**The str function**

* Str: compactly display the internal structure of an R object
  + - A diagnostic function and an alternative to “summary”
    - It is especially well suited to compactly display the (abbr) contents of (possibly nested) list
    - Roughly 1 line per object

Str(str)

Str(lm)

X <- rnorm(100, 2, 4)

Summary(x)

Str(x)

F <- gl(40, 10)

Str(f)

Summary(f)

Library(datasets)

Head(airquality)

M <- matrix(rnorm(100), 10, 10)

Str(m)

S <- split(airquality, airquality$Month)

Str(s)

**Simulation – Generating Random Numbers**

* Functions for probability distributions in R
  + - Rnorm
      * Generate random normal variates with a given mean and standard deviation
    - Dnorm
      * Evaluate the normal probability density at a point
    - Pnorm
      * Evaluate the cumulative distribution function for a normal distribution
    - Rpois
      * Generate random Poisson variates with a given rate
* Actually
  + - D for density
    - R for random number generation
    - P for cumulative distribution
    - Q for quantile function
* Working with normal distributions requires using these four functions

Dnorm(x, mean = 0, sd = 1, log = FALSE)

Pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)

Qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)

Rnorm(n, mean = 0, sd = 1)

* Example

X <- rnorm(10)

X <- rnorm(10, 20, 2)

* To regenerate the same random number TWICE:

Set.seed(1)

Rnorm(5)

* Generating Poisson data

Rpois(10, 1)

Rpois(10, 2)

Rpois(10, 20)

Ppois(2, 2) #cumulative distribution

Ppois(4, 2)

Ppois(6, 2)

**Simulation – Simulating a Linear Model**

Set.seed(20)

X <- rnorm(100)

E <- rnorm(100, 0, 2)

Y <- 0.5 + 2 \* x + e

Summary(y)

Plot(x, y)

* **What if x is a binary?**

Set.seed(10)

X <- rbinorm(100, 1, 0.5)

E <- rnorm(100, 0, 2)

Y <- 0.5 + 2 \* x + e

Summary(y)

Plot(x, y)

* Simulate a Poisson model?

Set.seed(1)

X <- rnorm(100)

X <- rpois(100, exp(log.mu))

Summary(y)

Plot(x, y)

**Simulation – random sampling**

* Sample function draws randomly from a specified set of (scalar) objects allowing you to sample from arbitrary distributions

Set.seed(1)

Samle(1:10, 4)

Sample(letters, 5)

Sample(1:10) #permutation

Sample(1:10, replace = TRUE) #sample with replacement

**Summary – simulation**

* Drawing samples from specific probability distributions can be done with x functions
* Standard distributions are built-in: normal, Poisson, binomial, exponential, gamma, etc.
* The sample function can be used to draw random samples from arbitrary vectors
* Setting the random number generator seed by Set.seed is critical, for REPRODUCIBILITY

**R profiler**

**Why is my code too slow?**

* Profiling is a sustematic wat to examine how much time is spent in different parts of a program
* Useful while trying to optimize your code
* Often code runs fine once, but what if you have to put in a loop of 1,000 iterations?
* Profiling is better than guessing

**On optimizing your code**

* Getting biggest impact on speeding up code depends on knowing where the code spends most of its time
* This cannot be done without profiling

**Principles of optimization**

* Design first, optimize later
* Remember never optimize first
* Measure data, don’t guess
* Remember this ^

**Using system.time()**

* Takes an arbitrary R expression as input and returns the amount of time taken to evaluate the expression
* Computes the time in seconds
  + - If there’s an error, gives time until the error occurred
* Returns an object of class proc\_time
  + - User time: time charged to the CPU for the expression
    - Elapsed time: normal time
* Usually user time = elapsed time

System.time( {

N <- 1000

R <- numeric(n)

For (I in 1:n) {

X <- rnorm(n)

R[i] <- mean(x)

}

})

* Beyond system.time()
  + - Tests certain functions, see whether they are taking too much time
    - Assume that you know where the problem is already

**Using Rprof()**

* Rprof() starts the profiler in R
* summaryRprof() summarizes the output from Rprof()
* DO NOT use system.time() and Rprof() together
* Rprof() keeps track of the function call stack at regularly sampled intervals and tabulates how much time is spend in each function
* Default sampling interval is 0.02 seconds

**Using summaryRprof()**

* Tabulates the R profiler output and calculates how much time is spend in which function
* There are two methods for normalizing the data
  + - “by.total” divides the time spend in each function by the total runtime
    - “by.self” does the same but first subtracts out time spent in functions above in the call stack

**Normalizing by “by.total” and “by.self”**

**Summary**

* Rprof() runs the profiler for performance of analysis of R code
* summaryRprof() summarizes the output of Rprof() and gives percent of time spent in each function
  + - 2 types of normalization
* Good to break your code into functions so that the profiler can give useful information about where time is being spent