**!** symbol means not

!is.na(…) #display the values only

**!=** not equal

**%/%** quotient without remainder

**%in%** tests whether elements of one vector can be found in another vector.

• x %in% y

**%%** remainder

**$** Data frame columns can be addressed using their names

trees$Height[trees$Volume < 17.5]

**aggregate()** calculate statistics for all values within different groups

aggregate(measurements ~ myfactor, data = mydata, FUN = function)

**all.equal()** tests approximate equality

**attach()** # to access variables directly

**c()** create a vector

**chron()** eg. chron(dates = x, format = c('y-m-d'))

**cut()** divides the range of x into intervals and codes the values in x according to which interval they fall.

**data.frame()** to construct data frames from vectors that already exist.

**dim()** gives both the number of rows and columns

**diff()** differences of the value in …

**dump("usefuldata", "useful.R")** Save usefuldata object for a future session.

This stores the command necessary to create the vector usefuldata into the file useful.R

**dump(c("nhtemp", "nhtempC"), "nhtemp.R")** To save existing objects nhtemp and nhtempC to a file called nhtemp.R on hard drive.

**dump(list = objects(), "all.R")** To save all of the objects that you have created during a session

**factor()**

levels() = "x" "y" …  
levels(grp)[1] <- "z" # change x to z

**format()** convert back for printing the date

**head()** see first few rows

**help()** obtain information about a built-in data frame

**install.packages("")**

**IQR()** difference between 1st and 3rd quartile of …

**is.na()** detects missing values

**ISOdate() & ISOdatetime()** functions are used when numerical values for the year, ... are known

**length()** the number of elements in …

**library()** to load a package.

**load()** to load the workspace in another directory

**ls()** prints a list of all objects in the current workspace

**max()** maximum value of …

**mean()** find the mean value eg.: mean(trees$Volume)

**median()** middle value of a data set

**min()** minimum value of …

**names()** see the names in a data frame:  
• eg. names(trees)

**ncol()** number of columns in a data frame

**nrow()** number of rows in a data frame

**objects()** prints a list of all objects in the current workspace

**options()** Control the number of digits in the output with the options() function.  
• options(digits=3)

**paste()**  
• paste(colors, "flowers")  
• paste("several", colors, "s", sep = "")  
• paste("I like", colors, collapse = ", "): allows components of the vector to be collapsed.

**paste0()** is the shorthand way to set sep = ""

**pmax()** pairwise maximum eg. pmax(x, y) pairwise max of x and y

**pmin()** pairwise minimum eg. pmin(x, y) pairwise min of x and y

**predict()**  
• eg. predict(litters.lm, newdata = data.frame(bodywt = 8.5, lsize = 6))

**q()** exit R.  
• q has three arguments: save, status, and runLast.  
• q("no") or q(save = "no") # quit without saving  
• to call the last parameter: q(runLast = FALSE)  
• OR q( , , FALSE)

**range()** difference between max and min of…how spread out the distribution of the measurements.

**read.table()** will read the data:  
• read.table("file.txt", header = TRUE)  
• if numbers are separated by commas:  
read.table("file.txt", header = TRUE, sep=",")

**rep()** repeated patterns:  
• rep(3, 12)  
• rep(seq(2, 20, by = 2), 2) = 2 4 6 8 10 … 2 4 6 8 10  
• rep(c(1, 4), c(3, 2)) = 1 1 1 4 4  
• rep(c(1, 4), each = 3) = 1 1 1 4 4 4  
• rep(1:10, rep(2, 10)) = 1 1 2 2 3 3 … 10 10

**rivers()** Rivers  
The rivers object contains measurements in miles.  
• To convert to kilometers: riversKm <- rivers \* 1.609  
• To convert to feet: riversFeet <- rivers \* 5280  
• Rivers[rivers > 2000]

**runif()** provides a model for the error in measurement, can stimulate values from this distribution.

**sample()** stimulate things like the results of the repeated tossing of a 6-sided dice.  
• diceTosses <- sample(1:6, size = 8, replace = TRUE)

**save.image("temp.RData")** save workspace image information to a file called temp.RData

**search()** see which packages are loaded.

**seq()** patterned vectors:  
• seq(1, 21, by = 2)

**setwd()** Set as Working Directory:  
• setwd("c:/mydata")

**sd()** standard deviation

**source("useful.R")** To retrieve the vector in a future session

**sort()** arranges the elements of … in ascending order

**stats::median()** force a function to be chosen from a particular package.

**str()** summary info about the structure

**strptime()** convert from strings to an internal numerical representation

**subset()** If you only want to see certain …  
• eg. chickHorsebean <- subset(chickwts, feed == "horsebean")

**substr()** to take substrings:  
• substr(x, start, stop)  
• x is a vector character, start will say which character to start, same for stop.

**substring()**

**summary()** provides information about the main features of a data frame

**table()** to display the numbers of …:  
• table(coinTosses)

**tail()** see last few rows

**ts()** create time series objects with the ts() function  
• Eg. JobsBC <- ts(jobs$BC, start = c(1995, 1), end = c(1996, 12), frequency = 12)  
• plot(jobsBC)

**unique()** to select the unique values.

**var()** the variance in data …

**which()** extracting the values that …  
• which(rivers > 2000) = 66, 68, 69, 70

**with()**  
• eg: with(trees, Volume)

**women()**  
The data set or data frame called women contains information on heights and weights of American women.

dataset1 <- read.table("file1.txt", header=TRUE, sep=" ", na.string=" ") #blanks missing values

**PLOTS**

plot()

Eg. Plot(c(1, 9) , c(0, 50), type = “n”, xlab=””, ylab=””)

pie() pie chart.

barplot() plot the bar chart

Eg. barplot(WorldPhones51, cex.names = .75, cex.axis = .75, main = “...”)

#cex.names argument reduced the size of the region names. Cex.axis reduced the labels on the vertical axis

#main argument sets the main title for the plot

Eg. barplot(VADeaths, beside = TRUE, legend = TRUE, ylim = c(0, 90), ylab = “Deaths per 1000”, main = “Death rates in Virginia”)

dotchart() dot chart

Eg. dotchart(WorldPhones51, xlab = “number of Phones...”)

#pch = 16 or other numbers to fill the dots or change their shape.

Eg. dotchart(VADeaths, xlim = c(0, 75), xlab = “Deaths per 1000”, main = “afja;fja;fja”, pch = 16)

hist() the main way to plot histograms #hist(x, ...)

Eg. hist(file, xlab = “x title”)

boxplot() are convenient for comparing distributions of data in two or more categories

Eg. boxplot(Sepal.Length ~ Species, data = iris, ylab = “Sepal length”, main = “Iris measurements”, boxwex = 0.5) #seperate side by side box plots for each species.

#Outlier – upper whisker – upper quartile – median – lower quartile – lower whisker – outlier

xyplot() # eg. xyplot(y ~ x | optionalfactor, data = mydata)

# eg. xyplot(leangth~ breadth | species, data = cuckoos, type = c(“p”, “smooth”))

#span =.75 #the span argument indicates what proportion of the data should be used to estimate each point of the smooth curve.

par() controls a very large number of parameters # eg. par(mar = c(5, 5, 5, 5) + 0.1)

text() adds text into the graph #eg. Text(6, 40, “...”) eg. Text(6, 20, “(6, 20)”, adj = c(0.5, 2))

points()

mtext() eg. Mtext(paste(“Margin”, 1:4), side = 1:4, line=3) mtext(text, side, line, ...) draws text in the margins

lines(x, y, ...) adds line segments

abline(a, b, ...) adds the line y = a + bx

abline(h=y, ...) adds a horizontal line

abline(v=x, ...) adds a vertical line

title(main, sub, xlab, ylab, ...) adds a main title, subtitle, an x-axis label/y-axis label

axis(side, at, labels, ...) adds an axis to the plot

box(...) adds a box around the plot region

mfrow=c(m, n) tells R to draw m rows and n columns of plots

mfg=c(i, j) draw the figure in row i and column j next

ask=TRUE tells R to ask before erasing a plot

cex=1.5 tells to expand characters

mar=c(side 1, side2, side 3, side 4)

oma=c(side1, side2, side3, side4) sets the outer margins

usr=c(x1, x2, y1, y2) sets the coordinate system within the plot wit x and y coordinates on the given ranges.

lcolor=rep(1:3, rep(2,3)) line color

color=rep(1:3, rep(2,3)) color of the dots or...

Factorial value

n <- 13

result <- 1

for (i in 1:n) result <- result \* i

result

or

factorial ()

**FUNCTION()**

Cor() the correlation between two numbers.

Corplot <- function (x, y, plotit) {

If (plotit == TRUE) plot(x, y)

cor (x, y)

}

rStdNorm <- function(n) {

Z <- 0

for (j in 1:12) {

U <- runif(n, min = -.5, max = .5)

Z <- Z + U

}

return(Z)

}

rStdNorm(…)

**Annuity**

annuityAmt <- function(n, R, i) {

R\*((1 + i)ˆ

n - 1) / i

}

annuityAmt(10, 400, 0.05)

**Eratosthenes**

Eratosthenes <- function(n) {

# Print all prime numbers up to n (based on the sieve of Eratosthenes

if (n >= 2) {

noMultiples <- function(j) sieve[(sieve %% j) != 0]

sieve <- seq(2, n)

primes <- c()

for (i in seq(2, n)) {

if (any(sieve == i)) {

primes <- c(primes, i)

sieve <- c(noMultiples(i), i)

}

}

return(primes)

} else {

stop("Input value of n should be at least 2.")

}

}

**Smoothing** a Scatterplot

smoother <- function(x, y, x.min, x.max, window) {

xpoints <- seq(x.min, x.max, len=401)

yaverages <- numeric(length(xpoints))

for (i in 1:length(xpoints)) {

indices <- which(abs(x - xpoints[i]) < window)

}

data.frame(x = xpoints, y = yaverages)

}

Within the for() loop just created, we add a line of code which assigns

the average of the values in y[indices] to yaverages[i]:

smoother <- function(x, y, x.min, x.max, window){

xpoints <- seq(x.min, x.max, len=401)

yaverages <- numeric(length(xpoints))

for (i in 1:length(xpoints)) {

indices <- which(abs(x - xpoints[i]) < window)

yaverages[i] <- mean(y[indices])

}

data.frame(x = xpoints, y = yaverages)

}

**Smoothing a Scatterplot**

smoother <- function(x, y, x.min, x.max, window=1) {

xpoints <- seq(x.min, x.max, len=401)

yaverages <- numeric(401)

for (i in 1:length(xpoints)) {

indices <- which(abs(x - xpoints[i]) < window)

if (length(indices) < 1) {

stop("Your choice of window width is too small.")

} else {

yaverages[i] <- mean(y[indices])

}

}

data.frame(x = xpoints, y = yaverages)

}

plot(faithful, pch=16, col="grey")

lines(doublesmoother(faithful$eruptions, faithful$waiting,

1.5, 5.0, c(1, 0.1)), col="blue", lwd=2)