

# Statistics for Data Analysis

N. Sairam

`sairam@cse.sastra.edu`

School of Computing, SASTRA University, Thanjavur.

Unit-I

# Introduction to R Programming

- A Programming Environment
- Object Oriented
- Open Source or Freeware
- Provide calculations on matrices
- Excellent graphical capabilities
- Supported by a large user network

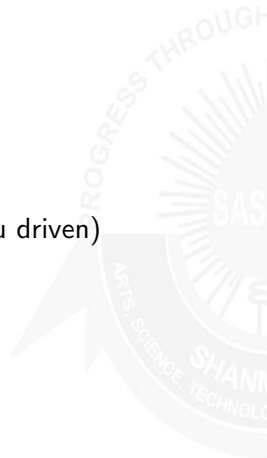


## Introduction to R Programming Contd..

- Includes routines for data summary and exploration, graphical representation and data modeling
- Create objects that are stored in the workspace(sometimes called image)
- Each object remains in the image unless it is explicitly deleted
- At the end of the session if the workspace is not saved it will be lost

# What R is not?

- A statistical package
- Menu Driven (But R-Studio is somewhat menu driven)
- Quick to learn
- A program with complex graphical interface



# How to Install R?

- [www.r-project.org](http://www.r-project.org)
- Download from CRAN
- Download the base package
- Download the contributed packages as needed



# A Short Introduction to R

- The command line prompt (`>`) is an invitation to start typing in commands or expressions
- R evaluates and prints out the result of any expression that is typed in at the command line in the console window (multiple commands may appear on the one line, with the semicolon (`;`) as the separator).
- Use of R as a calculator
- Example: `> 2+2`
- `[1] 4`
- `>`
- The first element is labeled `[1]` denotes just one element
- The final `>` prompt indicates that R is ready for another command

# A Short Introduction to R

- Anything that follows a `#` on the command line is taken as comment and ignored by R
- A continuation prompt, by default `+`, appears following a carriage return when the command is not yet complete

# Plotting using R

- Consider the following table that gives the internal marks of first 5 students of our class

Roll Number	Marks
1	32
2	40
3	36
4	35
5	26

- `RollNumber <- c(1,2,3,4,5)`
- `Marks <- c(32,40,36,35,26)`
- `plot(Marks~RollNumber,pch=16)`



## Explanation for the example given in the previous slide

- The `< -` is a left angle bracket (`<`) followed by a minus sign (`-`). It means "the values on the right are assigned to the name on the left."
- The objects `RollNumber` and `Marks` are vectors which were each formed by joining (concatenating) separate numbers together. Thus `c(32,40,36,35,26)` joined the numbers 32, 40, 36, 35, 26 together to form the vector `Marks`
- The construct `Marks~RollNumber` is a graphics formula. The `plot()` function interprets this formula to mean "Plot `Marks` as a function of `RollNumber`" or "Plot `Marks` on the y-axis against `RollNumber` on the x-axis."
- The setting `pch=16` (where `pch` is "plot character") gives a solid black dot

# Data Frames

- It is convenient to group the two vectors together into an object that is called a data frame
- Data frames are the preferred way for organizing data sets that are of modest size
- Data frames can be considered as a rectangular row by column layout, where the rows are observations and the columns are variables

## getwd(),ls()

- getwd()-give the name of the working directory
- ls()-list the contents of the workspace



# Quitting

- `q()`
- `.RData`-All the objects that remain in the working directory are saved in an "image" file that has the name `.RData`. This file is an "image" of the workspace immediately before quitting the session, and will be used to restore the workspace when a new session is again started in that directory

# Uses of R

- R offers an extensive collection of functions
  - Most of the calculations that users may wish to carry out, beyond simple command line computations, involve explicit use of functions. There are of course functions for calculating the sum (`sum()`), mean (`mean()`), range (`range()`), length of a vector (`length()`), for sorting values into order (`sort()`) and so on
- R will give numerical or graphical data summaries
- R is an interactive programming language

# Vectors in R

- Vectors, factors and uni-variate time series are all uni-variate objects that can be included as columns in a data frame
- Vectors in R
  - The vector modes that will be described here (there are others) are "numeric", "logical", and "character"
  - Examples: `c(2, 3, 5, 2, 7, 1)`, `3:10`, `c(T, F, F, F, T, T, F)`, `c("SoC", "SoME", "SEEE", "SCBT")`
  - The first two vectors above are numeric, the third is logical, and the fourth is a character vector
  - The use of the global variables `F(=FALSE)` and `T(=TRUE)` as a convenient shorthand when logical values are entered

# Concatenation and Subsets of Vectors

- Concatenation of Vectors
  - `c()` may be used to concatenate any combination of vectors and vector elements
  - Example: Refer BB
  - Subsets of vectors
    - Specify the indices of the elements that are to be extracted
    - Example: Refer BB
    - Use negative subscripts to omit the elements in nominated subscript positions
    - Example: Refer BB
    - Specify a vector of logical values. The elements that are extracted are those for which the logical value is TRUE
    - Example: Refer BB
    - Elements of vectors can be given names
    - Refer BB

# Patterned Data and Missing Values

- `seq()`-To create a sequence of data
- Example: Refer BB
- `rep()`-To repeat a sequence 'n' times
- Example: Refer BB
- Missing Values: NA
- Factors: A factor is stored internally as a numeric vector with values 1, 2, 3, ..., k. The value k is the number of levels. The levels are character strings.
- Example: Consider a survey that has data on 691 females and 692 males. If the first 691 are females and the next 692 males, we can create a vector of strings that holds the values thus: `gender <- c(rep("female",691), rep("male",692))`
- We can change this vector to a factor, by entering: `gender <- factor(gender)`. Internally, the factor `gender` is stored as 691 1s, followed by 692 2s. It has stored with it a table that holds

1	female
---	--------



## Factors Contd..

- In most contexts that seem to demand a character string, the 1 is translated into female and the 2 into male
- The values female and male are the levels of the factor
- By default, the levels are in sorted order for the data type from which the factor was formed, so that female precedes male. Hence: `levels(gender)`. [1] "female" "male"
- Note: if gender had been an ordinary character vector, the outcome of the above levels command would have been NULL
- The order of the factor levels determines the order of appearance of the levels in graphs and tables that use this information. To cause male to come before female, use:  
`gender <- factor(gender, levels=c("male", "female"))`
- This syntax is available both when the factor is first created, and later to change the order in an existing factor

## Factors Contd..

- Note: Specifying "Male" in place of "male" in the levels argument will cause all values that were "male" to be coded as missing
- `ordered()`- generates factors whose values can be compared using the relational operators `<`, `≤`, `>`, `≥`, `==` and `!=`.
- Time Series: The function `ts()` converts numeric vectors into time series objects. Frequently used arguments of `ts()` are `start`, `frequency` and `end`
- `window()`- to extract a subset of the time series

# Assignment

- 1 The following ten observations, taken during the years 1970-79, are on October snow cover. (Snow cover is in millions of square kilometers):

year	snow.cover
1970	6.5
1971	12.0
1972	14.9
1973	10.0
1974	10.7
1975	7.9
1976	21.9
1977	12.5
1978	14.5
1979	9.2

- (i) Enter the data into R, (ii) Plot snow.cover versus year,  
(iii) Use the hist() command to plot a histogram of the snow

# Assignment

- ② Input the following data, on damage that had occurred in space shuttle launches prior to the disastrous launch of Jan 28 1986. These are the data, for 6 launches out of 24, that were included in the pre-launch charts that were used in deciding whether to proceed with the launch.

Temp	Erosion	Blowby	Total
53	3	2	5
57	1	0	1
63	1	0	1
70	1	0	1
70	1	0	1
75	0	2	1

Enter these data into a data frame, with (for example) column names temperature , erosion , blowby and total. Plot total incidents against temperature.

# Data frames and matrices

- Fundamental to the use of R modelling and graphics functions
- More general object than a matrix in the sense that different columns may have different modes
- All the elements of any column must have same mode

## Displaying rows in a data frame

- `head()` → To display the first "n" rows of the data frame
- Example: `head(Cars93.summary,n=3)`
- `head()` and `tail()` are available also for use with objects other than data frames
- Square bracket notation: `Cars93.summary[1:3,]`

# Data frames are a specialized type of list

- list → Arbitrary collection of R objects, brought together into a simple data structure
- Most of R's modeling functions return their output as a list
- List can be joined using the function `c()`. That is they are vectors
- Important aspects of the syntax for working with data frames apply also to lists
- "row" and "column" have relevance to lists

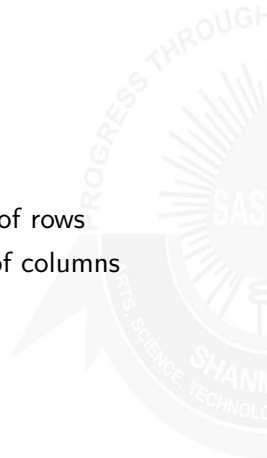
# Attaching of Data Frames

- In repeated computation with the same data frame, it is very difficult to keep repeating the name of the data frame
- By attaching a data frame, its columns can be referred to by name, without further need to give the name of the data frame
- Example: `attach(Cars93.summary); Min.passengers`
- Detaching data frames that are no longer in use reduces the risk of a clash of variable names
- Example: `detach(Cars93.summary)`
- Attaching of data frame extends the search list, which is the list of "databases" where R looks for objects



## Column and Row names

- `rownames(Cars93.summary)` → Extract names of rows
- `colnames(Cars93.summary)` → Extract names of columns



## Subsets of data frames

- `Cars93.summary[1:3,2:3]` → extract the first three rows and second and third column
- `Cars93.summary[,2:3]` → extract all the rows and second and third column
- `Cars93.summary[,c("No.of Cars","abbrev")]`
- `Cars93.summary[,-c(2,3)]` → omit column 2 and 3
- `subset()` → Alternate way to extract rows and columns
- Example: `subset(Cars93.summary,subset=1:2)`
- `Cars93.summary[1,]` → data frame
- To obtain vector of values use `unlist(Cars93.summary[1,])`

# Assignment of (new) names to the columns of a data frame

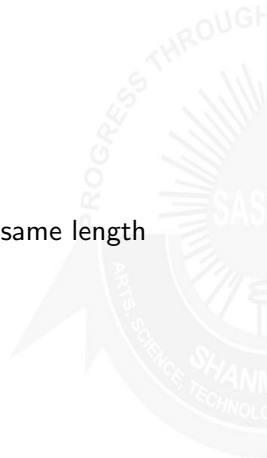
- The functions `names()` (or `colnames()`) and `rownames()` can be used to assign new names, perhaps shortening them
- Example: `names()`

# stacking and unstacking

- For stacking columns of a data frame, that is, placing successive columns one under the other, the function `stack()` is available
- The `unstack()` function reverses the stacking operation. For example, `unstack(Jobs)`
- Example: `Stack` and `UnStack`

# Matrices

- Example: Matrices
- `cbind()` → combine two or more vectors of the same length and type together into a matrix
- Example: `cbind()`



# Note

- 1 Any data frame where all columns hold data that is all of the same type, that is, all numeric or all character or all logical, can alternatively be stored as a matrix
- 2 Storage of numeric data in matrix rather than data frame format can speed up some mathematical and other manipulations when the number of elements is large
- 3 Matrix elements are stored in column order in one long vector, that is, columns are stacked one above the other, with the first column first
- 4 The extraction of submatrices has the same syntax as for data frames. Thus `fossilfuelmat[2:3,]` extracts rows 2 and 3 of the matrix `fossilfuelmat`
- 5 The `names()` function returns `NULL` when the argument is a matrix. But, the functions `rownames()` and `colnames()`, which can be used either with data frames or matrices

# Functions in R

- Builtin functions
  - `all()`→returns TRUE if all values are TRUE
  - `any()`→returns TRUE if any values are TRUE
  - `args()`→information on the arguments to a function
  - `cat()`→prints multiple objects, one after the other
  - `cumprod()`→cumulative product
  - `cumsum()`→ cumulative sum
  - `diff()`→form vector of first differences. N. B. `diff(x)` has one less element than `x`

## Built in functions

- `history()`→displays previous commands used
- `is.factor()` →returns TRUE if the argument is a factor
- `is.na()`→returns TRUE if the argument is an NA. Similar functions:`is.logical()`, `is.matrix()`
- `length()`→number of elements in a vector or of a list
- `ls()`→list names of objects in the workspace
- `mean()`→mean of the elements of a vector
- `median()`→median of the elements of a vector
- `order()`→`x[order(x)]` sorts `x` (by default, NAs are last)
- `print()`→prints a single R object
- `range()`→minimum and maximum value elements of vector



# Built in functions

- `sort()`→sort elements into order, by default omitting NAs
- `rev()`→reverse the order of vector elements
- `str()`→information on an R object
- `unique()`→form the vector of distinct values
- `which()`→locates TRUE indices of logical vectors
- `which.max()`→locates (first) maximum of a numeric vector
- `which.min()`→locates (first) minimum of a numeric vector
- `with()`→do computation using columns of specified data frame

# Data summary functions

- `table()` → Form a table of contents
- Example: `table(Sex=tinting$sex, AgeGroup=tinting$agegp)`
- By default, `table()` ignores NAs
- `xtabs()` → Form a table of totals
- `sapply()` → applies a nominated function to each column of a data frame, or to each element of a list
- Example: `sapply(jobs[, -7], range)`

## with()

- `with()` → a convenient alternative to attaching a data frame, executing one or more lines of code, and detaching the data frame
- Example: `with(tinting, table(Sex=sex, AgeGroup=agegp))`
- Utility functions and general functions: `help()`, `ls()`, `print()`, `dir()`

## Exercise

- For the data frame possum (DAAG package):
  - Use the function `str()` to get information on each of the columns.
  - Using the function `complete.cases()`, determine the rows in which one or more values is missing. Print those rows. In which columns do the missing values appear?
- For the data frame ais (DAAG package):
  - Use the function `str()` to get information on each of the columns. Determine whether any of the columns hold missing values.
  - Make a table, that shows the numbers of males and females for each different sport. In which sports is there a large imbalance (e.g. by a factor of more than 2:1) in the numbers of the two sexes?

# User Written functions

- See the demo (meanandsd-user defined fn,distance-relational op,match-selection and matching)
- Identification of rows involving missing values-complete.cases() and na.omit()
- na.omit() omits any rows that contain missing values

# Graphics in R

- Base graphics are provided by the graphics package that is automatically attached at startup
- It includes the function `plot()` for creating scatterplots, and the functions `points()`, `lines()`, `text()`, `mtext()` and `axis()` that add to existing plots

# plot()

- Syntax: `plot(x, y, main="title", sub="subtitle", xlab="X-axis label", ylab="y-axis label", xlim=c(xmin, xmax), ylim=c(ymin, ymax))`
- The basic command is: `plot(y~x)` or: `plot(x, y)` where `x` and `y` must be the same length
- Putting labels on the points
  - `xlim`: Used to specify `x` axis limits
  - Example: `plot(Brainwt~Bodywt, xlim=c(0,300))`
  - `labels`=label names → possible only if limits are specified
  - Example: `labels=row.names(primates)`
  - `pos`=position value → Possible values for position: 1-below, 2-Left, 3-above, 4-to the right

# title()

- Add labels to a plot
- Syntax: `title(main="main title", sub="sub-title", xlab="x-axis label", ylab="y-axis label")`
- Many other graphical parameters (such as text size, font, rotation, and color) can also be specified in the `title()` function.



# Text Annotations

- Text can be added to graphs using the `text( )` and `mtext( )` functions
- `text( )` places text within the graph while `mtext( )` places text in one of the four margins
- Syntax: `text(location, "text to place", pos, ...)` `mtext("text to place", side, line=n, ...)`
- location → location can be an x,y coordinate
- Alternatively, the text can be placed interactively via mouse by specifying location as `locator(1)`
- pos → position relative to location. 1=below, 2=left, 3=above, 4=right
- If you specify pos, you can specify `offset=` in percent of character width
- side → which margin to place text. 1=bottom, 2=left, 3=top, 4=right

# Text Annotations

- you can specify `line=` to indicate the line in the margin starting with 0 and moving out
- you can also specify `adj=0` for left/bottom alignment or `adj=1` for top/right alignment
- Other common options are `cex`, `col`, and `font` (for size, color, and font style respectively).

## axis()

- create custom axes using the axis( ) function
- Syntax: axis(side, at=, labels=, pos=, lty=, col=, las=, tck=, ...)
- Parameters:
  - side → an integer indicating the side of the graph to draw the axis (1=bottom, 2=left, 3=top, 4=right) at a numeric vector indicating where tick marks should be drawn labels a character vector of labels to be placed at the tickmarks (if NULL, the at values will be used)
  - pos → the coordinate at which the axis line is to be drawn. (i.e., the value on the other axis where it crosses)
  - lty → line type
  - col → the line and tick mark color
  - las → labels are parallel (=0) or perpendicular(=2) to axis
  - tck → length of tick mark as fraction of plotting region (negative number is outside graph, positive number is inside, 0 suppresses ticks, 1 creates grid lines) default is -0.01

## axis()

- The option `axes=FALSE` suppresses both x and y axes.  
`xaxt="n"` and `yaxt="n"` suppress the x and y axis respectively

## points() and lines()

- points() → add points to a plot
- lines() → add lines to a plot
- Difference: only in the default argument for the parameter type
- The default for points() is type = "p", and that for lines() is type = "l". (Explicitly specifying type = "p" causes either function to plot points, type = "l" gives lines.)

## Minor Tick marks

- The `minor.tick( )` function in the `Hmisc` package adds minor tick marks
- Syntax: `library(Hmisc) minor.tick(nx=n, ny=n, tick.ratio=n)`
- `nx` is the number of minor tick marks to place between x-axis major tick marks. `ny` does the same for the y-axis. `tick.ratio` is the size of the minor tick mark relative to the major tick mark. The length of the major tick mark is retrieved from `par("tck")`

## Adding References

- Add reference lines to a graph using the `abline( )` function
- `abline(h=yvalues, v=xvalues)`
- Other graphical parameters (such as line type, color, and width) can also be specified in the `abline( )` function

# legend

- Add a legend with the `legend()` function
- Syntax: `legend(location, title, legend, ...)`
- Parameters:
  - location → There are several ways to indicate the location of the legend
  - You can give an x,y coordinate for the upper left hand corner of the legend
  - You can use `locator(1)`, in which case you use the mouse to indicate the location of the legend
  - You can also use the keywords "bottom", "bottomleft", "left", "topleft", "top", "topright", "right", "bottomright", or "center"
  - If you use a keyword, you may want to use `inset=` to specify an amount to move the legend into the graph (as fraction of plot region)



# legend

- Parameters:
  - title→A character string for the legend title (optional)
  - legend→ A character vector with the labels
  - Other options→ If the legend labels colored lines, specify col= and a vector of colors
  - If the legend labels point symbols, specify pch= and a vector of point symbols
  - If the legend labels line width or line style, use lwd= or lty= and a vector of widths or styles
  - To create colored boxes for the legend (common in bar, box, or pie charts), use fill= and a vector of colors
  - Other common legend options include bty for box type, bg for background color, cex for size, and text.col for text color. Setting horiz=TRUE sets the legend horizontally rather than vertically

## Fine control parameter settings

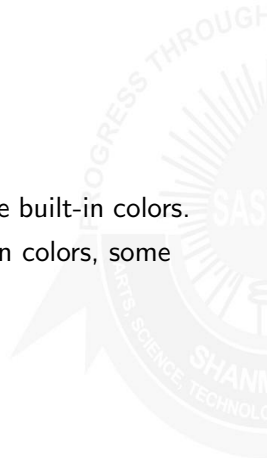
- `pch`→controls the choice of plotting symbol
- `cex`→ (character expansion) controls the size of the plotting symbol
- `col`→ controls the color
- These parameters can be set when a function such as `plot()` or `points()` is called, in which case they apply only to that function call
- `xlim,ylim`→determine the limits in the x and y directions respectively
- When `xlim` and/or `ylim` is not set explicitly, the range of data values determines the limits
- In any case, the axis is by default extended by 4% relative to those limits
- The setting `pty="s"` gives a square plot.

# Changing Parameter Settings

- `par()` → change parameter settings more permanently
- Example: `par(cex=1.2)`
- `cex.axis, cex.labels` → size of the axis annotation can be controlled and control the size of the axis labels
- `par(mfrow=c(m,n))` → get an m by n layout of graphs on a page

# Use of Color

- The default palette is a small selection from the built-in colors.
- `colors()` → returns the 657 names of the built-in colors, some of them aliases for the same color



# Plotting of expressions and mathematical symbols

- `symbols()`
  - This function draws symbols on a plot
  - One of six symbols; circles, squares, rectangles, stars, thermometers, and boxplots, can be plotted at a specified set of x and y coordinates
  - Specific aspects of the symbols, such as relative size, can be customized by additional parameters.
- `symbols(0, 0, circles=0.95, bg="gray", xlim=c(-1,2.25), ylim=c (-1,1), inches=FALSE) text(1.75, 0, expression(" Area" == pi*phantom(" ")*italic(r)^2))`

# Identification and location on the figure region

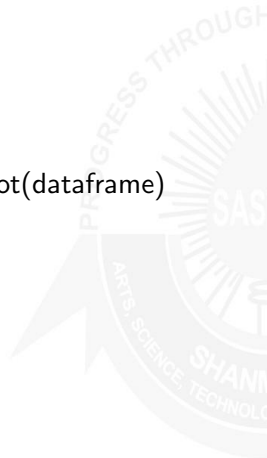
- `identify()`
  - `identify` reads the position of the graphics pointer when the (first) mouse button is pressed
  - It then searches the coordinates given in `x` and `y` for the point closest to the pointer
  - If this point is close enough to the pointer, its index will be returned as part of the value of the call
- `locator()`
  - Reads the position of the graphics cursor when the (first) mouse button is pressed.
  - Syntax: `locator(n = 512, type = "n", ...)` where "`n`" is the maximum number of points to locate. Valid values start at 1 and `type` is type One of "`n`", "`p`", "`l`" or "`o`". If "`p`" or "`o`" the points are plotted; if "`l`" or "`o`" they are joined by lines

## Identification and location on the figure region

- The user positions the cursor at the location for which co-ordinates are required, and clicks the left mouse button
- Depending on the platform, the identification or labeling of points may be terminated by pointing outside of the graphics area and clicking, or by clicking with a button other than the first
- If continued, the process will terminate after some default number  $n$  of points, which the user can set

## Other Graphs

- Plot methods for objects other than vectors-plot(dataframe)
- Lattice graphics versus base graphics-xyplot()
- hist()- histogram
- density()





# Good and bad graphs

- Ordinary Graphs
  - Graphs that are unlikely to mislead is preferred
  - Focus on important features and avoid distracting features
  - Lines that are intended to attract can be thickened
- Scatter Plots
  - Intention is typically to draw attention to the points
  - If the number of points is minimum, use of heavy black dots or other filled symbols are preferred instead of lines or curves
  - If the number of points are more and there are substantial overlap then it is better to use open symbols
  - If there is an extensive overlap, ink will fill that region more densely
- Colors
  - Colors, or gray scales, can often be used to distinguish groupings in the data
  - The eye has difficulty in focusing simultaneously on widely separated colors that are close together on the same graph

# Lattice (trellis) graphics

- Lattice graphics allows the use of the layout on the page, the choice of plotting symbols and colors of symbols, and the layout with panels, to represent important aspects of data structure

# Introduction

- When a researcher has a new set of data to analyze, the following questions arise
  - What is the best way to begin?
  - What forms of data exploration will draw attention?
  - What are the checks that will make it plausible that the data really will support an intended formal analysis?
  - What mix of exploratory analysis and formal analysis is appropriate?
  - What attention should be paid to analyses that other researchers have done with similar data?

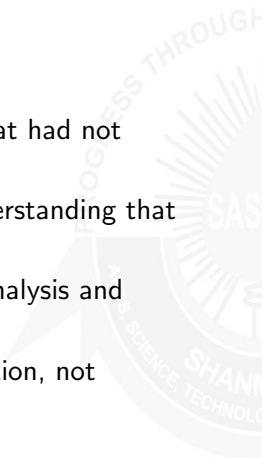
## Points to be addressed in this section

- Emphasize the importance of looking for patterns and relationships
- Numerical summaries, such as an average, can be very useful, but important features of the data may be missed without a glance at an appropriate graph
- The best modern statistical software makes a strong connection between data analysis and graphics, combining the computers ability to crunch numbers and present graphs with the ability of a trained human eye to detect pattern
- Often, careful consideration is needed, to choose a graph that will be effective for the purpose that is in hand

# Revealing views of the data

- Exploratory Data Analysis (EDA) is a name for a collection of data display techniques that are intended to let the data speak for themselves, prior to or as part of a formal analysis
- EDA formalizes and extends the use of graph to check their data
- Even if a data set has not been collected in a way that makes it suitable for formal statistical analysis, exploratory data analysis techniques can often be used to glean clues from it
- An effective EDA display presents data in a way that will make effective use of the human brains abilities as a pattern recognition device

# Roles of EDA

- 
- ① EDA may suggest ideas and understandings that had not previously been contemplated
  - ② EDA results may challenge the theoretical understanding that guided the initial collection of the data.
  - ③ EDA allows the data to criticize an intended analysis and facilitates checks on assumptions
  - ④ EDA techniques may reveal additional information, not directly related to the research question

# Views of a Single Sample

- Histogram:
  - The histogram is a basic (and over-used) EDA tool for displaying the frequency distribution of a set of data
  - The area of each rectangle of a histogram is proportional to the number of observations whose values lie within the width of the rectangle
  - A mound-shaped histogram may make it plausible that the data follow a normal distribution (the bell curve)
  - In small samples, however, the shape can be highly irregular
  - The appearance can depend on the choice of breakpoints(bins), which is a further reason for caution in interpreting the shape

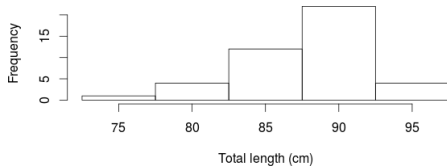
## Example

```
library(DAAG)
# Ensure that the DAAG package is attached
## Form the subset of possum that holds
#data on females only
fossum <- subset(possum, sex=="f")
attach(fossum)
hist(totlngth, breaks =72.5 + (0:5) * 5, ylim = c(0, 22),
     xlab="Total length(cm)", main ="A: Breaks at 72.5, 77.5,
     ...")
hist(totlngth, breaks =75 + (0:5) * 5, ylim = c(0, 22),
     xlab="Total length (cm)", main="B: Breaks at 75, 80,
     ...")
```

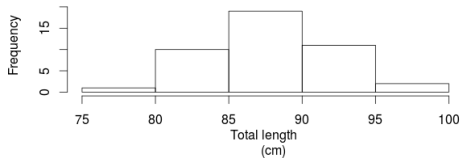


# Example

**B: Breaks at 75, 80, ...**



**A: Breaks at 72.5, 77.5, ...**



## Difference between the two Figures of the previous slide

- The only difference in the construction of the two plots is the choice of breakpoints, but one plot suggests that the distribution is asymmetric (skewed to the left), while the other suggests symmetry

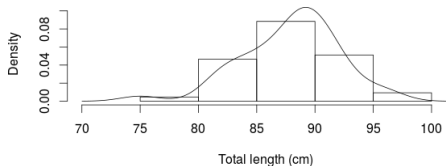
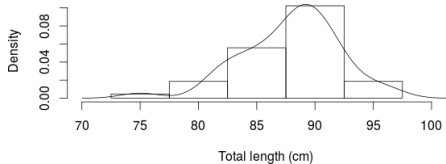
# Histogram vs Density Estimate

- A histogram is a crude form of a density estimate
- A better alternative is, often, a smooth density estimate
- The width of histogram bars must be chosen somewhat subjectively
- Density estimates require the choice of a bandwidth parameter that controls the amount of smoothing
- Anyway, in both cases, the software has default choices that can work reasonably well

# Histogram vs Density Estimate Example

```
dens <- density(totlngth)
xlim <- range(dens$x); ylim <- range(dens$y)
hist(totlngth, breaks = 72.5 + (0:5) * 5, probability = T,
     xlim = xlim, ylim = ylim,
     xlab="Total length (cm)", main=" ")
lines(dens)
hist(totlngth, breaks = 75 + (0:5) * 5, probability = T,
     xlim = xlim, ylim = ylim,
     xlab="Total length (cm)", main=" ")
lines(dens)
par(mfrow=c(1,1)); detach(fossum)
```

# Histogram vs Density Estimate Example



# Histogram vs Density Estimate Example

- The height of the density curve at any point is an estimate of the proportion of sample values per unit interval, locally at that point
- In Figures the cell of the histogram between the breakpoints (87.5, 92.5] has a frequency of 22
- The total frequency is 43, and the width of the cell is 5, this corresponds to a density of  $\frac{22}{43 \times 5} = 0.102$ , which is just a little smaller than the height of the density curve at its highest point or mode

# Histogram vs Density Estimate Example

- Density curves are preferable to histograms for drawing attention to particular forms of non-normality, such as that associated with strong skewness in the distribution, but are still not an adequate tool
- A more effective way of checking for normality - the normal probability plot
- Density curves are useful for estimating the population mode, i.e., the value that occurs most frequently

# The stem-and-leaf display

- The stem-and-leaf display is a fine-grained alternative to a histogram, for use in displaying a single column of numbers
- Example:
  - r code for stem-and-leaf plot of the heights of the 37 rowers in the ais data set
  - `with(ais, stem(ht[sport=="Row"]))`

The decimal point is 1 digit(s) to the right of the |

```
15 | 6
16 |
16 | 5
17 | 4
17 | 5678899
18 | 00000011223
18 | 55666668899
19 | 123
19 | 58
```



# The stem and leaf display Example

- The data have been rounded to the nearest centimeter
- The numbers that are displayed are, in order of magnitude, 156, 165, 174, . . . .
- The display has broken these down as  $150 + 6$ ,  $160 + 5$ ,  $170 + 4$ , . . . .
- The column of numbers on the left of the vertical bars (15, 16, . . . ) comprises the stem
- These are the tens of centimeters parts of the numbers
- The leaf part for that number (6, 5, 4, . . . ) is what remains after removing the stem
- These are printed, in order, to the right of the relevant vertical bar

# Box plot

- Like the histogram, the box-plot is a coarse summary
- It allows a trained eye to comprehend at a glance specific important features of the data
- Example:
  - r code for a boxplot of total lengths of females in the possum data set
  - `with(fossum, boxplot(totLngth, horiz=TRUE))`
  - `bwplot(~totLngth, data=fossum)`



## Box plot

- One point lies outside the "whiskers" to the left, and is thus flagged as a possible outlier
- An outlier is a point that, in some sense, lies away from the main body of the data
- In identifying points that are flagged as possible outliers, the normal distribution is taken as the standard
- Using the default criterion one point in 100 will on average, for data from a normal distribution, be flagged as a possible outlier
- Thus, in a boxplot display of 1000 values that are drawn at random from a normal distribution, around 10 will be plotted out beyond the box-plot whiskers and thus flagged as possible outliers

# Patterns in univariate time series

## Definition (Time Series)

A sequence of random variables measuring certain quantity of interest over time

- Convention:
  - In applications, a time series is a sequence of measurements of some quantity of interest taken at different time points
  - Usually, the measurements are observed at equally spaced time intervals, resulting in a discrete-time time series
  - If  $t$  is continuous, then we have a continuous-time time series. The series becomes a stochastic process
  - Notation:  $X_t$  or  $Y_t$  or  $Z_t$  for a discrete time time series and  $X(t)$  or  $Y(t)$  or  $Z(t)$  for a continuous case
  - $X_t$  can be a continuous random variable or a discrete random variable, e.g. counts or the number of wins of a football team in a season

# Basic Objective of Time Series Analysis

- The objective of (univariate) time series analysis is to find the dynamic dependence of  $X_t$ , i.e. the dependence of  $X_t$  on its past values  $\{X_{t-1}, X_{t-2}, \dots\}$
- A linear model means  $X_t$  depends linearly on its past values
- The term "univariate time series" refers to a time series that consists of single (scalar) observations recorded sequentially over equal time increments. Example: monthly  $\text{CO}_2$  concentrations
- Although a univariate time series data set is usually given as a single column of numbers, time is in fact an implicit variable in the time series

# Univariate Time Series

- If the data are equi-spaced, the time variable, or index, does not need to be explicitly given
- The time variable may sometimes be explicitly used for plotting the series
- However, it is not used in the time series model itself

## Patterns in bivariate data

- The scatter plot is a simple but important tool for the examination of pairwise relationships
- The following figure shows data from a tasting session where each of 17 panelists assessed the sweetness of each of two milk samples, one with four units of additive, and the other with one unit of additive. The line  $y = x$  has been added
- There is a positive correlation between assessments for the two samples
- If one was rated as sweet, by and large so was the other
- The line  $y = x$  assists in comparing the two samples
- Most panelists (13 out of 17) rated the sample with four units of additive as sweeter than the sample with one unit of additive

## Bivariate data example

```
library(DAAG)
xyrange <- range(milk)
plot(four~one, data = milk, xlim = xyrange,
     ylim = xyrange, pch = 16, pty="s")
# with rugs
library(DAAG)
xyrange <- range(milk)
plot(four~one, data = milk, xlim = xyrange,
     ylim = xyrange, pch = 16, pty="s")
# pty="s": square plotting region
rug(milk$one)
rug(milk$four, side = 2)
abline(0, 1)
```



## Example with and without rugs

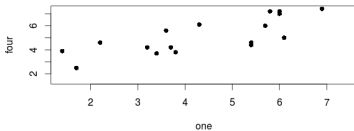


Figure: without rugs

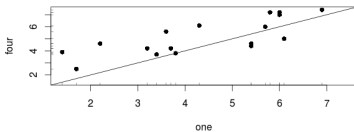


Figure: with rugs

# Fitting a smooth trend curve

- The following figure shows data from a study that measured both electrical resistance and apparent juice content for a number of slabs of kiwifruit

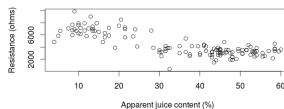
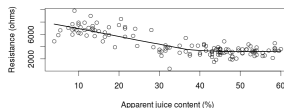


Figure: without smooth curve



## Fitting a smooth trend curve

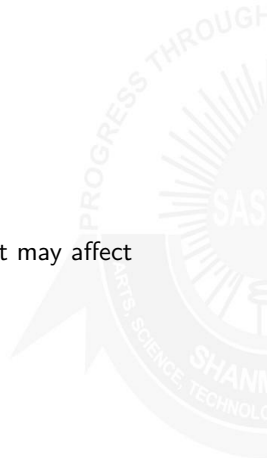
```
plot(ohms~juice, xlab="Apparent juice content (%)",  
ylab="Resistance (ohms)", data=fruitohms)  
with(fruitohms, lines(lowess(juice, ohms), lwd=2))
```

## Fitting a smooth trend curve

- The fitted smooth curve shows a form of response that is clearly inconsistent with a straight line
- It suggests an approximate linear relationship for juice content up to somewhat over 35%
- Once the juice content reaches around 45%, the curve becomes a horizontal line, and there is no evident further change in resistance
- There is no obvious simple form of equation that might be used to describe the curve
- A curve fitted using `lowess()` or another such smoothing function can provide a useful benchmark against which to compare the curve given by a theoretical or other mathematical form of equation that the data are thought to follow

# Data Summary

- Data summaries may:
  - ① be of interest in themselves
  - ② give insight into aspects of data structure that may affect further analysis
  - ③ be used as data for further analysis



## Example

```
library(DAAG)
tab <- with(nswpsid1, table(trt, nodeg, useNA="ifany"))
print(tab)
dimnames(tab) <- list(trt=c("none", "training"),
                      educ = c("completed", "dropout"))
print(tab)
```

# Tables

- If  $x_1, x_2, \dots, x_n$  are all columns (factors or vectors) of the same length and each is supplied as an argument to `table()`, the result is an  $n$ -way table
- For example, `table(x1, x2, x3)` gives a three-way table
- The first argument defines rows, though it is printed horizontally if there is just one column
- The second argument defines columns
- The table slices (rows by columns) that correspond to different values of the third argument appear in succession down the page, and so on

# Mosaic Plots

- We have visualized relationships among quantitative/continuous variables
- But how to visualize categorical variables?
- For single categorical variable, we can use a bar or pie chart
- If there are two categorical variables, we can look at a 3D bar chart
- But what happens if there are more than two categorical variables?
- One approach is to use mosaic plots



# Mosaic Plots

- In a mosaic plot, the frequencies in a multidimensional contingency table are represented by nested rectangular regions that are proportional to their cell frequency
- Color and or shading can be used to represent residuals from a fitted model
- Mosaic plots can be created with the `mosaic()` function from the `vcd` library

## Example

- Consider the Titanic dataset available in the base installation
- It describes the number of passengers who survived or died, cross-classified by their class (1st, 2nd, 3rd, Crew), sex (Male, Female), and age (Child, Adult)
- A well-studied dataset

# R-code for cross classification

```
ftable(Titanic)
```

			Survived	No	Yes
Class	Sex	Age			
1st	Male	Child		0	5
		Adult		118	57
	Female	Child		0	1
		Adult		4	140
2nd	Male	Child		0	11
		Adult		154	14
	Female	Child		0	13
		Adult		13	80
3rd	Male	Child		35	13
		Adult		387	75
	Female	Child		17	14
		Adult		89	76
Crew	Male	Child		0	0
		Adult		670	192
	Female	Child		0	0
		Adult		3	20

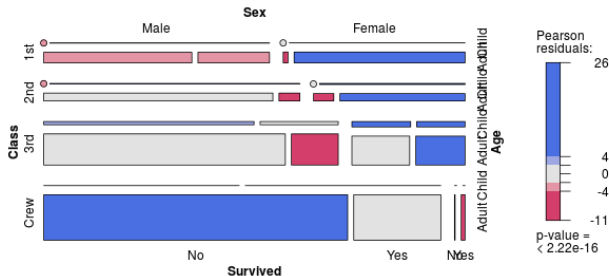
# mosaic()

- Syntax
  - ① `mosaic(table)`, where `table` is a contingency table in array form
  - ② `mosaic(formula, data=)` where `formula` is a standard R formula, and `data` specifies either a data frame or table
- Adding the option `shade=TRUE` will color the figure based on Pearson residuals from a fitted model (independence by default) and the option `legend=TRUE` will display a legend for these residuals

# R-Code for Titanic Example

```
library(vcd)
mosaic(Titanic, shade=TRUE, legend=TRUE)
and
library(vcd)
mosaic(~Class+Sex+Age+Survived, data=Titanic, shade=TRUE,
       legend=TRUE)
```

# Example



## Example

- The formula version gives you greater control over the selection and placement of variables in the graph
- Great deal of information packed into this one picture
- If we move from crew to first class, the survival rate increases precipitously
- Most children were in third and second class
- Most females in first class survived, whereas only about half the females in third class survived
- There were few females in the crew, causing the Survived labels (No, Yes at the bottom of the chart) to overlap for this group

## Example

- Extended mosaic plots add color and shading to represent the residuals from a fitted model
- In this example, the blue shading indicates cross-classifications that occur more often than expected, assuming that survival is unrelated to class, gender, and age
- Red shading indicates cross-classifications that occur less often than expected under the independence model
- The graph indicates that more first-class women survived and more male crew members died than would be expected under an independence model
- Fewer third-class men survived than would be expected if survival was independent of class, gender, and age



# Generating Frequency Tables

- R provides several methods for creating frequency and contingency tables

`table(var1, var2, ..., varN)`- Creates an N-way contingency table from N categorical variables (factors)

`xtabs(formula, data)`- Creates an N-way contingency table based on a formula and a matrix or data frame

`prop.table(table, margins)`- Expresses table entries as fractions of the marginal table defined by the margins

`margin.table(table, margins)`- Computes the sum of table entries for a marginal table defined by the margins

`addmargins(table, margins)`- Puts summary margins (sums by default) on a table

`ftable(table)`- Creates a compact "flat" contingency table

# One Way Tables Example

```
mytable <- with(Arthritis, table(Improved))  
mytable
```

```
Improved  
None    Some Marked  
42      14      28
```

```
prop.table(mytable)
```

```
Improved  
None      Some      Marked  
0.5000000 0.1666667 0.3333333
```

## Two Way Tables

- For two-way tables, the format for the `table()` function is `mytable <- table(A, B)`, where A is the row variable, and B is the column variable
- Alternatively, the `xtabs()` function allows you to create a contingency table using formula style input
- The format is `mytable <- xtabs( A + B, data=mydata)` where mydata is a matrix or data frame
- In general, the variables to be cross-classified appear on the right of the formula (that is, to the right of the `~`) separated by `+` signs
- If a variable is included on the left side of the formula, its assumed to be a vector of frequencies

# Example

```
mytable <- xtabs(~ Treatment+Improved, data=Arthritis)
```

	Improved		
Treatment	None	Some	Marked
Placebo	29	7	7
Treated	13	7	21

```
margin.table(mytable, 1)
```

Treatment	
Placebo	Treated
43	41

```
prop.table(mytable, 1)
```

	Improved		
Treatment	None	Some	Marked
Placebo	0.6744186	0.1627907	0.1627907
Treated	0.3170732	0.1707317	0.5121951

## addmargins()

- addmargins() function to add marginal sums to these tables

```
addmargins(mytable)
```

```
addmargins(prop.table(mytable))
```

```
addmargins(prop.table(mytable, 1), 2)
```

```
addmargins(prop.table(mytable, 2), 1)
```

- Note: The table() function ignores missing values (NAs) by default
- To include NA as a valid category in the frequency counts, include the table option useNA="ifany"

# Example

```
> addmargins(mytable)
Improved
Treatment None Some Marked Sum
Placebo 29 7 7 43
Treated 13 7 21 41
Sum 42 14 28 84
> addmargins(prop.table(mytable))
Improved
Treatment None Some Marked Sum
Placebo 0.34523810 0.08333333 0.08333333 0.51190476
Treated 0.15476190 0.08333333 0.25000000 0.48809524
Sum 0.50000000 0.16666667 0.33333333 1.00000000
> addmargins(prop.table(mytable, 1), 2)
Improved
Treatment None Some Marked Sum
Placebo 0.6744186 0.1627907 0.1627907 1.0000000
Treated 0.3170732 0.1707317 0.5121951 1.0000000
> addmargins(prop.table(mytable, 2), 1)
Improved
Treatment None Some Marked
Placebo 0.6904762 0.5000000 0.2500000
Treated 0.3095238 0.5000000 0.7500000
Sum 1.0000000 1.0000000 1.0000000
```

# CrossTable()

- The CrossTable() function produces two-way tables
- It is available in the gmodels package



## Descriptive statistics by group

- When comparing groups of individuals or observations, the focus is usually on the descriptive statistics of each group, rather than the total sample
- `aggregate()`-function to obtain descriptive statistics by group
- Example:

```
vars <- c("mpg", "hp", "wt")  
aggregate(mtcars[vars], by=list(am=mtcars$am), mean)  
aggregate(mtcars[vars], by=list(am=mtcars$am), sd)
```



# Example

```
> aggregate(mtcars[vars], by=list(am=mtcars$am), mean)
  am      mpg      hp      wt
1  0 17.14737 160.2632 3.768895
2  1 24.39231 126.8462 2.411000
> aggregate(mtcars[vars], by=list(am=mtcars$am), sd)
  am      mpg      hp      wt
1  0 3.833966 53.90820 0.7774001
2  1 6.166504 84.06232 0.6169816
```



## Example Contd..

- Suppose instead of `am=mtcars$am`, if we use `mtcars$am` alone then the column heading will be `Group.1`

```
> aggregate(mtcars[vars], by=list(mtcars$am), mean)
```

	Group.1	mpg	hp	wt
1	0	17.14737	160.2632	3.768895
2	1	24.39231	126.8462	2.411000

- Use the assignment to provide a more useful column label
- If you have more than one grouping variable, you can use code like `by=list(name1=groupvar1, name2=groupvar2, ... , groupvarN)`

# Limitations of aggregate()

- aggregate() only allows you to use single value functions such as mean, standard deviation, and the like in each call
- It wont return several statistics at once

## by()

- If we want to return several statistics at once, we can use `by()` function
- Syntax: `by(data, INDICES, FUN)` where `data` is a data frame or matrix, `INDICES` is a factor or list of factors that define the groups, and `FUN` is an arbitrary function
- Example:

```
dstats <- function(x)(c(mean=mean(x), sd=sd(x)))  
> by(mtcars[vars], mtcars$am, dstats)
```

# Extensions

- The doBy package and the psych package also provide functions for descriptive statistics by group
- They aren't distributed in the base installation and must be installed before first use
- The summaryBy() function in the doBy package has the syntax: `summaryBy(formula, data=dataframe, FUN=function)` where the formula takes the form `var1 + var2 + var3 + ... + varN ~ groupvar1 + groupvar2 + ... + groupvarN`
- Example:

```
summaryBy(mpg+hp+wt~am, data=mtcars, FUN=mean)
```

```
      am mpg.mean  hp.mean  wt.mean  
1    0 17.14737 160.2632 3.768895  
2    1 24.39231 126.8462 2.411000
```

# Standard Deviation and Inter Quartile Range

- An important measure of variation in a population is the population standard deviation (often written  $\sigma$ ), which is almost always unknown
- The variance  $\sigma^2$ , which is the square of the standard deviation, is widely used in formal inference
- The sample standard deviation, used to estimate the population standard deviation when a random sample has been taken, is

$$s = \sqrt{\frac{\sum (x - \bar{x})^2}{n - 1}}$$

- In R, use the function `sd()` to calculate the standard deviation, or `var()` to calculate the variance
- The standard deviation is in the same units as the original measurements
- For  $s$  to be an accurate estimate of  $\sigma$ , use large samples

# Example

```
summaryBy(length~species, data=cuckoos, FUN=sd)  
sapply(split(cuckoos$length, cuckoos$species), sd)
```

```
      species length.sd  
1 hedge.sparrow 1.0494373  
2 meadow.pipit 0.9195849  
3 pied.wagtail 1.0722917  
4         robin 0.6821229  
5   tree.pipit 0.8800974  
6           wren 0.7542262
```

hedge.sparrow	meadow.pipit	pied.wagtail	robin	tree.pipit
1.0494373	0.9195849	1.0722917	0.6821229	0.8800974
wren				
0.7542262				

# Degrees of Freedom

- The denominator  $n-1$  is the number of degrees of freedom remaining after estimating the mean
- With one data point, the sum of squares about the mean is zero, the degrees of freedom are zero, and no estimate of the variance is possible
- The degrees of freedom are the number of data values, additional to the first data value
- Degrees of freedom are reduced by 1 for each model parameter that is estimated

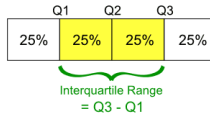


## Other Measures of Variability

- The standard deviation is similar in concept to the inter-quartile range  $H$  is the difference between the first and third quartiles. (The region between the lower and upper quartiles takes in 50% of the data)
- For data that are approximately normally distributed, note the approximate relationship  $s \approx 0.75H$ .
- If data are approximately normally distributed, one standard deviation either side of the mean takes in roughly 68% of the data

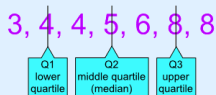
# Inter Quartile Range

The "Interquartile Range" is from Q1 to Q3:



To calculate it just **subtract Quartile 1 from Quartile 3**, like this:

Example:



The **Interquartile Range** is:

$$Q3 - Q1 = 8 - 4 = 4$$

# Median Absolute Deviation

- Median Absolute Deviation is calculated using the function `mad()`
- This calculates the median of the absolute deviations from the median
- By default this is multiplied by 1.4286, to ensure that in a large sample of normally distributed values the value returned should approximately equal the standard deviation
- `sapply(split(cuckoos$length, cuckoos$species), mad)`

hedge.sparrow	meadow.pipit	pied.wagtail	robin	tree.pipit
1.03782	0.44478	1.48260	0.74130	0.88956
wren				
0.74130				