$$

with the corresponding weights of

$$p_1, p_2, \cdots, p_n,$$

then the mean of this numbers is called the expectation of x and is computed as

$$E(x) = p_1.x_1 + p_2.x_2 + \cdots + p_n.x_n. \tag{3.3}$$

From this definition one can define the variance as

$$var(x) = E(x^2) - E(x)^2. \tag{3.4}$$

Covariance. Just as the mean and variance provide single-number summaries of the of single variables, covariance is a single-number summary of the two variables. Roughly speaking, covariance measures a tendency of two variables to go up or down together, relative to their expected values: positive covariance between x and y indicates that when x goes up, y also tends to go up, and negative covariance indicates that when x goes up, y tends to go down. Here is the precise Definition:
The covariance between two variables x and y is

$$Cov(x, y) = E((x - Ex) \times (y - Ey))$$

and

(3.5) $$\rho = Cov(x, y)/(sd(x).sd(y))$$

is called the correlation coefficients.

As we see from the definitions of covariance and the correlation coefficients, the covariance is a real number while the correlation ranges between -1 and 1.

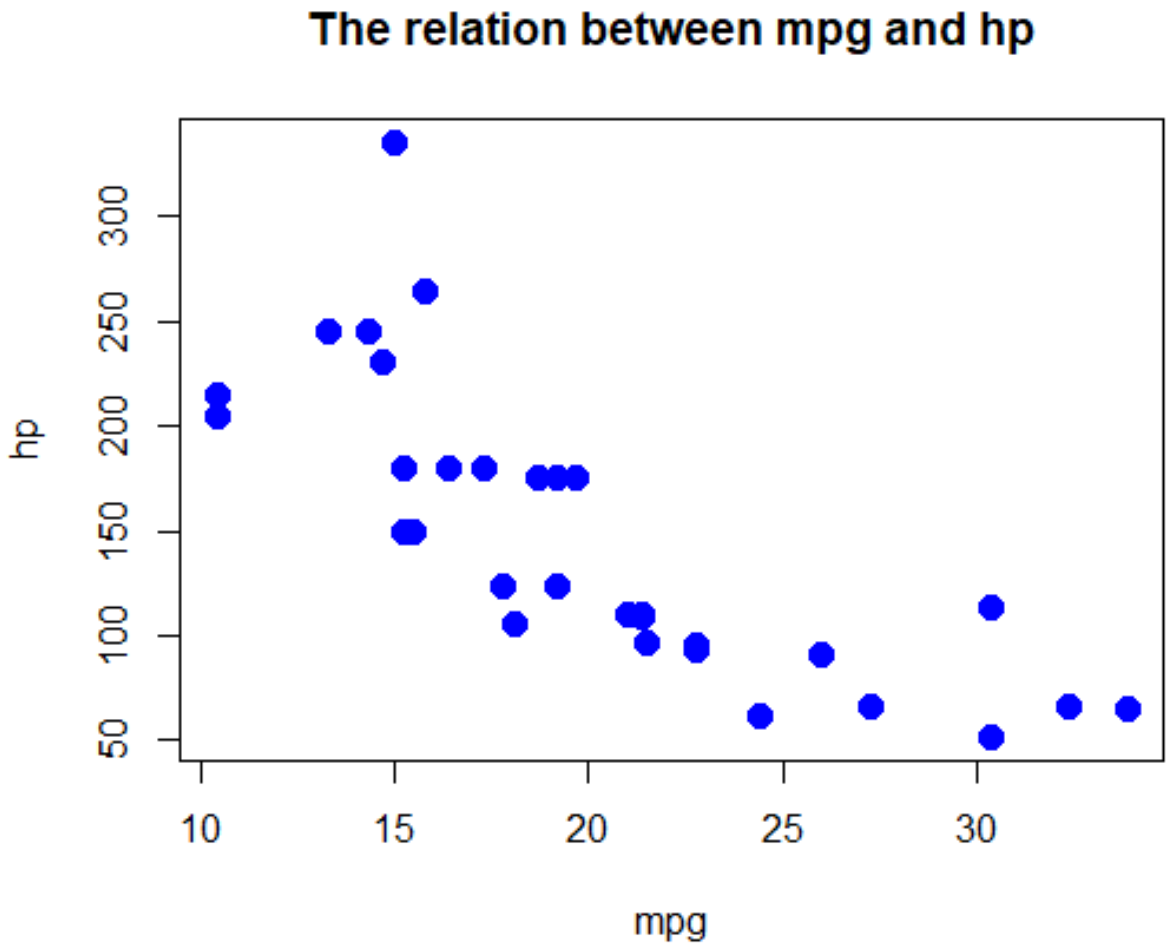
Mtcars is a basis R dataset which has the following variables.
A data frame with 32 observations on 11 (numeric) variables.

```
data("mtcars")
attach(mtcars)   # This make to work variables without data name
mtcars
```

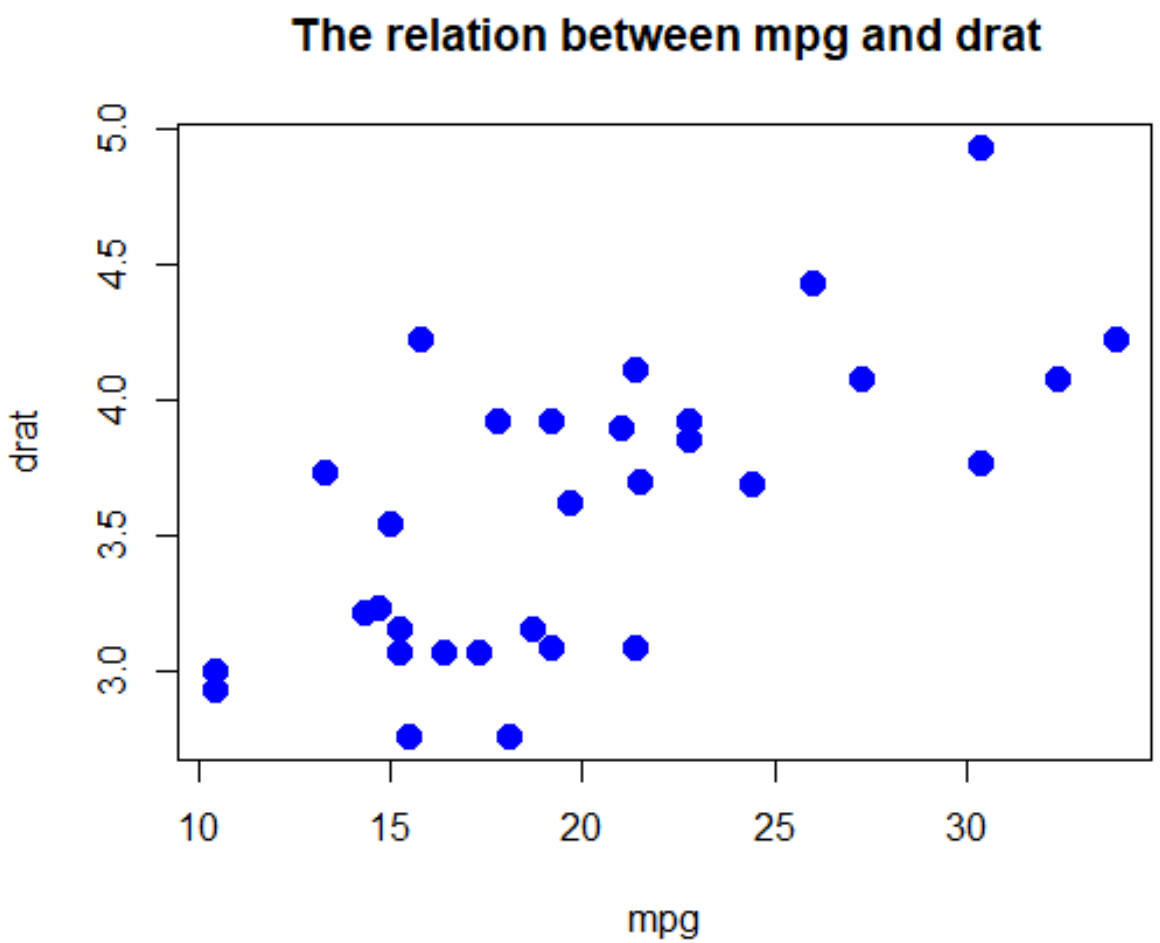
[, 1]	mpg	Miles/(US) gallon
[, 2]	cyl	Number of cylinders
[, 3]	disp	Displacement (cu.in.)
[, 4]	hp	Gross horsepower
[, 5]	drat	Rear axle ratio
[, 6]	wt	Weight (1000 lbs)
[, 7]	qsec	1/4 mile time
[, 8]	vs	Engine (0 = V-shaped, 1 = straight)
[, 9]	am	Transmission (0 = automatic, 1 = manual)
[,10]	gear	Number of forward gears
[,11]	carb	Number of carburetors

Let us examine the relationship of some these variables.
We see that when mpg increases, hp almost decreases which show that two variables negatively correlated.
The plot() function with two variable names returns the scatter plot of one variable versus the other.

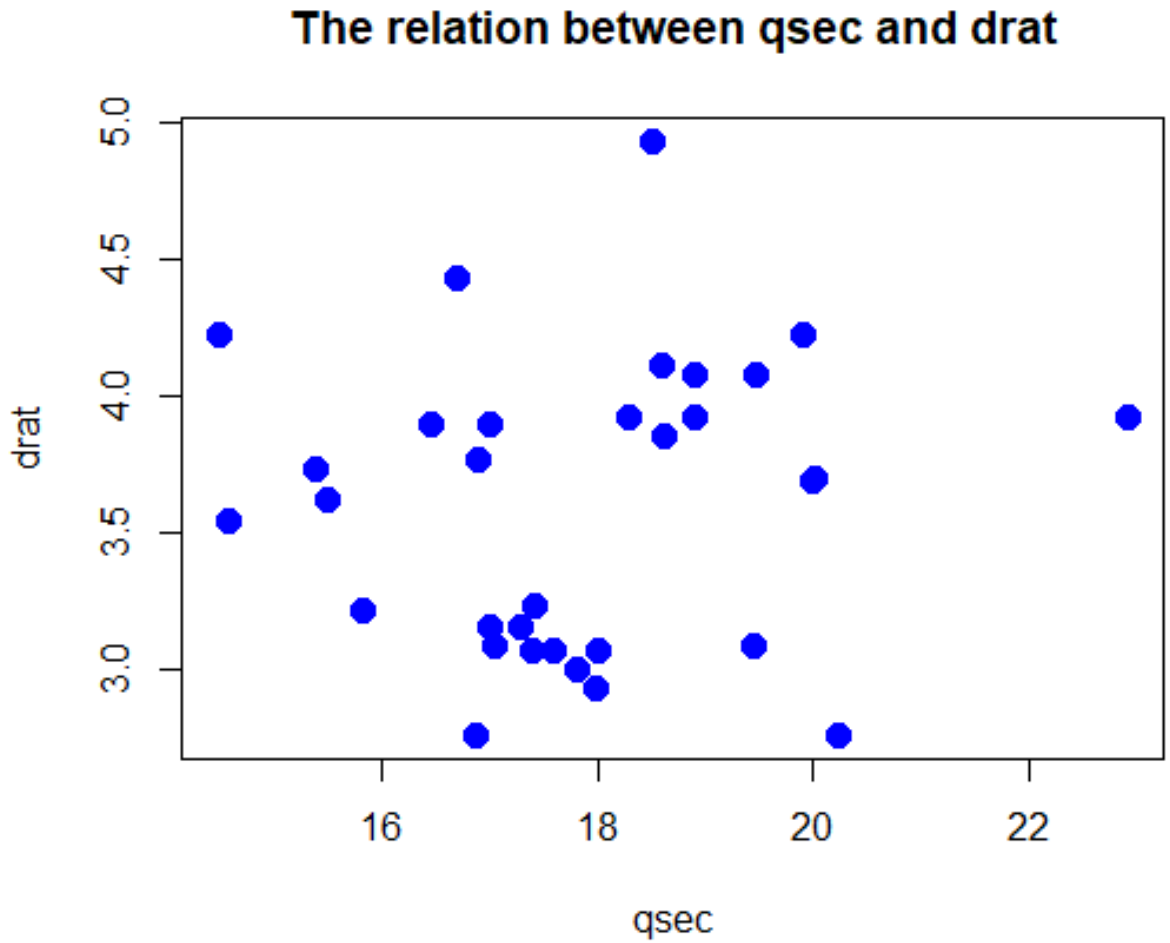
```
plot(mpg, hp, col='blue', pch = 19,
    main='The relation between mpg and hp', cex=1.5)
```



plot(mpg, drat, col='blue', main='The relation between mpg and drat', cex=1.5)



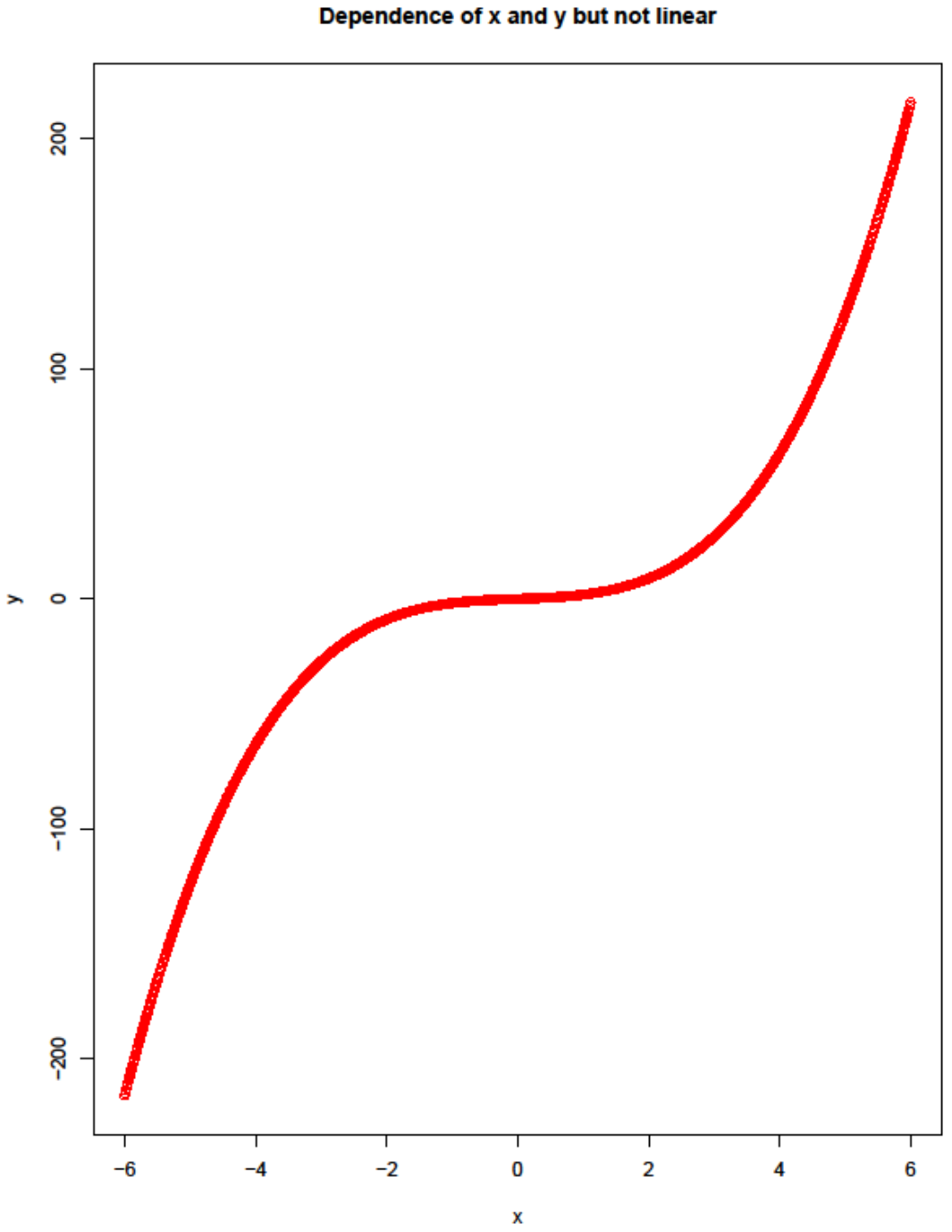
plot(qsec, drat, col='blue', main='The relation between qsec and drat', cex=1.5)



We see that there is weak positive relation between mpg and drat, but the relation between qsec and drat almost zero.
detach(mtcars) # This command detaches the dataset from the environment.

Example of dependence of x and y but it is not linear.

```
x <- seq(-6, 6, 0.01)
y <- x^3
plot(x, y, col='red', main='Dependence of x and y but not linear')
```

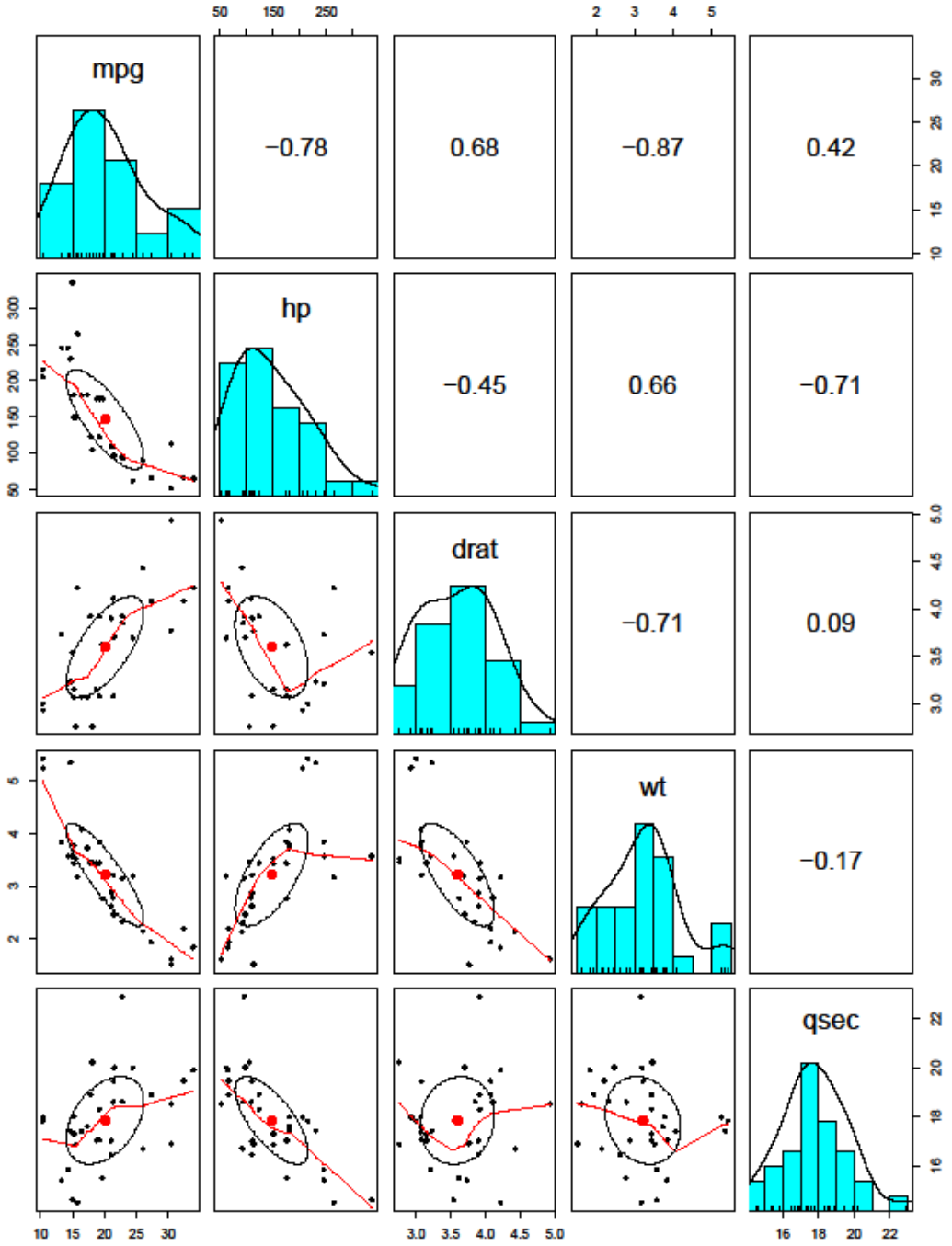


Having defined the codependence or equivalently Pearson correlation coefficients, We can now compute it among any number of variables.

Example 1

```
library(psych)
data(mtcars)
d <- c("mpg","hp","drat", "wt", "qsec")
pairs.panels(mtcars[,d])
```

We see that all correlation coefficients lies between -1 and 1. The smallest is -0.78 between mpg and hp variable. The Largest one is 0.66 between hp and wt. The number 0.09 which is close to 0 is between drat and qsec variables.

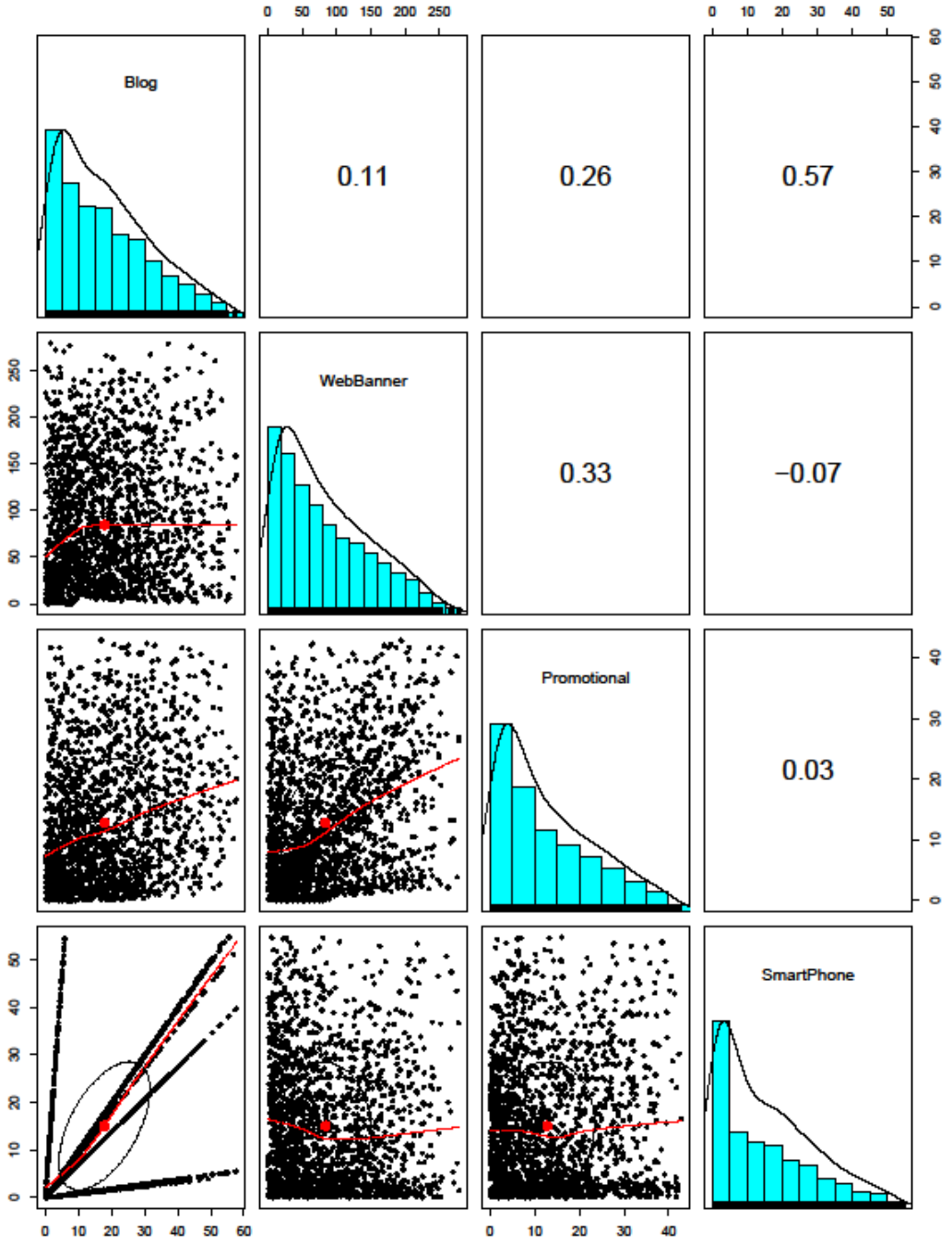


Example 2

Data is available from
https://github.com/farzaliizadi/Model-selection-for-advertising-dataset

```
Advertising <- read.csv('Advertising.csv')
library(psych)
pairs.panels(Advertising[, 4:7)
```

Note. Because of space limitation we selected only 4 variables



4.1 Probability

Pascal and Fermat, the fathers of the probability theory invented gambling problem related to the expected outcomes. As such most of the earliest work in probability theory was about games and gambling relying on symmetry properties. The basic idea then is that of a random sampling, dealing from a well-shuffled deck of cards or a well-stirred urn of balls. For these reasons, the concepts of randomness and probability are central to statistics. Every natural phenomenon has the concept of randomness as its backbone, that is any real-world problem that we encounter, can not be predicted in advance because of the randomness. So the model describing the problem should be probabilistic in nature and probability theory is the study of uncertainty. To this end, some basic knowledge of probability theory is necessary. It is an empirical fact that most experiments and investigations are not perfectly reproducible. The rate of irreproducibility is different from one event to another. The outcomes of physical experiments can be accurate to many decimal places, whereas biological experiments are typically much less reliable. However, the study of data as something coming from statistical distributions is crucial for understanding the statistical methods. In this section we present the basic ideas of probability and the functions that R has for random sampling and handling of theoretical distributions. Let us perform an experiment whose outcome is not predictable in advance but the set of all possible outcomes of this experiment is known. We call the set of all possible outcomes the *sample space and we denote it by S.*

An event: A subset of sample space S.

Simple event: A single outcome of sample space.

Some examples are as follows:

1. 1. In a flipping of a coin we have

$S = \{H, T\}$

where *H* means that the outcome of the toss is a head and *T* that it is a tail. Clearly each outcome {H} or {T} Is a simple event.

2. For a two coin we get

S={HT, HH, TH, TT}

3. In a rolling a die, the the sample space is

$S = \{1, 2, 3, 4, 5, 6\}$

In this case each single set such as {1} or {4} is a simple event and any non-empty subset of S e.g., {3,6} or {1,3,5} is an event.

4. If we flip a coin 3 times, the sample space is

S = {HHH, HHT, HTH, HTT, THH, THT, TTH, TTT}

5. If we roll a white and red dies together,

we get the sample space as

		White Die					
		1	2	3	4	5	6
Red Die	1	(1,1)	(2,1)	(3,1)	(4,1)	(5,1)	(6,1)
	2	(1,2)	(2,2)	(3,2)	(4,2)	(5,2)	(6,2)
	3	(1,3)	(2,3)	(3,3)	(4,3)	(5,3)	(6,3)
	4	(1,4)	(2,4)	(3,4)	(4,4)	(5,4)	(6,4)
	5	(1,5)	(2,5)	(3,5)	(4,5)	(5,5)	(6,5)
	6	(1,6)	(2,6)	(3,6)	(4,6)	(5,6)	(6,6)

Simulation of examples in R

We have already used the sample() function to generate s of random positive integers.

One can also sample the character strings. In the case of a coin toss we do the following:

```
sample(c("H","T"), 1)
```

Here we toss a coin one time. The outcome can be one H or one T. For two coins we use to get HT or TH and nothing else.

```
sample(c("H","T"), 2)
```

```
[1] "H" "T"
```

```
sample(c("H","T"), 2)
```

```
[1] "T" "H"
```

To get the other cases of TH of HT we add replace = TRUE

```
sample(c("H","T"), 2, replace = TRUE)
```

```
[1] "T" "T"
```

```
sample(c("H","T"), 2, replace = TRUE)
```

```
[1] "H" "T"
```

3 or more coins

sample(c("H","T"), 3, replace = TRUE)
[1] "T" "T" "H«

or
sample(c("H","T"), 10, replace = TRUE)
 [1] "T" "H" "H" "H" "T" "H" "T" "T" "H" "T„

Similarly, for throwing a die, we do
sample(6, 1)
[1] 4
sample(6, 1)
[1] 2

For 2, 3, …, 6 dies, we use
sample(6, k), where k = 2, 3, …, or 6.
sample(6, 6)
[1] 1 3 2 6 4 5

Finally, for more than 6 dies we add replace = TRUE
sample(6, 10, replace = TRUE)
 [1] 6 5 1 3 3 2 5 2 6 4
Note: We get only one simple event for each case.

We can sample with success and failures too.

```
sample(c("succ", "fail"), 5, replace=T)
[1] "fail" "succ" "succ" "succ" "succ«
```

Playing with a deck of cards: We have four different types of cards namely CLUBS, DIAMONDS, HEARTS, SPADES where each type has labels as A, 2, 3, …, 10, J, Q, and K.
So we repeat these items 4 times and then combine them with 4 different types.

```
u <- rep(c("A",2:10,"J","Q","K"), 4)
v <- c("CLUBS", "DIAMONDS", "HEARTS", "SPADES")
cards = paste(u, v)
sample(cards,4)
```

"8 SPADES" "9 SPADES" "A CLUBS" "10 SPADES"

4.2 Random variables and probabilities

A function from sample space into real numbers which we denote it by X.
In the examples of tossing the coins let us suppose that the value of X on a given outcome is the number of heads occurring. Then we have the following values for X.

1. X: S={H, T} → R where X({H})=1 and X({T)} =0

2. X: S={HH, HT, TH, TT} → R where X({HH}) = 2, X({HT})=X({TH}) = 1, and X({TT}) = 0.

3. Similarly the values of X on a sample space of 3 coins are 0, 1, 2, 3.

4. In a rolling a die we have X: S = {1, 2, 3, 4, 5,6} -→ {1, 2, 3, 4, 5,6}
where X({i}) = i for I =1,2,…,6.

5. If we roll a two dies, one white and the other red we define X({I, j}) =i+j for I, j = 1,2,…,6.
It is easy to see that the values of X in this case is the set {2, 3, 4, 5,…, 11,12}.

So far in our examples the random variable X took only discrete values of integer numbers. However, if the sample space is the weight or height of a population, then X can take a real numbers.
Having said that, a random variable is a random number drawn from a population. A data point then will be a realization of some random variable. Notice that the random terminology is coming from the random procedure or experiment.

Flip a coin 3 times

X: S ---> R

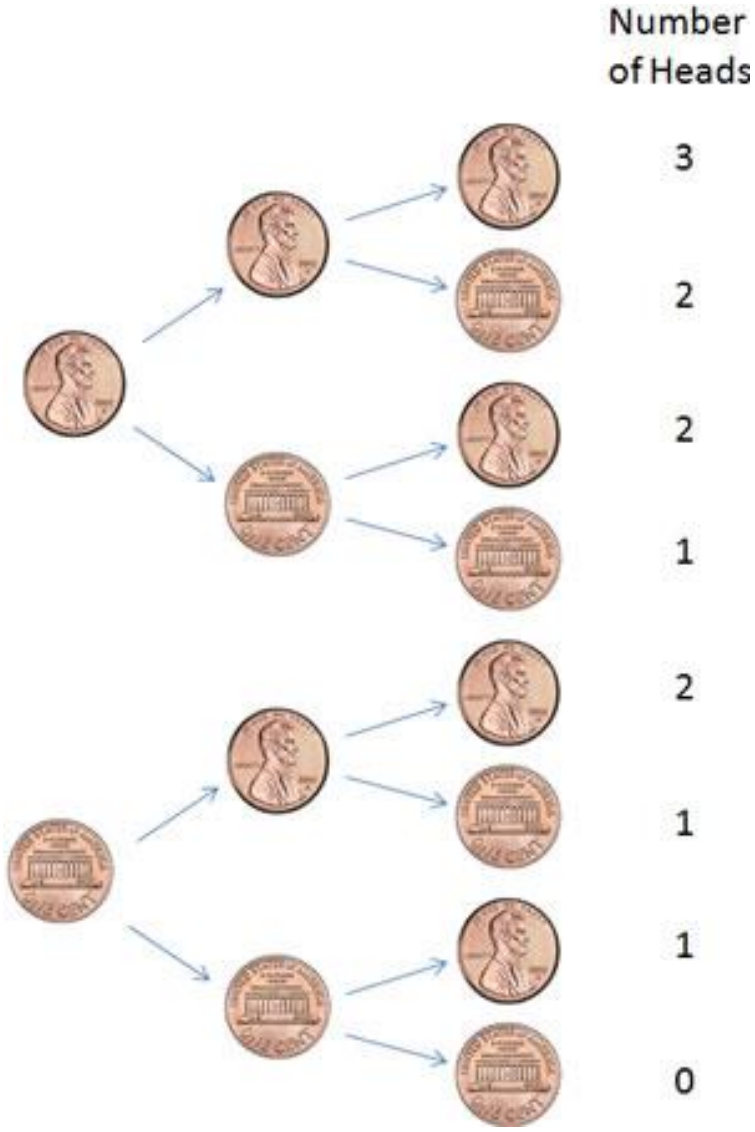
X(s) = # of heads

S = {HHH, HHT, HTH, HTT, THH, THT, TTH, TTT}

We have seen that in a performing an experiment its single event or outcome is not predictable in advance. Once observed, the value of the random variable is known. To fully describe a random variable prior to observing it, we need to indicate the probability that the random variable is some value or in a range of values. We refer to a description of the range and the probabilities as the distribution of a random variable. By probability we mean the likelihood of our random variable having some value. As the sample space S has the complete likelihood of occurring, its probability is 1. By denoting the probability function by capital letter P we get P(S) = 1.

Then the empty event had P({}) = 0 and consequently and non-empty event which is not the whole sample space S has the probability between 0 and 1.

For a situation where all events are equally likely and the total number of events is finite, we define the probability as



P(A) = # of events in A / # of events in S,

where A is an event in S.

Notice that for continuous random variable or

Equivalently for infinite number of events we will discuss in the following sections.

Before we write down the principals of the probability, let us mention some more definitions.

Disjoint or mutually exclusive events: The events which don't occur at the same time.

Complementary events : Events which are mutually exclusive and they union make up whole sample space.

Complement: When A happens, then A^c doesn't happen, where A^c is the complement of A.

From these facts along with previous definition of a probability we see that:

1. Probability of any event A is non-negative.

2. If A an B are two disjoint events, then

P(AUB) = P(A) + P(B),

where AUB is the union of the events A and B.

3. The probability of all possible events is equal to 1.

Note: If the events A and B are not disjoint,

then P(AUB) = P(A) + P(B) – P(A and B)

Examples in R

Let us use the sample function but we unequal probabilities in this case.

sample(c("succ", "fail"), 2,replace= TRUE, prob=c(0.65, 0.35))

[1] "succ" "succ"

sample(c("succ", "fail"), 2,replace= TRUE, prob=c(0.65, 0.35))

[1] "succ" "fail”

The probability of selecting a heart from a deck of cards

P(heart) = 13/54=0.25

```
u <- c("A",2:10,"J","Q","K")
v <- c("HEARTS")
cards = paste(u, v)
sample(cards,1)
```

[1] "3 HEARTS"

Since we can get all combinations of the elements of the vector with HEARTS, the total number is 13.

Flip a coin 10 times and get 5 heads, then P(H) = 5/10 = 50%

```
sample(c("H","T"), 10, replace = TRUE)
```

[1] "T" "H" "H" "H" "T" "H" "T" "T" "H" "T“

Margin tables

Traffic is a data frame in MASS library where
the limit variable = was the speed limit?
y = traffic accident count for that day.

```
str(Traffic)
'data.frame':          184 obs. of  4 variables:
 $ year : int  1961 1961 1961 1961 1961 1961 1961 1961 1961 1961 ...
 $ day  : int  1 2 3 4 5 6 7 8 9 10 ...
 $ limit: Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
 $ y    : int  9 11 9 20 31 26 18 19 18 13 ...


 tb <- table(Traffic$limit, Traffic$y)
 margin.table(tb, margin=1)
 no yes
115  69
sum(tb[1,]) / sum(tb)
[1] 0.625                          # Returns the probability of accidents with no speed limit over the total accidents.
```

More examples on margin table

```
> data <- read.table('court.txt')
> str(data)
'data.frame':          154 obs. of  2 variables:
 $ action: chr  "no-commit" "commit" "no-commit" "commit" ...
 $ charge: chr  "no-guilty" "guilty" "guilty" "no-guilty" ...
tb1 <- table(charge, action)
tb1
                    action
charge     commit    no-commit
  guilty         44             42
  no-guilty    38             30

  margin.table(tb1, margin=2)
           action
   commit       no-commit
       82               72
  sum(tb1[,2]) / sum(tb1)   # no-commit
[1] 0.4675325
```

Probability(guilty or no-commit) = P(guilty) + P(no-commit) – P(guilty and no-commit)

= 44+42/154 + 42+30/154 – 42/154 = 44+42+30/154 =

```
[1] 86.19481
```

Independent events: The events A and B are independent if only if P(A and B) = P(A) * P(B)

Example. In tossing a fair die, A = {2, 4, 6} and B = {1, 2, 3, 4}. Then A and B = {2, 4},

P(A and B) = 2/6 = P(A)*P(B) = (1/2) * (2/3). So A and B are independent.

Remark: If A and B are disjoint events with positive probabilities. Can they be independent? No.

Since P(A)*P(B) is positive yet P(A and B) = P(Empty set) = 0.

Conditional probability: The probability of B occurring given that A occurring is denoted by P(B occurring |A occurred) and it is equal to P(B and A)/P(A).

Example 1. In a die example above, let A = {1,2,3,5} and B = {1, 2,4,5,6}. Then

P(B occurring |A occurred) = P(B and A) / P(A) = P{1, 2, 5} / P{1,2,3,4} = 3/6 / 4/6 = 3/4.

Conditional probability Example 2

A medical test for disease D has outcome + and - . Here we used read.table to read directly the text data into a data frame data, where D^c is complement of D.

```
data <- read.table(header = T, text='
          D          D^c
  +     0.0081   0.0900
  -     0.0090    0.9010
‘)
```

P(+|D)= P(+, D)/P(D) = 0.0081/(0.0081+0.0090)

[1] 0.4736842

P(-|D.c) = P(-,D.c)/P(D.c) = 0.9010/(0.9010+0.0900)

[1] 0.9091826

Suppose you go for a test and get positive. What is the probability you have The disease?

P(D|+) + P(+, D)/P(+) = 0.0081/(0.0081+0.9000)

[1] 0.008919722

4.5 Statistical distributions

So far, we have discussed random variables whose possible outcomes are some discrete set such as the number of heads in filliping a coin, two coins, or three coins that is {0,1}, {0, 1, 2}. We have also seen the values of a random variable as the getting the set {1, 2, . . . ,6} when we toss a die. We can go further to study random variables with discrete sets consisting of {0,1,2,…} On filliping a coin an arbitrary number of times. In this case the realization of the probability distribution is the function dbinom(k, size = n, prob = p) function which is called the probability mass function, pmf, for a coin getting k head for n trials With parameter p being the probability of a head in an individual trial. In a probability course this function returns the value of

P(X = k) = choose(n, k)*p^k *(1-p)^(n-) k
where k = 0, 1, 2 ,3 , …, n and p is the proportion
of the getting one head. To simulate this formula for n=0, p=1/5, and k=0:7 we first generate an empty vector bi, the with a loop Over k we compute values of the function P(X=k) for k=0:7 and then add the values to the vector bi one by one using concatenation.

```
bi <- c()
for(k in 0:7){ b <- choose(10,k) * (1/5)^k * (4/5)^(10-k)
bi <- c(bi, b)}
bi
[1]   0.107374182  0.268435456  0.301989888 0.201326592
[5]   0.088080384  0.026424115  0.005505024 0.000786432
```

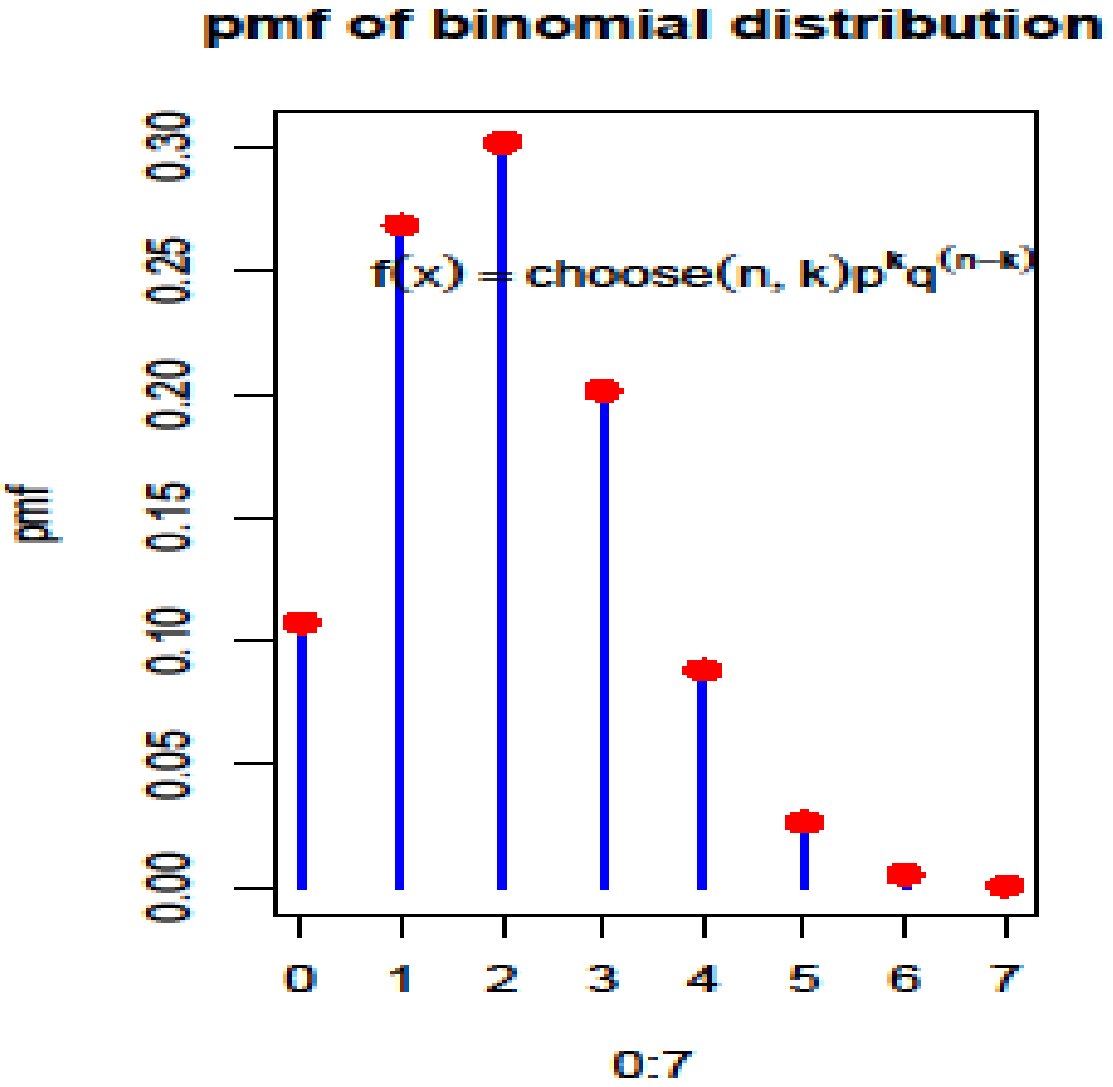
We can get this numbers simply by using the dbinom() as

```
pmf <- dbinom(0:7, size=10, prob=1/5); pmf
[1]   0.107374182   0.268435456  0.301989888  0.201326592
[5]   0.088080384   0.026424115  0.005505024  0.000786432
```

We can also plot the values by vertical thin lines as follows:

```
plot(0:7,pmf, type='h', col = "blue", lwd = 2,
     main = "pmf of binomial distribution")
text(5, 0.25, expression(f(x) ==choose(n,k)*p^k*(1-p)^(n-k)))
points(0:7, pmf, pch=16, cex=1.3,col='red')
```

The type ='h' plots a histogram with very thin lines. xlab, ylab, and main are labels for the axes and the title respectively.
Finally, points puts the points on the top of the lines.
Each case shows the probability of getting
0 to 7 heads from Trials 10 trails with p= 1/5.
Notice that the pmf function as a text on the plot too.



The probability that there are six or fewer heads is denoted by P(x<=6) and called the cumulative density function, cdf.

```
sum(dbinom(0:6, size=10, prob=1/2))                # To compute this we may use
[1] 0.8281
```

The main function to do this is pbinom() which we can use as:

```
pbinom(6, size=10, p = 1/2)
[1] 0.8281
```

```
sum(dbinom(7:10, size = 10, prob=1/2))                # For 7 or more
[1] 0.1719
```

Or equivalently

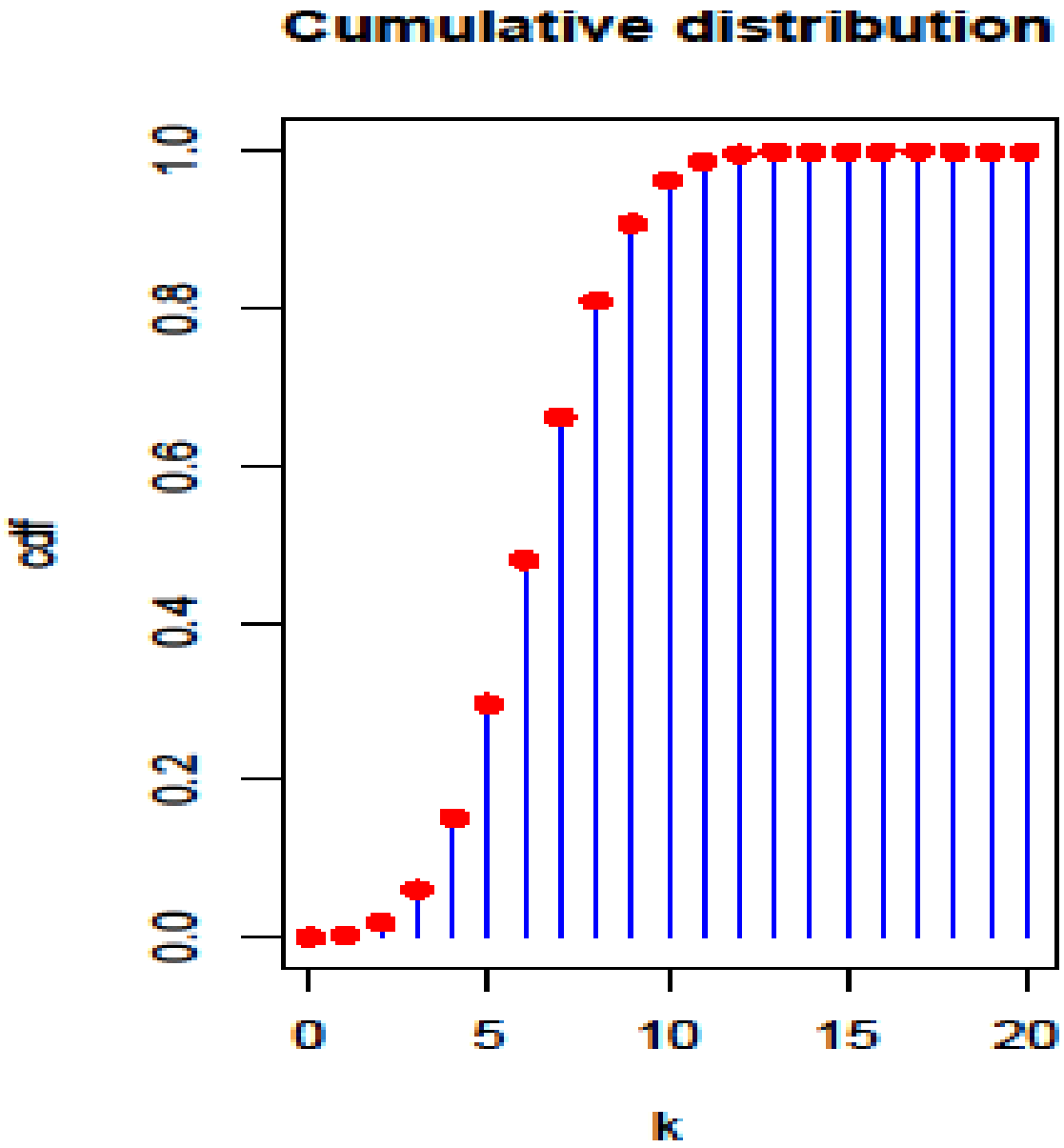
```
1 - pbinom(6, size =10, p=1/2)
[1] 0.1719
```

```
pbinom(6,size=10,p=1/2, lower.tail=FALSE) # k = 6 not 7!
[1] 0.1719
```

Here we used lower.tail = FALSE to get the upper tail of the distribution. The default is lower.tail = TRUE.

To plot the cdf we use the following parameters

```
n <- 20; p <- 1/3
cdf <- pbinom(0:20, size=n, prob=p)
plot(0:20, cdf, type= "h",
    main= “Cumulative distribution", xlab="k",
    ylab = "cdf", col="blue")
points(0:20, cdf, pch=16, cex=1, col="red")
```



For quantiles we first generate the quantiles vector
then apply the function qbinom().

```
 v <- c(.01,.05,.1,.2,.5,.8,.95,.99)
qbinom(v, size=100, p=1/5)
[1]  0  3  7 17 46 77 93 98

qbinom(v, size=100, p=1/5,lower.tail = FALSE )
[1]   2   7  12  23  54  83  97 100

qbinom(v, size=100, p=1/5,lower.tail = FALSE,log.p = F )
[1]   2   7  12  23  54  83  97 100
```

Finally for 10 binomial random numbers from 1 to 20 we have

```
rbinom(10, size=20, prob=1/4)
 [1] 4 6 5 2 4 6 6 7 4 2
```

Command	What it does
`help(distributions)`	shows documentation on distributions
`dbinom(k,n,p)`	PMF $P(X = k)$ for $X \sim \text{Bin}(n, p)$
`pbinom(x,n,p)`	CDF $P(X \leq x)$ for $X \sim \text{Bin}(n, p)$
`qbinom(a,n,p)`	ath quantile for $X \sim \text{Bin}(n, p)$
`rbinom(r,n,p)`	vector of r i.i.d. $\text{Bin}(n, p)$ r.v.s
`dgeom(k,p)`	PMF $P(X = k)$ for $X \sim \text{Geom}(p)$
`dhyper(k,w,b,n)`	PMF $P(X = k)$ for $X \sim \text{HGeom}(w, b, n)$
`dnbinom(k,r,p)`	PMF $P(X = k)$ for $X \sim \text{NBin}(r, p)$
`dpois(k,r)`	PMF $P(X = k)$ for $X \sim \text{Pois}(r)$
`dbeta(x,a,b)`	PDF $f(x)$ for $X \sim \text{Beta}(a, b)$
`dchisq(x,n)`	PDF $f(x)$ for $X \sim \chi^2_n$
`dexp(x,b)`	PDF $f(x)$ for $X \sim \text{Expo}(b)$
`dgamma(x,a,r)`	PDF $f(x)$ for $X \sim \text{Gamma}(a, r)$
`dlnorm(x,m,s)`	PDF $f(x)$ for $X \sim \mathcal{LN}(m, s^2)$
`dnorm(x,m,s)`	PDF $f(x)$ for $X \sim \mathcal{N}(m, s^2)$
`dt(x,n)`	PDF $f(x)$ for $X \sim t_n$
`dunif(x,a,b)`	PDF $f(x)$ for $X \sim \text{Unif}(a, b)$

The table above gives R commands for working with various named distributions. Commands analogous to `pbinom`, `qbinom`, and `rbinom` work for the other distributions in the table. For example, `pnorm`, `qnorm`, and `rnorm` can be used to get the CDF, quantiles, and random generation for the Normal. For the Multinomial, `dmultinom` can be used for calculating the joint PMF and `rmultinom` can be used for generating random vectors. For the Multivariate Normal, after installing and loading the `mvtnorm` package `dmvnorm` can be used for calculating the joint PDF and `rmvnorm` can be used for generating random vectors.

From binomial to Poisson

For binomial distribution with large n, calculating the distribution mass function is absolutely difficult however when n → infinity, p → 0, np → a constant number lambda, then

Binomial(n, p)= choose(n,k)p^k (1-p)^(n-k) ---> Poisson(lambda) = exp(lambda)lambda^k/factorial(k).

For example for n >= 100 and p <= 0.01, we can use Poisson with lambda = np <= 20 to approximate it.

Problem: Suppose 1 in 1000 light bulbs are defective. Let X denotes the number of defective light bulbs in a group of size 10000. what is the probability that at least 5 of them are defective. Notice that

n >= 100, p=0.001 <=0.01, and n*p=1000*0.001= 10 <= 20.

Before to go for the solution of this problem let us first introduce the Poisson pdf, cdf, quantiles and random number functions. The Poisson distribution can also be used for the number of events in other specified intervals such as distance, area or volume. It is also used in situations where we are counting the number of successes in a particular region or interval of time, and there are a large number of trials, each with a small probability of success. For example, the number of emails one receives in an hour. The pdf is given by

P(X = k) = exp(lamda)lambda^k/factorial(k),

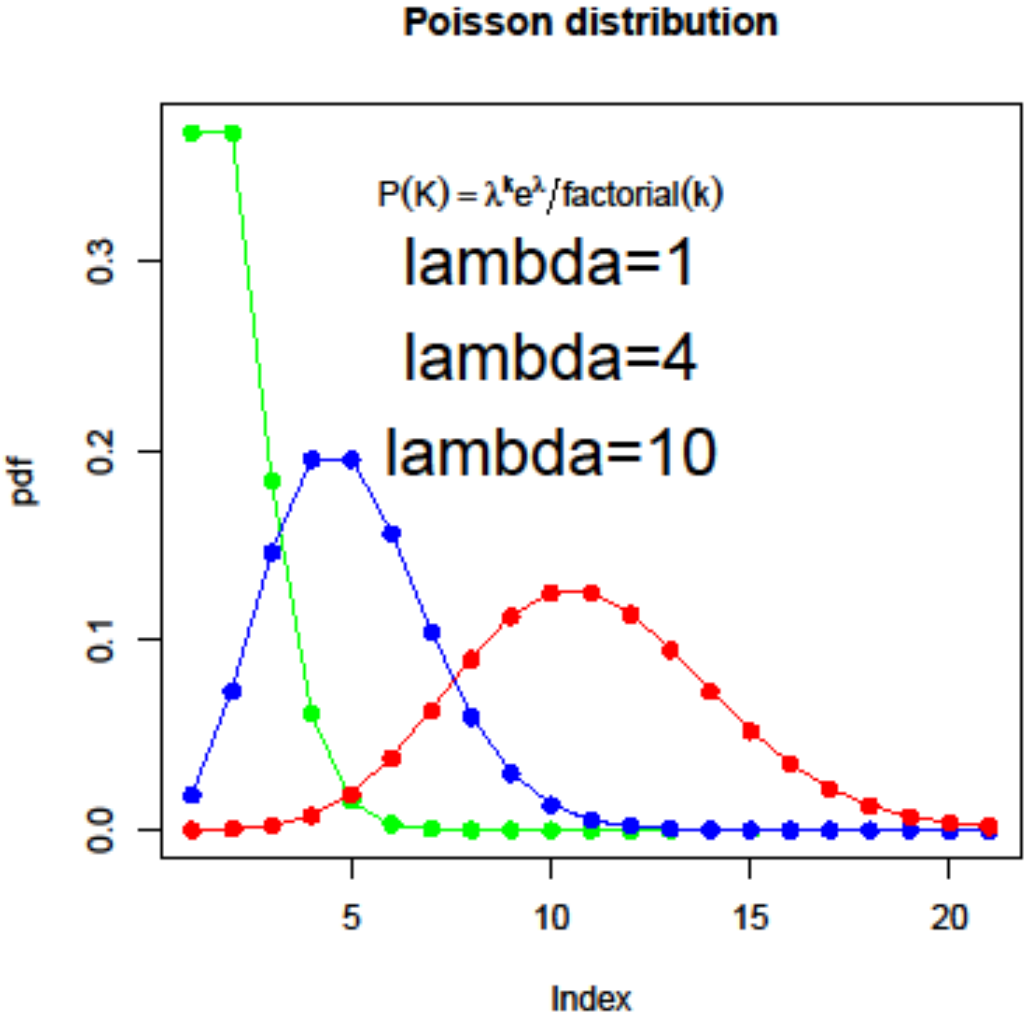
where k=0, 1, 2, …, n,… and lambda is called the rate parameter of the distribution.

The pdf function for Poisson distribution in R is dpois(k, lambda=p)

```
pdf <- dpois(0:20 , lambda=1)
plot(pdf,  pch=16, cex=1,col='green', main= "Poisson distribution" )
text(10, 0.335, expression(P(X=K)==lambda^k*e^lambda/factorial(k)))
lines(pdf,  pch=16, cex=1,col='green',text(10,0.3,"lambda=1", cex=2))

points(pdf,  pch=16, cex=1,col='blue')
lines(pdf,  pch=16, cex=1,col='blue',text(10,.25,"lambda=4",cex=2))
pdf <- dpois(0:20 , lambda=10)
points(pdf,  pch=16, cex=1,col='red')
lines(pdf,  pch=16,cex=1,col='red',text(10,.20,"lambda=10",cex=2))
```

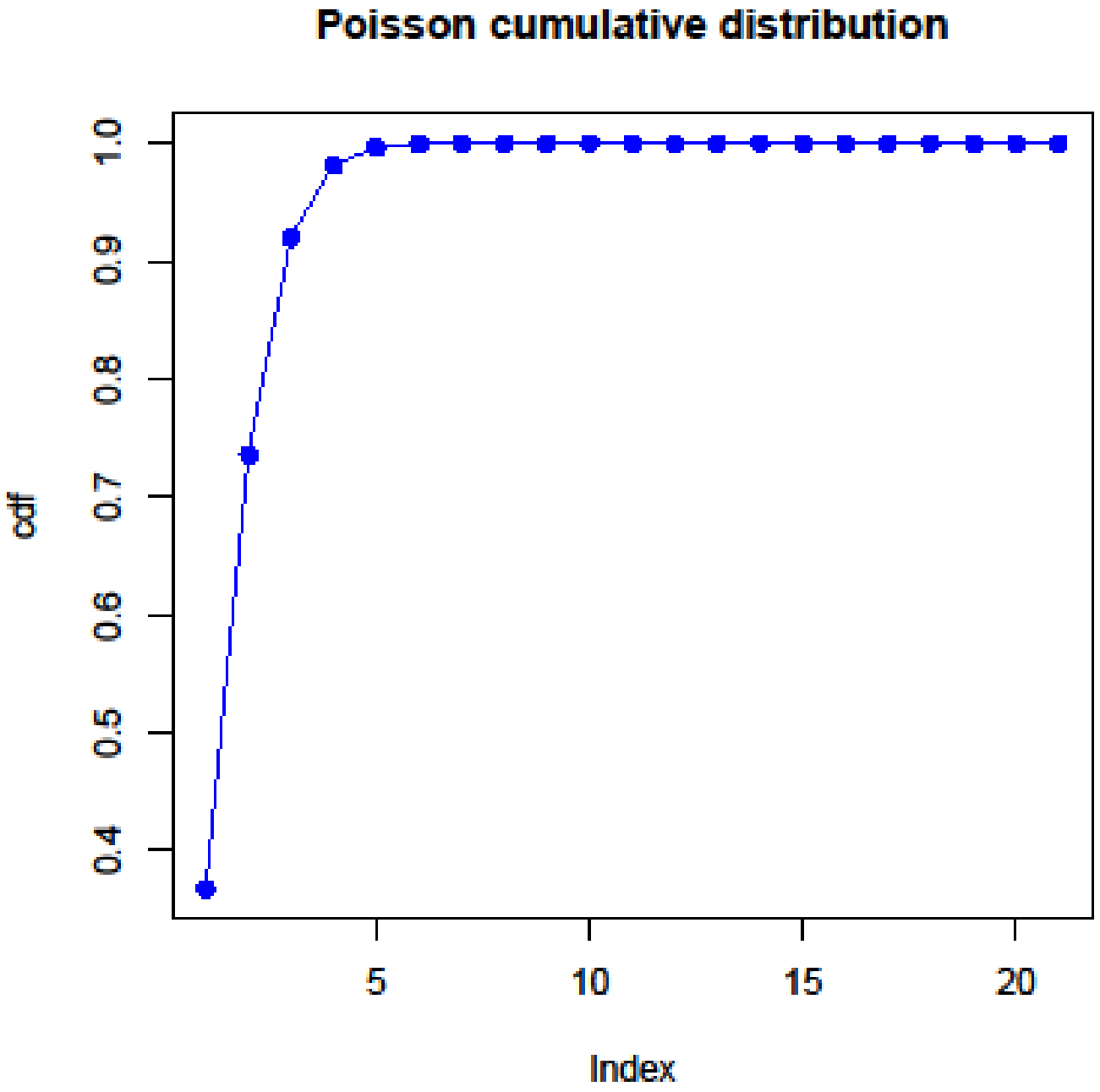
pch Adjust plot symbols (?pch).
cex Adjust size of text and symbols on a graphic
col Adjust color of objects drawn (?colors).
lwd Adjust width of lines drawn.

pch Adjust plot symbols (?pch).
cex Adjust size of text and symbols on a graphic
col Adjust color of objects drawn (?colors).



The cdf function for Poisson distribution in R is ppois(k, lambda=p)

```
cdf <- ppois(0:20 , lambda=1)
plot(cdf,  pch=16, cex=1,col='blue’,
main= "Poisson cumulative distribution" )
lines(cdf,  pch=16, cex=1,col='blue’,
text(10,0.3,"lambda=1", cex=2))
```



Similar to the binomial, we can also create a plot of the Poisson quantile function.
Let's create a sequence of values to which we can apply the qpois() function:

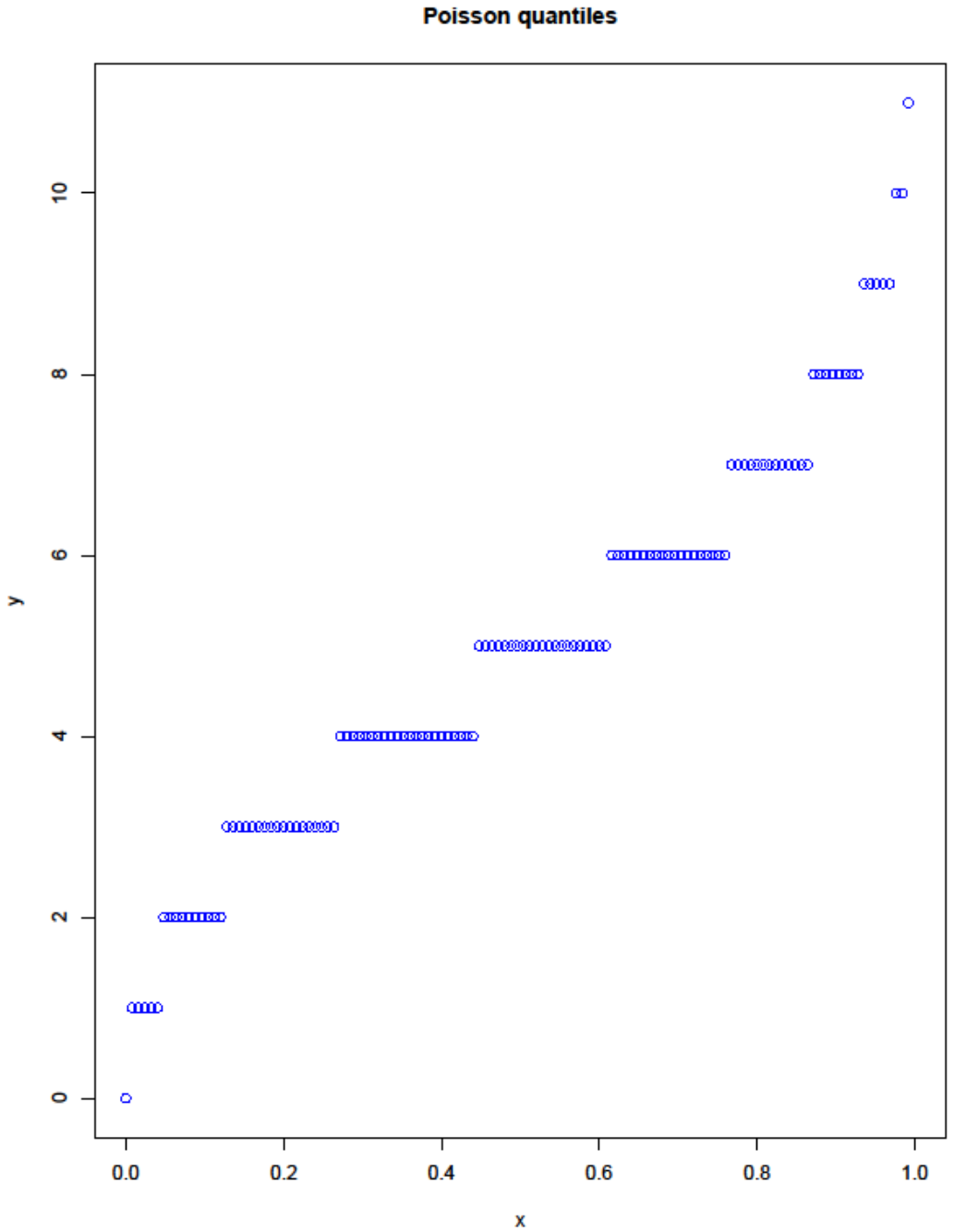
```
x <- seq(0, 1, by = 0.008)   # Specify x-values for qpois function
```

Now, we can apply the qpois function with a lambda of 5 as follows:

```
Y <- qpois(x, lambda = 5)                # Apply qpois function
```

With the plot function, we can illustrate our output:

```
plot(y)                                  # Plot qpois values
```



Finally rpois() returns the Poisson random numbers.

```
rpois(20, lambda = 5)
 [1] 1 5 4 3 5 6 6 5 9 3 2 3 9 4 4 2 4 6 7 8
```

Having seen the 4 different Poisson functions we now turn to solve the defective bulb problem.
According to the probability law \Sum{0}{infinity} P(X = k) =1, we have

(Probability with x <= 4) + (Probability with X >= 5) = 1.

Therefore for lambda = n*p=10, we get

(Probability with X >= 5) = 1- (Probability with x <= 4)

= 1- [Poisson(0) + Poisson(1) + Poisson(2) + Poisson(3) + Poisson(4)]

```
 d <- dpois(0, lambda = 10) + dpois(1, lambda =10) + dpois(2, lambda =10)

                    + dpois(3, lambda =10 ) + dpois(4, lambda = 10)
```

[1] 0.02648329

probability = 1-d
Probability

[1] 0.9972306

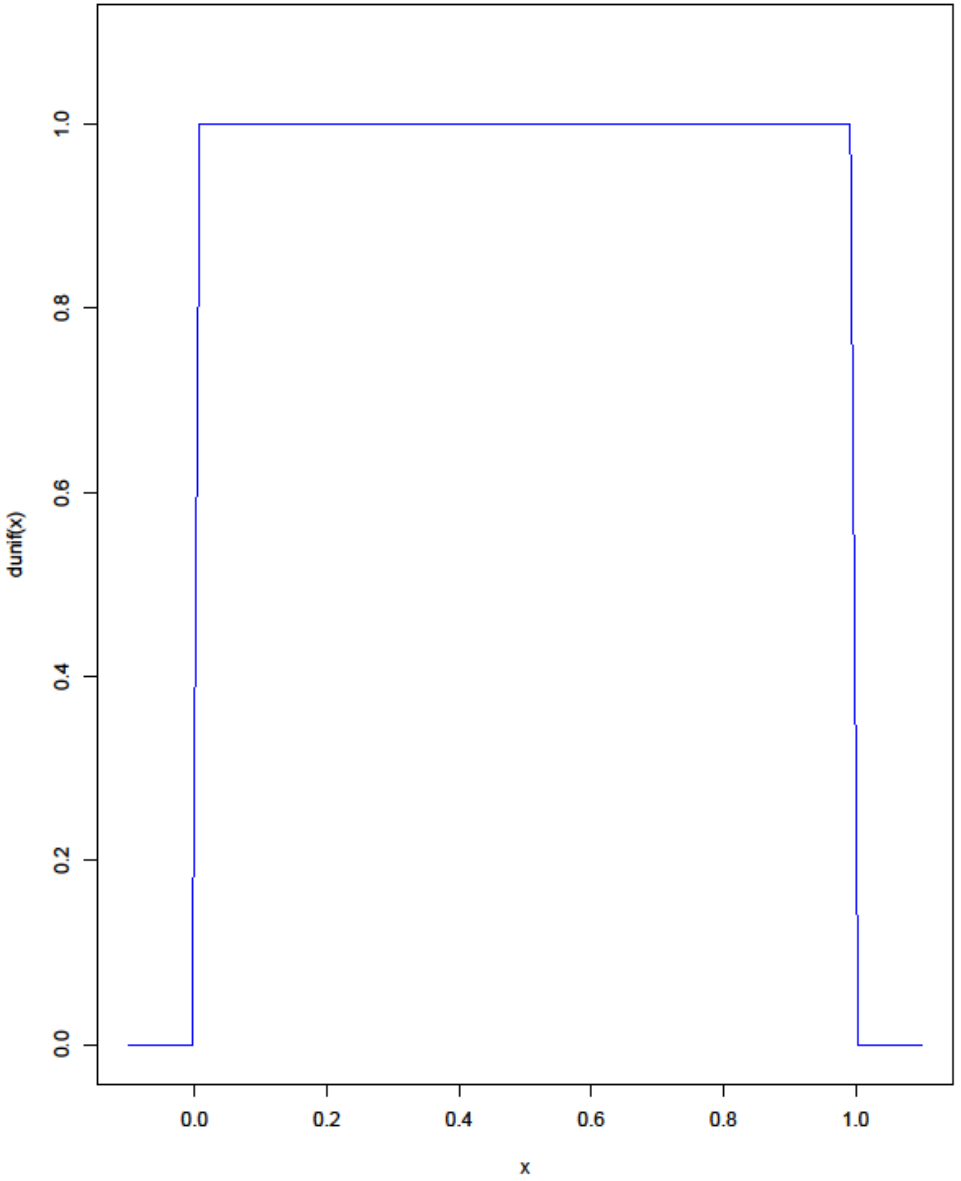
Uniform distribution

sometimes also known as a rectangular distribution, is a distribution t

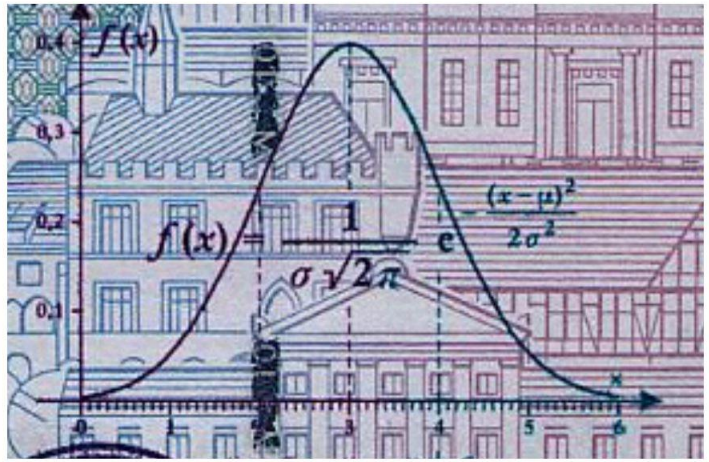
R code:

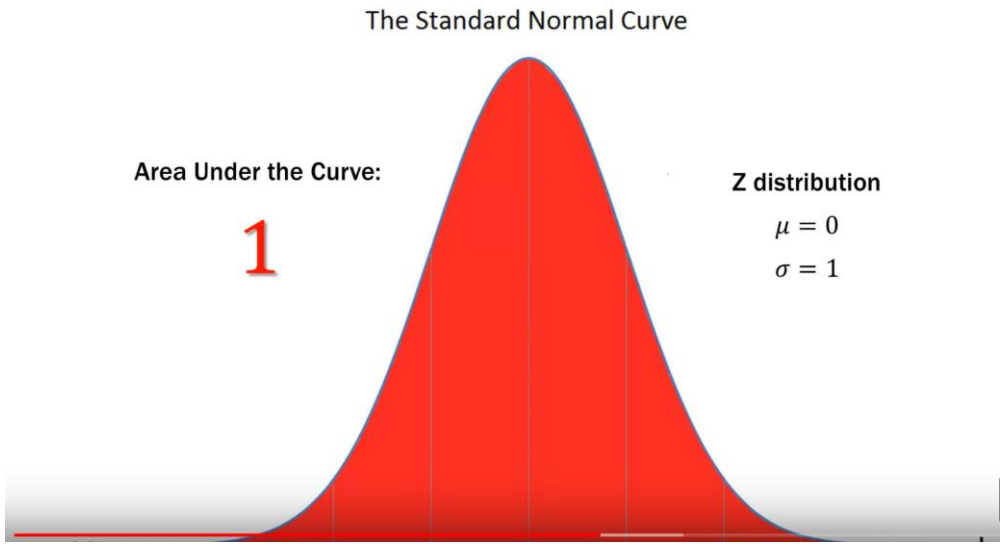
```
x <- runif(100)
d <- density(x)
curve(dunif, -0.1, 1.1, ylim=c(0, max(d$y, 1)), col='blue')
```

The x axis can be taken as time between min and max.



The Gaussian distribution





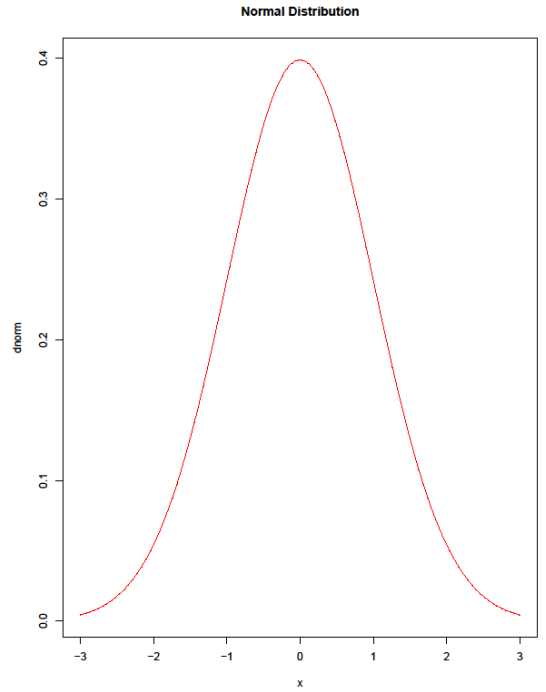
$$\int_{-\infty}^{\infty} e^{-x^2}\,dx = \sqrt{\pi}.$$

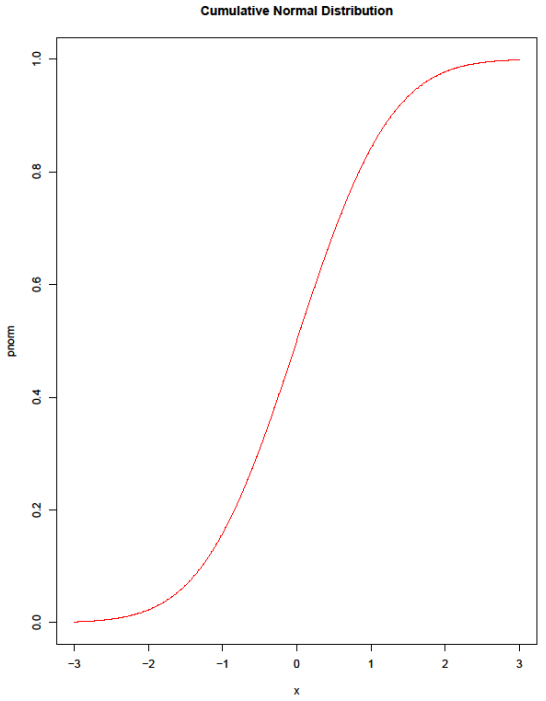
"A mathematician is one to whom *that* is as obvious as that twice two makes four is to you. Liouville was a mathematician."

—Lord Kelvin

Normal or Gaussian Distribution

```
plot(dnorm, -3, 3, main = "Normal Distribution", col='red')
plot(pnorm, -3, 3, main = "Cumulative Normal Distribution", col='red')
# Normal cdf# probability(q <= 0) is .5
pnorm(0, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
[1] 0.5
# lower.tail = TRUE, log.p = FALSE are defaults
norm(0, mean = 0, sd = 1)
[1] 0.5
pnorm(1, mean = 0, sd = 1, lower.tail = FALSE, log.p = FALSE)
[1] 0.1586553
# The area between -1.96 and 1.96
pnorm(1.96,lower.tail=TRUE) - pnorm(-1.96,lower.tail=TRUE)
[1] 0.9500042
# The area between -1 and 1
pnorm(1,lower.tail=TRUE) - pnorm(-1,lower.tail=TRUE)
[1] 0.6826895
# The area between -2 and 2
pnorm(2,lower.tail=TRUE) - pnorm(-2,lower.tail=TRUE)
[1] 0.9544997
# The area between -3 and 3
pnorm(3,lower.tail=TRUE) - pnorm(-3,lower.tail=TRUE)
[1] 0.9973002
```





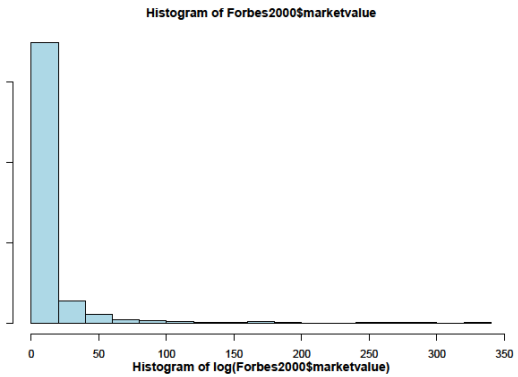
```
# The area between -2.575829 and 2.575829
pnorm(2.575829,lower.tail=TRUE) - pnorm(-2.575829,lower.tail=T)
[1] 0.99
# Normal quantiles
qnorm(0.005, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
[1] -2.575829
qnorm(0.025, mean = 0, sd = 1, lower.tail = F, log.p = FALSE)
[1] 1.959964
# qnorm is the inverse of pnorm
qnorm(pnorm(3))
[1] 3
# The left and right sides of a 95% confidence interval
c(qnorm(.025), qnorm(.975))
[1] -1.959964  1.959964
```

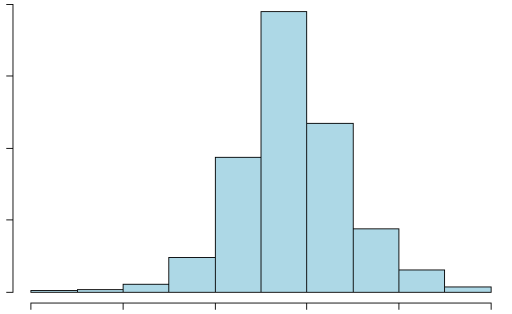
In package(AER), there is a data frame named CPS1985. Let us see the head of CPS1985.

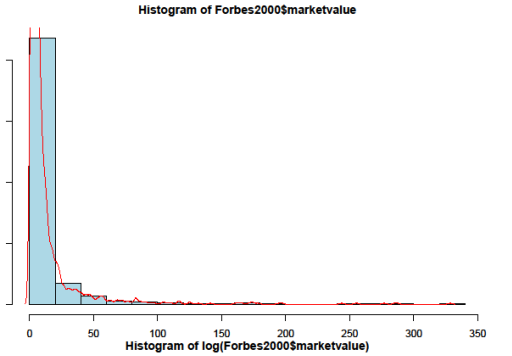
```
head(CPS1985)
     wage education experience age ethnicity region gender occupation       sector union married
1    5.10         8         21  35  hispanic  other female     worker manufacturing    no     yes
1100 4.95         9         42  57      cauc  other female     worker manufacturing    no     yes
2    6.67        12          1  19      cauc  other   male     worker manufacturing    no      no
3    4.00        12          4  22      cauc  other   male     worker        other     no      no
4    7.50        12         17  35      cauc  other   male     worker        other     no     yes
5   13.07        13          9  28      cauc  other   male     worker        other    yes      no

str(CPS1985)
'data.frame':	534 obs. of  11 variables:
 $ wage      : num  5.1 4.95 6.67 4 7.5 ...
 $ education : num  8 9 12 12 12 13 10 12 16 12 ...
 $ experience: num  21 42 1 4 17 9 27 9 11 9 ...
 $ age       : num  35 57 19 22 35 28 43 27 33 27 ...
 $ ethnicity : Factor w/ 3 levels "cauc","hispanic",..: 2 1 1 1 1 1 1 1 1 1 ...
 $ region    : Factor w/ 2 levels "south","other": 2 2 2 2 2 2 1 2 2 2 ...
 $ gender    : Factor w/ 2 levels "male","female": 2 2 1 1 1 1 1 1 1 1 ...
 $ occupation: Factor w/ 6 levels "worker","technical",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ sector    : Factor w/ 3 levels "manufacturing",..: 1 1 1 3 3 3 3 3 1 3 ...
 $ union     : Factor w/ 2 levels "no","yes": 1 1 1 1 1 2 1 1 1 1 ...
 $ married   : Factor w/ 2 levels "no","yes": 2 2 1 1 2 1 1 1 2 1 ...
```

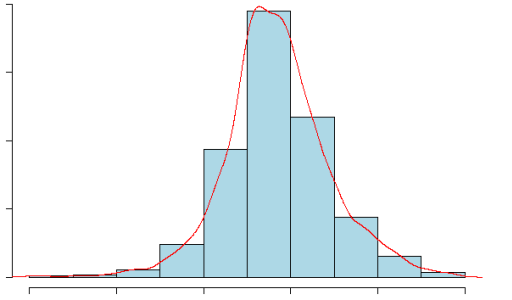
In the code below, mfrow=c(2,1) creates a window of 2 rows and 1 column. Mar=c(1,1,3,3) set the four margins of the window. hist() function plots the histogram of the marketvalue column of the data from Forbes2000. We see that the histogram is skewed to the right. In the next line of the code using the log value one can make the plot symmetric. The final lines, plots the density plots over the histograms.

```
par(mfrow=c(2,1), mar=c(1,1,3,3))
hist(Forbes2000$marketvalue, col='lightblue')
# Right skewed

hist(log(Forbes2000$marketvalue), col='lightblue’)
# Symmetric

hist(Forbes2000$marketvalue, freq = F, col='lightblue')
lines(density(Forbes2000$marketvalue), col='red’)
# Right skewed

hist(log(Forbes2000$marketvalue), freq = F, col='lightblue')
# Symmetric
lines(density(log(Forbes2000$marketvalue)), col='red’)
```



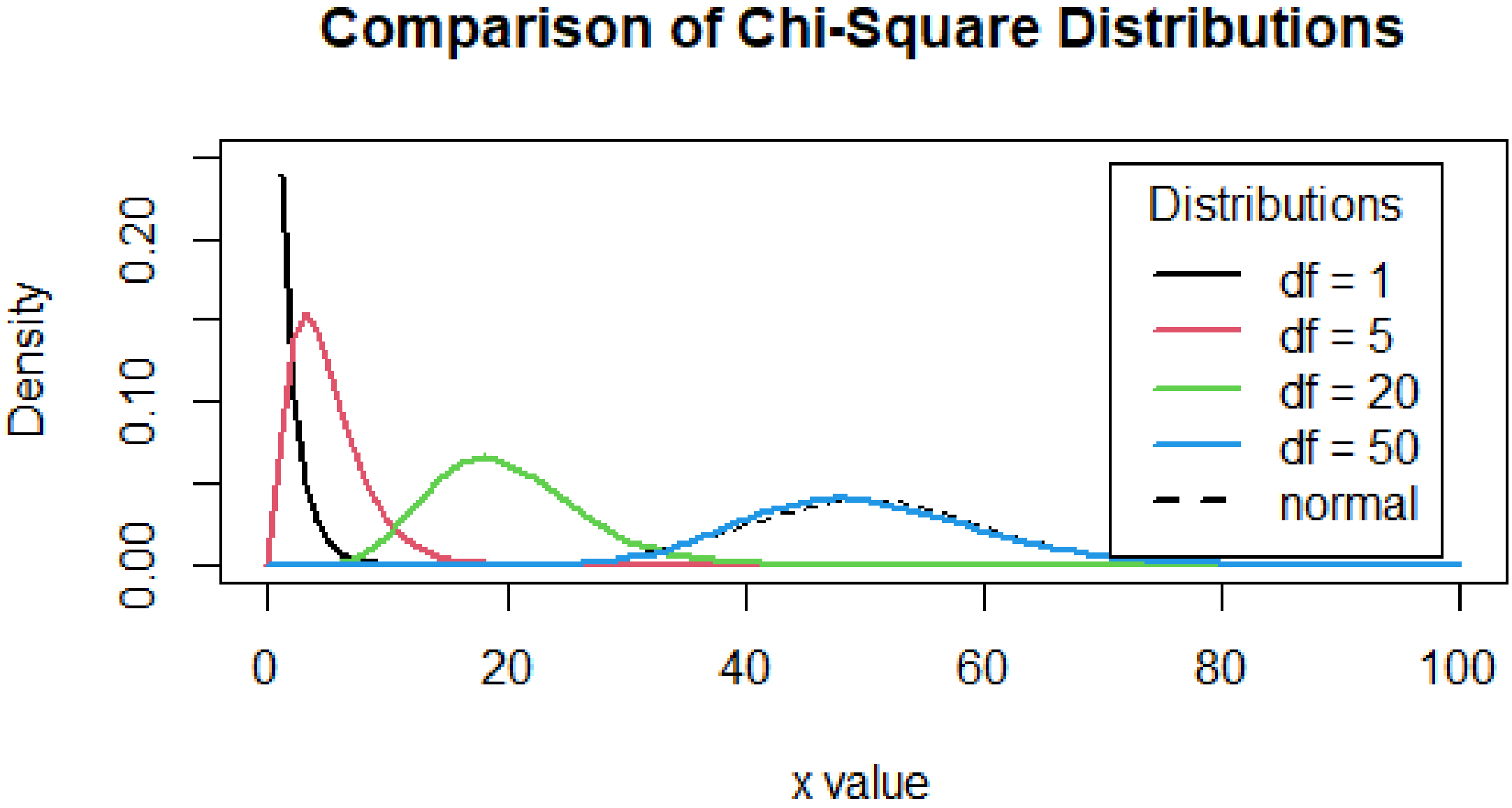






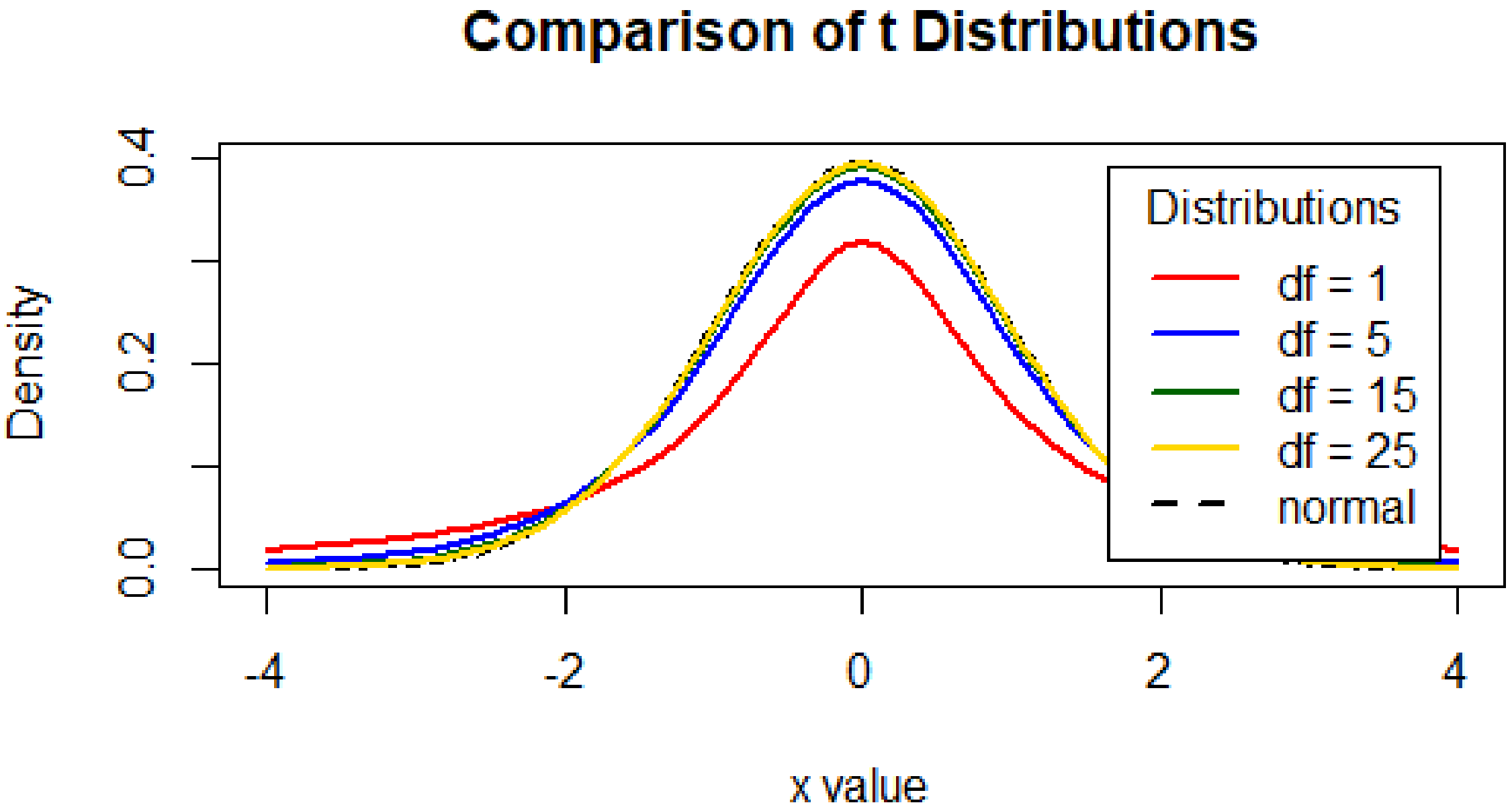
Comparing normal distribution with different chisq distributions

```
x <- seq(0, 100, length = 100)
hx <- dnorm(x,mean = 50,sd = sqrt(2*50)) #normal density
degf <- c(1, 5, 20, 50)
colors = 9:12
#colors <- c("red", "blue", "darkgreen", "gold", "black")
labels <- c("df = 1", "df = 5", "df = 20", "df = 50", "normal")
plot(x, hx, type = "l", lty = 2, xlab = "x value",
    ylab = "Density", main = "Comparison of Chi-Square Distributions",ylim = c(0,0.25))

for (i in 1:4) {
  lines(x, dchisq(x,degf[i]), lwd = 2, col = colors[i]) # dchisq is the chi-square density
}

legend("topright", inset = .05, title = "Distributions",
       labels, lwd = 2, lty = c(1, 1, 1, 1, 2), col = colors)
```


Comparison of Chi-Square Distributions
Density
0.00
0.10
0.20
0
20
40
60
80
100
x value
Distributions
df = 1
df = 5
df = 20
df = 50
normal

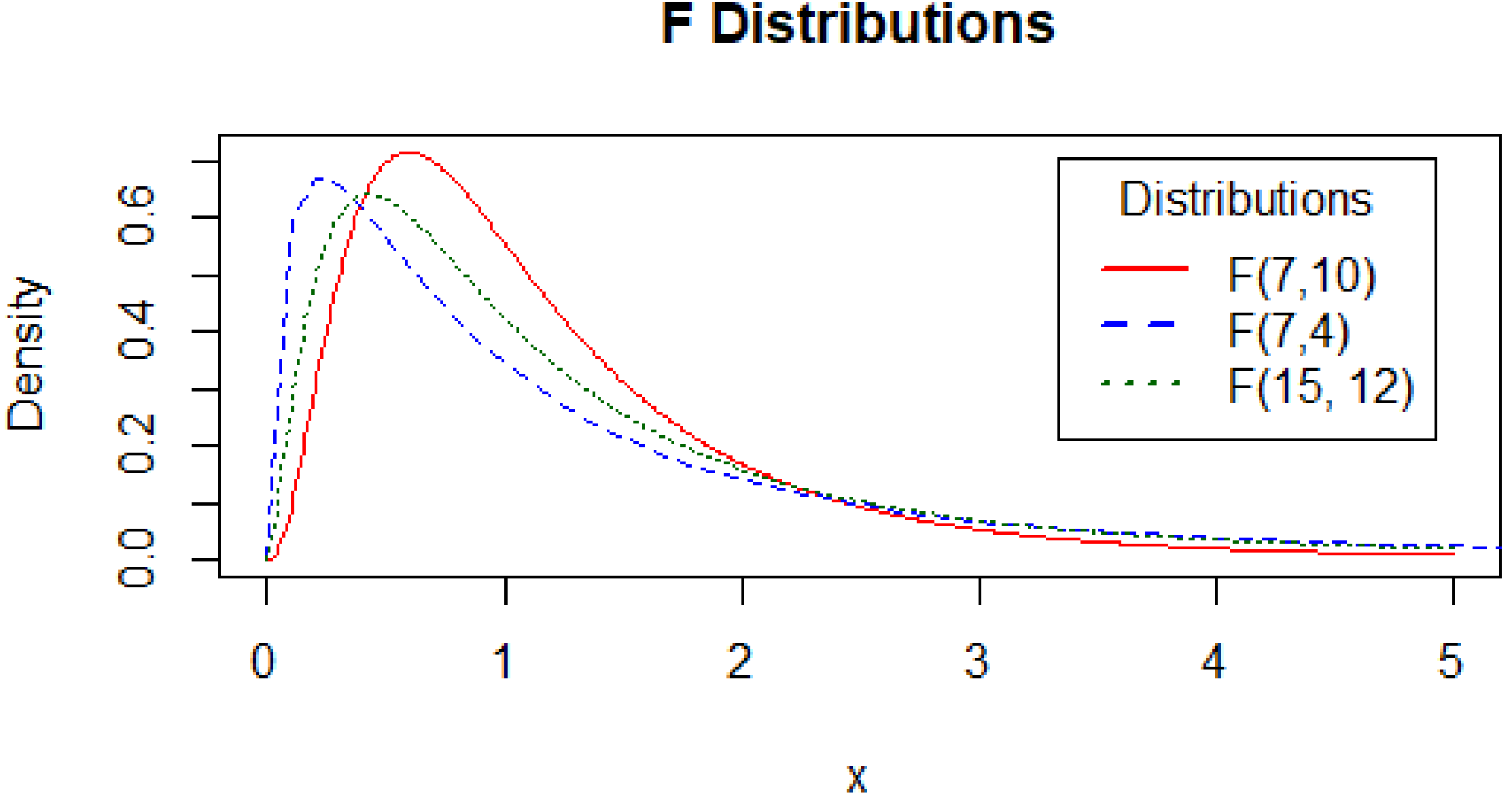
Comparing normal distribution with different t – distributions

```
degf <- c(1, 5, 15, 25)
colors <-  c("red", "blue", "darkgreen", "gold", "black")
labels <- c("df = 1", "df = 5", "df = 15", "df = 25", "normal")

plot(x, hx, type = "l", lty = 2, xlab = "x value",
    ylab = "Density", main = "Comparison of t Distributions", col=1)

for (i in 1:4) {
  lines(x, dt(x,degf[i]), lwd = 2, col = colors[i]) # dt is the t density
}

legend("topright", inset = .05, title = "Distributions",
       labels, lwd = 2, lty = c(1, 1, 1, 1, 2), col = colors)
```


Comparison of t Distributions
Density
0.0
0.2
0.4
-4
-2
0
2
4
x value
Distributions
df = 1
df = 5
df = 15
df = 25
normal

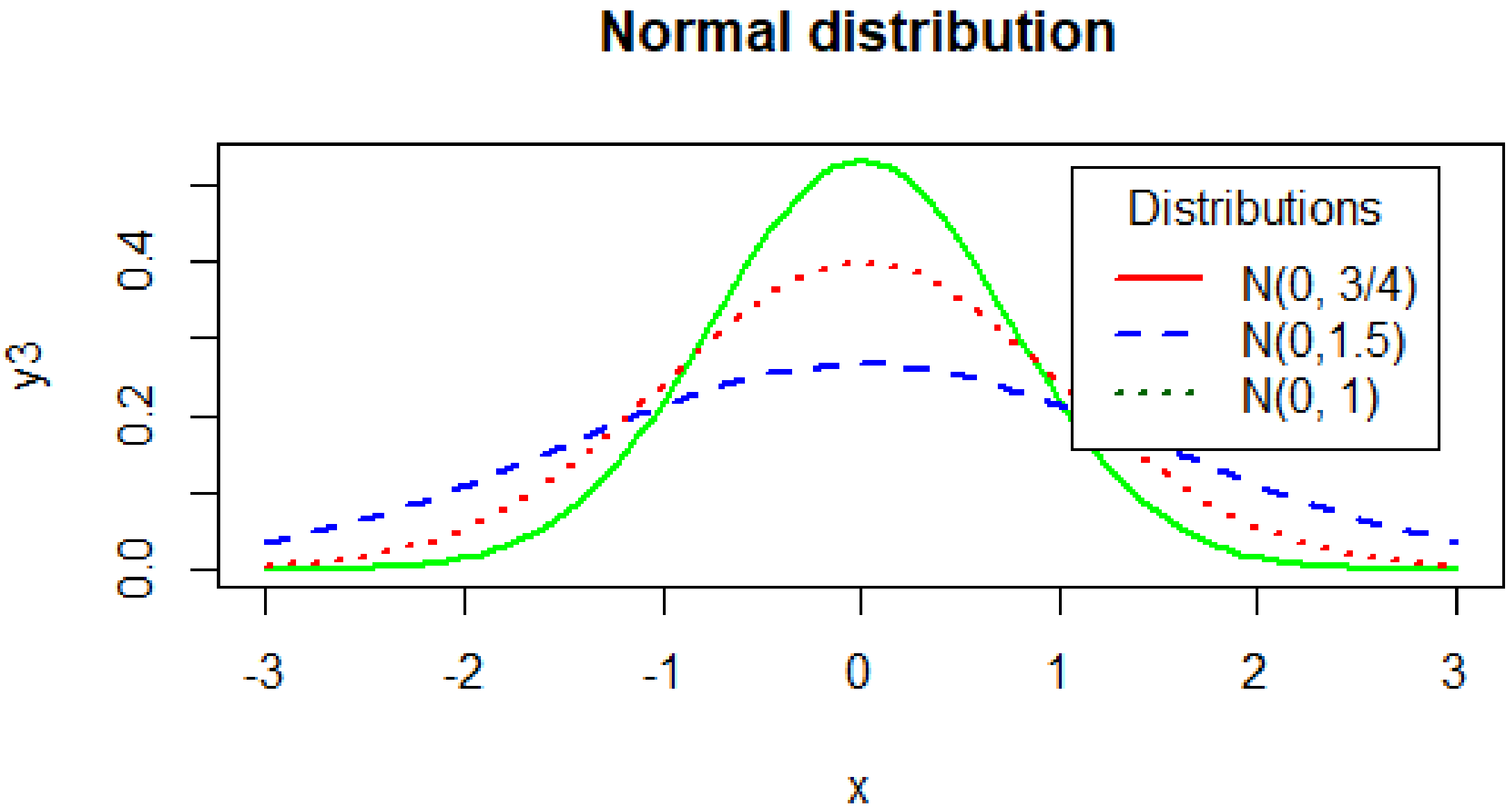
F – distributions with different degrees of freedom

```
colors <- c("red", "blue", "darkgreen")
x <- seq(0,5, by = 0.01)
y <- seq(0,10, by = 0.1)
plot(x, df(x,7,10),type = "l", col = colors[1],
    ylab = "Density",xlab = "x",main = "F Distributions")
lines(y, df(y,3,4),type = "l",lty=2, col = colors[2])
lines(x, df(x,5,5),type = "l",lty = 3,col = colors[3])
labels <- c("F(7,10)","F(7,4)", "F(15, 12)")
legend("topright", inset = .05, title = "Distributions",
      labels, lwd = 2, lty = c(1, 2, 3), col = colors)
```

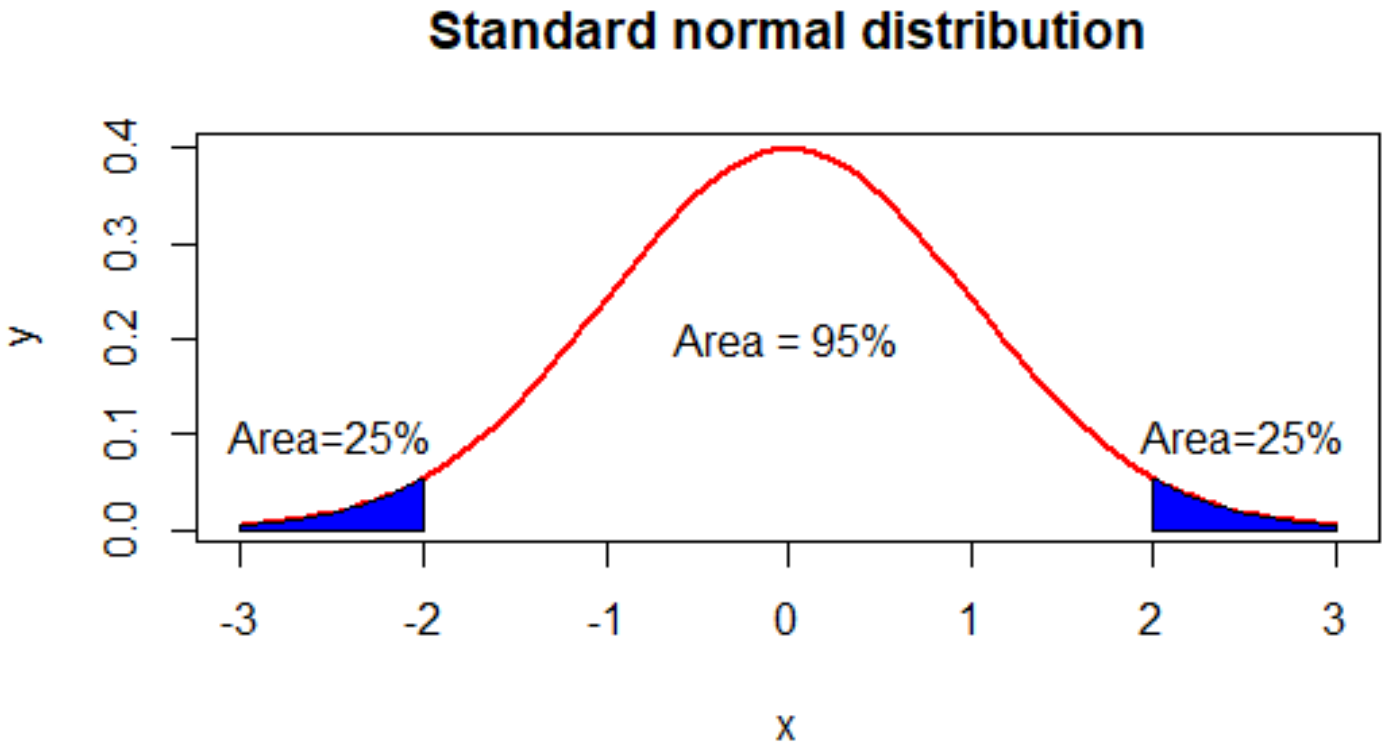

F Distributions
Density
0.0
0.2
0.4
0.6
0
1
2
3
4
5
x
Distributions
F(7,10)
F(7,4)
F(15, 12)

Normal distribution for mean = 0 and three different standard variations

```
set.seed(1234)
x=seq(-3,3,length=100)
y3=dnorm(x,mean=0,sd=3/4)
plot(x,y3,type="l",lwd=2,col="green", main="Normal distribution")
y2=dnorm(x,mean=0,sd=1.5)
lines(x,y2,type="l",lwd=2,col="blue", lty=2)
y1=dnorm(x,mean=0,sd=1)
lines(x,y1,type="l",lwd=2,col="red", lty=3)
legend("topright",c("sigma=1/2","sigma=2","sigma=1"),
       lty=c(1,1,1),col=c("green","blue","red"))

labels <- c("N(0, 3/4)","N(0,1.5)", "N(0, 1)")
legend("topright", inset = .05, title = "Distributions",
       labels, lwd = 2, lty = c(1, 2, 3), col = colors)
```

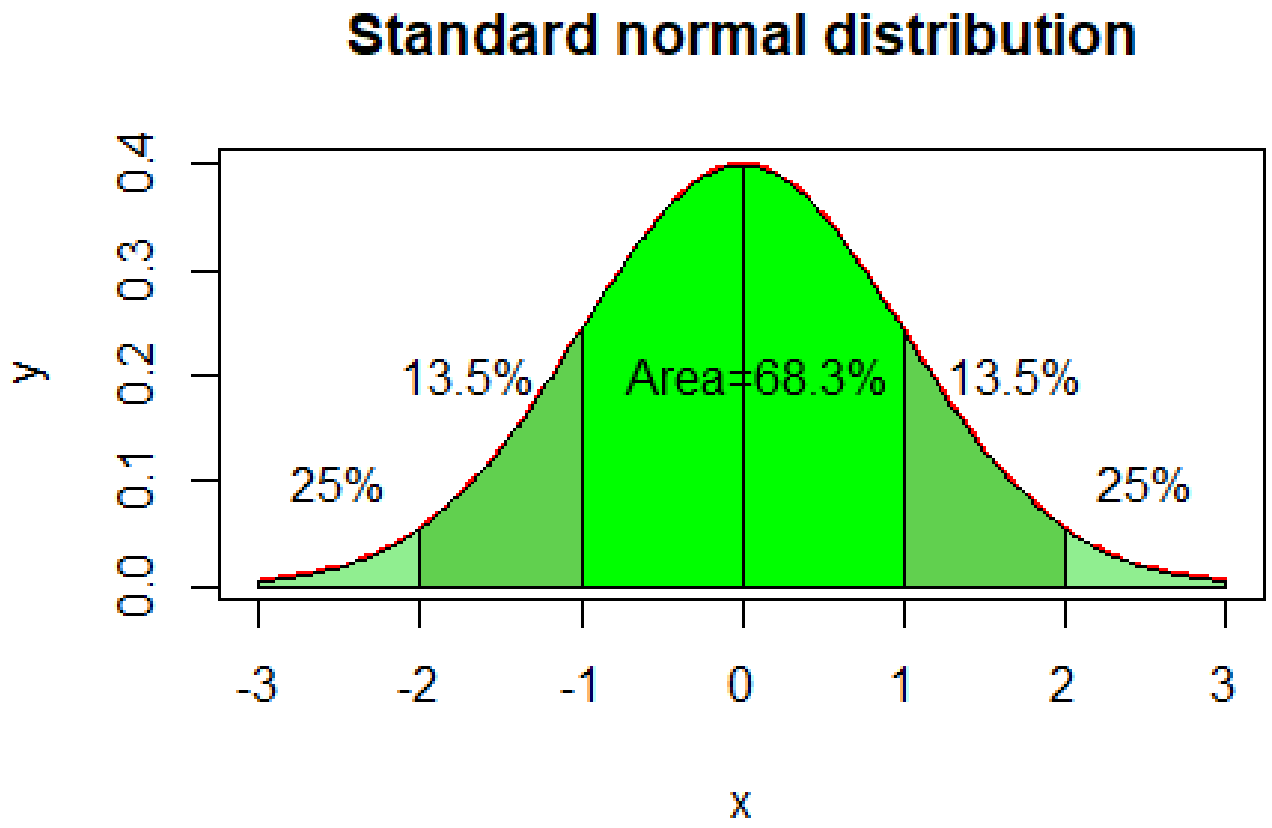

Normal distribution
y3
0.0
0.2
0.4
-3
-2
-1
0
1
2
3
x
Distributions
N(0, 3/4)
N(0,1.5)
N(0, 1)

```
pnorm(1.96, mean=0, sd=1)- pnorm(-1.96, mean=0, sd=1) #area under bell curve at z<= 1.96
[1] 0.95
x=seq(-3,3,length=100)
y=dnorm(x,mean=0,sd=1)
plot(x,y,type="l",lwd=2,col="red", main= 'Standard normal distribution')
x=seq(-3,-2,length=50)
z=seq(2,3,length=50)
y=dnorm(x,mean=0,sd=1)
w=dnorm(z,mean=0,sd=1)
polygon(c(-3,x,-2),c(0,y,0),col="blue")
polygon(c(2,z,3),c(0,w,0),col="blue")
text(-2.5, 0.1, "Area=25%")
text(2.5, 0.1, "Area=25%")
text(0, 0.2, "Area = 95%")
```



68.3 - 95.4 - 99.7 rule of thumb

```
x=seq(-3,3,length=100); y=dnorm(x,mean=0,sd=1)
plot(x,y,type="l",lwd=2,col="red", main= 'Standard normal distribution')
x=seq(-3,-2,length=50); z=seq(2,3,length=50)
y=dnorm(x,mean=0,sd=1); w=dnorm(z,mean=0,sd=1)
polygon(c(-3,x,-2),c(0,y,0),col='lightgreen')
polygon(c(2,z,3),c(0,w,0),col='lightgreen')
x1=seq(-2,-1,length=50); z1=seq(1,2,length=50)
y1=dnorm(x1,mean=0,sd=1); w1=dnorm(z1,mean=0,sd=1)
polygon(c(-2,x1,-1),c(0,y1,0),col=139)
polygon(c(1,z1,2),c(0,w1,0),col=139)
x2=seq(-1,0,length=50); z2=seq(0,1,length=50)
y2=dnorm(x2,mean=0,sd=1); w2=dnorm(z2,mean=0,sd=1)
polygon(c(-1,x2,0),c(0,y2,0),col='green')
polygon(c(0,z2,1),c(0,w2,0),col='green')
text(0.1, 0.2, "Area=68.3%")
text(-2.5, 0.1, "25%"); text(2.5, 0.1, "25%")
text(-1.7, 0.2, "13.5%"); text(1.7, 0.2, "13.5%")
```

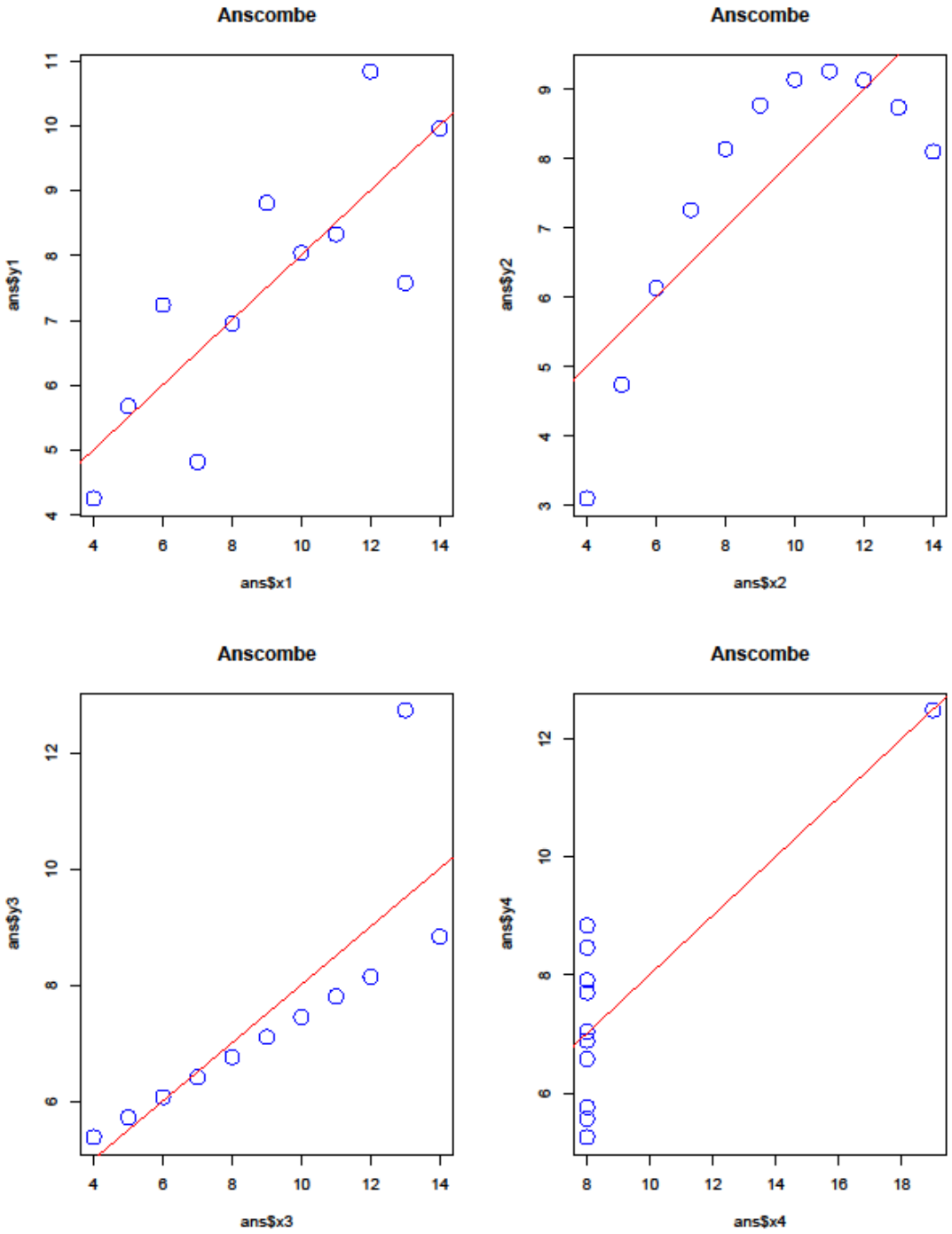



```
pnorm(3, mean=0, sd=1)- pnorm(-3, mean=0, sd=1)
[1] 0.997
pnorm(2, mean=0, sd=1)- pnorm(-2, mean=0, sd=1)
[1] 0.954
pnorm(1, mean=0, sd=1)- pnorm(-1, mean=0, sd=1)
[1] 0.683
```

Anscombe's quartet as an example of many similar statistics

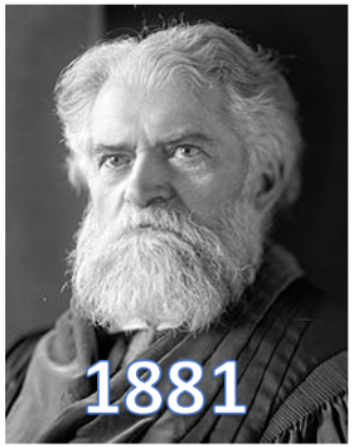
Data is available from:
https://github.com/farzaliizadi/data-sets
All four dataset have the same
Mean over x axis
Mean over y axis
regression line
sum of squared residuals

```
par(mfrow=c(2,2))   # divide the plots window into
2 rows and 2 columns.
lm(ans$y1~ans$x1)  # lm stands for linear model
plot(ans$x1,ans$y1, col='blue', main='Anscombe')
abline(a=3.0025, b=0.4997, col='red')
lm(ans$y2~ans$x2)
plot(ans$x2,ans$y2, col='blue',main='Anscombe')
abline(a=3.0025, b=0.5, col='red')
lm(ans$y3~ans$x3)
plot(ans$x3,ans$y3, col='blue', main='Anscombe')
abline(a=3.0025, b=0.5, col='red')
lm(ans$y4~ans$x4)
plot(ans$x4,ans$y4, col='blue', main='Anscombe')
abline(a=3.0017, b=0.4999, col='red')
```



Newcomb and Benford

- "That the ten digits do not occur with equal frequency must be evident to any one making much use of logarithmic tables, and noticing how much faster the first pages wear out than the last ones." (Newcomb, 1881)
- Benford observed the first digit of numbers in 20 different datasets.



Benford's law for the first digit

```
benlaw <- function(d) log10(1 + 1 / d)
benlaw(1)
[1] 0.30103

df <- data.frame(digit = 1:9, probability = benlaw(1:9))
ggplot(df, aes(x = digit, y = probability)) +
geom_bar(stat = "identity", fill = "dodgerblue") +
xlab("First digit") + ylab("Expected frequency") +
scale_x_continuous(breaks = 1:9, labels = 1:9) +
ylim(0, 0.33) + theme(text = element_text(size = 25))
```