A Quick Introduction to R

Introduction and examples

Deepayan Sarkar

Indian Satistical Institute, Delhi

January 26, 2010

What is R

R provides an environment in which you can perform statistical analysis and produce graphics. It is actually a complete programming language, although that is only marginally described in this book.

—Peter Dalgaard, **"Introductory Statistics with R"**, 2002

- R can be used as a toolbox for standard statistical techniques.
- Some knowledge of R programming essential to use it well.
- For advanced users, the main appeal of R is as a programming environment suited to data analysis.

More information available at the R Project homepage:
`http://www.r-project.org`

Our goal for the first set of lectures

. . . is basically to get comfortable using R. We will learn

- to do some elementary statistics
- to use the documentation / help system
- about the language basics
- about data manipulation

We will learn about other specialized tools later when they are required.

Plan

- Overview
    - Interacting with R
    - Basic concepts
    - Examples
- Revisit some aspects in more details
    - Important aspects of the language
    - Data manipulation
    - Graphics
- Whatever else seems interesting ...
- Reference: Dalgaard, *“Introductory Statistics with R”*
- More references: `http://www.r-project.org/doc/bib/R-jabref.html`

Interacting with R

R usually 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

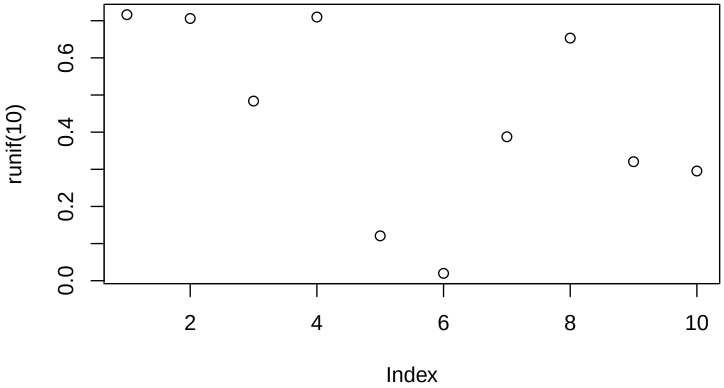
Simple Examples

```
> 2 + 2

[1] 4

> exp(-2) ## exponential function

[1] 0.1353353

> log(100, base = 10)

[1] 2

> runif(10)

 [1] 0.1797446 0.0987207 0.1134377 0.6498423 0.1958337
 [6] 0.6757325 0.3708451 0.7699026 0.6719360 0.4750887
```

- The last command generates 10 $U(0, 1)$ random variables.
- The result (printed) is a vector of 10 numbers.
- `exp()`, `log()`, and `runif()` are *functions*.
- Most useful things in R are done by functions.

Simple Examples: plotting

```
> 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).

```
> x <- 2
> yVar2 = x + 3
> s <- "this is a character string"
> x

[1] 2

> yVar2

[1] 5

> s

[1] "this is a character string"

> x + x

[1] 4
```

Variables

Possible variable names are very flexible. However, note that

- variable names cannot start with a digit
- names are case-sensitive
- some common names are already used by R, e.g.,
  `c`, `q`, `t`, `C`, `D`, `F`, `I`, `T`,
  and should be avoided

Vectorized arithmetic

- The elementary data types in R are all vectors
- 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
```

Vectorized arithmetic

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

> log(bmi)

[1] 2.975113 3.101093 3.041501 3.216102 3.446107 2.9
```

Vectorized arithmetic

- When two vectors are not of equal length, the shorter one is *recycled*. 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
```

Scalars from Vectors

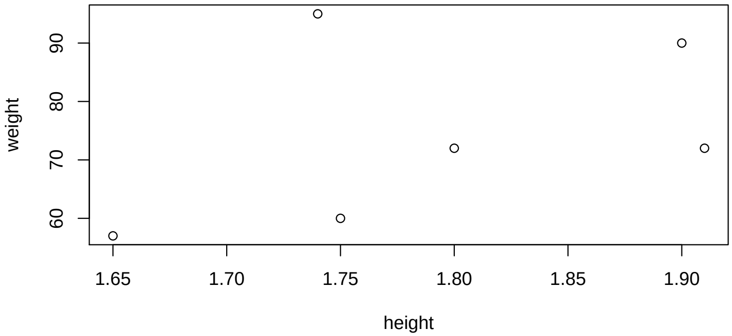
- Many functions summarize a data vector by producing a scalar from a vector. For example

```
> sum(weight)

[1] 446

> length(weight)

[1] 6

> avg.weight <- mean(weight)
> avg.weight

[1] 74.33333
```

Graphics

- The simplest way to produce R graphics output is to use the `plot()` function:

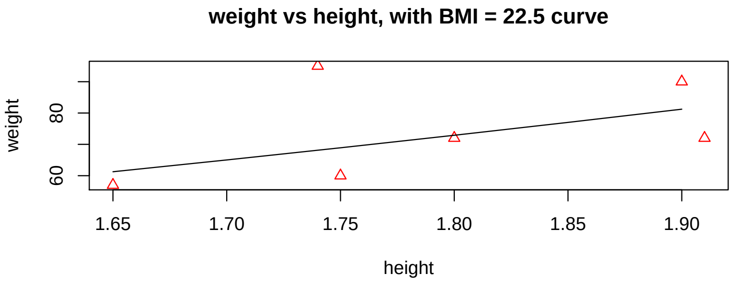
```
> plot(x = height, y = weight)
```



Graphics

- Optional arguments control details of the plot
- Once created, plots can also be enhanced

```
> plot(x = height, y = weight, pch = 2, col = "red")
> hh <- c(1.65, 1.70, 1.75, 1.80, 1.85, 1.90)
> lines(x = hh, y = 22.5 * hh^2)
> title(main = "weight vs height, with BMI = 22.5 cu
```



Descriptive Statistics

Simple summary statistics: *mean, median, s.d., variance*

```
> x <- rnorm(100)
> mean(x)

[1] 0.004956176

> sd(x)

[1] 1.01652

> var(x)

[1] 1.033313

> median(x)

[1] -0.1392815
```

Descriptive Statistics (contd)

Simple summary statistics: *quantiles, inter-quartile range*

```
> xquants <- quantile(x)
> xquants

        0%        25%        50%        75%       100%
-2.0322084 -0.7706084 -0.1392815  0.6314622  2.9148352

> xquants[4] - xquants[2]

     75%
1.402071

> IQR(x)

[1] 1.402071

> quantile(x, probs = c(0.2, 0.4, 0.6, 0.8))

       20%        40%        60%        80%
-0.8330161 -0.3401113  0.1294751  0.9969309
```

The `summary()` function

- When applied to a numeric vector, `summary()` produces a nice summary display:

```
> summary(x)

     Min.   1st Qu.    Median      Mean   3rd Qu.
-2.032000 -0.770600 -0.139300  0.004956  0.631500  2
```

- The output of `summary()` can be different when applied to other objects.

The Iris Dataset

- Let's look at a real dataset: The Iris data is one of many already available in R (type `data()` for a full list).

```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Spec
1          5.1         3.5          1.4         0.2  set
2          4.9         3.0          1.4         0.2  set
3          4.7         3.2          1.3         0.2  set
4          4.6         3.1          1.5         0.2  set
5          5.0         3.6          1.4         0.2  set
6          5.4         3.9          1.7         0.4  set
> 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.
 [14] 4.3 5.8 5.7 5.4 5.1 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.
 [27] 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0 5.5 4.9 4.
 [40] 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.
 [53] 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.
 [66] 6.7 5.6 5.8 6.2 5.6 5.9 6.1 6.3 6.1 6.4 6.6 6.8 6.
 [79] 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5 5.
 [92] 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.
```

The Iris Dataset

```
> 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.
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.
 $ Species     : Factor w/ 3 levels "setosa","versicolor
```

- 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?

The `summary()` function revisited

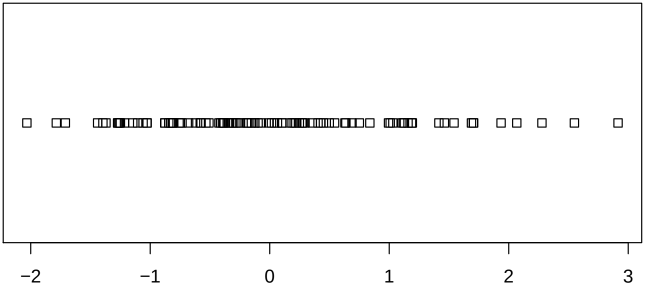
```
> summary(iris)
  Sepal.Length    Sepal.Width     Petal.Length
 Min.   :4.300   Min.   :2.000   Min.   :1.000
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600
 Median :5.800   Median :3.000   Median :4.350
 Mean   :5.843   Mean   :3.057   Mean   :3.758
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100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

- Note the different format of the output.
- `Species` is summarized differently because it is a *categorical variable* (more commonly called *factor* in R).

Graphical display: Strip Plots

- Data analysis should always start with a graphical study
- The simplest plot of numeric data is a *strip plot*

```
> stripchart(x) ## x = rnorm(100)
```



Add-on packages

- Built-in R graphics is not very effective for multivariate data
- Packages
  - R allows the use of add-on packages
  - Usually a collection of new R functions and datasets
  - Can be used to extend the functionality of R.
  - Writing new packages fairly simple.
- General-purpose R packages for visualizing multivariate data
  - lattice
  - ggplot2

Grouped Display

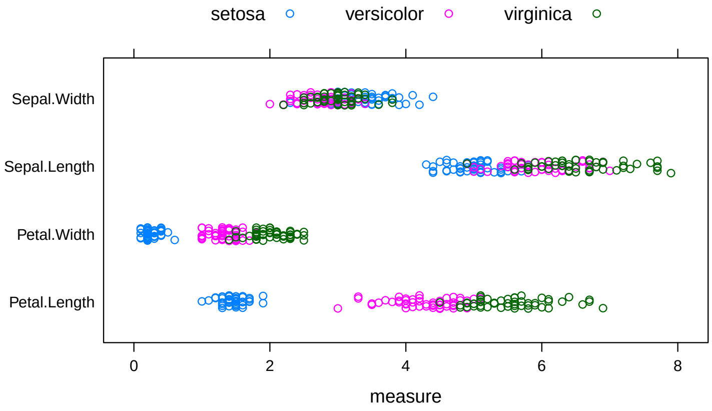
- More informative strip plot using lattice
- Needs data in a slightly different structure

```
> iris2 <-
+     reshape(iris, varying = list(names(iris)[1:4]),
+             v.names = "measure",
+             timevar = "type",
+             times = names(iris)[1:4],
+             direction = "long")
> str(iris2, give.attr = FALSE)
```

```
'data.frame': 600 obs. of  4 variables:
 $ Species: Factor w/ 3 levels "setosa","versicolor",..:
 $ type   : chr  "Sepal.Length" "Sepal.Length" "Sepal.Le
 $ measure: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ id     : int  1 2 3 4 5 6 7 8 9 10 ...
```

Grouped Display

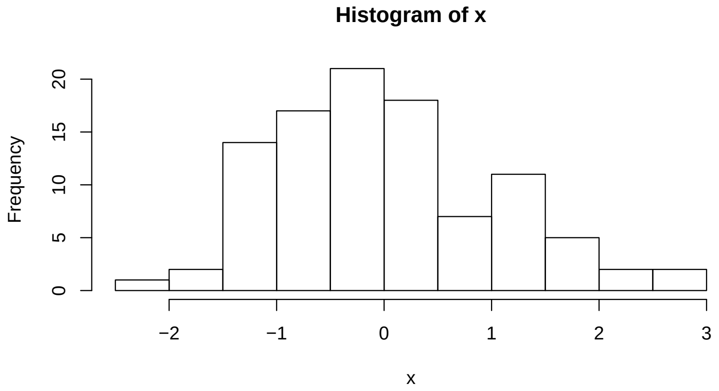
```
> library(package = "lattice")
> stripplot(type ~ measure, iris2, groups = Species,
+           jitter = TRUE, auto.key = list(columns = 3))
```



Histograms

- Strip plots not useful for large data sets.
- Most popular graphical summary for numeric data: *histogram*.

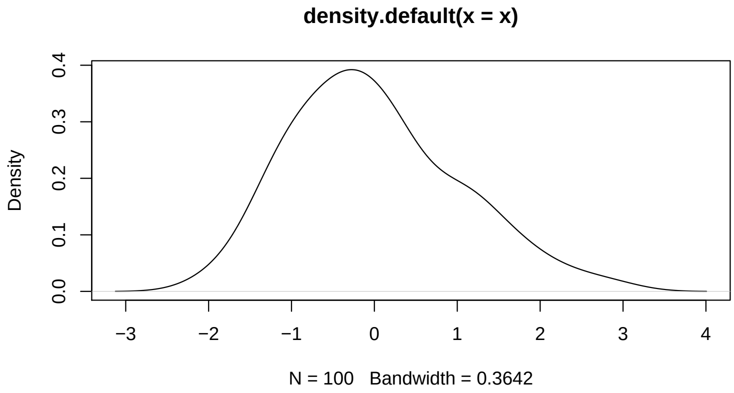
```
> hist(x)
```



Density Plots

- Density plots are generalized histograms.

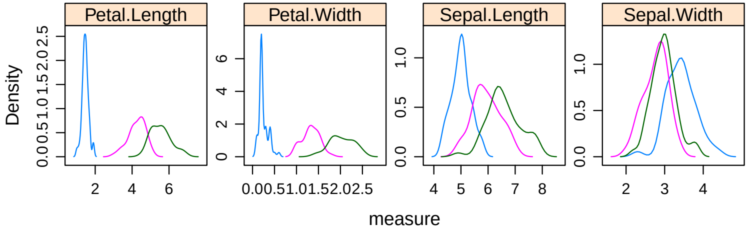
```
> plot(density(x))
```



Grouped Density Plots

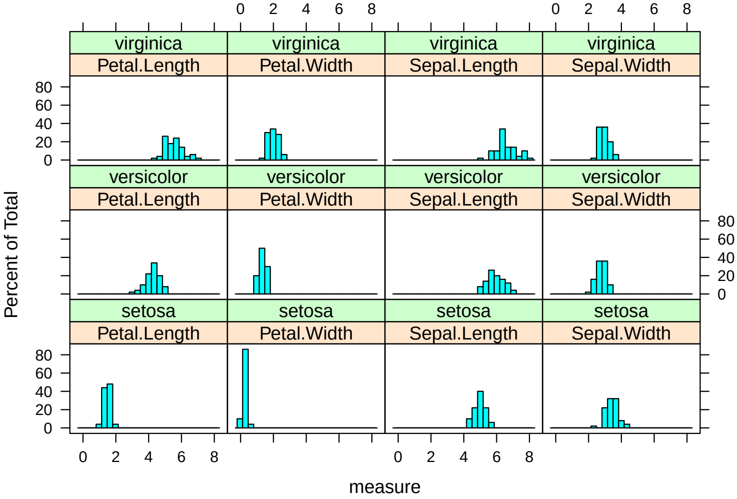
- Again, lattice functions are more suitable for grouped data.

```
> densityplot(~ measure | type, data = iris2,
+             groups = Species, scales = "free",
+             plot.points = FALSE)
```



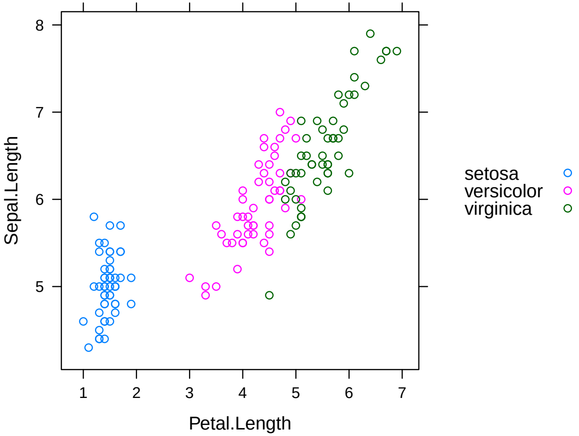
Grouped Histogram

```
> histogram(~measure | type + Species, iris2, nint = 25)
```



Grouped Scatter Plot

```
> xyplot(Sepal.Length ~ Petal.Length, data = iris,
+         groups = Species, aspect = 1,
+         auto.key = list(space = "right"))
```



Categorical Data

We have already seen one example:

```
> summary(iris$Species)

    setosa versicolor  virginica
        50         50         50
```

Let us try to predict the Species using other measurements.

- What's the best measure to use?
- What are good thresholds?

Discretizing

A continuous measure can be converted into a factor using the `cut()` function:

```
> iris$PL.disc <- cut(iris$Petal.Length,
+                     breaks = c(0, 2.5, 4.75, 7))
> iris$SL.disc <- cut(iris$Sepal.Length,
+                     breaks = c(0, 5.5, 6.25, 8))
> str(iris)
```

```
'data.frame': 150 obs. of  7 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.
 $ Species     : Factor w/ 3 levels "setosa","versicolor
 $ PL.disc     : Factor w/ 3 levels "(0,2.5]","(2.5,4.75
 $ SL.disc     : Factor w/ 3 levels "(0,5.5]","(5.5,6.25
```

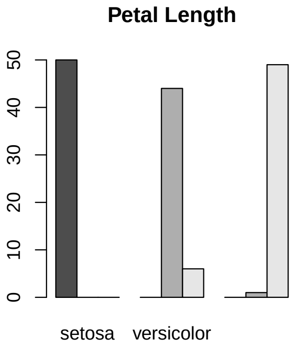
Tables

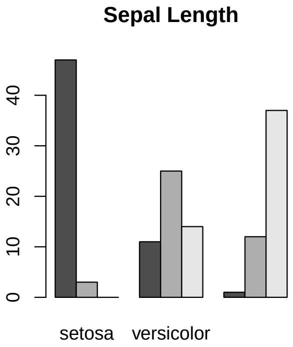
Association between categorical variables summarized by *tables*.

```
> PL.tab <- xtabs(~ PL.disc + Species, iris)
> SL.tab <- with(iris, table(SL.disc, Species))
> PL.tab

            Species
PL.disc       setosa versicolor virginica
  (0,2.5]         50          0         0
  (2.5,4.75]       0         44         1
  (4.75,7]         0          6        49

> SL.tab

            Species
SL.disc       setosa versicolor virginica
  (0,5.5]         47         11         1
  (5.5,6.25]       3         25        12
  (6.25,8]         0         14        37
```

Graphical Display of Tables: Bar chart

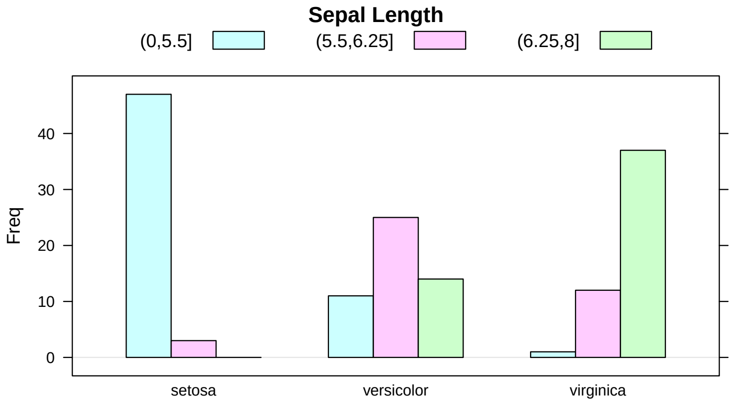
```
> par(mfrow = c(1,2))
> barplot(PL.tab, beside = TRUE, main = "Petal Length")
> barplot(SL.tab, beside = TRUE, main = "Sepal Length")
```





Graphical Display of Tables: Bar chart

```
> barchart(t(SL.tab), horizontal = FALSE, stack = FALSE,
+           auto.key = list(columns = 3),
+           main = "Sepal Length")
```



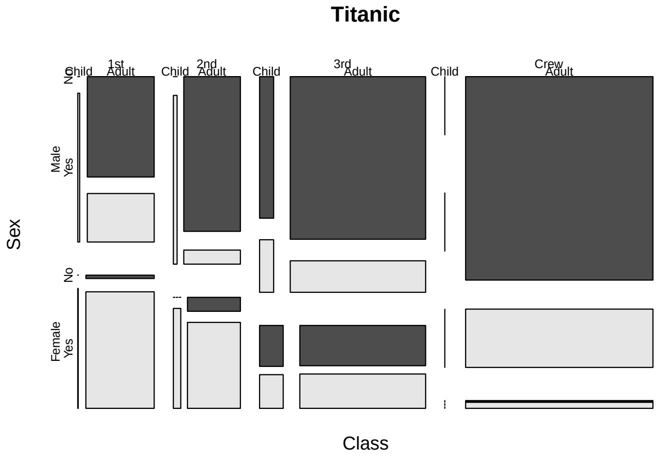
Higher Dimensional Tables

The built-in `Titanic` data set is a cross-tabulation of 4 characteristics of 2201 passengers on the Titanic

```
> dimnames(Titanic)

$Class
[1] "1st"  "2nd"  "3rd"  "Crew"

$Sex
[1] "Male"   "Female"

$Age
[1] "Child" "Adult"

$Survived
[1] "No"  "Yes"
```

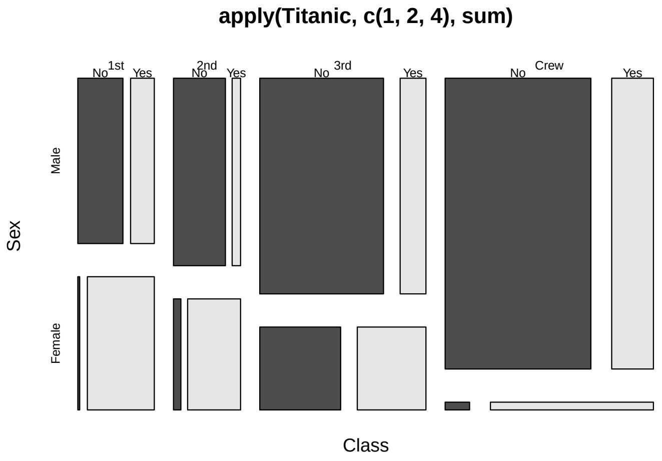
Titanic Survivors

```
> mosaicplot(Titanic, color = TRUE)
```



Titanic Survivors (simplified)

```
> mosaicplot(apply(Titanic, c(1, 2, 4), sum),
+                 color = TRUE)
```



Getting help

R has too many tools for anyone to remember them all, so it is very important to know how to find relevant information using the help system.

- `help.start()`
  Starts a browser window with an HTML help interface. One of the best ways to get started. Has links to a very detailed manual for beginners called ‘An Introduction to R’, as well as topic-wise listings.
- `help(topic)`
  Displays the help page for a particular topic or function. Every R function has a help page.
- `help.search("search string")`
  Subject/keyword search

Getting help (contd)

- The `help()` function provides topic-wise help. When you know which function you are interested in, this is usually the best way to learn how to use it. There's also a short-cut for this; use a question mark (`?`) followed by the topic. The following are equivalent:

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

- When you want to know about a specific subject, but don't know which particular help page has the information, the `help.search()` function (shortcut: `??`) is very useful. For example, try

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

Getting help (contd)

The help pages can be opened in a browser as well:

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

The help pages are usually detailed (but terse). Among other things, they often contain

- A ‘See Also’ section that lists related help pages
- A Description of what the function returns
- An ‘Examples’ section, with actual code illutrating how to use the documented functions. These examples can actually be run directly using the `example` function. e.g., try

  ```
  > example(plot)
  ```

apropos()

Another useful tool is the `apropos()` function:

```
> apropos("plot")
 [1] "assocplot"
 [2] "barplot"
 [3] "barplot.default"
 [4] "biplot"
 [5] "boxplot"
 [6] "boxplot.default"
 [7] "boxplot.matrix"
 [8] "boxplot.stats"
 [9] "bwplot"
[10] "cdplot"
[11] "contourplot"
[12] "coplot"
[13] ".__C__recordedplot"
[14] "densityplot"
[15] "dotplot"
[16] "fourfoldplot"
[17] "interaction.plot"
```

R packages

R makes use of a system of *packages*

- Each package is a collection of routines with a common theme
- The core of R itself is a package called base
- A collection of packages is called a *library*
- Some packages are already loaded when R starts up. Other packages need be loaded using the `library()` function

R packages

Several packages come pre-installed with R.

```
> ip <- installed.packages()
> rownames(ip)[ip[, "Priority"] %in%
+               c("base", "recommended")]
```

```
 [1] "base"        "boot"        "class"       "cluster"
 [5] "codetools"   "datasets"    "foreign"     "graphics"
 [9] "grDevices"   "grid"        "KernSmooth"  "lattice"
[13] "MASS"        "Matrix"      "methods"     "mgcv"
[17] "nlme"        "nnet"        "rpart"       "spatial"
[21] "splines"     "stats"       "stats4"      "survival"
[25] "tcltk"       "tools"       "utils"
```

R packages

- There are also many (more than 300) other packages contributed by various users of R available online, from the Comprehensive R Archive Network (*CRAN*):
  `http://cran.fhcrc.org/web/packages/`
- The *Bioconductor* project provides an extensive collection of R packages specifically for bioinformatics
  `http://www.bioconductor.org/packages/release/Software.html`

R packages

- It is fairly easy for anyone to write new R packages. This is one of the attractions of R over other statistical software.
- Some packages are already loaded when R starts up. At any point, The list of currently loaded packages can be listed by the `search()` function:

```
> search()

 [1] ".GlobalEnv"          "package:lattice"
 [3] "package:tools"       "package:stats"
 [5] "package:graphics"    "package:grDevices"
 [7] "package:utils"       "package:datasets"
 [9] "package:methods"     "Autoloads"
[11] "package:base"
```

R packages

- Other packages can be loaded by the user.
- For example, the ISwR package contains datasets used in Dalgaard, “Introductory Statistics with R”.
- This can be loaded by:

```
> library(ISwR)
```

- New packages can be downloaded and installed using the `install.packages()` function.
- For example, to install the ISwR package (if it’s not already installed), one can use

```
> install.packages("ISwR")
> library(help = ISwR)
```

- The last call gives a list of all help pages in the package.