Getting started with R

Eric Feigelson (Penn State) edf@astro.psu.edu

2nd East Asian Workshops in Astrostatistics Summer 2018

Adapted from R scripts in Appendix B, *Modern Statistical Methods for Astronomy With R Applications*, Eric D. Feigelson & G. Jogesh Babu 2012 http://astrostatistics.psu.edu/MSMA (http://astrostatistics.psu.edu/MSMA)

R is a powerful software environment for data analysis, graphics, and especially statistical analysis. It is available free to the public at www.r-project.org with easily installed binaries for Linux, MacOS and Windows. This notebook provides an introduction to R designed for students and researchers in astronomy. Some familiarity with scripting languages like Matlab or IDL is helpful.

Some basic information:

- R is driven by the command line in the R console. Double-click the R icon or (in Linux) type 'R' in the terminal to open the console.
- R has >100,000 'functions' that perform various tasks. Each function uses () brackets to list arguments and parameters.
- Every function has a 'help(fn)' page telling how it is used, what operation is run, and what output it produces with references an examples that can be cut-and-paste into an R console.
- A hash mark (#) denotes comments in an R script. A semi-colon (;) has the same action as a carriage return
- The following commands can be run interactively in Jupyter, or cut-and-pasted into a separate R console.

In [ ]:
```
getwd()                              # find working directory.
# setwd('/Users/myself/Rdir')   # set your personal working directory
setwd('/Users/ericfeigelson/Desktop/Rdir')
getwd()
sessionInfo()                        # learn about your environment
citation()                           # quote this citation in any publicati
on using R
```

In [ ]:
```
# II. Create and characterize a vector

a <- c(33, 44, 92, 58)      # combine numbers into a vector
length(a)
ls()                        # list names of objects in your environmen
t
class(a)                    # state the `class' of an R object (descri
bed in III below)
str(a)                      # state the structure of an R object
a                           # state the contents of an R object
write(file='output', a)     # write an ASCII file into the working dir
ectory
save(file='output_bin', a)  # write a binary file
Save(file='output_bin', a)  # error because 'Save' is not a known func
tion. R is case sensitive.

# Manipulation of vector values
mean(a) ; median(a)
min(a)  ; max(a)
sum(a)
summary(a)                  # many R objects have a built-in 'summary'
function

# Manipulation of vector indices
a[1:4]
a[3]            # Note that R vectors start with index 1 not 0, unlike
Python
a > 40          # logical operation
sum(a[a>40])    # note the self-referential use of vector/array indice
s here
which.max(a)
match(44, a)
```

In [ ]:
```
# Manipulation of vector values
mean(a) ; median(a)
min(a)  ; max(a)
sum(a)
```

In [ ]:
```
# Manipulation of vector indices
a[1:4]
a[3]            # Note that R vectors start with index 1 not 0, unlike
Python
a > 40          # logical operation
sum(a[a>40])    # note the self-referential use of vector/array indice
s here
which.max(a)
match(44, a)
```

In [ ]:
```
# III. R classes: list and data frame

# R objects are placed into `classes': numeric, character, logical,
# vector, matrix, factor, data.frame, list, and dozens of others desig
ned
# by advanced R functions and CRAN packages. plot, print, summary func
tions
# are adapted to class objects; see e.g. methods(summary).

# The list class allows a hierarchical structure of heterogeneous R ob
jects.
# Here we make a hierarchical list, use 'str' to show its contents, an
d
# access an element of the list using the $ delimiter

b_list <- list(star=c('Sirius', 'Procyon'), SpTy=c('O','B','A'),
Hubble_km.s=68)
str(b_list)
b_list[['SpTy']] = list(subtype=seq(0.1:0.9, by=0.1))
str(b_list)
b_list$SpTy$subtype[1:3]
```

In [ ]:
```
# Make and write a data.frame, a 2D array with column names
d <- data.frame(cbind(seq(1:4), a, a^3))  # Bind columns into data fra
me
names(d) <- c('ID', 'a_value', 'a_cubed') # Column names for data fram
e
d2 <- d[-4,-1]                            # Remove 4th row and 1st col
umn
d ; d2
write.table(d, file='d.txt', quote=FALSE, row.names=FALSE)
```

In [ ]:
```
# Download and use a CRAN package
install.packages('xtable')                # Download a CRAN package
library(xtable)                           # Bring LaTeX package into s
ession
print(xtable(d), file='d.tex')            # Write table with LaTeX for
mat
```

In [ ]:
```
# R help files give essential information on all functions in a
# standard format
help(cat)  #  Examine help files for many of the functions used above
```

In [ ]:
```
# IV. Arithmetic, algebra, trigonometry, and formatting numerics

5 + 3 ; 5-3 ; 5*3 ; 5/3 ; 5^3
x <- 5 ; y <- 3
x+y

sin(0)  ; sin(pi/2)
ang <- seq(0, pi/2, length=30)
sin(ang)

trunc(12345.6789) ; round(12345.6789)
format(12345.6789, digits=2, scientific=TRUE)

log(20)  ;  log10(20)   # log() in R is base-e. Use log10() for base-10 logarithms.
```

In [ ]:
```
# V. Astrophysical calculations of galaxy distances
# The `function' function: Note how one function uses another
# This how R builds new capabilities based on old capabilities in a compact syntax.

# First, make a simple calculation without functions
z <- seq(0.0, 0.5, 0.1)
z
H_0 <- 68                                  # km/s/Mpc,  Planck value
speed.light <- 3.0E5    # km/s
dist <- speed.light*z / H_0
dist

# Now, make a more complicated calculation with function
Omega_m <- (0.022068 + 0.12029) / (H_0/100)^2
Omega_Lambda <- 0.6825                     # Planck values

E.H0 <- function(redshift) {sqrt(Omega_m*(1+redshift)^3 + Omega_Lambda)}

lum.dist <- function(redshift) {
        luminosity.distance = (speed.light/H_0) * integrate(E.H0, 0, redshift)$value
        return(luminosity.distance)  }

distGR <- Vectorize(lum.dist)(z)

# Finally, plot the results
plot(z, distGR, type='l', lty=2, lwd=2, ylab='Distance (Mpc)')
lines(z, dist, lty=1, lwd=2)
legend(0.0, 2500, lty=c(1,2), lwd=c(2,2), title='Galaxy distances',
        legend=c('Euclidean', expression(Lambda*CDM)))
```

In [ ]:
```
# VI. Examine, summarize and plot univariate distributions

set.seed(1)
x <- sample(seq(0.01, 3, length.out=500))
y <- 0.5*x + 0.3^(x^2) + rnorm(500, mean=0, sd=(0.05*(1+x^2)))
xy <- cbind(x, y)

plot(xy, pch=20)
summary(x) ; summary(y)         # Summarizes properties of an R object

par(mfrow=c(1,2))               # Set up a two-panel figure
boxplot(x,  notch=T, main='Boxplot for X')
boxplot(y,  notch=T, pch=20, cex=0.5, main='Boxplot for Y')
dev.copy2pdf(file='box.pdf')

par(mfrow=c(1,1))
hist(x, breaks=30, main='', xlim=range(x), ylim=c(0,100), xlab='Yvar',
col='royalblue4')
hist(y, breaks=30, main='', xlab='Yvar', col='#ee3b3b70', add=T) # add
=TRUE suppresses a new plot

qqnorm(y, pch=20, cex=0.5)  # Quantile function of y compared to norma
l distribution
qqline(y)                                       # Expected relationshi
p if y is normal

plot(ecdf(x), pch=20, cex=0.0, verticals=TRUE, main='',ylab='EDF',xlab
='')
plot(ecdf(y), pch=20, cex=0.0, verticals=TRUE, add=T)
text(2.0,0.5,"X") ; text(1.4,0.8,"Y")             # text adds annotati
on within a plot
dev.copy2pdf(file='ecdf.pdf')
```

In [ ]:
```
# VII. Arrays, data frames and filtering

xy <- cbind(x, y)  ;  str(xy)   # Here xy is an `array' of numbers cre
ated by `column bind'
xy <- as.data.frame(xy)         # A data.frame associates names to the
columns
names(xy) <- c('Xvar', 'Yvar')

high_x1 <- xy[xy[,1]>2,]        # Rows where the first column value ex
ceeds 2
high_x2 <- subset(xy, xy[,1]>2) # Another way to extract rows
setequal(high_x1, high_x2)      # test equality of two vectors
```

In [ ]:
```
# VIII. Sampling and bootstrapping

trials <- sample.int(length(xy[,1]),20) # 20 random rows
xy[trials,]

trials <- sample.int(length(xy[,1]),20, replace=T) # 20 bootstrap resa
mples
xy[trials,]

median(xy[,2])                  # Estimate the standard error of the m
edian of Yvar
mad(xy[,2]) / sqrt(500) # Median absolute deviation estimate of median
s.e.

library(boot)                   # The following function in a base-R l
ibrary
med <- function(x,index) median(x[index])
# Read help(boot) to understand its output list structure
boot(xy[,2], med, R=1000) # Bootstrap estimate of median s.e.
hist(boot(xy[,2], med, R=1000)$t, breaks=50, xlab='Bootstrap median of
Yvar')
```

In [ ]:
```
# IX. Bivariate plots and correlation tests

par(mfrow=c(1,2))
plot(xy, pch=20, cex=0.5)       # Scatterplot. See help(points) for sy
mbol shapes.
plot(log10(xy), pch=20, cex=0.5, xlab='log(Xvar)', ylab='log(Yvar)')

length(x[x>2])          # State length of a vector.  Use `dim' for an
array or data.frame.
cor.test(x[x>2],y[x>2], method='pearson')       # Parametric hypothesi
s test for bivariate correlation
cor.test(x[x>2],y[x>2], method='kendall')       # Nonparametric hypoth
esis test for bivariate correlation
```

In [ ]:
```
# X.  Some programming features

# Two high-quality introductions to R:
#    From the R Core Team: https://cran.r-project.org/doc/manuals/R-in
tro.html
#    From Carnegie-Mellon University: http://www.stat.cmu.edu/~cshaliz
i/statcomp/14/

# A list of the ~30 important CRAN packages embedded in the base-R env
ironment
library()
```

```
# A full list of ~400 functions in R's `base' package
library(help = "base")

# Statistics in base R (~400, thousands more in CRAN and elsewhere in
R)
library(help='stats')

ls()                    # List current contents of environment

# Programming utilities including:
# Use `source' to bring in external R scripts
# Use `edit' to edit an R object
# Use 'environment' to segregate a collection of objects
# Functions `debug', `trace' and `browser' assist with code testing
# Function 'process.events' allows low-level handling of R commands
library(help = 'utils')

# Loops:  for( i in 1:100) { ... }
# Program flow control:  if/else, ifelse, switch, while, repeat, next,
break, stop
foo <- 2
if(foo == 1) cat('Hello world!') else cat('Do nothing')

# Graphics and devices in base R (other packages in CRAN)
library(help='graphics')
library(help='grDevices')

# Parallel computing control in base R (dozens of other HPC packages a
re in CRAN)
library(help='parallel')

# Run an R script residing on disk
help(source)

# Save R objects (or your full environment) onto disk
help(save) ; help(load)

# Save or load history of R commands
help(savehistory)  ;  help(loadhistory)

# Connections, pipes, sockets, URLs, clipboard, compression, etc.
help(connections)

# Interact with host computer
Sys.info()
system('ls -l')
system.time(fft(seq(0,1,length.out=1000000)))   # A million fast Fouri
er transforms

# Construct composite strings using 'paste'
```

```
# Extract postions of a string using `substring'
band_ir <- 'J'
paste('NGC1068',band_ir,'FITS', sep='.')

# FITS format reader/writer
install.packages('FITSio') ; library(FITSio)

# IDL Astro Library translated into R
install.packages('astrolibR') ; library(astrolibR)

# Wrapping R functions from Python
### pip install rpy2
### import rpy2
### import rpy2.robjects as robjects
### R = robjects.r
### ranGauss = R.rnorm(100)
### print ranGauss
```