Frequently Applied Command

**#check if the objective is numeric:** when the entry of one function could be numeric or character, one could use if condition to check

if ( as.numeric(x)){do something} # the as.numeric(x) check if x is numeric and if yes it carries out the statement

**#subset a data frame** selecting by the name of the column

x <- data.frame(cbind(C1=1, C2=1:8, C3=c("A","B","C","D","E","F","G","H")))

x[, c(“C1”,”C2)]

or

x <- as.data.frame(cbind(x[, c(1:3)]), stringAsFactors = FLASE) # using cbind to select c(x:y) columns with no column names (later given name colnames(x) <- c(“name”))

**#subset a data frame** where the elements of one columns equal to certain value (could be numeric or character):

x <- data.frame(cbind(C1=1, C2=1:8, C3=c("A","B","C","D","E","F","G","H")))

C1 C2 C3

1 1 1 A

2 1 2 B

3 1 3 C

4 1 4 D

5 1 5 E

6 1 6 F

7 1 7 G

8 1 8 H

x[x["C3"] == "A", ]

C1 C2 C3

1 1 1 A

Or

X <- which(x[,”C3] == “A”)

**#if / else if** structure control

if (condition) {expression

} else if (condition) {expression

} else if (condition) {expression

}

**count how many elements are NA value**

length(which(is.na(x)))

using “!” for the non-NA value

length(which(!is.na(x)))

**loop over the different levels of elements in a column:** loop over country, month and so on???

**rank a data frame with multiple columns:** first rank by values than by names…

result <- data[order(data[,”Value”], data[,”Name”], na.last = NA), ]

**Data Types**

Everything in R is object

The most basic object is a vector

A vector can only contain objects of the same class, except for a *list* (special type of vector)

c() is used to create vectors of objects

Index of list in R console has double brackets and other objects has only single brackets

x <- list()

Matrices are special type of vector in R with a *dimension* attribute. This attribute is itself an integer vector of length 2 (nrow, ncol)

x <- factor(c()) Factor are used to represent categorical data (unordered or ordered); it is an integer vector where each integer has a *label*

NAN – undefined mathematical operations NA – pretty much everything else

x <- data.frame() Data Frames is a **key** data type used in R to store tabular data; a special type of list where the length of each element of the list is the number of rows

All R objects can have namesx

**Control Structure of R**

if, else/for/while/repeat/break/next/return

if(condition) {do something} {else do something else} #test logical condition

use shift + enter to write multiple “do something” in {}

signing a complete if/else condition to an objective for clear viewing

for(sequence) {do something}

seq\_along(x) create a sequence with the length of vector x

seq\_len(nrow(x))

while(logical expression) {do something} #more readable

*next* is to skip certain iteration: for (i in 1:100) { if(i<=20) {next}

**Lexical Scoping**

An *environment* is a collection of (symbol, value) pairs, i.e. x is a symbol and 3.14 might be its value

A function + an environment = a closure or function closure

Lexical scoping allows “function over function”

**Coding Standard**

Always use text files / text editor

Indent your code (tab width 4 – 8)

Limit the width of your code (80 columns?)

Limit the length of individual functions

**Dates and Times**

as.Date(“dates”)

Sys.time

POSIXlt() as a list and POSIXct() as an integer (seconds of that day)

*strptime(some\_datestring, “%B %d, %Y %H:%M”)* convert dates in character string into time format

**Loop Function**

*slpit* is useful in conjuction with *lapply*

*lapply(list, FUN,…)* is applied over a **list** and **always** returns a list; the “…” can past arguments from *FUN* to *lapply*

Anonymous function (with any not yet taken function name) could be used in *lapply* to execute the action. And this function goes away after the execution.

*sapply* simplify the result resulting from using *lapply*

*apply(X, MARGIN, FUN, …)* is applied over an **array** and the *margin* indicates the dimension which is *NOT* get collapsed

Functions like *rowSums, rowMeans, colSums* and *colMeans* exist and are equivalent to *apply*

*mapply* is a multivariate apply of sorts which applies a function in parallel over a set of arguments. For example *mapply(rep, c(1,2,3), c(1,2,3)* will create a list with [[1]] 1, [[2]] 2, 2, [[3]] 3, 3, 3

*tapply* is used to apply a function over subsets of a vector. Use together with *gl()* to generate the levels in a vector.

*split* is often used with the above loop function

**Loop Functions – split** video contains some stuff which I might need to watch again

**Debugging Tools**

traceback, debug, browser, trace and recover are serial of debugging tools

**Simulation**

Functions for probability distributions ins R: rnorm/dnor/pnor/rpois

set.seed()

**Sample**

sample(1:10 , 4) sample 4 from the vector

sample(1:10, replace =TRUE) returns vector with 10 elements with replacement

Profiling

system.time() given the time needed to execute an expression

user time is the time charged to the CPU(s) for this expression, while elapsed time is the “wall clock” time

elapsed time > user time if the CPU spends a lot of time waiting around; elapsed time < user time if your machine has multiple cores/processors

Rprof() start the profiler in R, if you don’t know where to look at (to use system.time())

summaryRprof() summarizes the output from Rprof()

DON’T use Rprof() and syste.time() together

by.self give you the time spend in that function subtracting the time spent in lower level functions