SIMPLE TUTORIAL in R

R download from
http://www.r-project.org

partially based in
http://www.ats.ucla.edu/stat/r/notes

```
# INTRODUCTION

# R is accused of being slow, memory-hungry, and able
# to handle only small data sets. This is completely true.

# Fortunately, computers are fast and have lots of memory. Data
# sets with a few tens of thousands of observations can be handled
# in 256Mb of memory, and quite large data sets with 2Gb of memory.

#.........................................................................

# BASICS

# Lines with comments in R begin with sign #

help(solve)     # information on any specific named function
?solve          # alternative

help.search("plot")         # for help pages related to "plot"
apropos("thing")            # for functions whose names contain "plot"

# We can search any function in http://www.r-project.org

#.........................................................................

# OBJECTS and BASICS

# We can store data in an "object":
N <- 1000
N = 1000     # alternative

y <- c(3.1,10.5,14,30,15,19)
x <- c(4,12,12,20,16,22)

strata <- c("Madrid","Barcelona","Lisboa")      # character values
ok.set <- c(T,T,F,T,F)                          # logical values

ls()     # display the names of the objects in the workspace

# If you type the "object name" you see what is stored in the object
N
y
x

# To see what objects have been created
objects()
```

```
# To remove an object
rm(x)
rm(y,z)

# Other ways to enter/create data

z <- seq(1,10)                # a sequence of values
z <- c(rep(3,4),rep(5,2))     # join sequences of values

majors <- c(rep("Forestry",3),rep("Fisheries",5),rep("Math",2),
            "Education",rep("Business",2))


setwd("C:/kk")      # set to wherever your data directory is located
getwd()             # check that you are in the correct directory


# Run an ascii program written in R
source("c:/.../program.R")

#.........................................................................

# READ in DATA from a DATA FILE

# The easiest format in a file has variable names in the first row:

# case id gender   deg   yrdeg field startyr year   rank admin
#  1    1    F    Other     92  Other    95     95  Assist    0
#  2    2    M    Other     91  Other    94     94  Assist    0
#  3    2    M    Other     91  Other    94     95  Assist    0
#  4    4    M      PhD     96  Other    95     95  Assist    0

# and fields separated by spaces.

salary <- read.table("c:/.../salary.txt", header=TRUE)

# Data from the file salary.txt are stored into the data frame
# object "salary".

# HINTS:
#
# Many statistical packages (SAS, SPSS) can save data as an EXCEL file.
# Import any type of data into R by using EXCEL and saving there
# the data file into a comma delimited (*.csv) format.
# Once the comma delimited file is created using the "Save As" feature
# in EXCEL you can import it into R using either the read.table() or the
# read.csv() function.

thing <- read.table("c:/.../myfile.csv", "header=T", sep=",")

# Alternatively, you can use read.csv()
thing <-read.csv("c:/.../myfile.csv","header=T")

# Alternatively, tou can use the clipboard:
# Open the *.xls file in EXCEL
# Select and copy the relevant cells in Windows
thing <- read.table(file="clipboard",sep="\t",header=T)

# The file "clipboard" instructs read.table to read the file from the Windows
# clipboard, and the separator option of "\t" notifies read.table that elements
# are separated by tabs.

# The same way form R to EXCEL:
# Going to EXCEL and issuing the "paste" command will put the matrix
# into the EXCEL worksheet.

write.table(mymatrix,file="clipboard",sep="\t",col.names=NA)

# Files for read.table can also 'live' on the web
fl2000 <- read.table("http://faculty.washington.edu/tlumley/
data/FLvote.dat", header=TRUE)

# Another type of commonly used ASCII data format is fixed format.
# In this format data are placed in a fixed column for each observation.
# It requires a codebook to specify which column corresponds to which variable.

# Example:
# data are in file 'datfix.txt:
#         195  094951
#         26386161941
#         38780081841
#         479700  870
#         56878163690
#         66487182960
#         786  069  0
#         88194193921
#         98979090781
#        107868180801
#
# variable name column number
# id 1-2
# a1 3-4
# t1 5-6
# gender 7
# a2 8-9
# t2 10-11
# tgender 12

# To read these data we use the read.fwf() function on fixed format data
# instead of the read.table() function.
# Here, we use the width argument which indicates the width of
# each variable instead of using the sep argument to indicate the start
# of each variable.

fixed <- read.fwf("schdat_fix.txt", width = c(2, 2, 2, 1, 2, 2, 1))
names(fixed) <- c("id", "a1", "t1", "gender", "a2", "t2", "tgender")

fixed           # check the data

# Sometimes we read data from other packages, such as Stata or SPSS.
library(foreign)  # library to read foreign datasets
# read.dta:   read Stata (.dta) data files
# read.spss:  read SPSS (.sav) data files

#...............................................................................
```

```
# SIMPLE ARITHMETIC OPERATIONS

x+1           # add a 1 to all values in x
x+y           # add x and y
5*x           # multiply all values in x by 5
x^2           # take the square of all values in x
sqrt(x)       # take the square root of each value in x
log(x)        # take the natural log of each value in x

# Example: a sequence of arithmetic operations instead of one step
xbar <- mean(x)
diffs <- (x - xbar)      # subtract mean of x from each value
diffs.sq <- diffs^2      # square all the differences
ssx <- sum(diffs.sq)     # this is Sum of Squares of X

ssx <- sum((x-mean(x))^2) # can be done in one step

#................................................................................

# LISTS and DATA.FRAMES

# Examples of lists
w <- list(strat1=c(3,2,3),strat2=c(8,10,12,15))
x <- list(people=c("Zoe","Rapunzel","Igor"),
state=c("AK","AL","AK"),age=c(20,28,98))

# Example:
# One way to make side-by-side boxplots: make a list of
# the values with each component in the list corresponding to
# a different sample

y <- list(sample1=c(18,12,9,7,15,20),sample2=c(18,11,12,22,23,30),
     sample3=c(35,42,32,37,41,41,38,39))
boxplot(y)

# OPERATIONS with lists
x <- list(one=c(18:36),two=c("AK","AL","AZ"),
three=c(T,T,F,T),four=matrix(1:12,3,4))

# Access to components
x[[1]]          # by order
x$one           # by name

# Access to elements within components
x[[1]][3:6]
x$one[3:6]

unlist()        # convert a list to a vector
unlist(x)       # handy for printing out returned values from function

# List version of apply is lapply()
# (see next item of matrices)
lapply(x,length)

# DATA FRAMES: a special kind of list object; number of elements must
# be the same for all components

muscle <- rnorm(n=10,mean=3,sd=1)
sex <- factor(rep(c("M","F"),c(6,4)))
speed  <- rep(0,10)
speed[1:6] <- rnorm(6,30-2*muscle[1:6],2)
```

```
speed[7:10] <- rnorm(4,40-2*muscle[7:10],2)
mydata <- data.frame(y=speed,x1=muscle,x2=sex)
mydata

# Dealing with variables

# Commands:
# rbind: combines rows of data
# merge: match merges two data frames
# dimnames: lists or assigns names of data frames
# cbind: combines columns of data
# sapply: applies a function to elements of a list
# factor: creates a categorical variable with value labels if desired
# table: creates frequency table

# Keeping and Dropping Variables
hs1 <- read.table("http://www.ats.ucla.edu/stat/R/notes/hs1.csv", header=T,
sep=",")
attach(hs1)

# Keeping only the observations where the reading score is 60 or higher.
hs1.read.well <- hs1[read >= 60, ]

# Comparing means of read in the original hs1 data frame and the
# new smaller hs1.read.well data frame.
mean(hs1.read.well$read)
mean(hs1$read)

# Keeping only the variables id, female, read and write from the
# hs1.read.well data frame.
names(hs1.read.well)
hs1.kept <- hs1.read.well[ , c(1, 2, 7, 8)]
names(hs1.kept)

# Dropping the variables ses and prog from the hs1.read.well data frame
names(hs1.read.well)
hs1.drop <- hs1.read.well[ , -c(4, 12)]
names(hs1.drop)
detach()

# Consider two files:
# hsmale.txt with the information for the males
# hsfemale.txt with the information for the females
# Combine these two files


hsfemale <- read.table('http://www.ats.ucla.edu/stat/R/notes/hsfemale.txt',
header=T, sep=",")
hsmale <- read.table('http://www.ats.ucla.edu/stat/R/notes/hsmale.txt',
header=T, sep=",")
table(hsfemale$female)
table(hsmale$female)

# Use the rbind function when we stack data because we combine rows of data
hsmasters <- rbind(hsfemale, hsmale)
table(hsmasters$female)
detach()

# Merge two data frames on a variable (or a list of variables).
# We use variable id which has the same name in both data sets.
# Specifying T in the all argument indicates that we want to keep
# all the observations from each data set rather than only keeping
# the observations that came from both data sets.

hsdem <- read.table('http://www.ats.ucla.edu/stat/R/notes/hsdem.txt',
header=T, sep=",")
hstest <- read.table('http://www.ats.ucla.edu/stat/R/notes/hstest.txt',
header=T, sep=",")
hsdem
hstest

hsdiss <- merge(hstest, hsdem, by="id", all=T)
hsdiss

# If the variable that we were merging on had different names in each
# data frame then we could use the by.x and by.y arguments.

# In the by.x argument we would list the name of the variable(s) that
# was in the data frame listed first in the merge function
# (in this case in hstest) and in the by.y argument we would name the
# variable(s) that was in the data frame listed second (in this case hsdem).

hsdiss.1 <- merge(hstest, hsdem, by.x="id", by.y="id", all=T)
hsdiss.1

# Other option by creating an indicator of which data set the observations
# came from

from <- data.frame(rep(1, length(hsdem$id)))
dimnames(from)[[2]] <- "from"
hsdem.1 <- cbind(hsdem, from)
hsdem.1

from <- data.frame(rep(1, length(hstest$id)))
dimnames(from)[[2]] <- "from"
hstest.1 <- cbind(hstest, from)
hstest.1

hsdiss.2 <- merge(hstest.1, hsdem.1, by.x="id", by.y="id", all=T,
suffix=c("test", "dem"))

attach(hsdiss.2)

hsdiss.2$both[!is.na(fromtest) & !is.na(fromdem)] <- "both"
hsdiss.2$both[is.na(fromtest)] <- "dem"
hsdiss.2$both[is.na(fromdem)] <- "test"
hsdiss.2

# Factor variables
hs0 <- read.table("http://www.ats.ucla.edu/stat/R/notes/hs0.csv", header=T,
sep=",")
attach(hs0)

# Check if any of the variables in the hs0 data frame are factor variables.
sapply(hs0, is.factor)

# Creating a factor (categorical) variable called schtyp.f for
# schtyp with value labels.
schtyp.f <- factor(schtyp, levels=c(1, 2), labels=c("public", "private"))
search()
detach()
attach(hs0)
```

```
# Checking the factor variable schtyp.f in a frequency table.
table(schtyp.f)
schtyp.f

# Creating a factor variable called female from gender with value labels.
female <- factor(gender, levels=c(0, 1), labels=c("male", "female"))
detach()
attach(hs0)

# Checking the factor variable female in a frequency table.
table(female)
table(race)
race[race==5] <- NA
detach()
attach(hs0)
table(race)

# Creating a variable called total = read + write + socst
total <- read+write+socst
detach()
attach(hs0)
mean(total)

# Creating a variable called grade based on total
grade <- 0
grade[total >= 80 & total < 110] <- 1
grade[total >= 110 & total < 140] <- 2
grade[total >= 140 & total < 170] <- 3
grade[total >= 170] <- 4
detach()
attach(hs0)
table(grade)

# Creating a factor variable called grade.f based on grade
grade.f <- factor(grade, levels=0:4, labels=c("F", "D", "C", "B", "A"))
detach()
attach(hs0)
is.factor(grade.f)
table(grade.f)

# Labels are nice when looking at frequency tables.
table(schtyp, gender)   # without labels
table(schtyp.f, female) # with labels
detach()

#................................................................................

#   OPERATIONS with VECTORS

# Examples of vectors with different types of "elements"

w <- c(3,2,1)                            # numeric valued
x <- c(T,T,F,F)                          # logical valued
y <- c("Jane","Jill","Jeff","Matt")      # character valued
z <- matrix(c(3,3,2,4,2,1),nrow=3,ncol=2)  # numeric valued matrix

# Accessing elements of a vector in 1 of 4 ways
y <- c(18,32,15,-7,12,19)

# Position in vector as positive integer
```

```
y[3:5]

# Excluding elements, position as negative integers
y[-c(1,5,6)]

# By element name
names(y) <- c("Joe","Bill","Karen","Helen","Ray","Paul")
y[c("Helen","Ray")]

# By logical conditions
y[y<15]

# Merging vectors
# cbind() combines vectors by columns
c1 <- c(10,20,30,40)
c2 <- c(5,10,15,20)
x <- cbind(c1,c2)
x

# rbind() combines vectors by rows
x <- rbind(c1,c2)
x

#..............................................................................

#   OPERATIONS with MATRICES

y <- c(18,32,15,-7,12,19)
x <- matrix(data=y,nrow=2,ncol=3)    # fill by columns first: it is the default

x <- matrix(data=y,nrow=2,ncol=3,byrow=T)   # fill rows first

dimnames(x) <- list(c("r1","r2"),c("a","b","c"))

apply(x,1,sum)   # sum across the 1st dimension, namely rows
apply(x,2,sum)   # sum across the 2nd dimension, columns
apply(x,1,min)

# Examples:
A <- matrix(c(1, -2,  3,
        4, -5, -6,
        7,  8,  9,
        0,  0, 10),
      4, 3, byrow=TRUE)
A

t(A)          # transpose a matrix
diag(A)       # diagonal matrix
sum(diag(A))  # trace of a matrix

B <- matrix(c(-5, 1, 3,
         2, 2, 6,
         7, 3, -4),
      3, 3, byrow=TRUE)

A+B
A-B
-A

# Product of matrices
A %*% B
```

```r
B %*% A

# Inverse of a matrix: solve()
# Example:
A <- matrix(c(2, 5, 1, 3), 2, 2, byrow=TRUE)
solve(A)

# Check the result:
A %*% solve(A)
solve(A) %*% A

det(A)    # determinant of a matrix
eigen(A)  # eigenvalues and eigenvectors

#.............................................................................

# CONDITIONS AND LOOPS

#  if (...condition....) {
#    ...code 1...
#  }
#  else {
#     ...code 2...
#  }

# while (...condition....)
# {...code...}

# for(rank of indices)
# {...code...}


# Example 1
x <- 10
y <- 2
if (y >1){
         x <- 2*x
         y <- 2*y
      } else{
         x <- 38
         x <-2*x
     }
x
y

# Example 2
cunt <- c(0,0,0,0)
n <- c(2,4,6,4)
for(i in 1:length(n)){
cunt <- c(cunt,rep(i,n[i]))
}
cunt

# Example 3
for (i in 1:10) print(i)
n <- 10
while (n > 0) {
cat(n,"is greater than 0 \n")
       n <- n - 1
}
```

```r
#.............................................................................

# USEFUL FUNCTIONS

x <- c(10.1, 9.9, 11.2, 4.15, 2.3)

prod(x)              # Product of vector elements
cumsum(x)            # Cumulative sums products
diff(x)              # Lagged differences
round(x,1)           # Rounding of numbers
sort(x)              # Sorting or ordering vectors
rev(1:12)            # Reverse elements
rank(x)              # Sample ranks

# Example: find a minimum of a function
x <- seq(0,5,0.001)
fx <- x^3-8*x-20
m <- order(fx)
fx[m[1]]
x[m[1]]

# Samples
# To take a sample of a specified size from the elements of x
# using either with or without replacement
# sample(x, size, replace, prob)

# Example
x <- 1:12

sample(x)                # a random permutation
sample(x,replace=TRUE)   # bootstrap sampling (for length(x) > 1)

#.............................................................................

# EXTEND THE LANGUAGE BY WRITING YOUR OWN FUNCTIONS

# namefunction <- function(args)
# {
#   ... code ...
# }

# Examples of functions:
y <- c(3.1,10.5,14,30,15,19)
x <- c(4,12,12,20,16,22)
z <- cbind(x,y)

sd <- function(x) sqrt(var(x))
sd(x)

circle.area <- function(radius) {
  area <- pi*radius^2
  return(area)
 }
circle.area(4)

mystudy <- function(x){
    par(mfrow=c(3,1))
    hist(x[,1])
    hist(x[,2])
    plot(x[,1],x[,2])
    par(mfrow=c(1,1))
    apply(x,2,summary)
}
mystudy(z)

#.............................................................................

# SIMPLE STATISTICS, SUMMARIES, and PLOTS

# Typical R functions:
#
# head:      display first n observations
# sapply:    applies a function to elements in a list
# colMeans:  column means
# colSums:   column sums
# rowSums:   row sums
# median:    calculates the median
# length:    calculates the count
# var:       calculates the variance
# sd:        calculates the standard deviation
# tapply:    applies a function to each cell of a ragged array
# cbind:     combining columns
# summary:   generic function provides a synopsis of an object
# hist:      histogram plot
# histogram: trellis histogram plot(s)
# boxplot:   box plot
# bwplot:    trellis box plot(s)
# stem:      stem-and-leaf plot
# barplot:   bar plot
# table:     frequency table
# cor:       calculates correlations
# lm:        fits a linear model
# plot:      generic plot function
# abline:    adds a line to an existing plot

# Example

hs0 <- read.table("http://www.ats.ucla.edu/stat/R/notes/hs0.csv", header=T,
sep=",")
attach(hs0)
hs0[1:20, ]

names(hs0)
vars <- hs0[ , 7:10]  # shorthand way of referring to read, write, math, science
head(vars, n=10)

# The na.rm=T argument for the mean function is used to remove missing
# observations from the computation of the means.

sapply(hs0, mean, na.rm=T)
sapply(vars, length)  # count

# the count for science is wrong, we create a new variable with only
# the nonmissing cases of science and then use the length function

science.good <- na.omit(science)
length(science.good)

sapply(vars, median, na.rm=T)   # median
sapply(vars, var, na.rm=T)      # variance
sapply(vars, sd, na.rm=T)       # standard deviation
sapply(vars, min, na.rm=T)
sapply(vars, max, na.rm=T)

# Tukey's five number summary
# - the maximum value
# - the 75th percentile
# - the 50th percentile
# - the 25th percentile
# - the minimum value
sapply(vars, fivenum, na.rm=T)

# We can also use the colMeans function to obtain the mean.
# We can specify the variables by their numbers as in the sapply
# or as variable names using cbind.
colMeans(vars, na.rm=T)

# Descriptive statistics can also be computed for a subset of the data frame:

# we are looking at the summary statistics for only those students
# who had a reading score of 60 or higher.
sapply(vars[read >= 60, ], mean, na.rm=T)
sapply(vars[read >= 60, ], median, na.rm=T)

# Obtaining the means of the variables write and science broken down by prgtype.
# Science is the only variable with missing observations and thus
# we use the na.rm to remove the missing observation.
tapply(write, prgtype, mean)
tapply(science, prgtype, mean, na.rm=T)

tapply(write, prgtype, length)  # count
tapply(write, prgtype, var)     # variance
tapply(write, prgtype, sd)      # standard deviation
tapply(write, prgtype, median)  # median

# Descriptive statistics for write by prgtyp in a much nicer display.

m   <- tapply(write, prgtype, mean)
v   <- tapply(write, prgtype, var)
med <- tapply(write, prgtype, median)
n   <- tapply(write, prgtype, length)
sd  <- tapply(write, prgtype, sd)
cbind(mean=m, var=v, std.dev=sd, median=med, n=n)

# More descriptive statistics including quantiles can be obtained by
# using the summary function.

summary(science)

#..............................................................................

# EXPLORING THE DATA THROUGH GRAPHS

library(lattice)  # load trellis graphics
hist(write)

# trellis graphs
histogram(~write, hs0, type="count")
histogram(~write | gender, hs0, type="count")  # histogram of write by gender

# Note: In R it is possible to change the number of bins by
# using the breaks argument in the hist function.
hist(write, breaks=15)
```

```
# Put several plots on one image
par(mfrow=c(2,1))
hist(write, breaks=15)
hist(write)

# boxplot of the variable write
boxplot(write)

# trellis graph of write by ses
bwplot(ses ~ write, hs0)

# trellis graph of boxplots of write by ses for each level of gender
bwplot(ses ~ write| gender, hs0)

# The graph shows ses by gender where the levels of ses are stacked
# on top of another
barplot(table(ses,gender), legend=c("low","medium","high"), ylim=c(0,135))

barplot(table(ses,gender), beside=T, legend=c("low","medium","high"),
ylim=c(0,60))

#................................................................................

# FREQUENCY TABLES

table(ses)

# The table of write shows that it is generally undesirable to
# obtain frequencies of continuous variables.
table(write)

table.vars <- hs0[ , c(1,5,6)]  # shorthand way of referring to gender, schtyp
and prgtype
sapply(table.vars, table)

# Crosstabulation of gender and ses.
tab1 <- table(gender,ses)
tab1

# Compute the row and column proportions and frequencies
# and a chisquare test of independence for the two-way table.

prop.table(tab1,1)  # row proportions
prop.table(tab1,2)  # column proportions
rowSums(tab1)       # row frequencies
colSums(tab1)       # column frequencies
summary(tab1)       # chi-square test of independence

# Correlations of write, read, math and science with listwise deletion
# of missing values.
# The correlations will not be calculated if there are missing values

cor(vars, use="complete.obs")

#................................................................................
```

```r
# ANALYZING DATA

hs1 <- read.table("http://www.ats.ucla.edu/stat/R/notes/hs1.csv",
header=T, sep=",")

attach(hs1)

# t.test: t-tests, including one sample, two sample and paired
# tapply: applies a function to each cell of a ragged array
# var: calculates the variance
# lm: fits a linear model (regression)
# anova: extracts the anova table from a lm object
# summary: generic function provides a synopsis of an object
# fitted: extracts the fitted values from a lm object
# resid: extracts the residuals from a lm object
# abline: generic function which adds a line to an existing plot
# glm: logistic regression
# drop1: compares model by dropping terms one at a time
# wilcox.test: non-parametric analyses
# kruskal.test: non-parametric analyses

# t-tests
# one-sample t-test, testing whether the sample of writing scores
# was drawn from a population with a mean of 50.
t.test(write, mu=50)

# paired t-test, testing whether or not the mean of write
# equals the mean of read.
t.test(write, read, paired=TRUE)

# two-sample independent t-test.
# use the tapply function to look at the variances of the variable
# write for each group of female.
tapply(write, female, var)
t.test(write~female, var.equal=TRUE)   # assuming equal variances
t.test(write~female, var.equal=FALSE)  # assuming unequal variances

# ANOVA

# In R you can use either the aov function or the anova function
# combined with the lm function.
# The anova function extracts the anova table from the linear model
# fitted by the lm function.
# The aov function only fits an anova model and we use the summary
# function to see all the output.

anova(lm(write~factor(prog)))
# is equivalent to
summary(aov(write~factor(prog)))

# two factors with interactions
anova(lm(write~factor(prog)*female))
summary(aov(write~factor(prog)*female))

# Analysis of covariance (ANCOVA)
# here, prog is the categorical predictor and read is the continuous covariate

anova(lm(write~factor(prog) + read))
summary(aov(write~factor(prog) + read))
```

```
# REGRESSION

summary(lm(write~female+read))

# plot function will produce multiple diagnostic plots when applied
# to an lm object. These plots include residual versus fitted plots,
# qqplots of the residuals as well as scatter plots with the regression
# line overlaid

lm2 <- lm(write~read+socst)
summary(lm2)
plot(lm2)      # plotting diagnostic plots of lm2

# Plotting all in one figure
par(mfrow=c(2,2))
plot(lm2)
```