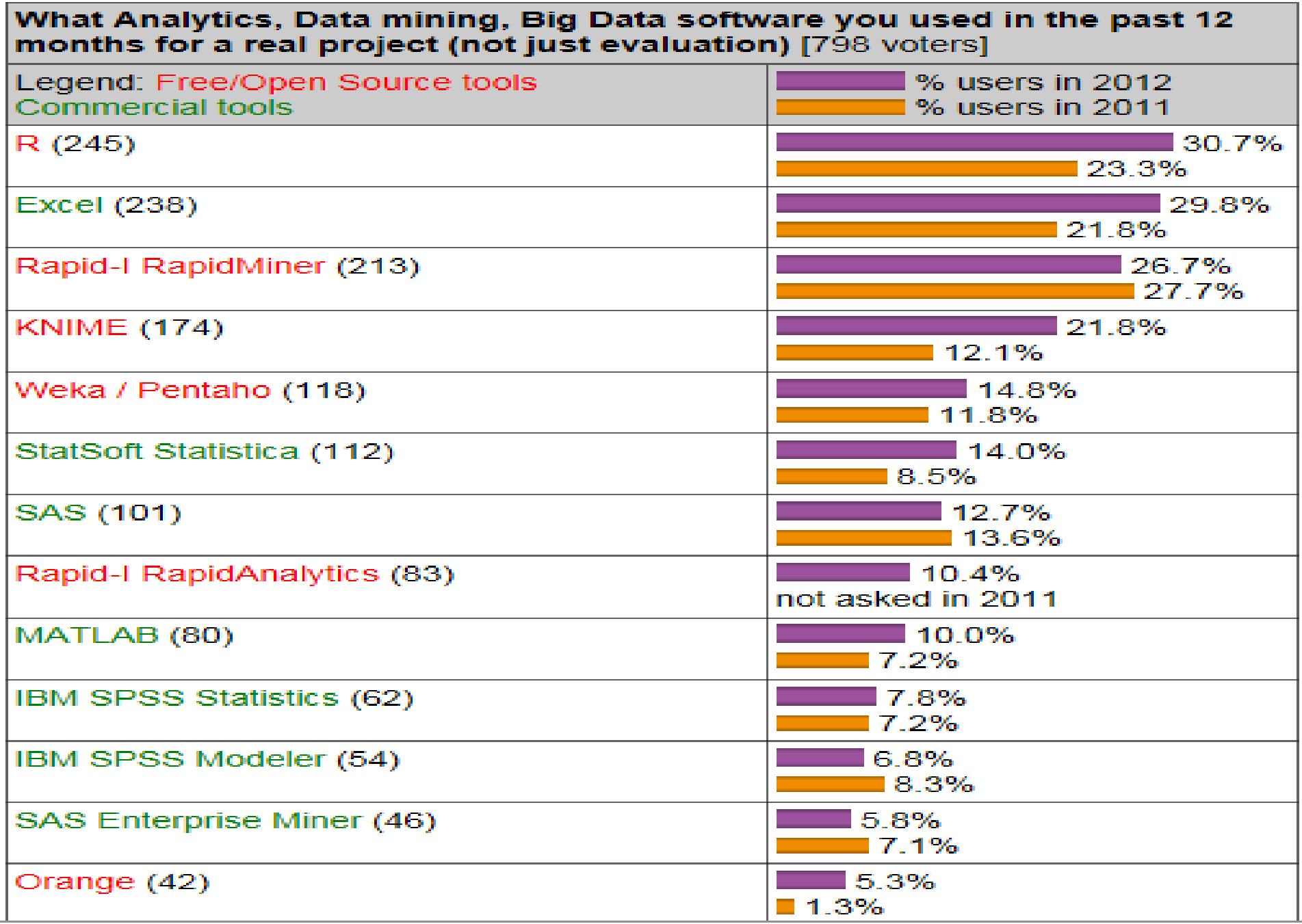
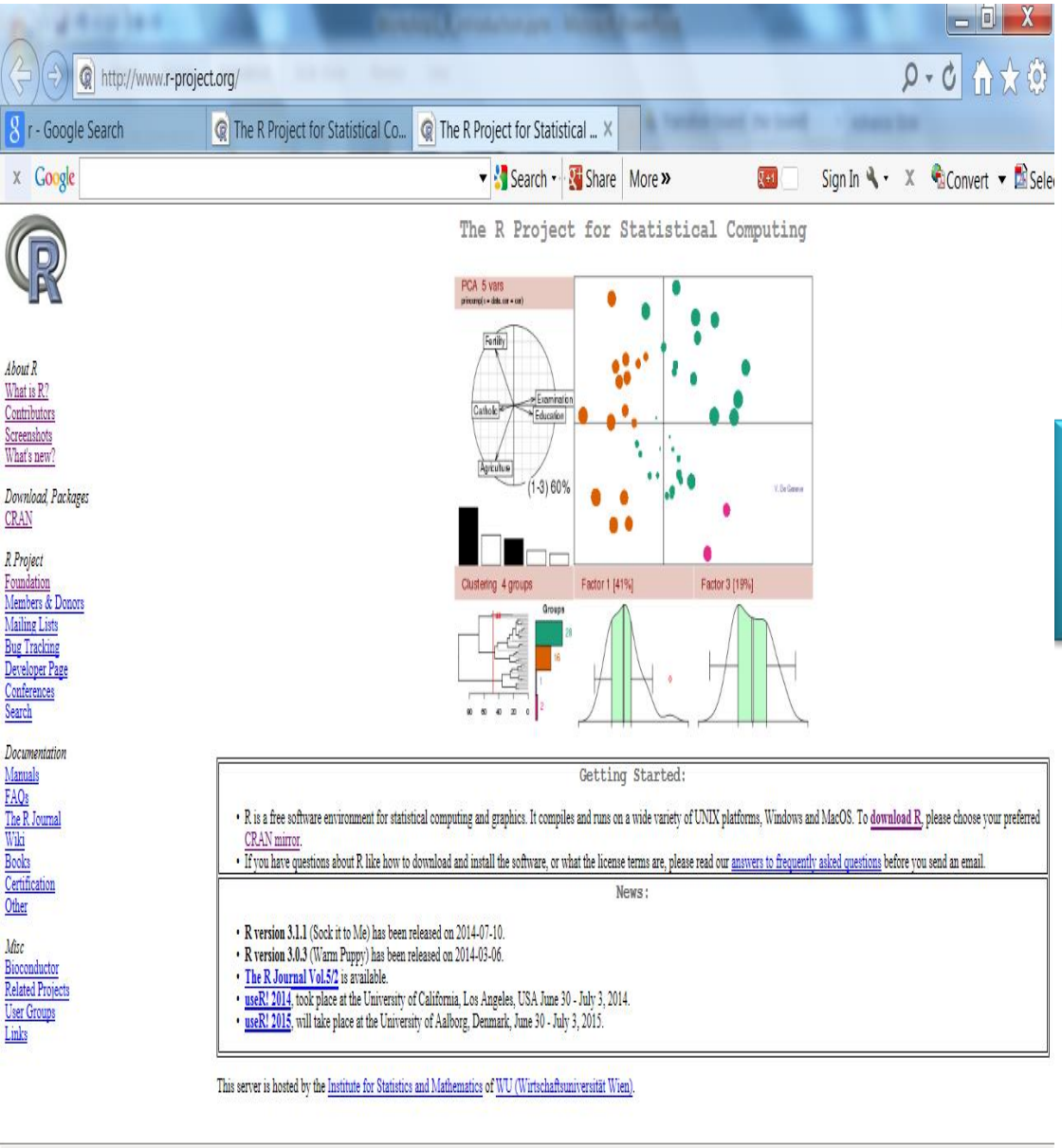
# R Introduction

Li Xiaoli

http://www.kdnuggets.com/polls/2012/analytics-data-mining-big-data-software.html



# What is R and why do we use it?




- Open source, most widely used for statistical analysis and graphics
- Extensible via dynamically loadable add-on packages
- > 5,887 packages on CRAN

http://cran.r-project.org/web/packages/

What are the packages on Network Analysis?

**Relatively easy to use**

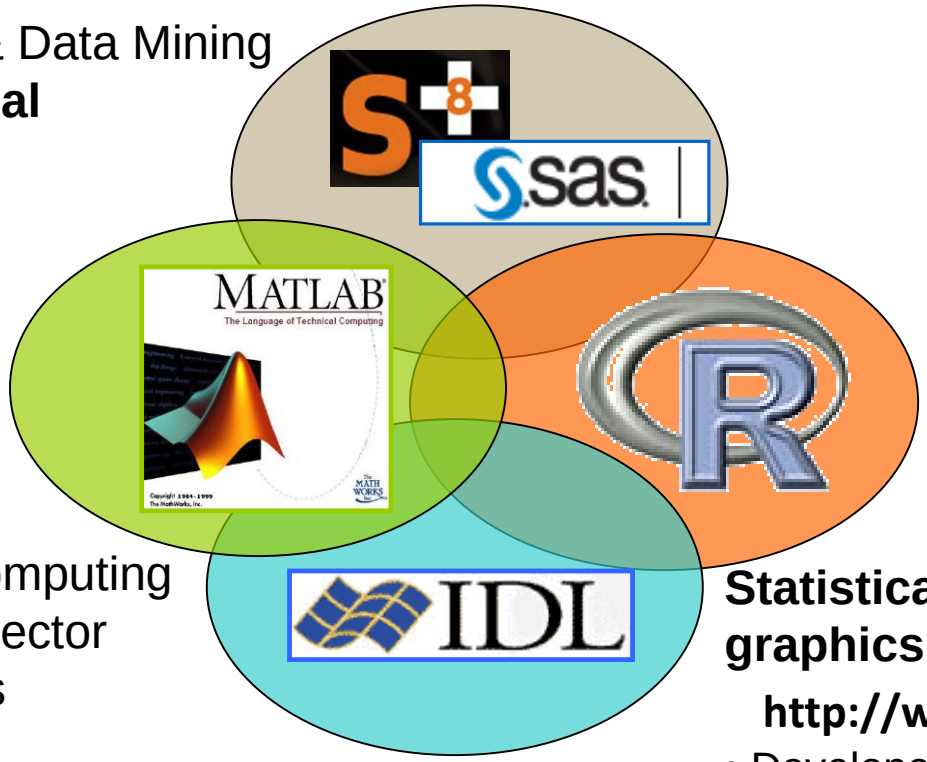
```
> v = rnorm(256)
> A = as.matrix (v,16,16)
> summary(A)
> library (fields)
> image.plot (A)
```

# Why R?

- **Open source** – It's free!
- **Statistical functions** – R is designed for statistical computing
- **Data Mining** – R is also widely used by data scientists
- **Econometrics**
- **Genetics**
- **…..**
- **Excellent graphing engine** – plot nice graphs with built-in functions and external packages like ggplot2 and heatmap.2
- **Easy to Use** – do more with less so you can spend more time thinking about the problem you are trying to solve

# Why R?

- Statistics & Data Mining
- **Commercial**




- Technical computing
- Matrix and vector formulations

- Data Visualization and analysis platform
- Image processing, vector computing

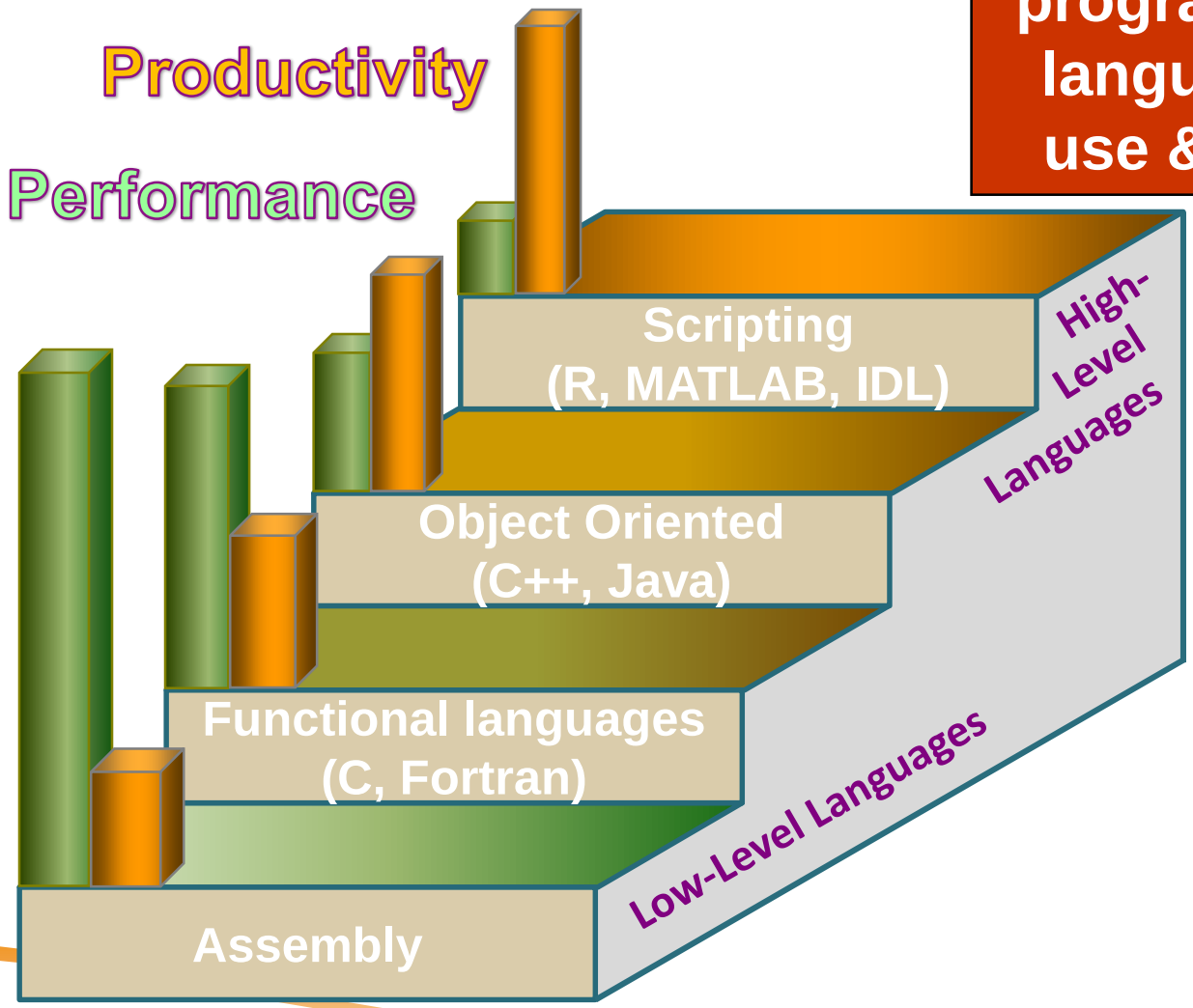
**Statistical computing and graphics**

**http://www.r-project.org**

- Developed by **R**. Gentleman & **R**. Ihaka
- Expanded by community as **open source**
- Statistically rich

# The Programmer's Dilemma

**What programming language to use & why?**



# Features of R

**R is an integrated suite of software for data manipulation, calculation, and graphical display**

- Effective data handling
- Various operators for calculations on arrays/matrices
- Graphical facilities for data analysis
- Well-developed language including conditionals, loops, recursive functions and I/O capabilities.

# R Installation

**Download R from**

http://www.r-project.org

**Rstudio R IDE software at**

http://www.rstudio.com

RStudio IDE is a powerful and productive user interface for R.

Video: Simple Introduction of Rstudio IDE

http://www.rstudio.com/products/RStudio/

# Getting help

- **From R GUI**
  - *help(function_name)*
    - *help(prcomp)*
  - *? function_name*
    - *?prcomp*
  - *help.search("topic")*
    - *??topic*
- **Search CRAN**
  - http://www.r-project.org
- **CRAN Task Views (for individual packages)**
  - http://cran.cnr.berkeley.edu/web/views/

# R Basics: Basic functions

```
# list installed packages
library()

# install a package
install.packages("ggplot2")

# load a library
library(ggplot2)

# search help files (free text search from all the help in the menu)
??mean

# help for a function (if you know what function  you are looking for)
?mean
```

# Variables and assignment

- Use variables to store values
- Three ways to assign variables
  - a = 6
  - a <- 6
  - 6 -> a
- Update variables by using the current value in an assignment
  - x = x + 1

# Basic usage: arithmetic in R

- You can use R as a calculator
- Typed expressions will be evaluated and printed out
  - Main operations: +, -, *, /, ^
  - Obeys order of operations
  - Use parentheses to group expressions
- More complex operations appear as *functions*
  - sqrt(2)
  - sin(pi/4), cos(pi/4), tan(pi/4), asin(1), acos(1), atan(1)
  - exp(1), log(2), log10(10)

# Vectors and vector operations

To create a vector:

```
# c() command to create vector x
 x=c(12,32,54,33,21,65)
# c() to add elements to vector x
 x=c(x,55,32)
```

```
# seq() command to create
sequence of number
 years=seq(1990,2003)
# to contain in steps of .5
 a=seq(3,5,.5)
# can use : to step by 1
years=1990:2003;
```

```
 # rep() command to create data
that follow a regular pattern
 b=rep(1,5)
 c=rep(1:2,4)
```

To access vector elements:

```
# 2nd element of x
 x[2]
# first five elements of x
 x[1:5]
# all but the 3rd element of x
 x[-3]
 # values of x that are < 40
 x[x<40]
# values of y such that x is < 40
 y[x<40]
```

To perform operations:

```
 # mathematical operations on vectors
y=c(3,2,4,3,7,6,1,1)
 x+y; 2*y;  x*y;  x/y;  y^2
```

# R Basics : Variable / Object assignment

```
# variable assignment
a = 1; b = 2; c = 3
# arrays (have the same type)
a = c(1,2,3)
```

c(...)This produces a vector of whatever is passed as an argument to c().

```
b = c("x", "y", "z")

c = 1:5

seq(1,2,by=.2) #This produces a sequence of numbers by a given increment

rep(1:5,3) #This produces a vector of repetitions of x by a given number of times.

a <- c(1,2,5.3,6,-2,4)  # numeric vector
b <- c("one","two","three")  # character vector
c <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE)  #logical vector
```

# Matrices & matrix operations

To create a matrix:

```
# matrix() command to create matrix A with rows and cols
A=matrix(c(54,49,49,41,26,43,49,50,58,71),nrow=5,ncol=2)
B=matrix(1,nrow=4,ncol=4)
```

To access matrix elements:

```
# matrix_name[row_no, col_no]
 A[2,1]   # 2nd row, 1st column element
 A[3,]   # 3rd row
 A[,2]     # 2nd column of the matrix
 A[2:4,c(2,1)]   # submatrix of 2nd-4th
elements of the 3rd and 1st columns
 A["KC",] # access row by name, "KC"
```

Statistical operations:

```
 rowSums(A)
 colSums(A)
 rowMeans(A)
 colMeans(A)
# max of each columns
 apply(A,2,max)
# min of each row
 apply(A,1,min)
```

Element by element ops:

```
2*A+3; A+B; A*B; A/B;
```

Matrix/vector multiplication:

```
A %*% B;
```

# Useful functions for vectors and matrices

- Find # of elements or dimensions
  - *length(v), length(A), dim(A)*
- Transpose
  - *t(v), t(A)*
- Matrix inverse
  - *solve(A)*
- Sort vector values
  - *sort(v)*
- Statistics
  - *min()*, *max()*, *mean()*, *median()*, *sum(), sd()*, *quantile()*
  - Treat matrices as a single vector (same with *sort()*)

# R Basics: data frames

```
codes = data.frame(id.x=1:4, code=c("B","B","A","D"))

colors = data.frame(id.y=1:4,
color=c("red","red",NA,"white"))

d = merge( codes, colors, by.x="id.x", by.y="id.y")
```

You can have multiple columns, and every column could have different data types, which is very useful in real life.

# R Basics: Assessing data in dataframes

```
# dataframes by column names
d$code
d$color
d[c("code", "color")]
```

# R Basics: Object properties

str(d) # structure of the data

summary(d)

is.na(d): which cell/location in your variable has missing value

length(c): how long is your variable (if it is a matrix, it gives you nothing)

dim(d): row by column

nrow(d): # of rows

ncol(d): # of columns

# R Basics: Simple statistics

```
mean(1:9)
max(4:8)
range(c)
sum(c)
quantile(c,0.95)
rank(c)
var(c)
sd(c)
cor(c,rnorm(5))
table(d$color)
table(d$code,d$color)
```

# R Basics: Simulation and Sampling Data

**# Simulate data**

rnorm(1000): randomly generate data with normal ditribution

rpois(10,4): simulate the poisson distribution

**# Sampling data**

sample(1:10): generate the random 10 data

sample(1:10, size = 5): sample 5 from my data

sample(1:10, size = 15, replace = TRUE)

set.seed(26011973)

sample(1:10, size = 15, replace = TRUE)

# R Basics: Defining and calling functions

```
power<-function(x,y) x**y  #power is the function name
power(2,3)

graphnormal<-function(n=1000) {
  d = rnorm(n)
  histogram(d)
}
graphnormal(10000)
```

# R Basics: Memory Management

```
# save object to file
save(d,file="d.RData")

# memory management (house keeping things)
objects()
ls()
rm(list=ls())

# load object from file
load("d.RData")
```

# Graphical display and plotting

- Most common  plotting function is *plot()*
  - **plot(*x*,*y*)** plots *y* **vs** *x*
  - ***plot*(*x*)** plots *x* **vs** 1: *length*(*x*)
- *plot*() has many options for labels, colors, symbol, size, etc.
  - Check help with *?plot*
- Use *points*(), *lines*(), or *text*() to add to an existing plot
- Use *x11*() to start a new output window
- Save plots with *png()*, *jpeg()*, *tiff()*, or *bmp()*

# R Basics: Graphical display and plotting

```
x=c(1,2,3, 4, 5)

y=c(1,4,9, 16, 25)

plot(x,y)

x11()

plot(x,y)

plot(x,y, main="My first Figure Using R",  sub="what is
  subtitle?")

plot(x,y, main="My first Figure Using R", col="red",
  sub="what is subtitle?", xlab="X-axis label", ylab="y-axix
  label", xlim=c(0, 6), ylim=c(0, 26))
```

# Reading data from files

- Large data sets are better loaded through the file input interface in R
- Reading a table of data can be done using the *read.table()* command:
  - *a <- read.table("a.txt")*
- The values are read into R as an object of type data frame. Various options can specify reading or discarding of headers and other metadata.
- A more primitive but universal file-reading function exists, called *scan()*
  - b = scan("input.dat");
  - *scan()* returns a vector of the data read

# Programming in R

- The following slides assume a basic understanding of programming concepts
- For more information, please see chapters 9 and 10 of the R manual:

http://cran.r-project.org/doc/manuals/R-intro.html

Additional resources

- *Beginning R: An Introduction to Statistical Programming* by Larry Pace
- Introduction to R webpage on APSnet:

http://www.apsnet.org/edcenter/advanced/topics/ecologyandepidemiologyinr/introductiontor/Pages/default.aspx

- The R Inferno:

http://www.burns-stat.com/pages/Tutor/R_inferno.pdf

# Conditional statements

- Perform different commands in different situations
- ***if*** *(condition) command_if_true*
  - Can add *else command_if_false* to end
  - Group multiple commands together with braces {}
  - ***if (cond1) {cmd1; cmd2;} else if (cond2) {cmd3; cmd4;}***
- Conditions use relational operators
  - ==, !=, <, >, <=, >=
  - Do not confuse = (assignment) with == (equality)
    - = is a command, == is a question
- Combine conditions with *and* (&&) and *or* (||)
  - Use & and | for vectors of length > 1 (element-wise)

# Loops

- Most common type of loop is the *for* loop
  - *for (x in v) { loop_commands; }*
  - *v* is a vector, commands repeat for each value in *v*
  - Variable *x* becomes each value in *v*, in order
  - **Example:** adding the numbers 1-10
    - *total = 0; for (x in 1:10) total = total + x;*
- Other type of loop is the *while* loop
  - *while (condition) { loop_commands; }*
  - Condition is identical to *if* statement
  - Commands are repeated until condition is false
    - Might execute commands 0 times if already false
- *while* loops are useful when you don't know number of iterations

# Lists

- Objects containing an ordered collection of objects
- Components do not have to be of same type
- Use *list()* to create a list:
  - *a <- list("hello",c(4,2,1),"class");*
- Components can be named:
  - *a <- list(string1="hello",num=c(4,2,1),string2="class")*
- Use [[position]] to access list elements
  - E.g., a[[2]]
- Running the *length()* command on a list gives the number of higher-level objects

# Writing your own functions

- Writing functions in R is defined by an assignment like:
  - *a <- function(arg1,arg2) { function_commands; }*
- Functions are R objects of type “function”
- Functions can be written in C/FORTRAN and called via *.C()* or *.Fortran()*
- Arguments may have default values
  - Example: *my.pow <- function(base, pow = 2) {return base^pow;}*
  - Arguments with default values become optional, should usually appear at end of argument list (though not required)
- Arguments are untyped
  - Allows multipurpose functions that depend on argument type
  - Use *class()*, *is.numeric()*, *is.matrix()*, etc. to determine type

# Useful R links

- R Home: http://www.r-project.org/
- R's CRAN package distribution: http://cran.cnr.berkeley.edu/
- Introduction to R manual: http://cran.cnr.berkeley.edu/doc/manuals/R-intro.pdf
- Writing R extensions: http://cran.cnr.berkeley.edu/doc/manuals/R-exts.pdf
- Other R documentation: http://cran.cnr.berkeley.edu/manuals.html

# References

Nagiza F. Samatova, William Hendrix, John Jenkins, Kanchana Padmanabhan, Arpan Chakraborty, Practical Graph Mining With R