**Using R to Analyze Soil Data: Getting Started**

Skye Wills – 12/28/18

Prepared for the Soil Technology – Measurement and Data Evaluation Course

Updated for Soil Correlation

Updated for Statistics for Soil Survey Course

Data and files for this exercise: <https://www.cloudvault.usda.gov/index.php/s/Pc3DNHEldS3EH3Q>

**Why R**

* R is free
* R is flexible
  + R is not just a statistics package, it’s a language.
  + R is designed to operate the way that problems are thought about.
  + R is both flexible and powerful.
  + Developers can work and share info
* R is reproducible
  + R scripts explicitly record error correction and data analysis
  + Analysis records the workflow – including decisions made
* R is the leading tool for data analysis
  + Graphics and Visualization
  + Powerful, cutting edge analytics
    - ‘packages’ are written for specific groups of users
    - Analysis can be automated

R is a statistical programming language. It is a dialect of the S language and is case sensitive. The R interface allows you to enter data and execute functions using a command prompt (>).

Packages are collections of code that run specific functions. They often include example data that can be used when executing those functions. While R comes with some standard, basic statistical functions; most work will require you to add additional packages. The packages are first installed and then stored in your library, where you can reference them (this does not require administrative privileges).

R is pushed to all computers with NASIS installed. Performance is improved if settings are adjusted to save packages on your desktop (not a shared drive). As new versions are CCE approved, they should be automatically updated on your machine. This is typically 1 – 3 versions behind the latest available for public download. While this can cause error messages, it rarely creates problems for the most commonly used packages.

Resources:

<http://www.burns-stat.com/documents/tutorials/why-use-the-r-language/>

**Use R**

**1. Introduction and Background**

There are many great technical guides (printed and online) to using R. They often assume that the user has a basic understanding of working with command driven programs (through scripts or codes), data preparation, and basic statistical analysis. The learning curve can be steep as you begin to use R; but once you get started; it’s easy to expand your skills into more sophisticated analyses.

Skye’s Favorite websites:

To get started….

CRAN: <http://cran.r-project.org/>

Quick R: <http://www.statmethods.net/>

Burns Statistics: <http://www.burns-stat.com/documents/tutorials/impatient-r/>

Gardener’s own: <http://www.gardenersown.co.uk/Education/Lectures/R/index.htm#inputting_data>

For (slightly) more advanced analyses….

CA Soil Resource Lab (R page): <http://casoilresource.lawr.ucdavis.edu/drupal/node/100>

Cookbook for R: <http://wiki.stdout.org/rcookbook/>

R Tutorial: <http://www.r-tutor.com/>

Stack Overflow (discussion forum that includes R questions): <http://stackoverflow.com/>

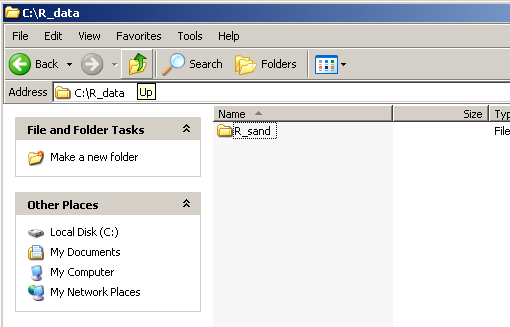
There are many forums, blogs and websites with examples, questions and answers about specific analyses and problems in R. Use search engines to your advantage – someone has nearly always tackled a problem before you – their script can be modified to fit your needs.

**For this exercise**, actions you are meant to take are in blue and specific R commands are in red. While example files (names in brown) are provided for periodic points through the exercise; it is expected that you will complete the entire exercise at your own pace and save your own versions of the files. Some portions of the instructions are repeated so that you can easily stop and restart the exercise at section breaks.

* 1. **Getting Started**

To complete this exercise it is useful to create a folder to store your R related materials.

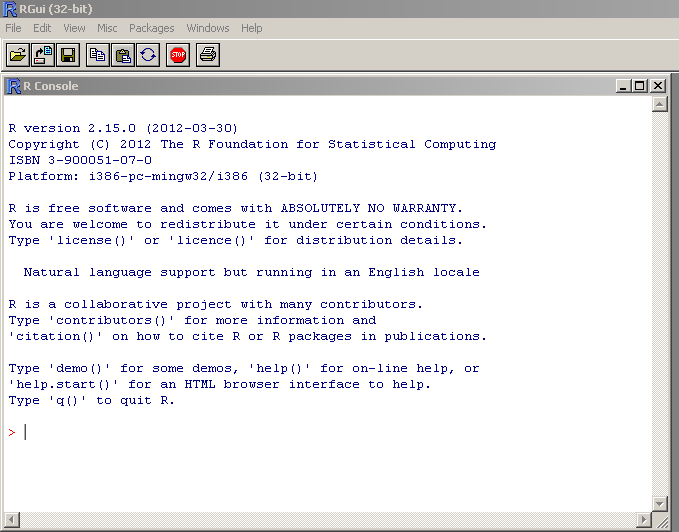
Create Folder to keep R stuff - C:\R\_data



1.2 Open R

Navigate to R (in the start and program menus) and Open R (3.4.0, or latest version)

This will open the R GUI console. *GUI – graphical user interface*



Command prompt

A note about working with R and Microsoft products: Microsoft puts some unusual formatting on text that can interfere with R commands (fancy “quotes”, for example). It is best not to copy and paste from Word to R. For this exercise, type directly into R. A later section will introduce you to ways to save and edit R commands.

1.3 Set up the R workspace

R will do mathematical calculations for you.

>log (100)

Calculates the natural log of 100

>log10(100)

Calculates a base 10 log of 100.

To use R, you type commands next to the command prompt ‘>’.

Find out what the current working directory is by typing ‘getwd()’ next to the command prompt.

> getwd()

Now change the working directory to the new folder you setup, C:\R\_data.

Try to copy and paste the path from windows explorer inside of quotations.

> setwd(“C:\R\_data”)

You will get an error that looks like this:

Error: '\R' is an unrecognized escape in character string starting ""C:\R"

**R does not treat ‘\’ as a separator.**

You can fix this error by placing your cursor next to the bottom command prompt ‘>’ and hitting the up arrow on your keyboard. Your last entry should appear.

> setwd("C:\R\_data")

If you press the up arrow again, the previous command will appear at the command prompt. You can use this function to move back through previous entries and redo or edit and redo them.

To fix this error, change the command so that it contains either double backslashes or single forward slashes.

> setwd("C:\\R\_data")

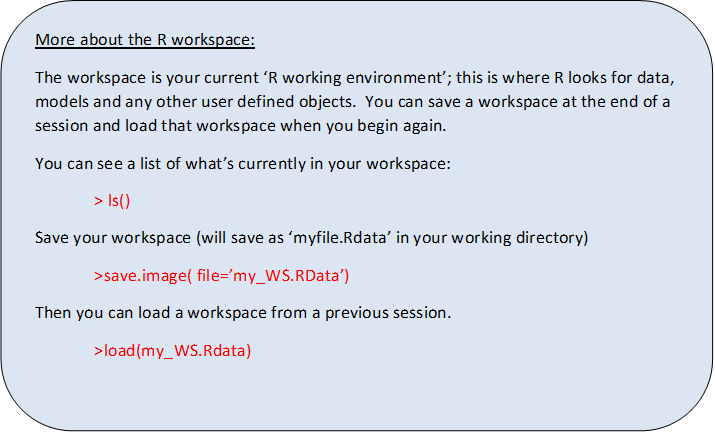
Or

> setwd("C:/R\_data")

Now the getwd() command should return the same path.



For more practice using commands in R, see Appendix A of ‘An Introduction to R’



**2. Preparing Data to Graph and Analyze**

When preparing your data for statistical analysis, a nicely formatted summary table is not appropriate. For entry into R (or any other stats package), your data needs to be basic and compact. We will typically (at least in the beginning) use dataframes in R, where columns contain different properties or variables and rows contain individual observations of that variable. This is the way you should arrange your data for statistical analysis.

*Sand content example*

You might collect or present your data like this:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| % Total Sand | | | | | | |
|  | **Cropland** | | **Rangeland** | | **Pastureland** | |
| **Old farm** | Ap | 19 | Ap | 23 | Ap | 22 |
| Bt | 21 | Bt | 34 | Bt | 23 |
| **City land** | Ap | 31 | Ap | 30 | Ap | 25 |
| Btk | 35 | Bt | 36 | Bt | 29 |
| **Out west** | A | 27 | A | 21 | A | 23 |
| Bt | 25 | Bw | 26 | Bw | 24 |

The data needs to be organized with total sand content as one long column – with headers for organization. Notice that spaces have been removed from column headers (required) and everything is now in lower case and abbreviated (not required, but it simplifies typing). There should be only 1 header row followed by data.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| location | landuse | master | depth | Sand |
| city | crop | A | 14 | 19 |
| city | crop | B | 25 | 21 |
| city | pasture | A | 10 | 23 |
| city | pasture | B | 27 | 34 |
| city | range | A | 15 | 22 |
| city | range | B | 23 | 23 |
| farm | crop | A | 12 | 31 |
| farm | crop | B | 31 | 35 |
| farm | pasture | A | 17 | 30 |
| farm | pasture | B | 26 | 36 |
| farm | range | A | 15 | 25 |
| farm | range | B | 24 | 29 |
| west | crop | A | 13 | 27 |
| west | crop | B | 29 | 25 |
| west | pasture | A | 11 | 21 |
| west | pasture | B | 31 | 26 |
| west | range | A | 14 | 23 |
| west | range | B | 24 | 24 |

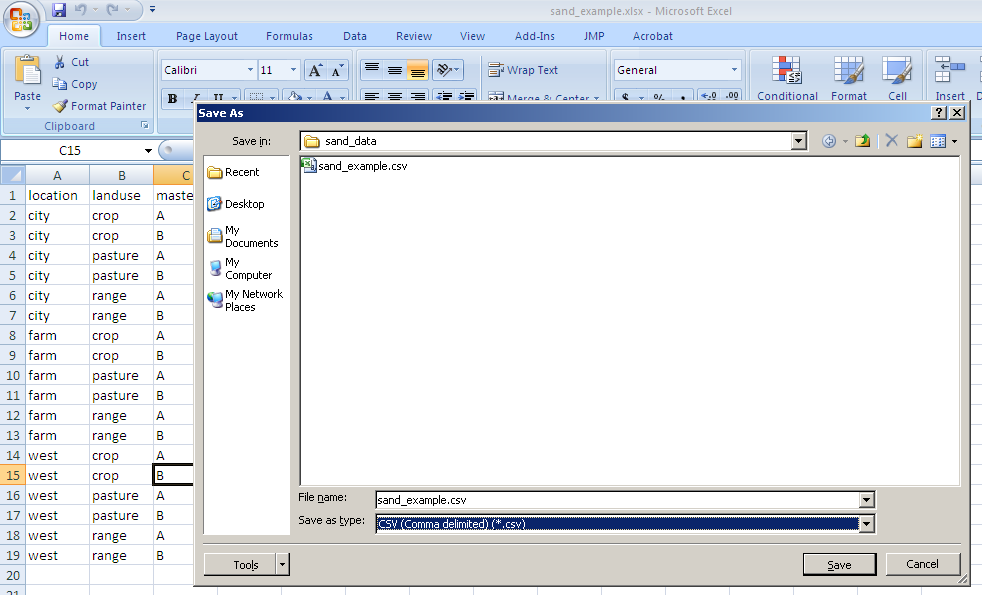
Create a new file in excel example\_sand.xlsx

This is an acceptable format; but it’s what is known as a WIDE layout. There are ways to make this data even longer using packages such as tidyr and dplyr. The benefits of this approach aren’t apparent until you have larger datasets and want to repeat a lot of commands. We will leave this as is for now, but some analyses are more efficient in longer formats (see <http://www.datacarpentry.org/R-ecology-lesson/03-dplyr.html> for more information).

It is important that you record and organize the meta-data (data about the data) for your project. You can do this in on excel work book with multiple worksheets (one for data, one for notes etc.) Alternatively, you can keep all the relevant files in a folder (csv or other data file, documents of project plan, R files etc.). Metadata should include:

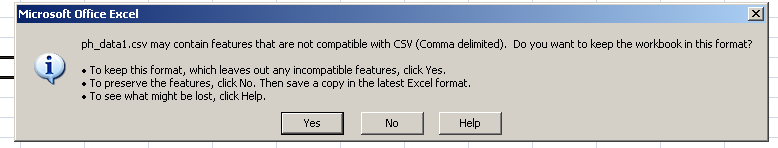
* notes about the project; where were the samples collected, by whom, pedon descriptions or user pedon ids
* explanation of the headers used; what method was used to determine sand content, how were master horizons determined, what definition was used for each land use
* analysis; summary tables, the location of any R files and notes on which commands were used and why

Then save data as a csv file (sand\_example.csv). To do this, make the data worksheet in the excel workbook active (it should be visible, by clicking the tab at the bottom of the window). Then go to the File menu at the upper left hand side and choose ‘Save as’. In the ‘Save As’ window, use the arrow to select ‘CSV (Comma delimited)(\*.csv)’



Close the .csv file. example\_sand.csv

Excel will ask you if you want to save the changes (click yes) and then it will warn you that some features may not save (click yes to save the file as csv).



While R can import excel files; it requires that a package be added to your library. Furthermore, any formatting or formulas in the excel file can cause the program to incorrectly read headers and data. It is best to use a file format that is simpler and easier for the program to interpret.

Comma separated files (csv’s) are a good way to store data and load it into R (and other programs). You can look at the file in the program notepad (Start-> Accessories->Notepad) and quickly detect that there is a single row for headers followed by data. CSV files are also small in size and easy to share.

**3. Import Data into R**

The basic command for importing data into R is ‘read.table’. The command is followed by the location of the file (remember to use / instead of \) and then some instructions for how to read the file. To create a dataset in R named ‘sand’ enter the command to import your previously saved data (remember to consider which case you’re using). read.csv() is specific to csv files and assumes the sep = “,”.

> sand <- read.table("C:/R\_data/sand\_example.csv", header = TRUE, sep=",")

header = TRUE – indicates that the first line contains the column headers

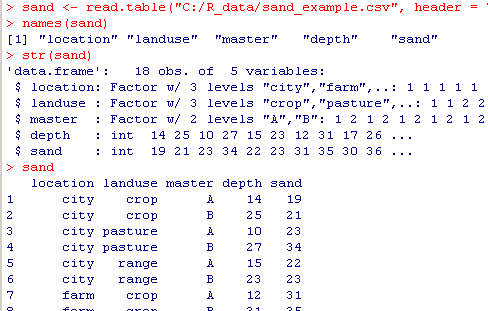
sep=’,’ – indicates that commas are used to delimit or separate data elements

There are other arguments you might want to use.

>help(read.table)

The help command will give you information about importing data including the full range of arguments you can use. (you can also try ??read.table to do a search for help on the command)

I like to use a command that returns the names (headers) of the data columns to check that I’ve imported the correct file and remind myself what the headers are. You can also use a display structure command to show you the structure of the data object. You can also enter the name of the table next to the command prompt to return the entire table; avoid this if your table is large.



**4. Install and Load Packages**

There are two ways to manage packages in R; with commands and through the GUI interface.

**R Packages**

Packages are collections of R functions, data and compiled code that are well-defined and referenced. While the standard R installation contains some basic statistics and graphics functions through a standard set of packages; you will likely want to expand your options using other packages. There are packages specifically for analysis of spatial maps and statistics, multi-panel graphs and even for analyzing soil profiles and transects.

4.1 Using Command Prompt to Manage Packages

Find out which packages have been installed:

>library()

Install a package you do not have currently downloaded. This will download the package so it is available to use. We will install the maps package. This package creates some nice base maps.

> install.packages("maps", dep=TRUE)

It will ask you to select a ‘CRAN mirror’ – this is a URL at a physical location that will be used to transmit data to you. The Comprehensive R Archive Network (CRAN) is a collection of sites that carry identical material for R. It is best to choose one that is close and reliable - USA.

USA(KS) or USA(IA) are good matches for Nebraska.

Select one and click OK at the bottom of the box.

To use the package, we must add it to our current library.

>library(maps)

To find more documentation about the maps package; request more information from R.

>??maps

This will send you to a webpage. We are interested in the maps:map documentation. At that website, you see the documentation about that function. There are a lot of options, but we’ll focus on the basics.

Usage (simple form): map(database, regions)

This means the command is “map” which will be followed by specific instructions (called arguments). In this case:

|  |  |
| --- | --- |
| database | character string naming a geographical database, or a list of x, y, and names obtained from a previous call to map. The string choices include a [world](http://127.0.0.1:24451/library/maps/help/world) map, three USA databases ([usa](http://127.0.0.1:24451/library/maps/help/usa), [state](http://127.0.0.1:24451/library/maps/help/state), [county](http://127.0.0.1:24451/library/maps/help/county)), and more (see the package index). The location of the map databases may be overridden by setting the R\_MAP\_DATA\_DIR environment variable. See [world](http://127.0.0.1:24451/library/maps/help/world) for further details. |
| regions | character vector that names the polygons to draw. Each database is composed of a collection of polygons, and each polygon has a unique name. When a region is composed of more than one polygon, the individual polygons have the name of the region, followed by a colon and a qualifier, as in michigan:north and michigan:south. Each element of regions is matched against the polygon names in the database and, according to exact, a subset is selected for drawing. The default selects all polygons in the database. |

Now we can call the map function from the maps package.

>map(‘usa’)

>map(‘state’)

When the region is left out, it defaults to showing all regions. We can specify a specific region.

>map(‘county’, ‘kansas’)

>map('county', region=c('Iowa', 'Illinois', 'Missouri'))

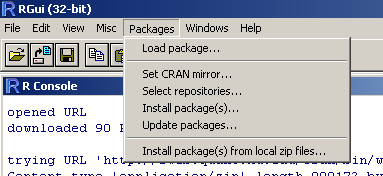
Now try your home state.

Try some of the examples included at the end of the map {maps} documentation (from the previous search or <http://cran.r-project.org/web/packages/maps/maps.pdf>)

4.2 Using the GUI to Manage Packages

Open R GUI (the R console should be active, if not, click in the window with the command prompt).

Navigate to the ‘Packages’ menu – select ‘Install Packages’



If you haven’t already done so in your current session; It will ask for a ‘CRAN mirror’ – this is a physical location that will be used to transmit data to you. It is best to choose one that is close and reliable - USA.

Then it will ask you to select packages (you can select more than one by holding down the cntrl key).

Scroll down, select ‘Rcmdr’ and Click OK at the bottom of the box.

If you have previously installed Rcmdr; you may get an error. The command

>detach(Rcmdr)

will remove it from your library and allow you to install it again.

You will see information move through the console. When the command prompt ‘>’appears at the bottom of the screen the process is complete. Once the package has downloaded, you need to load the package.

Navigate to ‘Packages’ menu again – select Load Package

Select Rcmdr and click OK (it may ask you to install additional packages, select the buttons for CRAN and OK)

You can also use the GUI interface to save your workspace under the file menu.

Rcmdr should open as a new window within R.

**5. Working with R scripts**

One key benefit of using R is that you can save scripts (collections of commands) that can be used to recreate or share you analyses. Be sure to update your csv file with any data corrections or additions so that your script always uses the most current data.

5.1 Create an R script

With the R console active, Navigate to the File menu and select ‘New script’

This will open a window called ‘R editor’.

Enter the command to import the sand data that you previously entered.

sand <- read.table("C:/R\_data/sand\_example.csv", header = TRUE, sep=",")

Enter the command to view the headers.

names(sand)

Add a command for a simple plot to the editor window:

plot(depth~sand, data = sand)

To execute, select the all lines and right click –‘Run line or selection’

This will return information about ‘sand’ to the R console, and open a window with a simple plot where depth is on the y axis and sand is on the y axis.

Navigate to the File menu and select ‘Save as’. Enter the name ‘import\_sand’. This window will be saved as with an ‘.R’ extension. import\_sand.R

You can open this file or another in a later R session.

5.2 Using a Saved R script

You may want to use a script you’ve previously saved or one provided to you by a colleague. The benefit of rerunning the script each time you need the analysis is that the results will be updated to match any changes (corrections, additions etc.) in your data. Keeping a copy with your data ensures that you can recreate your work. With experience, you can customize scripts from others to do the analyses you need.

Open/activate the R GUI console

Navigate to the File menu

Select ‘Open script’

Use windows explorer to navigate to the previously provided created or use another script provided with the tutorial (also available online <https://www.cloudvault.usda.gov/index.php/s/z6jdj5dVpMniNr3>

more\_sand.R

A new window will open up within R – called ‘R editor’.

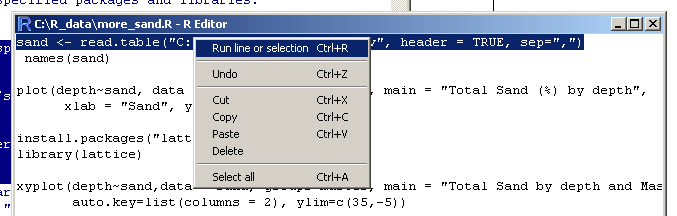
When using someone else’s script ensure that the path to the data is correct and that any other changes are made before running.

Change the data import step to the correct path on your computer.

sand <- read.table("C:/XXXXX/sand\_example.csv", header = TRUE, sep=",")

Place your cursor in the first line.

Then right click with your mouse and select ‘Run line or section’

******

This is a data input step – next time you open R, you will need to import the dataset again. If you update the file sand\_example.csv, the changes will be reflected when you rerun the analysis.

Select the next line, use Ctrl+R (or right click and select ‘run’)

plot(depth~sand, data = sand, ylim =c(50,-1), main = "Total Sand (%) by depth",

xlab = "Sand", ylab = "Depth")

This recreates the plot from the last exercise with arguments that alter the y-axis and labels.

How do you read more about the options (arguments available)?

>help(plot)

Now, install and load a library that will allow us to execute a more complicated plot.

install.packages("lattice", dep=TRUE)

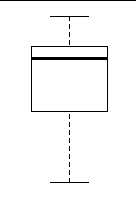
library(lattice)

and run the command for a new type of plot. Note that this plot has the same initial elements as the previous plot with the addition of a grouping variable (master) and a legend (auto.key=list…)

xyplot(depth~sand,data = sand, groups=master, main = "Total Sand by depth and Master", auto.key=list(columns = 2), ylim=c(35,-5))

*>boxplot(property ~ group)*

*Boxplots (or box and whisker graphs) are my favorite kind of graphs. They can show the distribution of an entire variable/property, or divide it by group. The box of a boxplot extends from the 1st quartile (25th quantile) to the 3rd quartile (75th quantile). The dark line indicates the median (2nd quartile or 50th quantile). The default setting for whiskers extend past the box - 1.5 x IQR.*

**

*IQR (interquaretile range) is the difference between the 1st and 3rd quartile.*

Next, select the lines for any command that you wish to execute, for instance:

**Creates a histogram.**

of sand:

hist(sand$sand)

**or depth:**

hist(sand$depth)

**The ‘$’ is used to indicate a particular column within the dataset (dataset$column). To get more information about how to use this function, including its usage and arguments that you can use to modify the default output;** use the help or ?? command**.**

**We can edit the script in R editor and save our changes for later.**

Add the following command:

hist(sand$sand, scale="frequency", breaks="Sturges", col="lightblue", xlab = "Total Sand %")

**In this example, we’ve changed the label of the x-axis and the color of the graph.**

Now try the boxplot function:

boxplot(sand~landuse, data = sand)

Notice that the range boxplot has a single circle on the graph above it. This indicates an outlier, or a value that is more than 1.5 x IQR. You should evaluate this data point to ensure that the number measured and entered is correct.

Now try the more complicated lattice version:

bwplot(sand~master|landuse, data=sand)

**See the full explanation of a boxplot to the right.**

**Adding notes to an R script**

Any text that you do not want R to act on (such as instructions, information, notes or comments) is preceded by a “#”. R will ignore the remainder of the line.

**You can add an explanation about your data or script. At the top of the R editor window enter:**

#this is a demo of how to use R editor

**Now select all and run the entire script. The additional text doesn’t affect the output.**

Save your R script using the file icon or in the File menu as more\_sand.R

Close R

It will ask you to save a workspace image, if you select yes, R can load the settings from your current session next time it is opened (such as your working directory).

5.3 Export Graphs

Open the R script you just saved more\_sand.R.

Edit the script to create multiple graph windows and export those graphs. (If you are starting a new session and have not saved a new version of more\_sand.R; you will need to edit the path for the read.table command)

Insert the command to create a new window before each graph.

windows()

Each graph will now be created in a new window instead of overwriting the previous one.

You can also direct graphic output to be saved in one of many formats.

|  |  |
| --- | --- |
| Function | Output to |
| pdf("graph.pdf") | pdf file |
| win.metafile("graph.wmf") | windows metafile |
| png("graph.png") | png file |
| jpeg("graph.jpg") | jpeg file |
| bmp("graph.bmp") | bmp file |
| postscript("graph.ps") | postscript file |

In the script it looks like this:

jpeg(‘graph.jpg’)

plot(x,y)

dev.off

The graph of plot(x,y) will be saved as a jpeg in your working directory. You can also enter the explicit path to save it in another place. Insert the name of the file you’re creating in place of ‘graph’.

Enter the following commands at the end of the R script.

png (‘sand\_boxplot.png’)

bwplot(sand~location|landuse, data=sand)

dev.off

Can you export other graphs from the exercise? In other file formats?

final\_sand.R

sand\_boxplot.png

depth\_histogrm.pdf

**6. R Commander**

R Commander (Rcmdr) is an expanded GUI that will allow you to point and click to import data and do basic analyses. It was created for students in introductory statistics courses, so they could see how the software worked without learning a large number of commands. It is only appropriate for simple, standard graphs and statistics.

In the R console, call Rcmdr from the library.

>library(Rcmdr)

This should open the R Commander window. If it is not visible, select it from you toolbar (it may be in a list with other R windows).

If you have not previously installed R Commander use

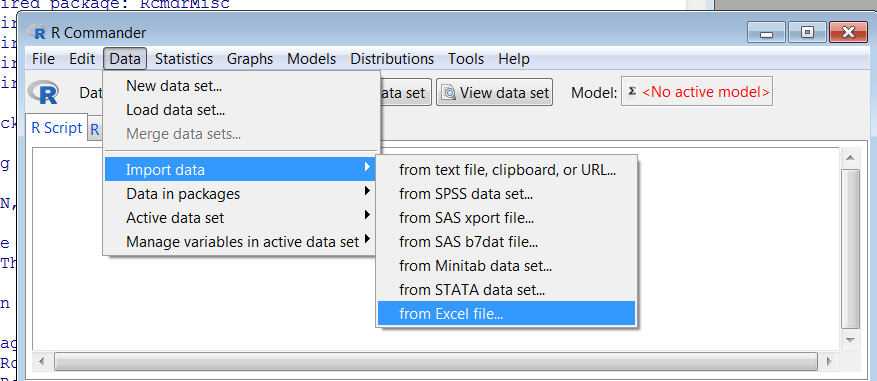
>install.packages("Rcmdr",dep=TRUE)

then

>library(Rcmdr)

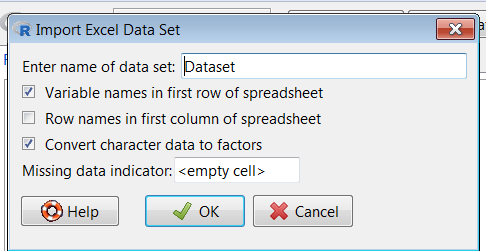
Now we will use the Rcmdr GUI to import data into R.

Navigate to the Data toolbar – scroll down to ‘Import data’ and then select ‘ from Excel, Access or dBase dataset…’



Name dataset.

Replace ‘Dataset’ with ‘sand2’ and select OK



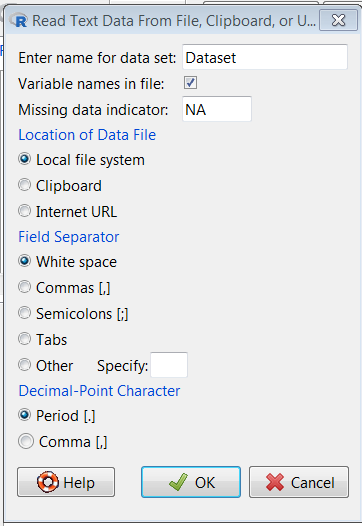
Navigate to file location using window explorer,

select proper excel file

select proper worksheet.

Alternate way to do it - text and csv files are smaller and easily transferable

Navigate menus and select: Data – Import data – from text file, clipboard or url….



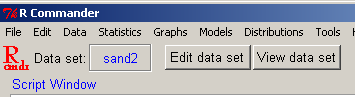
Complete the import box: name – change to ‘sand2’, check the box ‘variable names in file, location = local file system, field separator = commas, decimal-point character = period

Pay attention to the field separator (csv’s use commas); decimal point (Americans typically use periods) and variable names in file (checking this box means that the header names will be included in the dataframe)

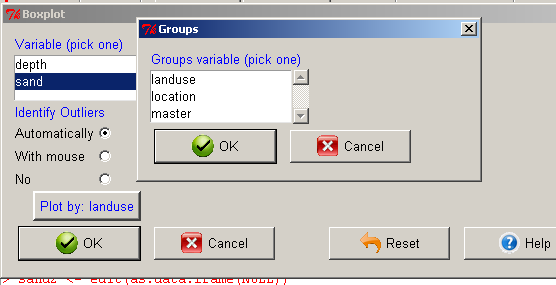
Navigate to csv file location using Windows explorer.

Now use the R Commander GUI functions to:

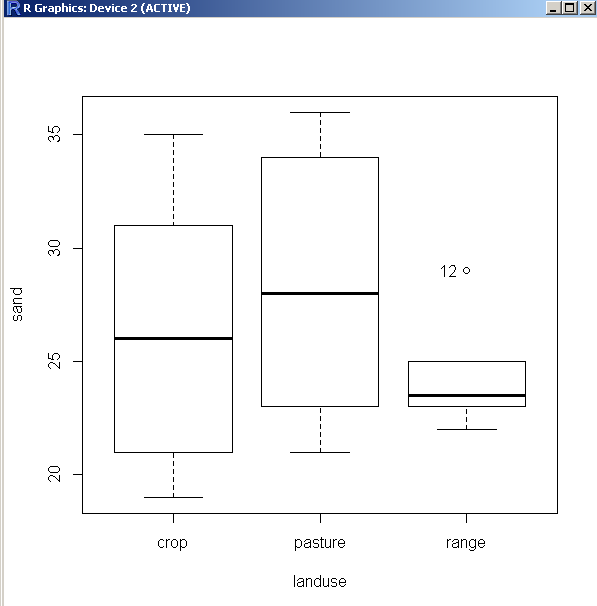
* View Dataset – confirm that data imported with 3 columns

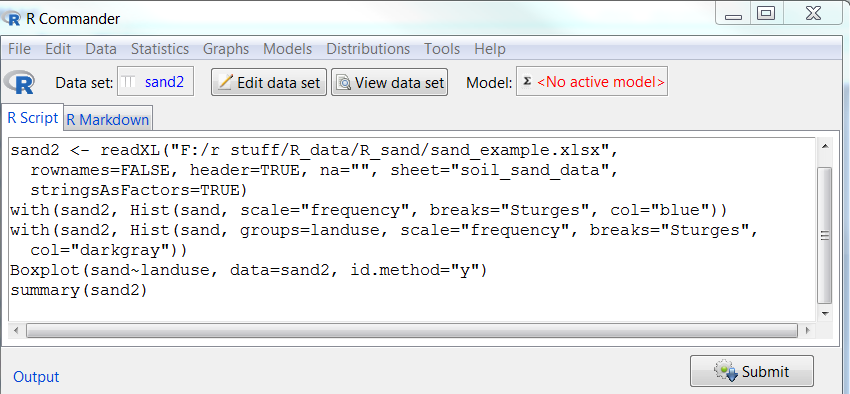


* Create graphs
  + Navigate Graphs menu
    - Select – Histogram…
      * Variable – sand
      * Number of bins – use <auto> first time, edit later
      * Click OK
      * Plot will appear in a separate window within the R GUI.
  + From the R Commander toolbar select the Graphs menu again
    - Select – Boxplot…
      * Variable – sand
      * Select ‘Plot by Groups’ button
        + Group variable – choose ‘landuse’
      * Options - Identify outliers – ‘automatically’
      * Select ‘OK’ button
      * Plot will appear in a separate window within the R GUI

****

*Boxplot output:*

**



If you want to change something slightly, edit the command line in the script window, select the entire command and hit ‘Submit’. Notice that commands in the R Commander script window are not preceded by ‘>’.

Use R help – or use an online search to find information about the desired function.

For instance, if you would like to change the colors of the bars in the histogram.

Edit the ‘col’ command:

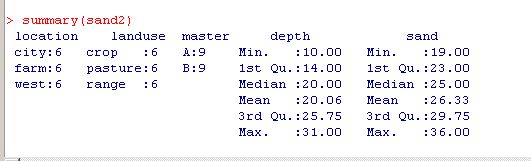
with(sand2, Hist(sand, scale="frequency", breaks="Sturges", col="blue")

Select the line

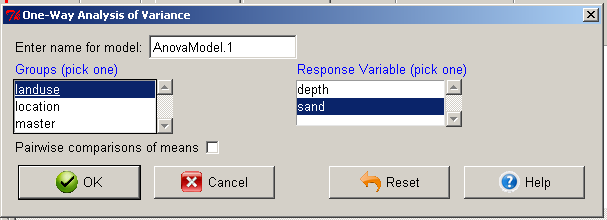
Hit ‘Submit’ button

To calculate basic summary statistics use the options in the Statistics menu. Here are a couple of examples of things you might want to do.

* Navigate to R Commander Statistics menu
  + Select summary
    - Active dataset
    - Results are returned in the ‘Output Window’
    - It gives the number of records for each categorical (name) variable and some basic measures of the continuous (numeric) variables



* + Select means
    - One-way ANOVA
      * Name – if you wish to reuse model, give it a unique name
      * Groups – landuse
      * Response Variable – sand
      * Hit OK



Example Ouput:

> summary(AnovaModel.1)

Df Sum Sq Mean Sq F value Pr(>F)

landuse 2 48 24.00 0.914 0.422

Residuals 15 394 26.27

> with(sand2, numSummary(sand, groups=landuse, statistics=c("mean", "sd"))

mean sd % data:n

crop 26.33333 6.022181 6

pasture 28.33333 6.022181 6

range 24.33333 2.503331 6

This result indicates that the sand content of these landuses are not significantly different ( Pr = 0.422). It should be noted that this simple analysis has not accounted for the two kinds of horizons analyzed (A and B) or the non-independent nature of multiple samples collected at each location. It also doesn’t tell you if comparing sand content between land uses was a reasonable thing to do.

For record keeping, you can save a script for later use in R commander or R GUI.

* To Save a script
  + Edit the ‘Script Window’ to reflect the analyses you want to recreate (including the data importation step)
  + Navigate to the R Commander File menu
    - save script as…
      * Rename sand\_rcmdr.R and save in your working directory

The same steps can be used to save the output window using ‘save ouput as’.

Close the R Commander window and move to the next section. Sand\_rcmdr.R

**6.1 Using a Saved R Commander Script**

You can open your saved R commander script in R commander or in R editor. However, sometimes scripts saved with R Commander will have things encoded for R commander that aren’t apparent and it won’t run directly from R editor. First we’ll open the script with R commander, then we’ll open it in the R editor.

6.1.1 Using a saved R commander script in R Commander

Open and view R Commander

Navigate to File, Open Script file sand\_rcmdr.R

It will ask you if you want to save the current log file (hit no to clear without saving)

Your saved script now appears in the ‘Script Window’.

Place your cursor on any command line and hit submit.

You will see the ‘output’ and graphs display as they did when you first executed them through the menu system of R Commander.

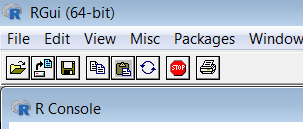
6.1.2 Using a saved R commander script in R editor

Open the RGUI console

Navigate to the File menu

Select ‘Open script’

Use windows explorer to navigate to the previously saved script sand\_rcmdr.R

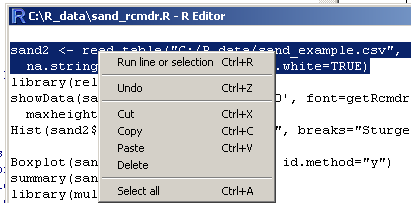


A new window will open up within R - ‘R editor’.

Select the first two lines of script using your mouse to left-click and drag.

Then right click with your mouse and select ‘Run line or section’ (you can also use the key stocke shortcut (cntr+r)

sand2 <- read.table("C:/R\_data/sand\_example.csv", header=TRUE, sep=",", na.strings="NA", dec=".", strip.white=TRUE)

******

This is a data input step – next time you open R, you will need to import the dataset again. If you update the file sand\_example.csv, the changes will be reflected when you rerun the analysis.

Select the next line and ‘run’

library(relimp, pos=4)

This loads a package that R Commander used to run the script.

Next, select the lines for any command that you wish to execute, for instance:

Hist(sand2$sand, scale="frequency", breaks="Sturges", col="blue")

**This recreates the histogram graph.**

**Enter the command**

**>help(hist)**

**to get more information about how to use this function including its usage and arguments that you can modify the default out put**

**We can edit this in R editor and save our changes for later.**

Hist(sand$sand, scale="frequency", breaks="Sturges", col="lightblue", xlab = "Total Sand")

**In this example, we’ve changed the label of the x-axis and the color of the graph.**

**At the top of the R editor window enter:**

#this is a demo of how to use R editor with an R Commander Script

Now select all and run the entire script.

Save your R script using the file icon or in the File menu.

Close R

7. R studio

*(Instructions developed on 0.96, should work on other versions, but it might look slightly different)*

R studio is a ‘super-package’ that allows you to interact with R more easily - (version 0.99.903 is CCE approved as of March, 2017).

Navigate to R Studio from your start menu.

When you open R Studio, you will see your screen split into quadrants

Source – these are script files that you have saved or are creating

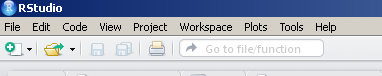
Console – this is the command prompt window for R

Workspace – keeps track of all data in use (which can be clicked and viewed through the source)

Plots – input and output space, includes files, packages and graphs that you create.

Rstudio Support has examples in the ‘Knowledge Base’ that will help you use R studio. <http://support.rstudio.org/help/kb/using-rstudio>

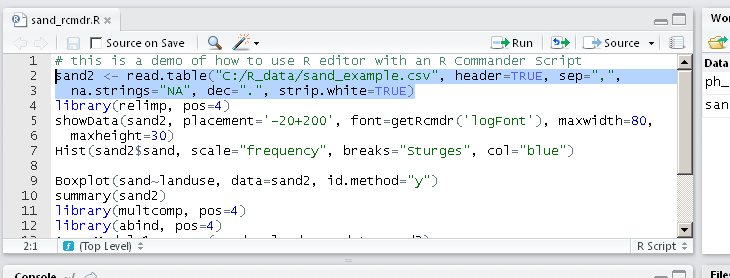
Open the script you created with R commander and edited in R editor (click the file symbol and navigate to the script)



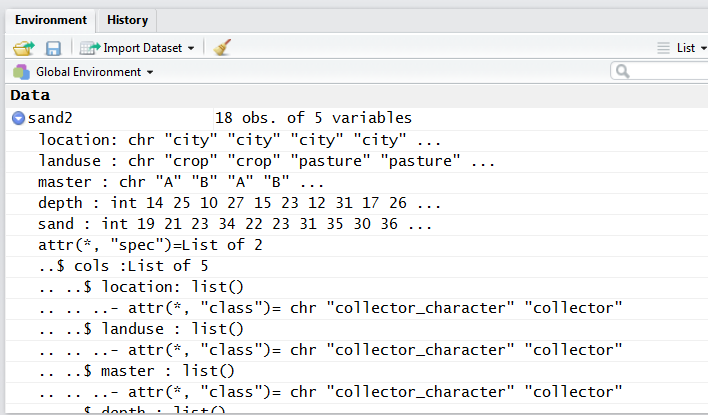
Open sand\_rcmdr.R

Run commands from within the script

Select first command (first three lines) and hit ‘run’ from the task bar above the script.

`

Notice that the command line is passed to the console (lower left) and the data file appears in the workspace (upper right) as sand2.



Click arrow next to the table name to see the columns, click the table icon once to view; notice that the console will show the corresponding command prompt ‘>view(sand2)’

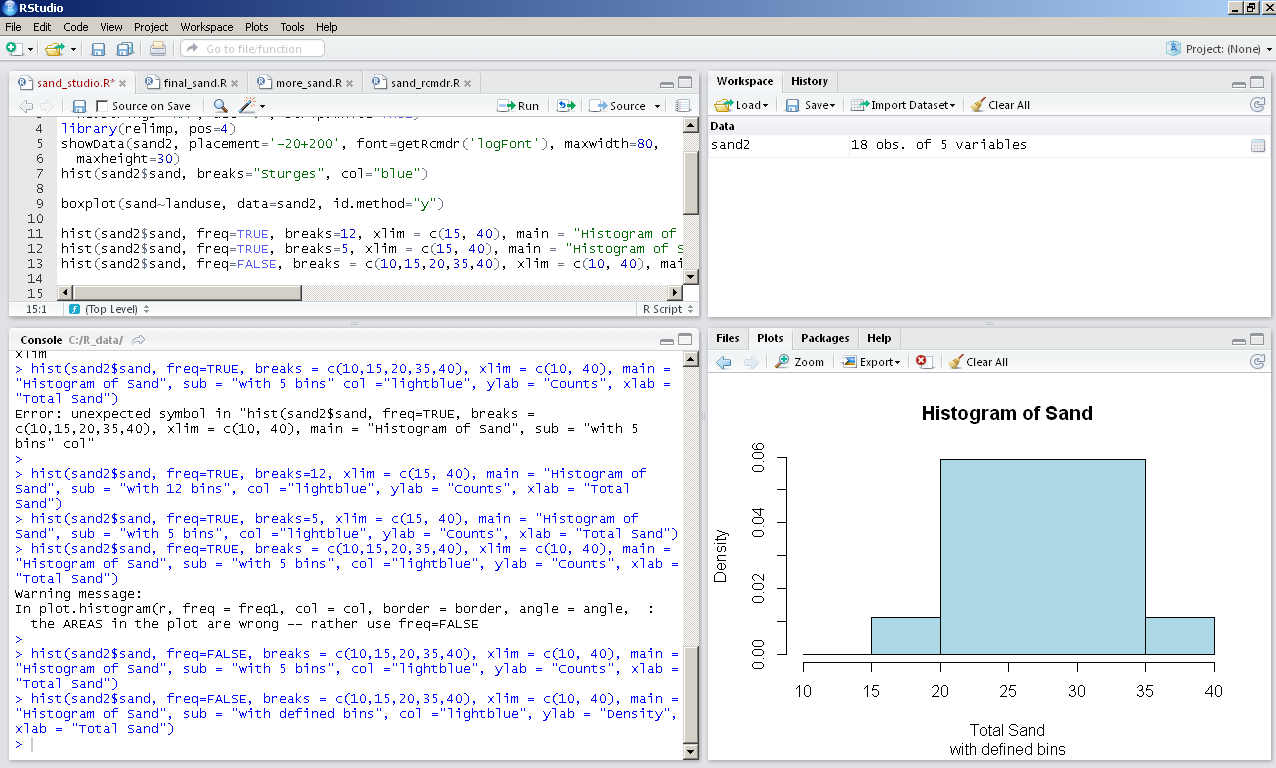
Select the command to create a histogram (begins with ‘hist’) and hit ‘run’

You will get an error message “Error: could not find function "Hist"

This is a function of the packages that loaded when you installed and loaded Rcmdr, but are not available now. Go to console, at command prompt (>) push the up arrow and edit the command to have a lower case ‘h’ as in ‘hist’ hit enter;



Now edit the ‘Hist’ command in the R script; do the same for Boxplot – to boxplot



Select both commands (hist and boxplot) and hit ‘run again

Ignore the warning message.

You will see the most recent graph in the Plot window; use the arrows to scroll through all graphs produced during this session.

Save your script. sand\_studio.R

Help features of R Studio

To learn more about the function you are using and the options/arguments available; take advantage of some of the help functions in R studio.

Place your cursor next to the command prompt (>) in the console (lower left).

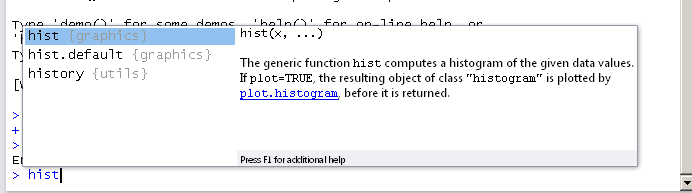
Type: > hist

Place your cursor at the end of ‘hist’ and hit tab – you’ll see a brief explanation of the functions and the name of the package it comes from {this can be handy for searching} hit the ‘F1’ key to get further explanation (equivalent to help(hist) in the console.

Look through the usage and arguments

Type: > ?hist

Reenter ‘hist’ function; evaluate the effects of changing color, breaks, freq, and labels

Try these examples: 

hist(sand2$sand, freq=TRUE, breaks=12, xlim = c(15, 40), main = "Histogram of Sand", sub = "with 12 bins", col ="lightblue", ylab = "Counts", xlab = "Total Sand")

hist(sand2$sand, freq=TRUE, breaks=5, xlim = c(15, 40), main = "Histogram of Sand", sub = "with 5 bins", col ="lightblue", ylab = "Counts", xlab = "Total Sand")

Notice how changing the ‘breaks’ argument alters the appearance of the graph. The breaks argument tells R how the individual values should be counted in bins or groups. The ‘xlim’ argument tells R where to set the upper and lower limit of the x-axis.

Now try:

hist(sand2$sand, freq=FALSE, breaks = c(10,15,20,35,40), xlim = c(10, 40), main = "Histogram of Sand", sub = "with predefined bins", col ="lightblue", ylab = "Counts", xlab = "Total Sand")

Note that this arbitrarily sets the bin breaks using a list – c(x1,x2,x3…..). This can be a good way to separate groups, but in a way that may alter the way you visualize the distribution.

This will work for any function in the console command prompt.

You can also search for help on any function (even if you don’t have the package installed

>help.search("histogram")

**We have just scratched the surface of what is possible with R and R studio.**

**There is a course:**

**Statistics for Soil Scientists -** [**http://ncss-tech.github.io/stats\_for\_soil\_survey/chapters/**](http://ncss-tech.github.io/stats_for_soil_survey/chapters/)

You can use R to:

Directly download data from NASIS or SDA – use the package soilDB

<http://ncss-tech.github.io/soilDB/docs/>

Do analysis by depth slices and produce graphs for transect visualization and analysis – use the package aqp

<http://casoilresource.lawr.ucdavis.edu/drupal/node/1020>

Evaluate Generalized Horizon Information

<http://ncss-tech.github.io/stats_for_soil_survey/chapters/4_exploratory_analysis/4_exploratory_analysis.html#8_soil_reports>

background - <http://ncss-tech.github.io/AQP/aqp/gen-hz-assignment.html>