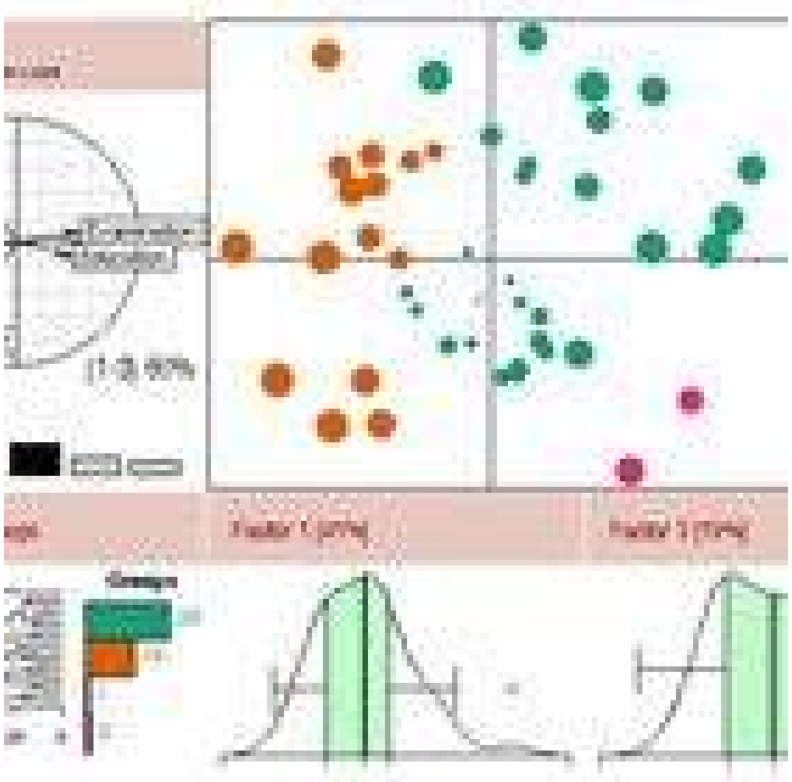
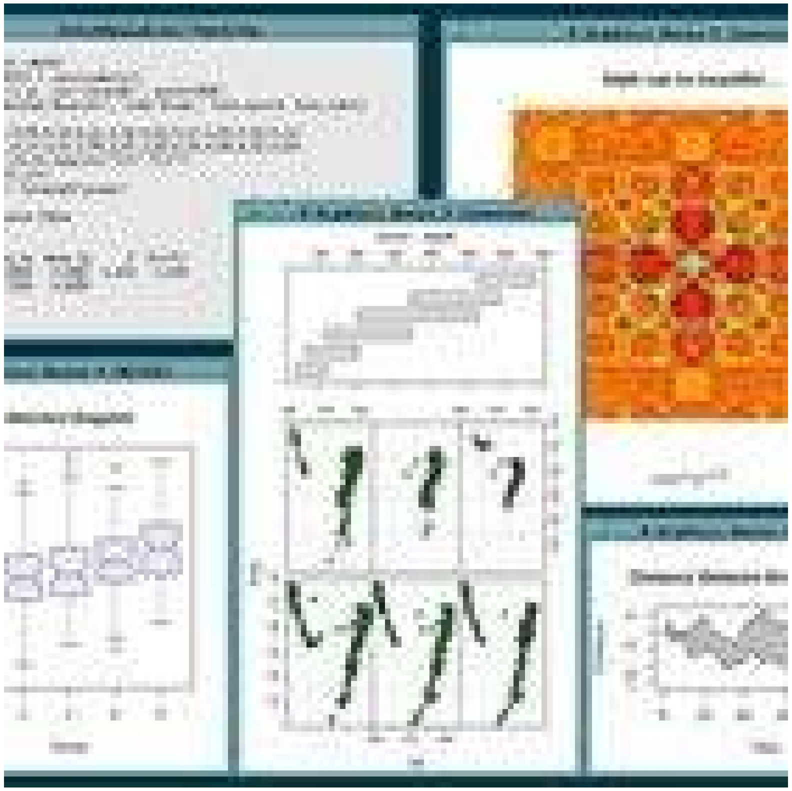
# SCS2111 – Laboratory II

Lecture 01

19.08.2015





# SCS2111 – Laboratory II

- 1L + 1P (Compulsary)
- Lecturers :
  - Dr. C. H. Magalla (champa@stat.cmb.ac.lk)
  - Mrs. Mindika Premachandra (amp@ucsc.cmb.ac.lk)
- Lectures : 1.00 – 2.00 pm Thursdays (Mini Aud.)
- Practicals :
  - 3.00 – 5.00 pm Wednesdays (IRQUE)
  - 2.00 – 5.00 pm Thursdays (Labs A & B, IRQUE)

# Learning Outcomes

- Be familiar with Mathlab/Octave and R environments
  - Be able to do basic mathematics and simple manipulations with Mathlab and R
  - Be able to do basic statistical operations with R
  - Use Mathlab and R for basic plotting
  - Learn scripting with Mathlab and R
  - Use different data analysis techniques in Mathlab and R

# Course Contents : Mathlab/Octave

- Manipulating Variables
- Basic Mathematics using Mathlab
  - E.g.: Linear Algebra, basic statistics, differentiation and integrals, Fourier transforms
- Basic plotting and curve fitting
- Programming Scripts and Functions
- Image Processing functions and Animations
- Debugging
- Data Structures and File Management
- Symbolics, Simulink, file I/O, building GUIs

# Course Contents: R

- Introduction to R environment, Getting Help, R Commands, Case sensitivity, Recall and correction of previous commands, R scripting and executing
- Simple manipulations: objects, vector arithmetic, arrays and matrices, lists and data frames
- Reading data from file
- R as a statistical package: Analysis of Qualitative and Quantitative data, probability distributions, Numerical measures, probability distributions

# Course Contents : R

- Graphical procedures: plotting, displaying multivariate data, display graphics
- Multivariate correlations: Bayes approach and Naïve Bayes classifier
- Principal Component Analysis
- Cluster Analysis: K means clustering, K nearest neighbours clustering
- Linear Discriminant Analysis
- Decision Tree Models (Tree-based) models

# Assessment

- Lab sheets
- Assignments
- Final Examination
- Rubric : 70% Theory, 30% Practicals?

Detailed assessment plan will be informed later

# What is R?

- R is a free software environment for statistical computing and graphics
- R was created by Ross Ihaka and Robert Gentleman at the University of Auckland
- A GNU project which is similar to the **S** language,
  - developed at Bell Laboratories by John Chambers and colleagues.
  - **R** can be considered as a different implementation of **S**.
- The source code for the R software environment is written primarily in C, Fortran, and R.
- Pre-compiled binary versions are provided for various OS
- http://www.r-project.org/index.html

# Why learn R?

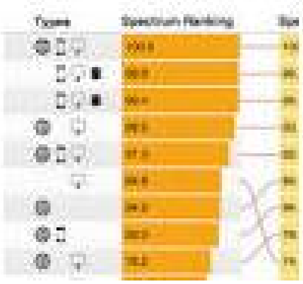
- R is FREE, easy to use, and open source.
  - Commercial options: SAS, SPSS
- The R language is widely used among statisticians and data miners for developing statistical software and data analysis
- The "de facto" standard for data analysis and data mining

- A complete programming language
- Comes with a large library of pre-defined functions
- Better suited for advanced users who want all the power in their hands
  - R supports matrix arithmetic
  - R's data structures include vectors, matrices, arrays, data frames (similar to tables in a relational database) and lists.
  - R's extensible object system includes objects for (among others): regression models, time-series and geo-spatial coordinates.


The 2015 Top Ten **Programming Languages**
IEEE Spectrum - Jul 20, 2015
What are the most popular **programming languages**? ... The big mover is **R**, a statistical computing language that's handy for analyzing and ...

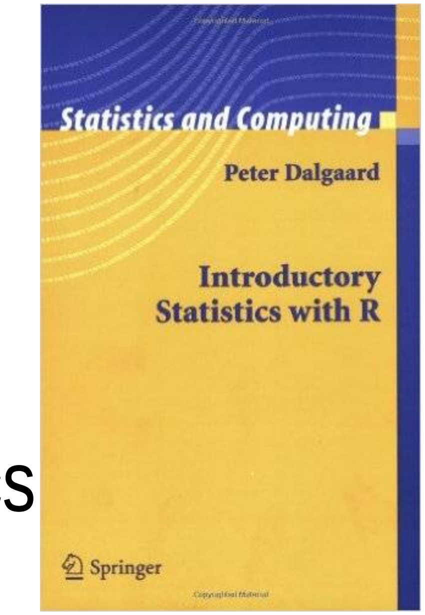
**R** Rises in IEEE Ranking of Top **Programming Languages**
ADT Magazine - Jul 21, 2015
IEEE Spectrum has followed up last year's report on the top **programming languages** with a new study that sees **R** making a big jump in the ...

In data science, **the R language** is swallowing Python
InfoWorld - Jul 24, 2015
It's always precarious to compare **programming languages**, given their ... While **R** is a language developed by and for statisticians, Python has ...




The Most Popular **Programming Languages** of 2015
ProgrammableWeb - Aug 4, 2015
While the top 5 remain unchanged, C has moved within touching distance of Java, and statistical **programming language R** has jumped from ...

# Learning Resources




- An excellent introductory book is by Peter Dalgaard, “Introductory Statistics with R", Springer (2002)

- *An Introduction to R*. *Notes on R: A Programming Environment for Data Analysis and Graphics*. *Version 2.15.2* (*2012-10-26*). W. N. Venables, D. M. Smith.

# Interacting with R

- R is an interpreted language; users typically access it through a command-line interpreter.
- There are also several graphical front-ends for it.
- Unlike languages like C, Fortran, or Java, R is an interactive programming langauge.
- This means that R works interactively, using a question-and-answer model:
  - Start **R**
  - Type a command and press **Enter**
  - **R** executes this command (often printing the result)
  - **R** then waits for more input
  - Type **q()** to exit

# Here are some simple examples:

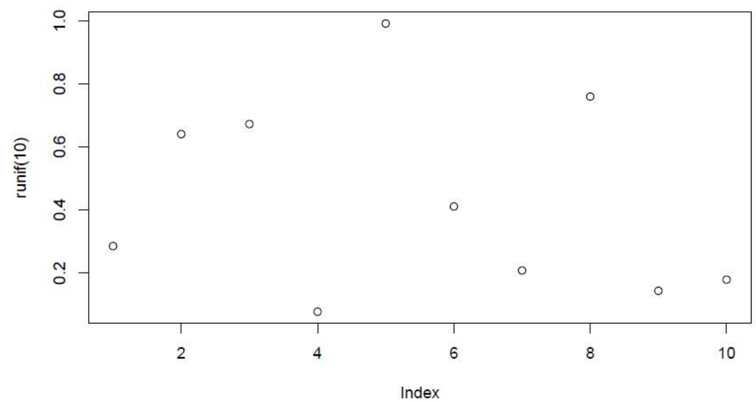
- Taken from AN INTRODUCTION TO R by Deepayan Sarkar

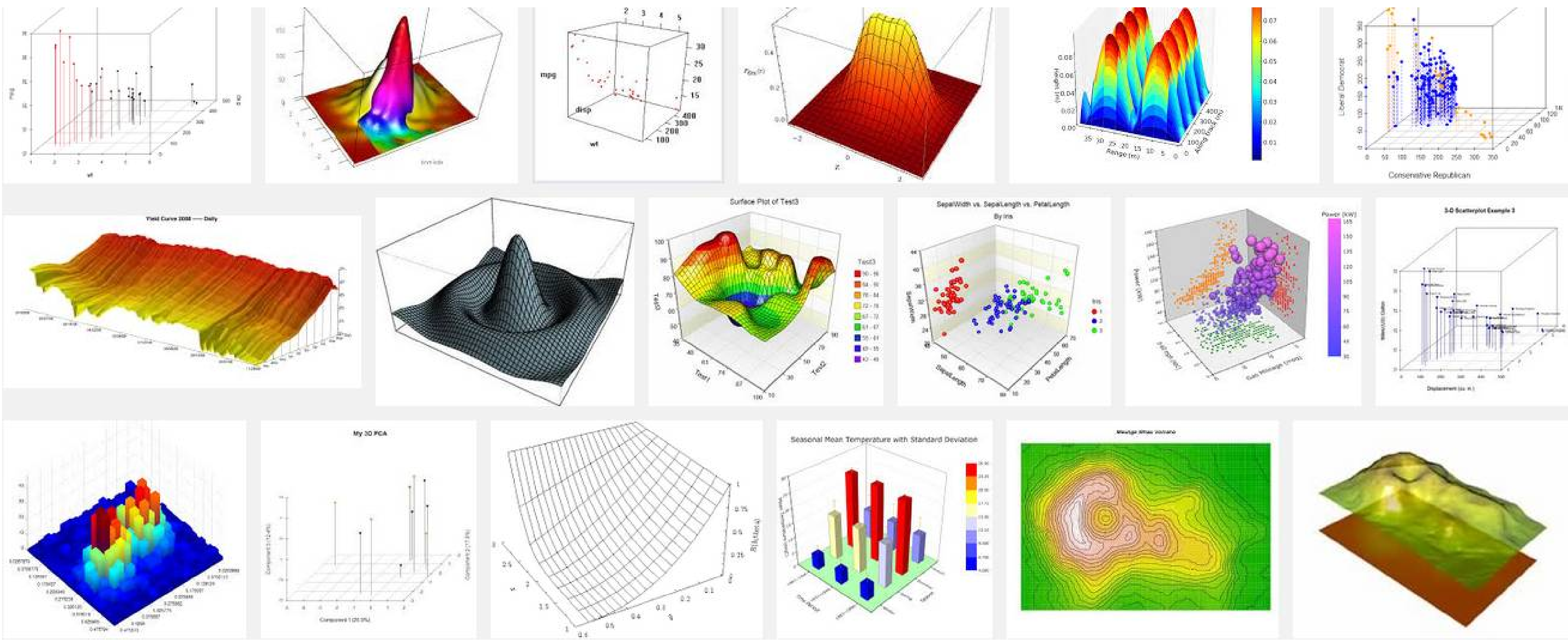
```
> 2 + 2
[1] 4
> exp(-2) ## exponential function
[1] 0.1353353
> log(100, base = 10)

[1] 2
> runif(10)
 [1] 0.39435130 0.98811744 0.07357143 0.16689946 0.80572031 0.05292909
 [7] 0.70498250 0.18781961 0.07865185 0.21618324
```

- The last command generates ten U(0; 1) random variables; the result (which is printed) is a vector of 10 numbers. **exp()**, **log()**, and **runif()** are functions.

# Plots

```
> plot(runif(10))
```



# Variables

- R has symbolic variables which can be assigned values.
- Assignment is done using the '<-' operator.
- The more C-like '=' also works (with some exceptions).

```
> s <- "this is a character string"
> s
[1] "this is a character string"
```

```
> x <- 2
> x + x
[1] 4
> yVar2 = x + 3
> yVar2
[1] 5
```

- Variable names can be almost anything, but they should not start with a digit, and should not contain spaces. Names are case-sensitive.
- Some common names are already used by R (c, q, t, C, D, F, I, T) and should be avoided.

# Vectorized arithmetic

- The elementary data types in R are all vectors; even the scalar variables we defined above are stored as vectors of length one.
- The **c(...)** construct can be used to create vectors:

```
> weight <- c(60, 72, 57, 90, 95, 72)
> weight
[1] 60 72 57 90 95 72
```

- To generate a vector of regularly spaced numbers, use

```
> seq(0, 1, length = 11)
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> 1:10
 [1]  1  2  3  4  5  6  7  8  9 10
```

- The c() function can be used to combine vectors as well as scalars,

```
> x <- seq(0, 1, length = 6)
> c(x, 1:10, 100)
 [1]   0.0   0.2   0.4   0.6   0.8   1.0   1.0   2.0   3.0   4.0   5.0   6.0
[13]   7.0   8.0   9.0  10.0 100.0
```

- Common arithmetic operations (including +, -, *, /, ^) and mathematical functions (e.g. sin(),cos(), log()) work element-wise on vectors, producing another vector:

```
> height <- c(1.75, 1.80, 1.65, 1.90, 1.74, 1.91)
> height^2
[1] 3.0625 3.2400 2.7225 3.6100 3.0276 3.6481
> bmi <- weight / height^2
> bmi
[1] 19.59184 22.22222 20.93664 24.93075 31.37799 19.736
> log(bmi)
[1] 2.975113 3.101093 3.041501 3.216102 3.446107 2.9824
```

- When two vectors are not of equal length, the shorter one is recycled.
  - E.g.: The following adds 0 to all the odd elements and 2 to all the even elements of 1:10:

```
> 1:10 + c(0, 2)
 [1]  1  4  3  6  5  8  7 10  9 12
```

# Summaries

- Many functions summarize a data vector by producing a scalar from a vector, e.g.,

```
> sum(weight)
[1] 446
> length(weight)
[1] 6
> avg.weight <- mean(weight)
> avg.weight
[1] 74.33333
```

- Simple summary statistics (mean, median, s.d., variance) can be computed from numeric vectors using appropriately named functions:

```
> x <- rnorm(100)
> mean(x)
[1] -0.1354077
> sd(x)
[1] 1.007307
> var(x)
[1] 1.014668
> median(x)
[1] -0.06083453
```

- Quantiles can be computed using the quantile() function.

- IQR() computes the inter-quartile range (**midspread** or **middle fifty)**.

```
> xquants <- quantile(x)
> xquants
         0%         25%         50%         75%        100%
-3.14440776 -0.74831291 -0.06083453  0.50980136  2.19369423
> xquants[4] - xquants[2]
     75%
1.258114
> IQR(x)
[1] 1.258114
> quantile(x, probs = c(0.2, 0.4, 0.6, 0.8))
       20%        40%        60%        80%
-1.0308886 -0.4388473  0.1236059  0.7357803
```

- The five-number summary (minimum, maximum, and quartiles) is given by fivenum(). A slightly extended summary is given by summary().

```
> fivenum(x)
[1] -3.14440776 -0.75013378 -0.06083453  0.51742360  2.19369423
> summary(x)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-3.14400 -0.74830 -0.06083 -0.13540  0.50980  2.19400
```

# Object-oriented progamming: classes and methods

- Let’s illustrate using a real dataset, one of the many datasets built into R (The well-known Iris data).
  - The dataset contains measurements on 150 flowers, 50 each from 3 species: Iris setosa, versicolor and virginica.
  - It is typically used to illustrate the problem of classification : Given the four measurements for a new flower, can we predict its Species?
  - Like most datasets, iris is not a simple vector, but a composite “data frame" object made up of several component vectors.
  - We can think of a data frame as a matrix-like object, with each row representing an observational unit (in this case, a flower), and columns representing multiple measurements made on the unit.

- The Iris data : The head() function extracts the first few rows, and the $ operator extracts individual components.

```
> head(iris) # The first few rows
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
> iris$Sepal.Length
  [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
 [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
 [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
 [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
 [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
 [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
[109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
[127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
[145] 6.7 6.7 6.3 6.5 6.2 5.9
```

- A more concise description is given by the str() function (short for “structure").

```
> str(iris)
'data.frame':      150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1
```

- As we can see,
  - the first four components of iris are numeric vectors,
  - but the last is a “factor". These are how R represents categorical variables.

- Let us now see the effect of calling summary() for different types of objects.
  - Note the different formats of the output.
  - Species is summarized by the frequency distribution of its values because it is a categorical variable, for which mean or quantiles are meaningless.
  - The entire data frame iris is summarized by combining the summaries of all its components.

```
> summary(iris$Sepal.Length)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  4.300   5.100   5.800   5.843   6.400   7.900
> summary(iris$Species)
    setosa versicolor  virginica
        50         50         50
> summary(iris)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
 Median :5.800   Median :3.000   Median :4.350   Median :1.300
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
       Species
 setosa    :50
 versicolor:50
 virginica :50
```

- R achieves this kind of object-specific customized output through a fairly simple object-oriented paradigm.
- Each R object has a class (“numeric", “factor", etc.).

```
> class(iris$Sepal.Length)
[1] "numeric"
> class(iris$Species)
[1] "factor"
> class(iris)
[1] "data.frame"
```

- summary() is what is referred to as a generic function, with class-specific methods that handle objects of various classes.
- When the generic summary() is called, R figures out the appropriate method and calls it.

- The rules are fairly intuitive.
- The last call gives the list of all available methods.

```
> methods(summary)
 [1] summary.aov              summary.aovlist          summary.aspell*
 [4] summary.connection       summary.data.frame       summary.Date
 [7] summary.default          summary.ecdf*            summary.factor
[10] summary.glm              summary.infl             summary.lm
[13] summary.loess*           summary.manova           summary.matrix
[16] summary.mlm              summary.nls*             summary.packageStatus*
[19] summary.PDF_Dictionary*  summary.PDF_Stream*      summary.POSIXct
[22] summary.POSIXlt          summary.ppr*             summary.prcomp*
[25] summary.princomp*        summary.srcfile          summary.srcref
[28] summary.stepfun          summary.stl*             summary.table
[31] summary.tukeysmooth*

   Non-visible functions are asterisked
```

- Objects of class “factor” are handled by summary.factor(),
- “data.frame”s are handled by summary.data.frame().
- There is no summary.numeric(), so numeric vectors are handled by summary.default().

# Getting Help

- help.start() starts a browser window with an HTML help interface.

- help(topic) displays the help page for a particular topic. Every R function has a help page.

```
> help(plot)
> ?plot
```

```
> help(plot, help_type = "html")
```

- help.search("search string") performs a subject/keyword search.

```
> help.search("logarithm")
> ??logarithm
```

- To directly run the examples given in help pages, use the example() function

```
> example(plot)
```

- The apropos() function, lists all functions (or other variables) whose name matches a specified character string.

```
> apropos("plot")
 [1] "assocplot"             "barplot"               "barplot.default"
 [4] "biplot"                "boxplot"               "boxplot.default"
 [7] "boxplot.matrix"        "boxplot.stats"         "cdplot"
[10] "coplot"                ".__C__recordedplot"    "fourfoldplot"
[13] "interaction.plot"      "lag.plot"              "matplot"
[16] "monthplot"             "mosaicplot"            "plot"
[19] "plot.default"          "plot.density"          "plot.design"
[22] "plot.ecdf"             "plot.function"         "plot.lm"
[25] "plot.mlm"              "plot.new"              "plot.spec"
[28] "plot.spec.coherency"   "plot.spec.phase"       "plot.stepfun"
[31] "plot.ts"               "plot.TukeyHSD"         "plot.window"
[34] "plot.xy"               "preplot"               "qqplot"
[37] "recordPlot"            "replayPlot"            "savePlot"
[40] "screeplot"             "spineplot"             "sunflowerplot"
[43] "termplot"              "ts.plot"
```

- Further Reading:
  - For a useful list of "standard" packages in R, see http://cran.fhcrc.org/doc/contrib/refcard.pdf
  - Browse through the index of help pages in specific packages, produced by

```
> library(help = base)
> library(help = graphics)
```

etc., and read the topics that seem interesting.

# Next…

- Importing data
- Packages
- Session management and serialization
- Expressions and Objects
- Functions
- Special values
- Vectors, Matrices, Arrays, Factors, Lists, Data Frames
- Indexing,
- Logical comparisons
- Modifying Objects
- Sorting