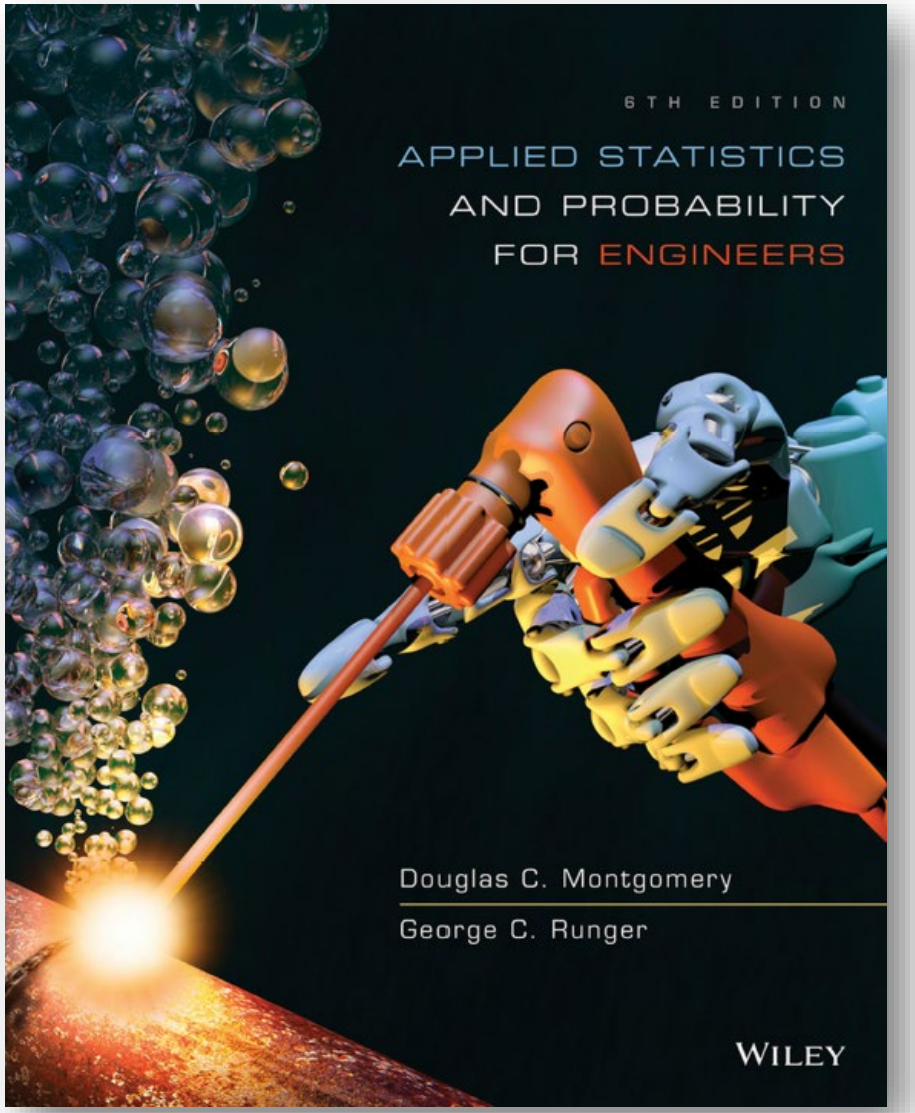


Probability and Statistics

King Faisal University

College of Science

Department of Mathematics and Statistics

Math 121

Dr. Reda Alshenawy

The Role of Statistics in Engineering:

The concepts of probability and statistics are powerful ones and contribute extensively to the solutions of many types of engineering problems.

We will encounter many examples of these applications during this semester.

Definition: Statistics is a science that Help us make decisions and draw conclusions in the presence of variability.

Example:

Civil engineers working in the transportation field are concerned about the capacity of regional highway systems. A typical problem related to transportation would involve data regarding this specific systems of non-work, home-based trips, the number of persons per household, and the number of vehicles per household. The objective would be to construct a model (based on the data) that relates the number of persons and the number of vehicles, regression analysis is used to produce such a model.

Application of Statistics in Computer Science:

There are a number of ways the roles of statisticians and computer scientists merge; consider the development of models and data mining. Typically, statistical approach to models tends to involve stochastic (random) models with prior knowledge of the data. The computer science approach, on the other hand, leans more to algorithmic models without prior knowledge of the data. Ultimately, these come together in attempts to solve problems.

Data mining processes for computer science have statistical counter arts.

Consider the following:

Steps in Computer Science	Steps in Statistics
Data collection/enrichment	Experimental design for the collection of data/noise reduction
Data exploration	classifying the distribution/variability
Analysis and Modeling	Group differences, dimension reduction; prediction; classification
Representation and Reporting	Visualization; communication

How else is statistics used in computer science?

Simulations (used to gain a greater understanding of a variety of systems) are truly a marriage of computing capability and statistics—the use of statistics within programming improves understanding of the underlying system leading to more meaningful results.

Statistics in software engineering leads to more conclusive determinations of quality and optimal performance.

The Engineering Method and Statistical Thinking

- An engineer is someone who solves problems of interest to society by the efficient application of scientific principles.
- The engineering, or scientific, method is the approach to formulating and solving these problems.

The steps in the engineering method are as follows:

1. Develop a clear and concise description of the problem.
2. Identify, the important factors that affect this problem or that may play a role in its solution.
3. Propose a model for the problem, using scientific or engineering knowledge of the phenomenon being studied. State any limitations or assumptions of the model.
4. Conduct appropriate experiments and collect data to test or validate the model or conclusions made in steps 2 and 3.

Cont.

5. Refine the model on the basis of the observed data.
6. Manipulate the model to assist in developing a solution to the problem.
7. Conduct an appropriate experiment to confirm that the proposed solution to the problem is both effective and efficient.
8. Draw conclusions or make recommendations based on the problem solution.

Cont.

- Engineers must know how to efficiently plan experiments, collect data, analyze and interpret the data, and understand how the observed data relate to the model they have proposed for the problem under study.
- The field of **statistics** deals with the collection, presentation, analysis, and use of data to make decisions, solve problems, and design products and processes. In simple terms, **statistics is the science of data.**

Variability and statistical thinking

- Statistical methods are used to help us describe and understand **variability**. By variability, we mean that successive observations of a system or phenomenon do not produce exactly the same result.
- **Statistical thinking** can give us a useful way to incorporate this variability into our decision-making processes.

Collecting Engineering Data

There are three basic methods of collecting data:

- **A retrospective study (using historical data):** use either all or a sample of the historical process data archived over some period of time.
- **An observational study:** the engineer observes the process or population, disturbing it as little as possible, and records the quantities of interest.

Cont.

- **A designed experiment:**

  the engineer makes *deliberate or purposeful changes in the controllable* variables of the system or process, observes the resulting system output data, and then makes an inference or decision about which variables are responsible for the observed changes in output performance.

Mechanistic and Empirical Models

Mechanistic model: is built from our underlying knowledge of the basic physical mechanism that relates these variables. However, if we performed this measurement process more than once, perhaps at different times, or even on different days, the observed current could differ slightly because of small changes or variations in factors that are not completely controlled.

Empirical model: uses our engineering and scientific knowledge of the phenomenon, but it is not directly developed from our theoretical or first-principles understanding of the underlying mechanism.

Introduction to R

R is a language and an environment for statistical computing and graphics. R is used to store, manage, and manipulate data; to create graphical displays of data using different graphical systems; to analyse data using standard statistical procedures; and to perform simulations. In short, everything a student or scientist may want or need to do with data can be done with R.

For the reader who wants his documents to update (graphs, tables, statistics, etc.)

Installing R

Visit one of the links below to download the latest version of R for your operating system:

- Microsoft Windows: http://cran.r-project.org/bin/windows/base/
- MacOS: http://cran.r-project.org/bin/macosx/
- Linux: http://cran.r-project.org/bin/linux/

R-4.4.2 for Windows

Download R-4.4.2 for Windows (83 megabytes, 64 bit)

README on the Windows binary distribution
New features in this version

This build requires UCRT, which is part of Windows since Windows 10 and Windows Server 2016. On older systems, UCRT has to be installed manually from here.

If you want to double-check that the package you have downloaded matches the package distributed by CRAN, you can compare the md5sum of the .exe to the fingerprint on the master ser

Frequently asked questions

- Does R run under my version of Windows?
- How do I update packages in my previous version of R?

Please see the R FAQ for general information about R and the R Windows FAQ for Windows-specific information.

Other builds

- Patches to this release are incorporated in the r-patched snapshot build.
- A build of the development version (which will eventually become the next major release of R) is available in the r-devel snapshot build.
- Previous releases

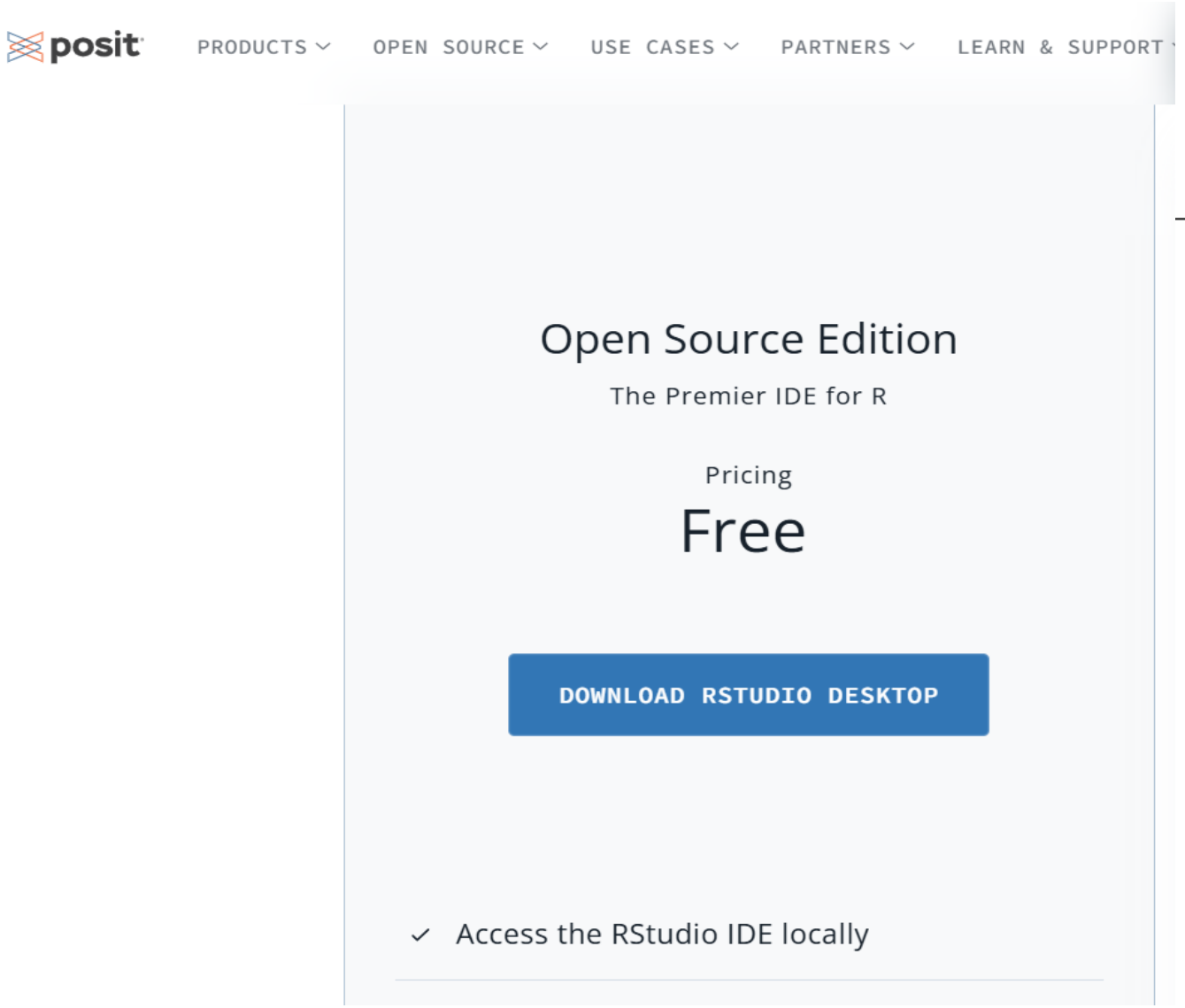
Note to webmasters: A stable link which will redirect to the current Windows binary release is <CRAN MIRROR>/bin/windows/base/release.html.

Last change: 2024-11-01

R Studio

R Studio is an application like Microsoft Word—except that instead of helping you write in English, R Studio helps you write in R. I use R Studio throughout the book because it makes using R much easier. Also, the R Studio interface looks the same for Windows, Mac OS, and Linux. That will help me match the book to your personal experience.

You can https://posit.co/products/open-source/rstudio/ for free. Just click the “Download R Studio” button and follow the simple instructions that follow. Once you’ve installed R Studio, you can open it like any other program on your computer—usually by clicking an icon on your desktop.


posit
PRODUCTS
OPEN SOURCE
USE CASES
PARTNERS
LEARN & SUPPORT
Open Source Edition
The Premier IDE for R
Pricing
Free
DOWNLOAD RSTUDIO DESKTOP
Access the RStudio IDE locally

Basic R Operations and Concepts

The R developers have written an introductory document entitled “An Introduction to R”. There is a sample session included which shows what basic interaction with R looks like. I recommend that all new users of R read that document, but bear in mind that there are concepts mentioned which will be unfamiliar to the beginner.

Below are some of the most basic operations that can be done with R. Almost every book about R begins with a section like the one below; look around to see all sorts of things that can be done at this most basic level.

2.3.1 Arithmetic

```
> 2 + 3          # add

[1] 5

> 4 * 5 / 6      # multiply and divide

[1] 3.333333

> 7^8            # 7 to the 8th power

[1] 5764801
```

Notice the comment character #. Anything typed after a # symbol is ignored by R. We know that 20/6 is a repeating decimal, but the above example shows only 7 digits. We can change the number of digits displayed with `options`:

```
> options(digits = 16)
> 10/3                    # see more digits

[1] 3.333333333333333

> sqrt(2)                 # square root

[1] 1.414213562373095

> exp(1)                  # Euler's constant, e

[1] 2.718281828459045

> pi

[1] 3.141592653589793

> options(digits = 7)     # back to default
```

Note that it is possible to set `digits` up to 22, but setting them over 16 is not recommended (the extra significant digits are not necessarily reliable). Above notice the `sqrt` function for square roots and the `exp` function for powers of e, Euler's number.

2.3.2 Assignment, Object names, and Data types

It is often convenient to assign numbers and values to variables (objects) to be used later. The proper way to assign values to a variable is with the <- operator (with a space on either side). The = symbol works too, but it is recommended by the R masters to reserve = for specifying arguments to functions (discussed later). In this book we will follow their advice and use <- for assignment. Once a variable is assigned, its value can be printed by simply entering the variable name by itself.

```
> x <- 7*41/pi    # don't see the calculated value
> x               # take a look

[1] 91.35494
```

When choosing a variable name you can use letters, numbers, dots ".", or underscore "_" characters. You cannot use mathematical operators, and a leading dot may not be followed by a number. Examples of valid names are: `x`, `x1`, `y.value`, and `y_hat`. (More precisely, the set of allowable characters in object names depends on one's particular system and locale; see An Introduction to R for more discussion on this.)

Objects can be of many *types*, *modes*, and *classes*. At this level, it is not necessary to investigate all of the intricacies of the respective types, but there are some with which you need to become familiar:

- integer: the values 0, ±1, ±2, . . . ; these are represented exactly by R.
- double: real numbers (rational and irrational); these numbers are not represented exactly (save integers or fractions with a denominator that is a power of 2, see [212]).
- character: elements that are wrapped with pairs of " or ';
- logical: includes `TRUE`, `FALSE`, and `NA` (which are reserved words); the `NA` stands for "not available", *i.e.*, a missing value.

You can determine an object's type with the `typeof` function. In addition to the above, there is the `complex` data type:

```
> sqrt(-1)                # isn't defined
[1] NaN
> sqrt(-1+0i)             # is defined
[1] 0+1i
> sqrt(as.complex(-1))    # same thing
[1] 0+1i
> (0 + 1i)^2              # should be -1
[1] -1+0i
> typeof((0 + 1i)^2)
[1] "complex"
```

Note that you can just type `(1i)^2` to get the same answer. The `NaN` stands for "not a number"; it is represented internally as `double` .

2.3.3 Vectors

All of this time we have been manipulating vectors of length 1. Now let us move to vectors with multiple entries.

Entering data vectors

The long way: If you would like to enter the data `74,31,95,61,76,34,23,54,96` into R, you may create a data vector with the `c` function (which is short for *concatenate*).

```
> x <- c(74, 31, 95, 61, 76, 34, 23, 54, 96)
> x

[1] 74 31 95 61 76 34 23 54 96
```

The elements of a vector are usually coerced by R to the the most general type of any of the elements, so if you do `c(1, "2")` then the result will be `c("1", "2")`.

A shorter way: The `scan` method is useful when the data are stored somewhere else. For instance, you may type `x <- scan()` at the command prompt and R will display `1:` to indicate that it is waiting for the first data value. Type a value and press `Enter`, at which point R will display `2:`, and so forth. Note that entering an empty line stops the scan. This method is especially handy when you have a column of values, say, stored in a text file or spreadsheet. You may copy and paste them all at the `1:` prompt, and R will store all of the values instantly in the vector `x`.

Repeated data; regular patterns: the `seq` function will generate all sorts of sequences of numbers. It has the arguments `from`, `to`, `by`, and `length.out` which can be set in concert with one another. We will do a couple of examples to show you how it works.

```
> seq(from = 1, to = 5)

[1] 1 2 3 4 5

> seq(from = 2, by = -0.1, length.out = 4)

[1] 2.0 1.9 1.8 1.7
```

Note that we can get the first line much quicker with the colon operator.

```
> 1:5

[1] 1 2 3 4 5
```

The vector `LETTERS` has the 26 letters of the English alphabet in uppercase and `letters` has all of them in lowercase.

Indexing data vectors

Sometimes we do not want the whole vector, but just a piece of it. We can access the intermediate parts with the [] operator. Observe (with x defined above)

```
> x[1]

[1] 74

> x[2:4]

[1] 31 95 61
```

```
> x[c(1,3,4,8)]

[1] 74 95 61 54

> x[-c(1,3,4,8)]

[1] 31 76 34 23 96
```

Notice that we used the minus sign to specify those elements that we do *not* want.

```
> LETTERS[1:5]

[1] "A" "B" "C" "D" "E"

> letters[-(6:24)]

[1] "a" "b" "c" "d" "e" "y" "z"
```

2.3.4 Functions and Expressions

A function takes arguments as input and returns an object as output. There are functions to do all sorts of things. We show some examples below.

```
> x <- 1:5
> sum(x)

[1] 15

> length(x)

[1] 5

> min(x)

[1] 1

> mean(x)       # sample mean

[1] 3

> sd(x)         # sample standard deviation

[1] 1.581139
```

2.4 Getting Help

When you are using R, it will not take long before you find yourself needing help. Fortunately, R has extensive help resources and you should immediately become familiar with them. Begin by clicking `Help` on `RGui`. The following options are available.

- Console: gives useful shortcuts, for instance, `Ctrl+L`, to clear the R console screen.
- FAQ on R: frequently asked questions concerning general R operation.
- FAQ on R for Windows: frequently asked questions about R, tailored to the Microsoft Windows operating system.
- Manuals: technical manuals about all features of the R system including installation, the complete language definition, and add-on packages.
- R functions (text)...: use this if you know the *exact* name of the function you want to know more about, for example, `mean` or `plot`. Typing `mean` in the window is equivalent to typing `help("mean")` at the command line, or more simply, `?mean` . Note that this method only works if the function of interest is contained in a package that is already loaded into the search path with `library`.
- HTML Help: use this to browse the manuals with point-and-click links. It also has a Search Engine & Keywords for searching the help page titles, with point-and-click links for the search results. This is possibly the best help method for beginners. It can be started from the command line with the command `help.start()` .

- Search help...: use this if you do not know the exact name of the function of interest, or if the function is in a package that has not been loaded yet. For example, you may enter `plo` and a text window will return listing all the help files with an alias, concept, or title matching `plo` using regular expression matching; it is equivalent to typing `help.search("plo")` at the command line. The advantage is that you do not need to know the exact name of the function; the disadvantage is that you cannot point-and-click the results. Therefore, one may wish to use the HTML Help search engine instead. An equivalent way is `??plo` at the command line.

- search.r-project.org...: this will search for words in help lists and email archives of the R Project. It can be very useful for finding other questions that other users have asked.

- Apropos...: use this for more sophisticated partial name matching of functions. See `?apropos` for details.

On the help pages for a function there are sometimes "Examples" listed at the bottom of the page, which will work if copy-pasted at the command line (unless marked otherwise). The `example` function will run the code automatically, skipping the intermediate step. For instance, we may try `example(mean)` to see a few examples of how the `mean` function works.

2.5 External Resources

There is a mountain of information on the Internet about R. Below are a few of the important ones.

- The R-Project for Statistical Computing, `http://www.r-project.org/` . Go there first.

- The Comprehensive R Archive Network, `http://cran.r-project.org/` . That is where R is stored along with thousands of contributed packages. There are also loads of contributed information (books, tutorials, *etc.*). There are mirrors all over the world with duplicate information.
- R-Forge, `http://r-forge.r-project.org/` . This is another location where R packages are stored. Here you can find development code which has not yet been released to `CRAN`.
- R Seek, `http://www.rseek.org` is a search engine based on Google specifically tailored for R queries.

The End

Thank you