**DRAFT R Reference Card for Soil Scientists**

**April 2016**

Note: x is used as the object example

# Import/Export/Save

**setwd("C:/workspace")**

set working directory

**getwd**( )current working

directory

**x <- read.table(filename, header = TRUE, sep = " ")** reads

a file in a table format

with a header row and

creates a dataframe

called x

**x <- read.csv("filename", header = TRUE)** reads a CSV file

with a header row and

creates a dataframe

called x

**write.table(x, file = "filename.txt", sep = "\t", header = TRUE)** writes a dataframe to a tab

delimited text file with

a header row

**write.csv(x, file = "filename.csv", header = TRUE)**

writes a dataframe to a

comma delimitated file

with a header row

**save.image( )** saves all objects

**save.image(file = "filename.RData")**

saves R workspace

**savehistory(file = "filename.Rhistory")** saves a

history of the commands

that have been executed during a R session

**loadhistory(file = "filename.Rhistory")** loads a

history file

**history(max.show = Inf)**

displays a list of all

previous commands;

max.show specifies

the number of lines displayed

# Packages

**install.packages("packagename", dep = TRUE)** installs a package

and its dependencies

**library(packagename)** loads

package

**remove.packages(packagename)**

uninstalls package

# Help

**help(command, package = packagename**)

**?command**

# Examining Data Structure

**View(x)** opens a spreadsheet-

style viewer

**ls()** list of all objects in

the search path

**str(x)** displays structure of

an R object

**class(x)**displays the object

class

**names(x**)displays column names

**head(x)** displays the first

parts of x

**tail(x)** displays the last part

of x

**nrow(x)** displays number of rows

**ncol(x)** displays number of

columns

**length(x)** object length or

characters in a

vector, dataframe, or

array

**unique(x**) returns the unique

values for factors

***EX: unique(SPC$landform.string****)*

*would return a list of*

*all landforms in the SPC*

*object*

# Editing and Summarizing Data

**summary(x)** summary of model

results or each column

in a matrix or dataframe

**as.factor(x$column**) change

a column to a factor

**ordered(x$column)** logical

flag to determine if the

levels in a factor

are ordered

**as.numeric(x$column)**

change the column to

numeric

**order(x$column)** returns a

permutation which

rearranges its first

argument into ascending

order; descending order:

**order(x$column, decreasing =**

**TRUE)**

**sort(x)** sorts a vector or

factor(partially) in

ascending or descending

order

***EX: sort(table(x$subgroup),***

***decreasing = TRUE)*** *would sort*

*a table depicting a*

*frequency count of*

*subgroup in decreasing*

*order*

***EX: sort(unique(x$hzname))***

*would return a vector of*

*unique horizon names in*

*ascending order*

**na.omit(x)** removes rows with NA

values

**new.object <- edit(x)** opens a

spreadsheet-like text

editor where edits can

be saved if a new object

is specified

**dput(x, file = " ")** writes an

ASCII text

representation of a R

object to a file or

connection. Useful for

converting a list (ex:

"2011MT081001"

"2011MT081009") into a

comma delimited string

(ex: c("2011MT0810001",

"2011MT0810009")) for

use in queries

**table(x)** displays contingency

table of the counts at

each combination of

factor levels

***EX:******table(x$subgroup, x$order)***

*would display a contingency*

*table of frequency counts*

*by subgroup and order*

**c( )** concatenates values into a

vector or list**. O**utput

type is determined from

the highest type of the

components in the

hierarchy NULL < raw <

logical < integer <

double < complex <

character < list <

expression

**which( )** returns the TRUE

indices of a logical

object, allowing for

array indices

***EX: which(x$partsizeclass !=***

***'sandy-skeletal')*** *returns*

*the row numbers of*

*object x where the*

*particle size class was*

*everything except sandy-*

*skeletal*

***EX: which(x$partsizeclass ==***

***'sandy-skeletal')*** *returns*

*the row numbers of*

*object x where the*

*particle size class is*

*sandy-skeletal*

***EX: which(x$depthtobedrock > 102****)*

*returns the row*

*numbers where depth to*

*bedrock is greater than*

*102*

***EX: which(x$depthtobedrock >= 102****)*

*returns the row*

*numbers where depth to*

*bedrock is greater than*

*or equal to 102*

***EX: which(x$partsizeclass %in%***

***c('loamy-skeletal', 'sandy-***

***skeletal'))*** *returns the row*

*numbers of objects where*

*the particle size class*

*is sandy-skeletal or*

*loamy-skeletal*

**ifelse(test, yes, no)** value with

the same shape as test

filled with elements

from either yes or no

**EX: ifelse(as.numeric(data$lime)**

**== 2, 1, 0)** would return a

numerical vector of 1s and

0s, 1 representing lime = 2

and 0 representing lime =

to anything other than 2

**grepl(pattern, vector)** searches

for matches to an

argument pattern within

each element of a

character vector

***EX: grepl('dyst', subgroup)*** *finds*

*the rows with the text dyst*

*in the object called*

*subgroup*

***EX: grep('loamy | sandy',***

***f$partsize\_class)*** *finds*

*pedons with loamy* ***or*** *sandy*

*partsize\_class*

***EX: grep('^sandy',***

***f$partsize\_class)*** *finds*

*pedons with partsize\_class*

*beginning with sandy*

***EX: grep('skeletal$',***

***f$partsize\_class)*** *finds*

*pedons with partsize\_class*

*ending with skeletal*

***EX: grep('loamy & sandy',***

***f$partsize\_class)*** *finds*

*pedons with loamy* ***and*** *sandy*

*partsize\_class*

**with(data, expr)** evaluates expr

in a local environment

constructed from data

***EX: with(data,table(hillslope\_pos,***

***argillic.horizon, useNA =***

***"ifany"))*** *generates a*

*frequency table from data*

*with two columns: hillslope\_pos and*

*argillic.horizon. This table*

*would include all NA values*

*since useNA is set to*

*“ifany”****.***

**subset(data, select = )** subsets

vectors, matrices or

dataframes

**aggregate(column1 ~ column2, data = , statistic)** splits the

data into subsets,

computes summary

statistic for each, and

returns the result in a

convenient form

***EX: aggregate(clay ~ horizon, data***

***= h, mean)*** *creates a table*

*of mean clay content for*

*each horizon in h*

***EX: aggregate(clay ~ horizon, data***

***= h, summary)*** *generates a*

*table of the minimum,*

*1st and 3rd quantiles,*

*median, mean, and*

*maximum clay content for*

*each horizon in h*

**round(x)** rounds the values in

its first argument to

the specified number of

decimal places (default

= 0)

***EX: table(round(data$clay))*** *would*

*generate a table with*

*rounded clay values*

**addmargins(table(x, z))** adds

column and row summaries

to a table made up of

one or more vectors (in

this example, x and z)

**prop.table(table(x), margin = NULL)\*100** returns a proportions

table

***EX: round(prop.table(table(data$ho***

***rizon, data$texture\_class),***

***margin = 1) \* 100)*** *would*

*generate a rounded*

*proportions table relative*

*to the rows, margin = 1*

*calculates for rows,*

*margin = 2 calculates*

*for columns, margin =*

*NULL calculates for*

*total observations*

**data.frame(x, z)** combines two

vectors (x and z in this

example), creating a

dataframe

**cbind( )** binds columns into a

matrix

**rbind( )** binds rows into a

matrix

# Exploratory Data Analysis

**hist(x)** creates a histogram

**histogram(~ clay + sand + total\_frags\_pct, data = x)**

using the lattice

package, would create

separate histograms for

clay, sand, and total\_frags\_pct

***EX: histogram(~ clay | horizon,***

***data = x)*** *would plot*

*multiple histograms,*

*one for every horizon in*

*object x*

**dp <- density** plots the

computed density plot

**plot(dp)** using the lattice

package, densityplot( )

produces a Kernel

Density Plot

***EX:******densityplot(~ clay + sand,***

***data = x, auto.key = TRUE)***

*would create a density plot*

*with two curves (clay and*

*sand), removing NA values*

**qqnorm(x)** generates a QQplot

with a line that

represents the quantiles

of a normal distribution. If

the data are normally

distributed, the data

points will fall very

close to the QQline

**qqline(x**)

**mean(x, na.rm = TRUE)** returns

the arithmetic mean of a

numerical vector after

removing NA values

**median(x)** returns the middle

measurement of a sample

set, and as such is a

more robust estimate of

central tendency than

the mean

**var(x**) computes the variance

**sd(x)** computes the standard

deviation

**quantile(x)** returns the min,

25th percentile, median

or 50th percentile, 75th

percentile, and max from

a numeric vector

**range(x)** returns the minimum

and maximum values

**plot(y ~ grp, data = x)** produces

a "box-and-whiskers"

plot where y represents

a numeric vector of data

values to be split into

groups according to

variable grp which is usually a factor

**boxplot(y ~ grp, data = x)** boxplot

**bwplot(y ~ grp, data = x )**

boxplot using the

lattice package

**vars <- c("clay", "sand", "phfield")** creates an object

called vars that

contains numerical

vectors clay, sand, and

phfield

**round(cor(data[vars], use = "complete.obs"), 2)** creates a

correlation matrix of

vars to 2 decimal places

from vars

**plot(a ~ b, data = x)** creates a

scatterplot of two

numeric variables (in

this example, a and b)

**xyplot(a ~ b, data = x)**

scatterplot using

lattice

**splom(data[vars])** using

lattice, creates a

scatterplot matrix for

all numeric variables

(vars) in a dataset

**pairs(data[vars])**

generates a scatterplot

matrix for all numeric

variables (vars)

**aspect <- data$aspect\_field**

**aspect <- circular(aspect,**

**template = "geographic", units =**

**"degrees", modulo = "2pi")** using

the circular package,

creates a circular object

(in this example, aspect)

to provide a numerical

summary

**rose.diag(aspect, bins = 8, col = "grey")** using the circular package,

rose.diag( )generates a

rose diagram plot of aspect

# Spatial

## Points and General Mapping Commands

**coordinates(dataframe) <- ~ x + y**

using the sp package,

promotes a dataframe to

a spatial object or

retrieves spatial

coordinates from a

spatial object

**proj4string(spatialobject) <- '+proj = longlat + datum = WGS84'**

sets the projection of a

spatial object

**map('county', 'statename'**)

using the maps package

plots counties and states

**points(x)** draws a sequence

of points at the

specified spatial extent

**bbox(spatialobject)** returns the

bounding box coordinates

for a spatial object

**spTransform(spatialobject, CRS("+init = epsg:code"))** using

the rgdal package, the

spTransform command will

reproject spatial objects

[Projection Codes](https://www.nceas.ucsb.edu/~frazier/RSpatialGuides/OverviewCoordinateReferenceSystems.pdf)

## Raster

*The following demonstrates two ways of creating a raster stack:*

**folder <- "C:/geodata/"**

sets file path

**files <- c(**

**elev = "ned30m.tif",**

**slope = "ned30m\_slope5.tif",**

**aspect = "ned30m\_aspect5.tif",**

**twi = "ned30m\_wetness.tif")**

list of file names

**geodata\_f <- sapply(files, function(x) paste0(folder, x))**

combines the folder

directory and file names

**geodata\_r <- stack(geodata\_f**)

creates a raster stack

**rasters <- stack(list.files(getwd( ),pattern = "img$", full.names = FALSE))** more efficient

process if all rasters

are in the working

directory, contain only

those rasters to be used

in creating the stack

and are of the same

format. The pattern

argument can be used to

specify input raster

file extension

**data <- data.frame( as.data.frame(spatialobject)[c("pedon\_id", "taxonname", "x\_std", "y\_std", "tax\_subgroup")], extract(rasterstack, spatialobject))** extracts raster

data to point data. In

this example, the output

dataframe object "data"

will contain the

following columns:

pedon\_id, taxonname,

x\_std, y\_std,

tax\_subgroup and

extracted raster values

for each row in the

spatial object for each

raster in the raster

stack

**data2 <- data.frame(spatialobject$column, extract(rasterstack, spatialobject))**

the data2 dataframe will

contain extracted raster

values and the single

column specified from

the spatial object

**writeRaster(r, filename = "C:/workspace/raster.tif", format = "GTiff", progress = "text", overwrite = TRUE)**

exports raster using the

raster package

**projectRaster(from, to, method = " ")** projects the values of a raster

object to a new raster

object with another

projection

## Vector

**readOGR(dsn = "C:/workspace/polygon.shp", layer = "polygon")** imports vector data

using the rgdal package

**writeOGR(pol, dsn = "C:/workspace/polygon.shp", layer = "polygon", driver = "ESRI Shapefile", overwrite\_layer = TRUE)**

exports vector data to ESRI

shapefile using the

rgdal package

# soilDB and AQP

**fetchNASISLabData()** fetches KSSL

laboratory pedon/horizon

layer data from a local

NASIS database

**fetchKSSL()**fetches KSSL data via

BBOX, MLRA, or series

Name query, from the

SoilWeb system: [**KSSL data demo**](https://r-forge.r-project.org/scm/viewvc.php/%2acheckout%2a/docs/soilDB/NASIS-component-data.html?root=aqp)

**fetchOSD()** fetches a limited

subset of horizon- and

site-level attributes for

named soil series from

the SoilWeb system

**fetchPedonPC()** fetches commonly

used site/horizon data

from a PedonPC v.5

database

**get(‘sites.missing.pedons’, envir=soilDB.env)** returns user

site ID’s for sites

missing pedons

**get(‘dup.pedon.ids’, envir=soilDB.env)** returns pedon

ID’s for sites with

duplicate pedon ID’s

**get(‘bad.pedon.ids’, envir=soilDB.env)** returns user

pedon ID’s for pedons with

inconsistent horizon depths

**get(‘bad.horizons’, envir=soilDB.env)** returns a

dataframe of horizon-

level information for

pedons with inconsistent

horizon depths

**get\_extended\_data\_from\_NASIS\_db( )**

extracts accessory tables

and summaries from a

local NASIS Database

## Functions Specific to a Soil Profile Collection Object

*Note: In the examples below, SPC will be used to represent a soil profile collection object.*

**siteNames(SPC)** returns a list

of the site column names

in the soil profile

collection object

**horizonNames(SPC)** returns a

list of the horizon

column names in the soil

profile collection

object

**s <- site(SPC**) creates an

object called s that

contains the site info

from a soil profile

collection object

**h <- horizons(SPC)** creates an

object called h that

contains the horizon

information from a soil

profile collection

object

**x <- as(SPC, 'data.frame')**

converts SPC into a data

frame called x

**?plotSPC** invokes a help

document containing a

detailed list of

arguments and examples

for plotting soil

profile collections

**x <- site(SPC)[, c('site\_id', 'x\_std', 'y\_std')]** creates an

object called x that

contains only the

site\_id, x\_std, and

y\_std columns out of the

site table

**idx <- grep('lithic', SPC$tax\_subgroup, invert = FALSE)**

would filter and create an

index, apply that index

to the SPC, and plot the

first 10 pedons in the

newly created index

object. invert = TRUE

would pattern match for

everything except for

lithic

**lithic.subgroup <- SPC[idx,** ]

saves the idx subset of

'lithic' soils for later

use

**sort(table(f$part\_size\_class[idx]), decreasing = TRUE)** uses the

index directly to

summarize a field

**plot(lithic.subgroup[1:10, ], label = 'site\_id')** plots first

10 profiles in the

lithic.subgroup index

labeled by site\_id

**findBtHorizons <- function(i) {**

**h <- horizons(i)**

extract horizons for

current profile

**idx <- grep('t', h$hzname)**

search for pattern 't' in

horizon designations

**h2 <- h[idx, ]**

subset h by the idx criteria

**res <- h2[, c('peiid', 'phiid',**

**'hzname', 'hzdept', 'hzdepb',**

**'clay', 'phfield')]**

pattern match

for "t" in the hzname

table and create a

dataframe of pedons with

hzname "t" with the

peiid, phiid, hzname,

hzdept, hzdepb, clay,

and phfield columns

**return(res)}**

“i” is a temporary object created inside of the context of this function based on the argument supplied; in the example above, the argument ‘i’ represents a single soil profile.

**l <- profileApply(SPC, FUN = findBtHorizons, simplify = FALSE)**

take the above function

and apply it to all

profiles in a soil

**Bt.horizons <- ldply(l)**using

Plyr package convert

list into a dataframe

**Bt.horizons.top <- ddply(Bt.horizons, 'peiid', summarise, depth\_to\_argillic\_cm = min (hzdept))** create a new

object called

Bt.horizons.top that

would contain a summary

of the minimum top depth

of the argillic horizon

for each peiid

**site(SPC) <- Bt.horizons.top**

rejoins the

Bt.horizons.top

dataframe to the site

data in the SPC using

the peiid

**slab( )** aggregate soil

properties along user-

defined 'slabs', and

optionally within groups

***EX: slab(SPC, fm = peiid ~ clay +***

***sand, slab.structure =***

***c(25,100), slab.fun = mean,***

***na.rm = TRUE)*** *create a table*

*summarizing the mean*

*clay and mean of sand*

*content for each peiid*

*between 25 and 100 cm*

In the sharpshootR package, the diagnosticPropertyPlot command can be used to generate a graphical description of the presence/absence of soil diagnostic properties.

**diagnosticPropertyPlot(f, v, k, grid.label = 'pedon\_id', dend.label = 'pedon\_id')**

where:

f = SoilProfileCollection object

v = a character vector

of site-level attribute

names that are Boolean

(e.g. TRUE/FALSE) data

k = an integer, number

of groups to highlight

grid.label = name of a

site-level attribute

(usually unique)

annotating the y-axis of

the grid dend.label = name

of a site-level attribute

(usually unique)

annotating dendrogram

terminal leaves

***EX: v <- c('ochric.epipedon', 'cambic.horizon', 'argillic.horizon', 'paralithic.contact', 'lithic.contact')***

***diagnosticPropertyPlot(***

***SPC, v, k = 5, grid.label = 'site\_id', dend.label = 'taxonname', sort.vars = FALSE)***

*generates diagnostic*

*property diagram*