UNIT VI Exploring Data in R

Syllabus Topics

Basic features of R, Exploring R GUI, Data Frames & Lists, Handling Data in R Workspace, Reading Data Sets & Exporting Data from R, Manipulating & Processing Data in R

6.1 What is R? Its advantages

- R is a programming language and software environment for statistical analysis, graphics representation and reporting.
- This programming language was named **R**, based on the first letter of first name of the two R authors (Robert Gentleman and Ross Ihaka), and partly a play on the name of the Bell Labs Language **S.**
- R is both software and a language considered as a dialect of the S language created by the AT&T Bell Laboratories. S is available as the software S-PLUS commercialized by Insightful. There are important differences in the designs of R and of S
- R is freely distributed under the terms of the GNU General Public Licence; its development and distribution are carried out by several statisticians known as the R Development Core Team
- R is available in several forms: the sources (written mainly in C and some routines in FORTRAN), essentially for UNIX and Linux machines, or some pre-compiled binaries for Windows, Linux, and Macintosh.
- The files needed to install R, either from the sources or from the pre-compiled binaries, are distributed from the internet site of the Comprehensive R Archive Network (CRAN) where the instructions for the installation are also available.
- R has many functions for statistical analyses and graphics; the latter are visualized immediately in their own window and can be saved in various formats (jpg, png, bmp, ps, pdf, emf, pictex, xfig; the available formats may depend on the operating system).
- The results from a statistical analysis are displayed on the screen; some intermediate results (P-values, regression coefficients, residuals . . .) can be saved, written in a file, or used in subsequent analyses.
- The R language allows the user, for instance, to program loops to successively analyse several data sets. It is also possible to combine in single program different statistical functions to perform more complex analyses.
- The R users may benefit from a large number of programs written for S and available on the internet, most of these programs can be used directly with R.

- R language is an open source program maintained by the R core-development team. It is a team of volunteer developers from across the globe.
    - R language used for performing statistical operations
    - It is available from the R-Project website www.r-project.org.
    - R is a command line driven program.
    - The user enters commands at the prompt (> by default) and each command is executed one at a time.
- R is a programming language, mainly dealing with the statistical computation of data and graphical representations. Many data science experts claim that R can be considered as a very different application, of its licensed contemporary tool, SAS.

    - **a software environment:**
        - statistics
        - graphics
        - programming
        - calculator
        - GIS
    - **a language to explore, summarize, and model data**
        - functions = verbs
        - objects = nouns

Advantages of R Programming:

1. Open source Language
2. Related to other Language
3. Cross platform Compatible
4. Advanced Statistical language
5. Outstanding Graphs
6. Vast community
7. Support Extensions
8. Flexible ‘n’ fun

6.2 Syllabus topic – Basic Features of R

The following are the important features of R,

- R is a well-developed, simple and effective programming language which includes conditionals, loops; user defined recursive functions and input and output facilities.
- R has an effective data handling and storage facility,
- R provides a suite of operators for calculations on arrays, lists, vectors and matrices.
- R provides a large, coherent and integrated collection of tools for data analysis.
- R provides graphical facilities for data analysis and display either directly at the computer or printing at the papers.
- It supports procedural programming with functions and object-oriented programming with generic functions. Procedural programming includes procedure, records, modules, and procedure calls. While object-oriented programming language includes class, objects, and functions.
- Packages are part of R programming. Hence, they are useful in collecting sets of R functions into a single unit.
- R programming features include database input, exporting data, viewing data, variable labels, missing data, etc.
- R is an interpreted language. Hence, we can access it through command line interpreter.
- R supports matrix arithmetic.
- It has effective data handling and storage facilities.
- R supports a large pool of operators for performing operations on arrays and matrices.
- It has facilities to print the reports for the analysis performed in the form of graphs either on-screen or on hardcopy.

6.3 Syllabus topic - Exploring R GUI

- R offers a remarkable variety of graphics such as demo (graphics) or demo(preps).each graphical function has a large number of options making the production of graphics very flexible.
- The way graphical functions work deviates substantially from the scheme sketched particularly, the result of a graphical function cannot be assigned to an object11 but is sent to a graphical device. A graphical device is a graphical window or a file.
- There are two kinds of graphical functions

1. High level plotting function
2. Low level plotting function

- The high-level plotting functions which create a new graph, and the low-level plotting functions which add elements to an existing graph.
- The graphs are produced with respect to graphical parameters which are defined by default and can be modified with the function par.
- How to manage graphics and graphical devices; we will then somehow detail the graphical functions and parameters.

6.3.1 Managing graphics

6.3.1.1 – opening several graphical devices

- When a graphical function is executed, if no graphical device is open, R opens a graphical window and displays the graph. A graphical device may be open with an appropriate function.
- The list of available graphical devices depends on the operating system. The graphical windows are called X11 under Unix/Linux and windows under Windows.
- In all cases, one can open a graphical window with the command x11() which also works under Windows because of an alias towards the command windows ().
- A graphical device which is a file will be open with a function depending on the format: postscript (), pdf(), png(), . . . The list of available graphical devices can be found with ?device
- The last open device becomes the active graphical device on which all subsequent graphs are displayed.
- The function dev.list() displays the list of open devices:

```
> x11(); x11(); pdf()
> dev.list()
  X11 X11 pdf 2 3 4
```

- The figures displayed are the device numbers which must be used to change the active device. To know what is the active device:

```
> dev.cur()
 pdf
 4
 and to change the active device:
> dev.set(3)
 X11
```

```
3
```

- The function dev.off() closes a device: by default the active device is closed, otherwise this is the one which number is given as argument to the function. R then displays the number of the new active device:

```
> dev.off(2)
 X11
  3
> dev.off()
 pdf
 4
```

- Two specific features of the Windows version of R are worth mentioning: a Windows Metafile device can be open with the function win.metafile, and a menu “History” displayed when the graphical window is selected allowing recording of all graphs drawn during a session (by default, the recording system is off, the user switches it on by clicking on “Recording” in this menu).

6.3.1.2 – partitioning graphics:

The function split.screen partitions the active graphical device. For example:

```
> split.screen(c (1, 2))
```

- Divides the device into two parts which can be selected with screen (1) or screen (2); erase.screen () deletes the last drawn graph. A part of the device can itself be divided with split.screen () leading to the possibility to make complex arrangements.
- These functions are incompatible with others (such as layout or coplot) and must not be used with multiple graphical devices. Their use should be limited, for instance, to graphical exploration of data.
- The function layout partitions the active graphic window in several parts where the graphs will be displayed successively. Its main argument is a matrix with integer numbers indicating the numbers of the “sub-windows”. For example, to divide the device into four equal parts:

6.4 Syllabus topic – Data frames and Lists

1. Data Frames:

- A data frame is a list with class "data.frame". There are restrictions on lists that may be made into data frames, namely

- The components must be vectors (numeric, character, or logical), factors, numeric matrices, lists, or other data frames.
- Matrices, lists, and data frames provide as many variables to the new data frame as they have columns, elements, or variables, respectively.
- Numeric vectors, logical and factors are included as is, and by default1 character vectors are coerced to be factors, whose levels the unique values are appearing in the vector.
- Vector structures appearing as variables of the data frame must all have the same length, and matrix structures must all have the same row size.

A data frame may for many purposes be regarded as a matrix with columns possibly of differing modes and attributes. It may be displayed in matrix form, and its rows and columns extracted using matrix indexing conventions.

6.4.1 Making Data Frames

– Objects satisfying the restrictions placed on the columns (components) of a data frame may be used to form one using the function

```
data.frame:
> accountants <- data.frame(home=statef, loot=incomes, shot=incomef)
```

– A list whose components conform to the restrictions of a data frame may be coerced into a data frame using the function as.data.frame ()

– The simplest way to construct a data frame from scratch is to use the read.table () function to read an entire data frame from an external file.

6.4.2 attach () and detach()

– The $ notation, such as accountants$home, for list components is not always very convenient. A useful facility would be somehow to make the components of a list or data frame temporarily visible as variables under their component name, without the need to quote the list name explicitly each time.

– The attach() function takes a ‘database’ such as a list or data frame as its argument. Thus suppose lentils is a data frame with three variables lentils$u, lentils$v, lentils$w. The attach

```
> attach(lentils)
```

– Places the data frame in the search path at position 2, and provided there are no variables u, v or w in position 1, u, v and w are available as variables from the data frame in their own right. At this point an assignment such as

```
> u <- v+w
```

- Does not replace the component u of the data frame, but rather masks it with another variable u in the working directory at position 1 on the search path. To make a permanent change to the data frame itself, the simplest way is to resort once again to the $ notation:

  ```
  > lentils$u <- v+w
  ```
- However the new value of component u is not visible until the data frame is detached and attached again.
- To detach a data frame, use the function

  ```
  > detach()
  ```
- More precisely, this statement detaches from the search path the entity currently at position 2. Thus in the present context the variables u, v and w would be no longer visible, except under the list notation as lentils$u and so on. Entities at positions greater than 2 on the search path can be detached by giving their number to detach, but it is much safer to always use a name, for example by detach(lentils) or detach("lentils")

Note: In R lists and data frames can only be attached at position 2 or above, and what is attached is a copy of the original object. You can alter the attached values via assign, but the original list or data frame is unchanged.

6.4.3 Working with data frames

- A useful convention that allows you to work with many different problems comfortably together in the same working directory is
  - gather together all variables for any well-defined and separate problem in a data frame under a suitably informative name;
  - when working with a problem attach the appropriate data frame at position 2, and use the working directory at level 1 for operational quantities and temporary variables;
  - before leaving a problem, add any variables you wish to keep for future reference to the data frame using the $ form of assignment, and then detach();
  - Finally remove all unwanted variables from the working directory and keep it as clean of left-over temporary variables as possible.
- In this way it is quite simple to work with many problems in the same directory, all of which have variables named x, y and z, for example.

6.4.4 Attaching arbitrary list.

- attach () is a generic function that allows not only directories and data frames to be attached to the search path, but other classes of object as well. In particular any object of mode "list" may be attached in the same way:

```
> attach(any.old.list)
```

- Anything that has been attached can be detached by detach, by position number or, preferably, by name.

6.4.5 Managing the search Path

- The function search shows the current search path and so is a very useful way to keep track of which data frames and lists (and packages) have been attached and detached. Initially it gives

```
> search() [1] ".GlobalEnv" "Autoloads" "package:base"
```

  where .GlobalEnv is the workspace.
- After lentils is attached we have

```
> search()
[1] ".GlobalEnv" "lentils" "Autoloads" "package:base"
> ls(2)
[1] "u" "v" "w"
```

- and as we see ls (or objects) can be used to examine the contents of any position on the search path.
- Finally, we detach the data frame and confirm it has been removed from the search path.

```
> detach("lentils") > search()
[1] ".GlobalEnv" "Autoloads" "package:base"
```

2. List:

- An R list is an object consisting of an ordered collection of objects known as its components.
- There is no particular need for the components to be of the same mode or type, and, for example, a list could consist of a numeric vector, a logical value, a matrix, a complex vector, a character array, a function, and so on. Here is a simple example of how to make a list:

```
> Lst <- list(name="Fred", wife="Mary", no.children=3, child.ages=c(4,7,9))
```

- Components are always numbered and may always be referred to as such. Thus if Lst is the name of a list with four components, these may be individually referred to as Lst[[1]], Lst[[2]], Lst[[3]] and Lst[[4]]. If, further, Lst[[4]] is a vector subscripted array then Lst[[4]][1] is its first entry.

- If Lst is a list, then the function length(Lst) gives the number of (top level) components it has.
- Components of lists may also be named, and in this case the component may be referred to either by giving the component name as a character string in place of the number in double square brackets, or, more conveniently, by giving an expression of the form

  > name$component_name for the same thing.
- This is a very useful convention as it makes it easier to get the right component if you forget the number.
- So in the simple example given above:

  Lst$name is the same as Lst[[1]] and is the string "Fred",

  Lst$wife is the same as Lst[[2]] and is the string "Mary",

  Lst$child.ages[1] is the same as Lst[[4]][1] and is the number 4.
- Additionally, one can also use the names of the list components in double square brackets, i.e., Lst[["name"]] is the same as Lst$name. This is especially useful, when the name of the component to be extracted is stored in another variable as in

  > x <- "name"; Lst[[x]]
- It is very important to distinguish Lst[[1]] from Lst[1]. '[[...]]' is the operator used to select a single element, whereas '[...]' is a general subscripting operator. Thus the former is the first object in the list Lst, and if it is a named list the name is not included. The latter is a sublist of the list Lst consisting of the first entry only. If it is a named list, the names are transferred to the sublist.
- The names of components may be abbreviated down to the minimum number of letters needed to identify them uniquely. Thus Lst$coefficients may be minimally specified as Lst$coe and Lst$covariance as Lst$cov.
- The vector of names is in fact simply an attribute of the list like any other and may be handled as such. Other structures besides lists may, of course, similarly be given a names attribute also.

6.4.2.1 constructing and modifying lists

- New lists may be formed from existing objects by the function list(). An assignment of the form

  > Lst <- list(name_1=object_1, ..., name_m=object_m)
- sets up a list Lst of m components using object 1, . . ., object m for the components and giving them names as specified by the argument names, (which can be freely chosen). If these names are omitted, the components are numbered only. The components used to form the list are copied when forming the new list and the originals are not affected.

- Lists, like any subscripted object, can be extended by specifying additional components. For example

  > Lst[5] <- list(matrix=Mat)

6.4.2.2 Concatenating lists:

- When the concatenation function c() is given list arguments, the result is an object of mode list also, whose components are those of the argument lists joined together in sequence.

  > list.ABC <- c(list.A, list.B, list.C)

- Recall that with vector objects as arguments the concatenation function similarly joined together all arguments into a single vector structure. In this case all other attributes, such as dim attributes, are discarded

6.5 Syllabus topic - Handling Data in R:

- Data handling means gathering and recording the information gathered and presents it in a way that is meaningful to others.
- Let us take an example, let's go back to the early 20's, do you remember of a phone directory, which consists of peoples name and their phone numbers. The names are arranged in alphabetical order that means the names are arranged in a systematic manner that is why it is possible to find the number of a particular person. This is an example of data handling as the data is arranged in such a way that is meaningful to others.
- There are two approaches are data handling
  1. Statistical approach
  2. Non statistical approach

1. Statistical approach:

- The statistical approach is arranging the data in a meaningful manner and extracting some information from the data which can be used to gain information about the data.
- Let us suppose that we have observations on the weight of 1,000 students in a random sequence, then after looking at the data, we can`t say anything about the distribution of the weights of the students. For having information about the above data, we have to arrange the data in a given order; we have to find the mean and standard deviation of the data. So these are some of the points which we have to keep in my mind before starting the data analysis for any data.
-

2. Non statistical approach:

- The non-statistical approach to the data handling simply arranging your data in a form that is meaningful to others. It can the simple arrangement of names according to the alphabetical order on a sheet of paper, so that when we want the information for a given person we can do it easily.

6.6 Syllabus topic - Reading data sets and Exporting data from R

- Reading data into a statistical system for analysis and exporting the results to some other system for report writing can be frustrating tasks that can take far more time than the statistical analysis itself, even though most readers will find the latter far more appealing.
- import and export facilities available either in R itself or via packages which are available from CRAN or elsewhere
- Statistical systems like R are not particularly well suited to manipulations of large-scale data.
- Some other systems are better than R we can make another system do the work!
- (For example Therneau & Grambsch (2000) commented that they preferred to do data manipulation in SAS and then use package survival (https://CRAN.R-project.org/package=survival) in S for the analysis.) Database manipulation systems are often very suitable for manipulating and extracting data: several packages to interact with DBMSs.
- There are packages to allow functionality developed in languages such as Java, Perl and python to be directly integrated with R code, making the use of facilities in these languages even more appropriate.
- It is also worth remembering that R like S comes from the UNIX tradition of small re-usable tools, and it can be rewarding to use tools such as awk and perl to manipulate data before import or after export.

6.6.1 Imports:

- The easiest form of data to import into R is a simple text file, and this will often be acceptable for problems of small or medium scale. The primary function to import from a text file is scan, and this underlies most of the more convenient functions [spread sheet like data].
- All statistical consultants are familiar with being presented by a client with a memory stick (formerly, a floppy disc or CD-R) of data in some proprietary binary format, for example 'an Excel spread sheet' or 'an SPSS file'.

- Often the simplest thing to do is to use the originating application to export the data as a text file (and statistical consultants will have copies of the most common applications on their computers for that purpose).
- However, this is not always possible, and what facilities are available to access such files directly from R. For Excel spread sheets, the available methods are summarized[Reading excel spread sheet]
- Data have been stored in a binary form for compactness and speed of access. One application of this that we have seen several times is imaging data, which is normally stored as a stream of bytes as represented in memory, possibly preceded by a header. Such data formats [Binary Files, Binary connections].
- For much larger databases it is common to handle the data using a database management system (DBMS). There is once again the option of using the DBMS to extract a plain file, but for many such DBMSs the extraction operation can be done directly from an R package
- Importing data via network connections is Network Interface

6.6.1.1. Encodings:

- Unless the file to be imported from is entirely in ASCII, it is usually necessary to know how it was encoded. For text files, a good way to find out something about its structure is the file command-line tool (for Windows, included in Rtools).
- This reports something like

```
text.Rd: UTF-8 Unicode English text
text2.dat: ISO-8859 English text
text3.dat: Little-endian UTF-16 Unicode English character data, with CRLF line terminators
intro.dat: UTF-8 Unicode text
intro.dat: UTF-8 Unicode (with BOM) text
```

- Modern Unix-alike systems, including macOS, are likely to produce UTF-8 files. Windows may produce what it calls ‘Unicode’ files (UCS-2LE or just possibly UTF-16LE1). Otherwise most files will be in a 8-bit encoding unless from a Chinese/Japanese/Korean locale (which have a wide range of encodings in common use).
- It is not possible to automatically detect with certainty which 8-bit encoding (although guesses may be possible and file may guess as it did in the example above), so you may simply have to ask the originator for some clues (e.g. ‘Russian on Windows’).

- ‘BOMs’ cause problems for Unicode files. In the Unix world BOMs are rarely used, whereas in the Windows world they almost always are for UCS-2/UTF-16 files, and often are for UTF-8 files.
- The file utility will not even recognize UCS-2 files without a BOM, but many other utilities will refuse to read files with a BOM and the IANA standards for UTF-16LE and UTF-16BE prohibit it. We have too often been reduced to looking at the file with the command-line utility od or a hex editor to work out its encoding.

6.6.2 Export to text Files

- Exporting results from R is usually a less contentious task, but there are still a number of pitfalls. There will be a target application in mind, and often a text file will be the most convenient interchange vehicle.
- Function cat underlies the functions for exporting data. It takes a file argument, and the append argument allows a text file to be written via successive calls to cat. Better, especially if this is to be done many times, is to open a file connection for writing or appending, and cat to that connection, and then close it.
- The most common task is to write a matrix or data frame to file as a rectangular grid of numbers, possibly with row and column labels.
- This can be done by the functions write.table and write.
- Function write just writes out a matrix or vector in a specified number of columns (and transposes a matrix).
- Function write.table is more convenient, and writes out a data frame (or an object that can be coerced to a data frame) with row and column labels.
- There are a number of issues that need to be considered in writing out a data frame to a text file.

1. Precision

Most of the conversions of real/complex numbers done by these functions is to full precision, but those by write are governed by the current setting of options (digits). For more control, use format on a data frame, possibly column-by-column.

2. Header lines:

R prefers the header line to have no entry for the row names, so the file looks like

```
                dist      climb     time
   Greenmantle  2.5        650      16.083...
```

Some other systems require a (possibly empty) entry for the row names, which is what write.table will provide if argument col.names = NA is specified. Excel is one such system.

3. Separator:

- A common field separator to use in the file is a comma, as that is unlikely to appear in any of the fields in English-speaking countries. Such files are known as CSV (comma separated values) files, and wrapper function write.csv provides appropriate defaults.
- In some locales the comma is used as the decimal point (set this in write.table by dec = ",") and there CSV files use the semicolon as the field separator: use write.csv2 for appropriate defaults.
- There is an IETF standard for CSV files (which mandates commas and CRLF line endings, for which use eol = "\r\n"), RFC4180 (see https://tools.ietf.org/html/rfc4180), but what is more important in practice is that the file is readable by the application it is targeted at. Using a semicolon or tab (sep = "\t") are probably the safest options.

4. Missing Values:

By default missing values are output as NA, but this may be changed by argument na. Note that NaNs are not treated as NA by write.table, but neither by cat nor write.

5. Quoting String:

- By default strings are quoted (including the row and column names). Argument quote controls if character and factor variables are quoted: some programs, for example Mondrian do not accept quoted strings (https://en.wikipedia.org/wiki/Mondrian_(software)).
- Some care is needed if the strings contain embedded quotes. Three useful forms are

```
> df <- data.frame(a = I("a \" quote"))
> write.table (df)
"a"
 "1" "a \" quote"
> write.table (df, qmethod = "double")
"a"
"1" "a "" quote"
 > write.table (df, quote = FALSE, sep = ",")
A
1,a " quote
```

The second is the form of escape commonly used by spread sheets.

6. Encodings:

- Text files do not contain metadata on their encodings, so for non-ASCII data the file needs to be targeted to the application intended to read it.
- All of these functions can write to a connection which allows an encoding to be specified for the file, and write.table has a file Encoding argument to make this easier.
- The hard part is to know what file encoding to use. For use on Windows, it is best to use what Windows calls 'Unicode'2 that is "UTF-16LE".
- Using UTF-8 is a good way to make portable files that will not easily be confused with any other encoding, but even macOS applications (where UTF-8 is the system encoding) may not recognize them, and Windows applications are most unlikely to. Apparently Excel: mac 2004/8 expected .csv files in "macroman" encoding (the encoding used in much earlier versions of Mac OS).

– Function write.matrix in package MASS (https://CRAN.R-project.org/package=MASS) provides a specialized interface for writing matrices, with the option of writing them in blocks and thereby reducing memory usage.

– It is possible to use sink to divert the standard R output to a file, and thereby capture the output of (possibly implicit) print statements. This is not usually the most efficient route, and the options (width) setting may need to be increased.

– Function write.foreign in package foreign (https: / / CRAN . R-project. org / package=foreign) uses write.table to produce a text file and also writes a code file that will read this text file into another statistical package. There is currently support for export to SAS, SPSS and State.

6.6.3. XML

– When reading data from text files, it is the responsibility of the user to know and to specify the conventions used to create that file, e.g. the comment character, whether a header line is present, the value separator, the representation for missing values and so on.

– A mark-up language which can be used to describe not only content but also the structure of the content can make a file self-describing, so that one need not provide these details to the software reading the data

– The eXtensible Mark-up Language – more commonly known simply as XML – can be used to provide such structure, not only for standard datasets but also more complex data structures.

- XML is becoming extremely popular and is emerging as a standard for general data markup and exchange. It is being used by different communities to describe geographical data such as maps, graphical displays, and mathematics and so on.
- XML provides a way to specify the file's encoding, e.g.

  <?xml version="1.0" encoding ="UTF-8"?>

  although it does not require it
- The XML (https://CRAN.R-project.org/package=XML) package provides general facilities for reading and writing XML documents within R.
- Package StatDataML (https://CRAN. R-project.org/package=StatDataML) on CRAN is one example building on XML (https:// CRAN.R-project.org/package=XML)
- Another interface to the libxml2 C library is provided by package xml2 (https://CRAN.R-project.org/package=xml2).
- Yaml is another system for structuring text data, with emphasis on human-readability: it is supported by package yaml(https://CRAN.R-project.org/package=yaml).

6.6.4 Reading and Writing data in R

6.6.4.1. Reading data in R

For reading, (importing) data into R following are some functions.

- read.table(), and read.csv(), for reading tabular data
- readLines() for reading lines of a text file
- source() for reading in R code files (inverse of dump)
- dget() for reading in R code files (inverse of dput)
- load () for reading in saved workspaces.

6.6.4.2. Writing data to files

Following are few functions for writing (exporting) data to files.

- write.table(), and write.csv() exports data to wider range of file format including csv and tab-delimited.
- writeLines() write text lines to a text-mode connection.
- dump() takes a vector of names of R objects and produces text representations of the objects on a file (or connection). A dump file can usually be sourced into another R session.
- dput() writes an ASCII text representation of an R object to a file (or connection) or uses one to recreate the object.

- save() writes an external representation of R objects to the specified file.

6.6.4.3. Reading data files with read.table ( )

The read.table() function is one of the most commonly used functions for reading data into R. It has a few important arguments.

- file, the name of a file, or a connection
- header, logical indicating if the file has a header line
- sep, a string indicating how the columns are separated
- colClasses, a character vector indicating the class of each column in the data set
- nrows, the number of rows in the dataset
- comment.char, a character string indicating the comment character
- skip, the number of lines to skip from the beginning
- stringsAsFactors, should character variables be coded as factors?

6.6.4.4 read.table ( ) and read.csv examples:

1. Read.table ( ):

R will automatically skip lines that begin with a #, figure out how many rows there are (and how much memory needs to be allocated). R also figure out what type of variable is in each column of the table.

```
read.table(file, header = FALSE, sep = "", quote = "\"'",
           dec = ".", row.names, col.names,
           as.is = !stringsAsFactors,
           na.strings = "NA", colClasses = NA, nrows = -1,
           skip = 0, check.names = TRUE, fill = !blank.lines.skip,
           strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#",
           allowEscapes = FALSE, flush = FALSE,
           stringsAsFactors = default.stringsAsFactors(),
           fileEncoding = "", encoding = "unknown", text)
```

- **file**: file name
- **header**: 1st line as header or not, logical
- **sep**: field separator
- **quote**: quoting characters

Following is a csv file example "tp.txt":

	t1	t2	t3	t4	t5	t6	t7	t8
r1	1	0	1	0	0	1	0	2
r2	1	2	2	1	2	1	2	1
r3	0	0	0	2	1	1	0	1
r4	0	0	1	1	2	0	0	0
r5	0	2	1	1	1	0	0	0
r6	2	2	0	1	1	1	0	0
r7	2	2	0	1	1	1	0	1
r8	0	2	1	0	1	1	2	0
r9	1	0	1	2	0	1	0	1
r10	1	0	2	1	2	2	1	0
r11	1	0	0	0	1	2	1	2
r12	1	2	0	0	0	1	2	1
r13	2	0	0	1	0	2	1	0
r14	0	2	0	2	1	2	0	2
r15	0	0	0	2	0	2	2	1
r16	0	0	0	1	2	0	1	0
r17	2	1	0	1	2	0	1	0
r18	1	1	0	0	1	0	1	2
r19	0	1	1	1	1	0	0	1
r20	0	0	2	1	1	0	0	1

```
> x <- read.table("tp.txt",header=T,sep="\t");
> is.data.frame(x)
[1] TRUE
> x

          X t1 t2 t3 t4 t5 t6 t7 t8
      1    r1  1  0  1  0  0  1  0  2
      2    r2  1  2  2  1  2  1  2  1
      3    r3  0  0  0  2  1  1  0  1
      4    r4  0  0  1  1  2  0  0  0
      5    r5  0  2  1  1  1  0  0  0
      6    r6  2  2  0  1  1  1  0  0
      7    r7  2  2  0  1  1  1  0  1
      8    r8  0  2  1  0  1  1  2  0
      9    r9  1  0  1  2  0  1  0  1
      10 r10  1  0  2  1  2  2  1  0
      11 r11  1  0  0  0  1  2  1  2
      12 r12  1  2  0  0  0  1  2  1
      13 r13  2  0  0  1  0  2  1  0
      14 r14  0  2  0  2  1  2  0  2
      15 r15  0  0  0  2  0  2  2  1
      16 r16  0  0  0  1  2  0  1  0
      17 r17  2  1  0  1  2  0  1  0
      18 r18  1  1  0  0  1  0  1  2
      19 r19  0  1  1  1  1  0  0  1
      20 r20  0  0  2  1  1  0  0  1

> ncol(x)
[1] 9
```

```
> nrow(x)
[1] 20

> rownames(x)
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10"  "11"  "12"  "13"  "14"  "15"
[16] "16"  "17"  "18"  "19"  "20"
```

R will automatically skip lines that begin with a #, figure out how many rows there are (and how much memory needs to be allocated). R also figure out what type of variable is in each column of the table.

2. Read.csv ( )

In R, we can read data from files stored outside the R environment. We can also write data into files which will be stored and accessed by the operating system. R can read and write into various file formats like csv, excel xml etc.

Input as CSV

– The csv file is a text file in which the values in the columns are separated by a comma. Let's consider the following data present in the file named input.csv.
– You can create this file using windows notepad by copying and pasting this data. Save the file as input.csv using the save As All files(*.*) option in notepad.

```
id,    name,        salary,             start_date,   dept
1,     Rick,        623.3,              2012-01-01,   IT
2,     Dan,         515.2,              2013-09-23,   Operations
3,     Michelle,    611,                2014-11-15,   IT
4,     Ryan,        729,                2014-05-11,   HR
5,     Gary,        843.25,             2015-03-27,   Finance
6,     Nina,        578,                2013-05-21,   IT
```

```
data <- read.csv("input.csv")
print(data)

      id,   name,    salary,   start_date,     dept
1      1    Rick     623.30    2012-01-01      IT
2      2    Dan      515.20    2013-09-23      Operations
3      3    Michelle 611.00    2014-11-15      IT
4      4    Ryan     729.00    2014-05-11      HR
5     NA    Gary     843.25    2015-03-27      Finance
6      6    Nina     578.00    2013-05-21      IT
7      7    Simon    632.80    2013-07-30      Operations
8      8    Guru     722.50    2014-06-17      Finance
```

Analysing CSV files:

```
data <- read.csv("input.csv")
print(is.data.frame(data))
print(ncol(data))
print(nrow(data))


[1] TRUE
[1] 5
[1] 8
```

Get the Maximum salary:

```
# Create a data frame.
data <- read.csv("input.csv")

# Get the max salary from data frame.
sal <- max(data$salary)
print(sal)

[1] 843.25

# Create a data frame.
data <- read.csv("input.csv")

# Get the max salary from data frame.
sal <- max(data$salary)

# Get the person detail having max salary.
retval <- subset(data, salary == max(salary))
print(retval)


      id    name  salary  start_date    dept
5     NA    Gary  843.25  2015-03-27    Finance


# Create a data frame.
data <- read.csv("input.csv")


retval <- subset( data, dept == "IT")
print(retval)


        id   name      salary   start_date   dept
1       1    Rick      623.3    2012-01-01   IT
3       3    Michelle  611.0    2014-11-15   IT
6       6    Nina      578.0    2013-05-21   IT

# Create a data frame.
data <- read.csv("input.csv")
```

```
retval <- subset(data, as.Date(start_date) > as.Date("2014-01-01"))
print(retval)

       id   name     salary   start_date    dept
3      3    Michelle 611.00   2014-11-15    IT
4      4    Ryan     729.00   2014-05-11    HR
5     NA    Gary     843.25   2015-03-27    Finance
8      8    Guru     722.50   2014-06-17    Finance

# Create a data frame.
data <- read.csv("input.csv")
retval <- subset(data, as.Date(start_date) > as.Date("2014-01-01"))

# Write filtered data into a new file.
write.csv(retval,"output.csv")
newdata <- read.csv("output.csv")
print(newdata)


  X      id   name      salary   start_date    dept
1 3      3    Michelle  611.00   2014-11-15    IT
2 4      4    Ryan      729.00   2014-05-11    HR
3 5     NA    Gary      843.25   2015-03-27    Finance
4 8      8    Guru      722.50   2014-06-17    Finance

# Create a data frame.
data <- read.csv("input.csv")
retval <- subset(data, as.Date(start_date) > as.Date("2014-01-01"))

# Write filtered data into a new file.
write.csv(retval,"output.csv", row.names = FALSE)
newdata <- read.csv("output.csv")
print(newdata)

      id    name      salary   start_date    dept
1      3    Michelle  611.00   2014-11-15    IT
2      4    Ryan      729.00   2014-05-11    HR
3     NA    Gary      843.25   2015-03-27    Finance
4      8    Guru      722.50   2014-06-17    Finance
```

6.6.4.4. Writing data files with write.table ()

Following are few important arguments usually used in write.table () function.

- x, the object to be written, typically a data frame
- file, the name of the file which the data are to be written to
- sep, the field separator string
- col.names, a logical value indicating whether the column names of x are to be written along with x, or a character vector of column names to be written

- row.names, a logical value indicating whether the row names of x are to be written along with x, or a character vector of row names to be written
- na, the string to use for missing values in the data

6.7 Manipulating & Processing Data in R.

– **Data manipulation in R** and data processing with Moreover, we will see three subset operators in R and how to perform R data manipulation like sub setting in R, sorting and merging of data in R programming language.

Creating subsets	Creating subgroups	Match( ) function
Sample( ) command	Merging Datasets	Traversing Data
Application	Merge( ) function	Formula Interface
Adding calculated field	Sorting and Ordering	Variables

6.7.1 What is Data Manipulation in R

– Data structures provide the way to represent data in data analytics. We can manipulate data in R for analysis and visualization.

– Before we start playing with data in R, let us see how to import data in R and ways to export data from R to different external sources like SAS, SPSS, text file or CSV file. One of the most important aspects of computing with data Manipulation in R and enable its subsequent analysis and visualization. Let us see few basic data structures in R.

1. Vector:

– These are ordered a container of primitive elements and are used for 1-dimensional data. Types – integer, numeric, logical, character, complex.

2. Matrices in R:

– These are Rectangular collections of elements and are useful when all data is of a single class that is numeric or characters.
Dimensions – two, three, etc.

3. List in R:

- These are ordered a container for arbitrary elements and are used for higher dimension data, like customer data information of an organization. When data cannot be represented as an array or a data frame, list is the best choice. This is so because lists can contain all kinds of other objects, including other lists or data frames, and in that sense, they are very flexible.

4. Data Frames:

- These are two-dimensional containers for records and variables and are used for representing data from spread sheets etc. It is similar to a single table in the database.

6.7.2 Creating subset data in R.

- Data size is increasing exponentially and doing an analysis of complete data is very time-consuming. So the data is divided into small sized samples and analysis of samples is done.
- The process of creating samples is called sub setting.
  Different methods of sub setting in R are:

1. $:

- The dollar sign operator selects a single element of data. When you use this operator with a data frame, the result is always a vector.

2. [[

- Similar to $ in R, the double square brackets operator in R also returns a single element, but it offers the flexibility of referring to the elements by position rather than by name. It can be used for data frames and lists.

3. [

- The single square bracket operator in R returns multiple elements of data. The index within the square brackets can be a numeric vector, a logical vector, or a character vector. For example: To retrieve 5 rows and all columns of already built-in dataset iris, below command is used:
- > iris[1:5,]
- Creating Subsets of Data in R lets see Sample ( ) function for Data Manipulation in R.

4. Sample ( ) command in R

- To create samples, sample() command is used and the number of samples to be drawn are mentioned.
  For example, to create a sample of 10 simulations of a die, below command is used:

```
> sample(1:6, 10, replace=TRUE)
```

It gives output as:

[1] 2 2 5 3 5 3 5 6 3 5

- Sample() should always produce random values. But it does not happen with the test code sometimes. If substituted with a seed value, the sample() command always produces random samples.

The seed value is the starting point for any random number generator formula. Seed value defines both, the initialization of the random number generator along with the path that the formula will follow.

Let us see how seed value is used.

> set.**seed**(1) //setting seed values for sample() command
>
> >**sample**(1:6, 10, replace=TRUE)

- This gives output as below:

  [1] 2 3 4 6 2 6 6 4 4 1

  Now let's move ahead in the R Data Manipulation tutorial with Applications of Subsetting Data.

5. Application of sub setting Data:

1. Duplicate data can be removed during analysis using duplicated () function in R.

- Duplicated ( ) function finds duplicate values and returns a logical vector that tells you whether the specified value is a duplicate of a previous value.

  >duplicated(c(1,2,1,3,1,4))
- This gives output as below:

  [1] FALSE FALSE TRUE FALSE TRUE FALSE

  For all those values which are duplicate in the sample, true is returned.

2. Missing data can be identified using complete.cases ( ) function in R:

- If during analysis, any row with missing data can identify and remove as below:
- complete.cases ( ) command in R is used to find rows which are complete.
- It gives logical vector with the value TRUE for rows that are complete, and FALSE for rows that have some NA values.
- Rows which have NA values can be removed using na.omit ( ) function as below:

  row_name <- na.**omit**(file_name)

6. Adding calculated Fields to Data.

- R makes it easy to perform calculations on columns of a data frame because each column is itself a vector.

  ```
  > x <- iris$Sepal.Length / iris$Sepal.Width
  >head(x)
  //Command to display the first five elements of the result
  ```

- It gives the output as:

  [1] 1.457143 1.633333 1.468750 1.483871 1.388889 1.384615

  1.with ( ) Function in R:

- To reduce the amount of typing and make code more readable, we use with() command as below:

  ```
  >y <- with(iris, Sepal.Length / Sepal.Width) //Command to calculate the ratio between the lengths and width of the sepals using the with() function
  >head(y)
  ```

  2. within ( ) function in R:

  ```
  >iris<- within(iris, ratio <- Sepal.Length / Sepal.Width)
  ```

- With() function allows you to refer to columns inside a data frame without explicitly using the dollar sign or even the name of the data frame itself.With and Within we can use interchangeably.

7. Creating Subgroups or Bins of Data:

- Cut() function groups values of a variable into larger bins. It creates bins of equal size and classifies each element into its appropriate bin.

  ```
  >cut(frost, 3, include.lowest=TRUE)
  ```

- This gives the result as a factor with three levels.
  The cut() function creates mathematical labels for the bins. The label names can be provided by the user.

  ```
  >cut(frost, 3, include.lowest=TRUE, labels=c("Low", "Med", "High"))
  ```

  1. cut ( ) function in R:

- Cut() function groups values of a variable into larger bins. It creates bins of equal size and classifies each element into its appropriate bin.

  ```
  > cut(frost, 3, include.lowest=TRUE)
  ```

- The cut ( ) function creates mathematical labels for the bins. The label names can be provided by the user.

>**cut**(frost, 3, include.lowest=TRUE, labels=**c**("Low", "Med", "High"))

2. table ( ) function in R:

> x <- **cut**(frost, 3, include.lowest=TRUE, labels=**c**("Low", "Med", "High"))

> table(x)

8. Combining and Merging Datasets in R.

– If you want to combine data from different sources in R, you can combine different sets of data in three ways:

1. **By Adding Columns using cbind ( ) in R:**

– If the two sets of data have an equal set of rows, and the order of the rows is identical, then adding columns makes sense. This can be done by using the data.frame or cbind() function.

2. **By Adding Rows using rbind ( ) function in R:**

– If both sets of data have the same columns and you want to add rows to the bottom, use rbind ( )

3. **By Combining Data With Different Shapes using merge() function in R:**

– The merge() function combines data based on common columns as well as rows. In database language, this is usually called joining data.For merging the existing data, using the merge()function is useful. You can use merge()to combine data only when certain matching conditions are satisfied.

9. Merge ( ) Function in R:

– The merge ( ) function is used to combine data frames.

```
merge(cold.states, large.states) Name Frost Area
```

– The merge ( ) function allows four ways of combining data:

1. **Natural join in R**

– To keep only rows that match from the data frames, specify the argument all=FALSE

2. **Full outer join in R**

– To keep all rows from both data frames, specify all=TRUE

3. **Left outer join in R**

– To include all the rows of your data frame x and only those from y that match, specify all.x=TRUE

4. **Right outer join in R**

- To include all the rows of your data frame y and only those from x that match, specify all.y=TRUE

  The merge()function takes a large number of arguments, as follows:

  x: A data frame

  y: A data frame

- by, by.x, by.y: Names of the columns common to both x and y. By default, it uses columns with common names between the two data frames.
- all, all.x, all.y: Logical values that specify the type of merge. The default value is all = FALSE

10. Match() function in R

- The R match() function returns the matching positions of two vectors or, more specifically, the positions of the first matches of one vector in the second vector.

```
> index <- match(cold.states$Name, large.states$Name)
```

- This is the command to search for large states that also occur in the data frame cold.states

```
> index
```

It gives output as:

```
[1] 1 4 NA NA 5 6 NA NA NA NA NA
```

11. Sorting and Ordering Data in R using sort() in R and Order() in R

A common task in data analysis and reporting is sorting information.

First create data frame and then we will sort it.

```
> some.states <- data.frame( + Region = state.region, + state.x77)
```

This is the command to create data frame some.states.

```
> some.states <- some.states[1:10, 1:3]
```

This will create subset of it.

By default, sorting is done in ascending manner if not specified.

```
> sort(some.states$Population) //Command to sort Population in ascending order
> sort(some.states$Population, decreasing=TRUE) //Command to sort Population in descending order
```

This is how sorting of data can be done in R.

Data frames can also be sorted as below:

```
>order.pop <- order(some.states$Population)
```

12. Traversing Data with the Apply ( ) Function in R

To traverse the data, R uses apply functions. The output of the apply() function depends on the data structure being traversed.

1. **Array or matrix**
   - The apply() function traverses either the rows or columns of a matrix, applies a function to each resulting vector, and returns a vector of summarized results
2. **List**
   - The lapply() function can traverse a list, it applies a function to each element, and returns a list of the results. Sometimes it is possible to simplify the resulting list into a matrix or vector.

   R Apply() function use as below:

   apply(X, MARGIN, FUN, ...)

   The apply() function takes four arguments as below:

- **X:** This is the data—an array (or matrix)
- **MARGIN:** This is a numeric vector that indicates the dimension over which to traverse—1 means rows and 2 means columns
- **FUN:** This is the function to apply (for example, sum or mean)
- **... (dots):** If the FUN function requires any additional arguments, they can add here.

In essence, the apply function allows us to make entry-by-entry changes to data frames and matrices. If MARGIN=1, the function accepts each row of X as a vector argument, and returns a vector of the results. Similarly, if MARGIN=2 the function acts on the columns of X. Most impressively, when we apply MARGIN=c(1,2) function to every entry of X.
Let us now discuss the variations of the apply() function:

1. lapply() function in R
2. sapply() function in R
3. tapply() function in R

We use it to create tabular summaries of data. This function takes three arguments:

- **X:** Refers to a vector
- **INDEX:** Refers to a factor or list of factors
- **FUN:** Refers to a function

An illustrative example

Consider the code below:

#Create the matrix

m<-**matrix**(**c**(**seq**(from=-98,to=100,by=2)),nrow=10,ncol=10)

# Return the product of each of the rows

apply(m,1,prod)

# Return the sum of each of the columns

apply (m,2,sum)

13. Formula Interface in R:

- The R formula interface allows you to concisely specify which columns to use when fitting a model, as well as the behaviour of the model for Data Manipulation in R.
- The formula operator + means to include a column, not to mathematically add two columns together

Operator	Example	Meaning
~	y ~ x	Model y as a function of x
+	y ~ a + b	Include columns a as well as b
–	y ~ a – b	Include a but exclude b
:	y ~ a : b	Estimate the interaction of a and b
*	y ~ a * b	Include columns as well as their interaction (that is, y ~ a + b + a:b)
|	y ~ a | b	Estimate y as a function of a conditional on b