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

Skye Wills – 1/1/20 - Updated for Statistics for Soil Survey Course

Data and files for this exercise are on the Stats for Soil Survey github page:

<https://github.com/ncss-tech/stats_for_soil_survey/tree/master/data/Pre-course>

Begin by doing the Stats for Soil Survey Pre-Course assignment:

<http://ncss-tech.github.io/stats_for_soil_survey/chapters/0_pre-class-assignment/pre-class-assignment.html>

**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://www.cookbook-r.com/>

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.

**There are errors and mistakes in this exercise – ON PURPOSE – if you can’t work through them, reach out to your mentor.**

* 1. **Getting Started**

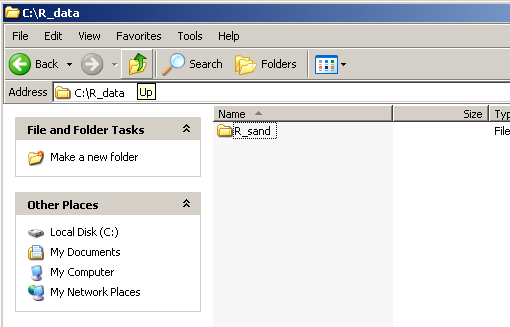
To complete this exercise, create a folder to store your R related materials.

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

this is an example you can replace all reference to C:\R-data with the path you use

Copy the contents of R\_sand into that folder

<https://github.com/ncss-tech/stats_for_soil_survey/tree/master/data/Pre-course>



1. Open R studio

*(Instructions developed on 1.1.423, 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

R studio will not work if you do not have R installed on your computer.

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

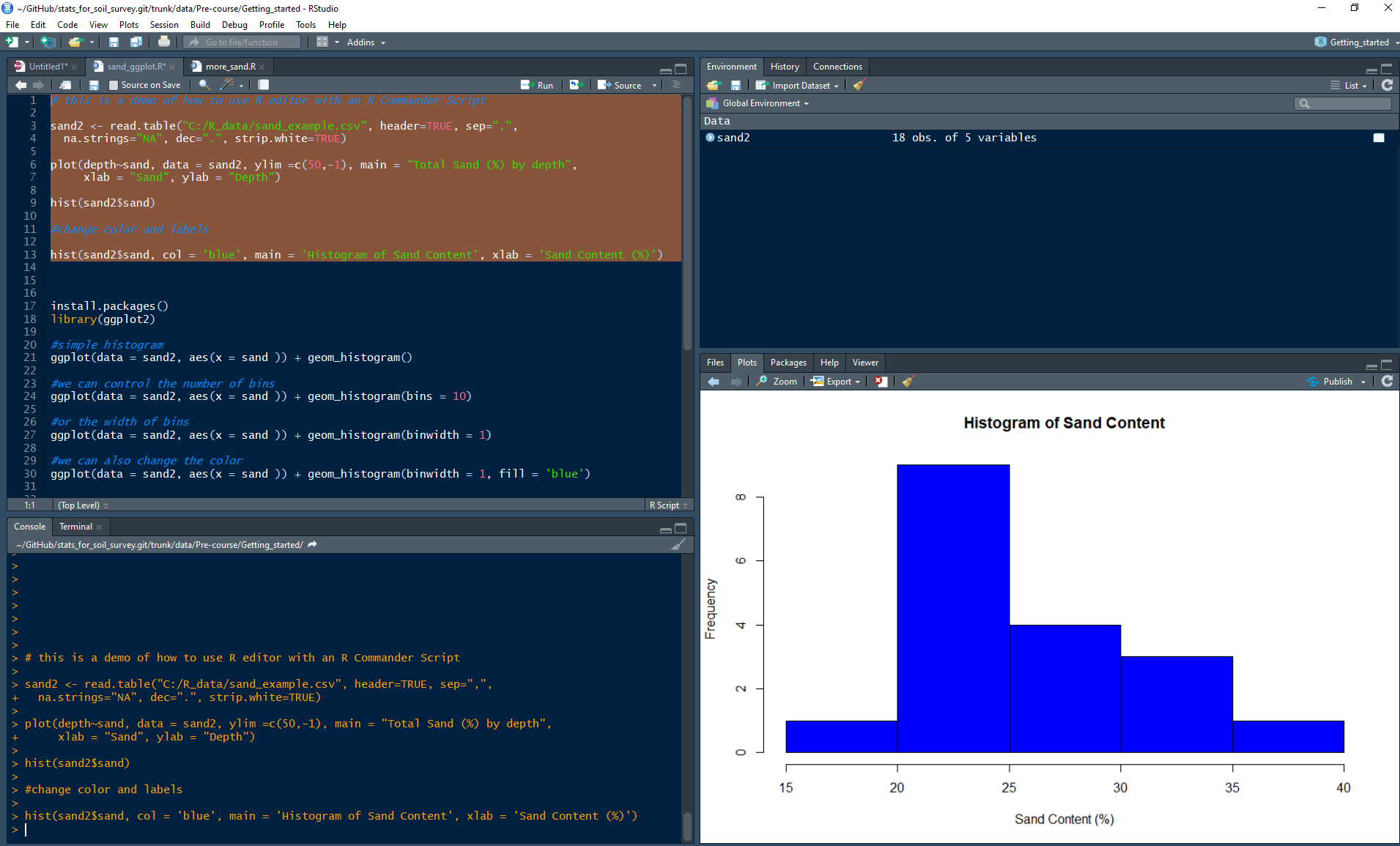
Workspace – keeps track of all data in use (which can be clicked and viewed through the source)

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

Rstudio Support has examples in the ‘resources – webinars and videos ’ that will help you use R studio. <https://resources.rstudio.com/>

<https://rstudio.com/resources/cheatsheets/>

NOTE: I’ve changed my default colors (Tools<global options<Appearance)



Console

Source

1.1 Give R a command

Navigate to the command prompt in the console

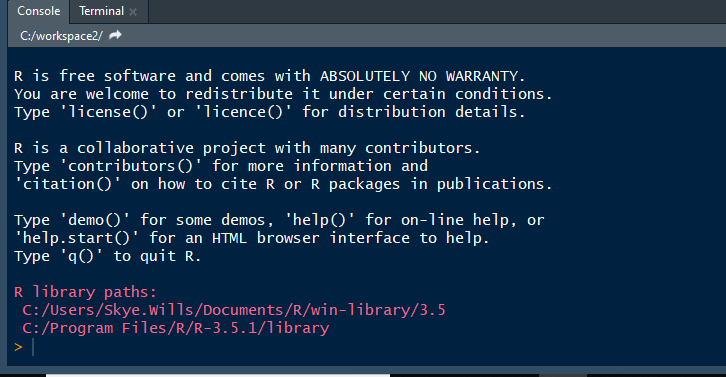
R will do mathematical calculations for you.

>log (100)

Calculates the natural log of 100

>log10(100)

Calculates a base 10 log of 100.



Command prompt

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

Try entering some simple math like 2+2 enter, you should see the answer returned in the console. When you see “>”, then the system is ready for you to inter another command.

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.

Command prompt

1.2 Set up the R workspace

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

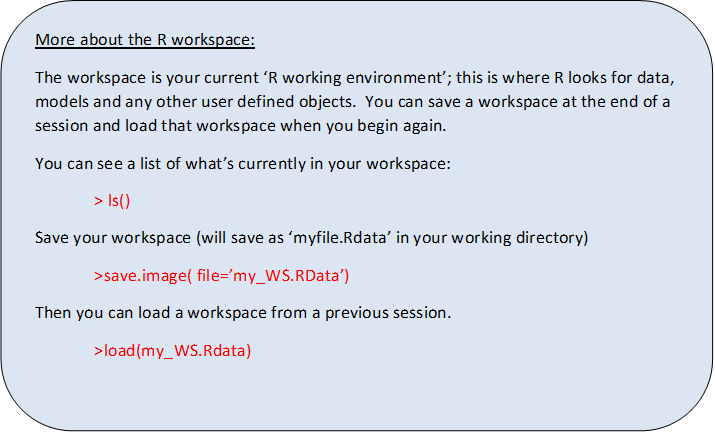
> setwd("C:/R\_data")

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



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

NOTE: you’ll also get an error if you haven’t created a folder with the path you’re pointing to.



**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 |

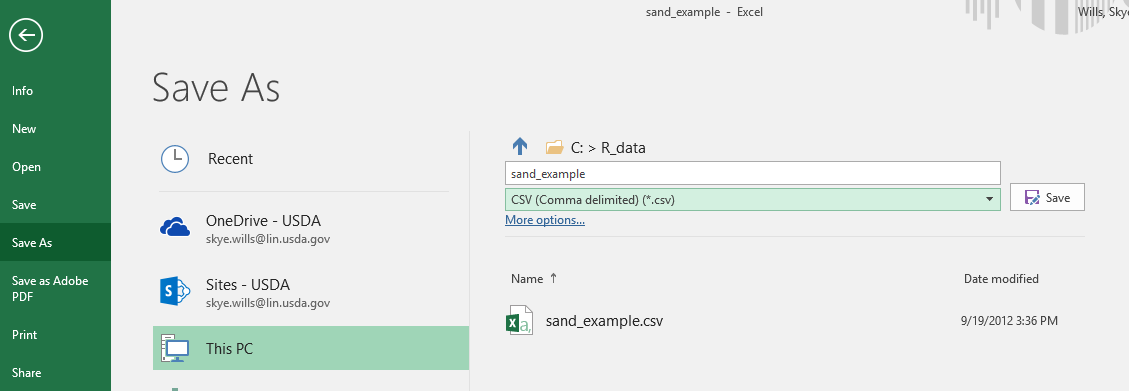
Create a new file in excel example\_sand.xlsx

This is an acceptable format; there is one kind of data stored in each column. Sometimes the terms wide and long tables/format are used. 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 one 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:

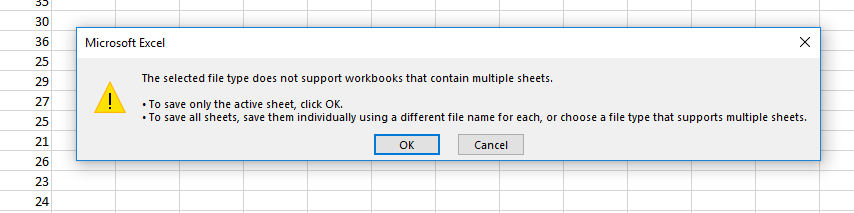
* notes about the project; where were the samples collected, by whom, pedon descriptions or user pedon ids
* explanation of the headers used; what method was used to determine sand content, how were master horizons determined, what definition was used for each land use
* analysis; summary tables, the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

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

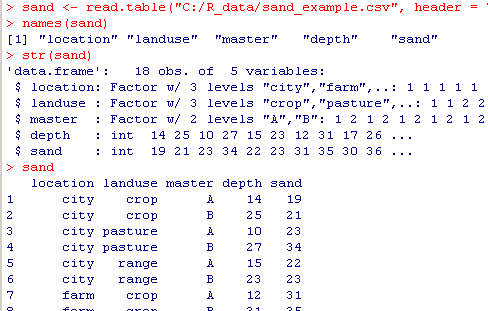
There are other arguments you might want to use.

>help(read.table)

The help command will open documentation in the help tab on the lower right. This document gives you information about the command requested including a full range of arguments you can use. You can usually also find these documents by search in your web browser. (you can also try ??read.table to do a search for help on the command line)

Explore the dataset

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**

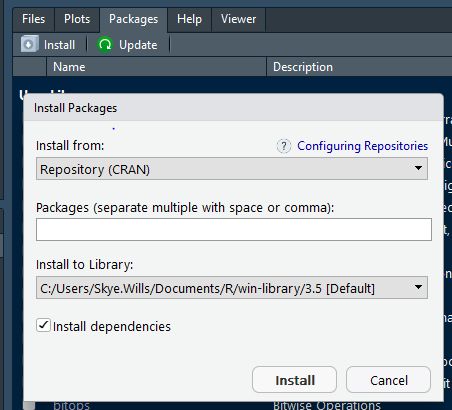
There are two ways to manage packages in R; with commands and through the GUI interface.

**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 Rstudio to Manage Packages

Navigate to the ‘Packages’ menu (typically lower right) – select ‘Install’



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.

Begin typing below pacakges – it will give you autofill options, the package name must be an exact match including capitalization.

Scroll down, type ‘lattice’ and Click Install at the bottom of the box.

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’ again – click on the checkbox next to the package you with to load.

Select ‘lattice’ and click OK

4.2 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)

**Warnings vs. Errors**

A warning is given if something didn’t go as expected. It alerts the user, but does not halt the execution of the command.

An error means that something is wrong and the command given cannot be executed.

For thi

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

>??maps

This will search help for you of all the places map might occur. We are interested in the maps:map documentation. This notation says package::command. If you see select that link, you’ll see 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 the function “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'))

Note that we give the argument a list with the c(). The individual parts of the list are separated by commas

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>)

**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

In the upper left of R Studio, Navigate to the File menu and select ‘New’ and ‘R script’

This will open a new tab in the source window called ‘Untitled’.

In the newly opened window:

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

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

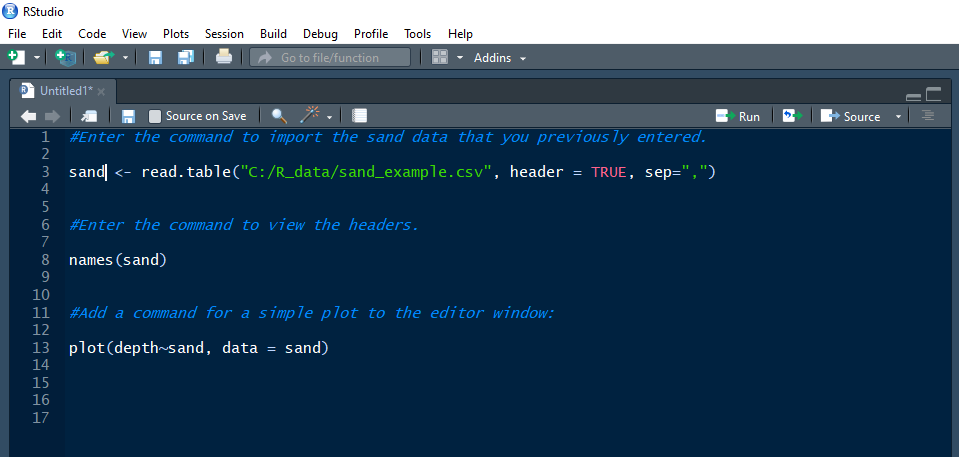
Enter the command to view the headers.

names(sand)

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

plot(depth~sand, data = sand)

To execute, move the cursor ‘|’ to a line and click on ‘Run’



Note that in this script there are explanations of the steps.

The ‘#’ indicates a comment (a line of text or notation) that R will ignore

To Execute all lines select/highlight all lines and right click –‘Run ’

This will return information in the script 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

Close the editor window - 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.

Navigate to the File menu and select ‘Open File’, navigate to the folder with the course material and open more\_sand.R.

more\_sand.R

A new tab will open in the source area.

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=",")

Replace XXXX with R\_data path to your files

Place your cursor in the first line.

Then right click with your mouse and select ‘Run’ – you can also use keystokes – CTrL-ENTER will also send this command to the console.

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, Ctrl+R (or right click and select ‘run’)

Names(sand)

See that the output is sent to the console

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. Note that we’ve given it new arguments that change the main title, and y axis).

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…)

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

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 any 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.**

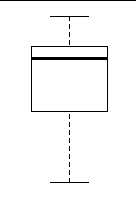
Now try the boxplot function:

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.

*>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.*

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.

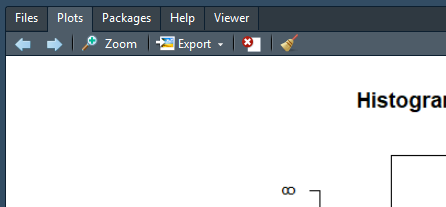
**You can add an explanation about your data or script. At the top of the R editor window enter:**

#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

View all graphs that have been created as part of this session with the arrows in the plot window.



Close R studio

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).

However, you don’t usually want to do this as you’ll want to be able to start each new session with a clean environment .

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)

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

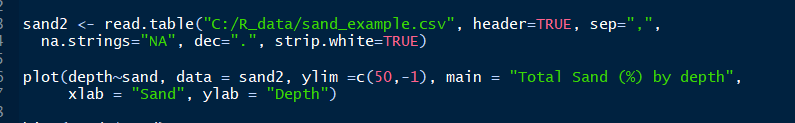
depth\_histogrm.pd

Exercise: Use a previously saved script in R studio

Open sand\_studio.R

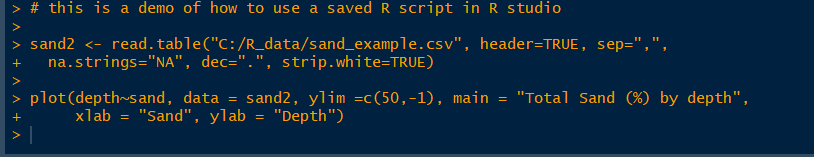
Run commands from within the script

Select first 2 command (first four lines) and hit ‘run’ from the task bar above the script (or cntl-r).

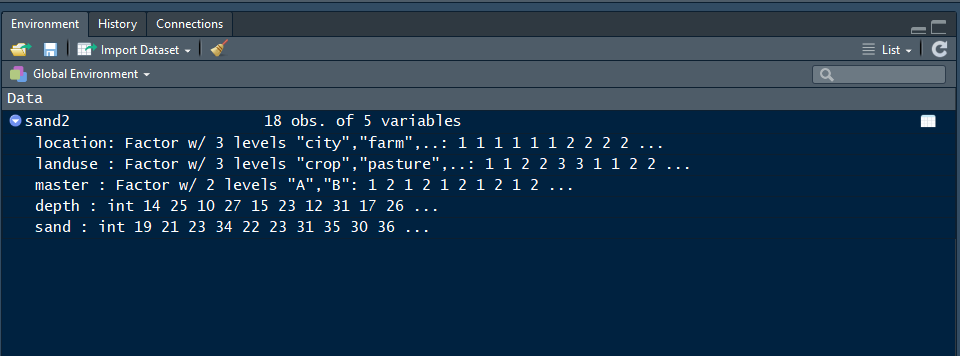


`

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)’

Now we will create a new kind of plot, a histogram (covered in Chapter 4 of Stats for Soil Survey)

Select and run

>hist(sand2$sand)

Can you change the color? The labels?

To get help

>?hist

The help page with tell you what arguments are available – scroll down find col and labels

Edit and submit with new arguments

>hist(sand2$sand, col = 'blue', main = 'Histogram of Sand Content', xlab = 'Sand Content (%)')

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

Now we will use a different package to make a graph: install and load ggplot2

ggplot uses a different approach for building plots (‘grammar of graphics’). It requires you to designate aesthetics of the plot (aes). Then you add a geometric object (geom) and you can control statistic objects, scales and other variables. It’s a very popular package and there is a lot of outline help. A google search will often lead you to the help you need

<https://ggplot2.tidyverse.org/>

Read the comments as you run the rest of the script – go line by line until you get an error

Error in FUN(X[[i]], ...) : object 'Master' not found

That looks like a complicated error – what does it mean?

The error is telling you that there is no object ‘Master’ – check the data you’re referring to…..

See that master is lowercase – edit the command and run again.

Complete the rest of the script – do a web search find a new element to add to your graph

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: > ggplot

You should see a pop-up box – if not place your cursor at the end and hit tab

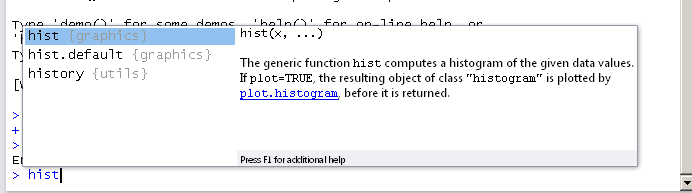
* you’ll see auto-complete options
* select one and you’ll see brief explanation of the functions and the name of the package it comes from {this can be handy for searching}
* hit the ‘F1’ key to get further explanation (equivalent to help(hist) in the console.

Look through the usage and arguments - some of this should look familiar from the last script we looked at.

Now try one with more options

Type: > geom\_

We can also go back and do this for the base R graphics such as histogram

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")

7. R markdown

R markdown is a file format that contains both text and executable bits of code.

<https://rmarkdown.rstudio.com/articles_intro.html>

This is an extremely brief introduction – but it’s not hard to get the basics

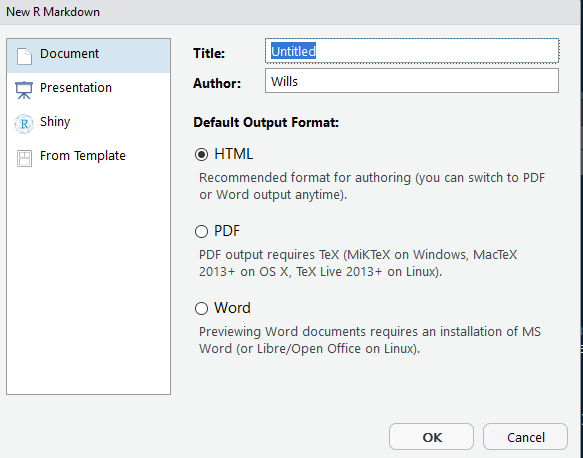
We’ll expect you to submit your final project as an R markdown file.

Open r studio

Open the sand\_studio.R file (file<open file)

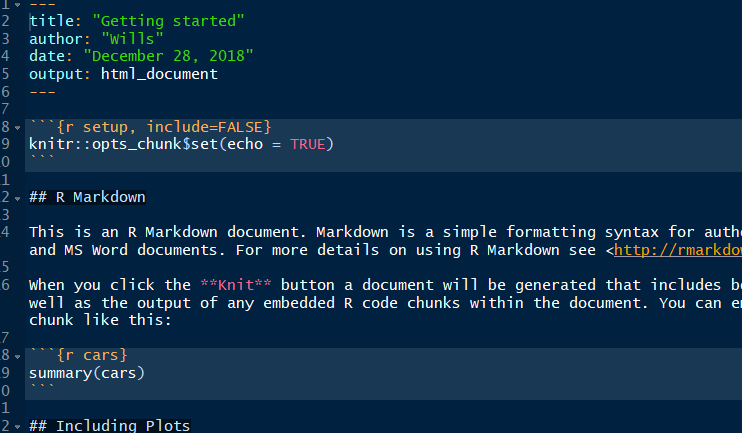
Create new R markdown file (file<new file< R markdown)

This should open the following pop-up



Change title, author and leave default input as html

It will open a new file in the source with some parts pre-populated



We will leave most of these. Note the top portion has the same information that we created in the pop-up. Don’t change this unless you have a good reason.

After the ‘---’

Areas that are not shaded are text, areas that are not are chunks.

To create a chunk you start with ``` (in the upper left of the keyboard)

Then {r name} – this can optionally be followed by options, each chunk is given a different name.

The middle of the chunk has the commands you want to execute.

Then end the chunk with ```

This pattern is repeated as needed.

You can add formatting to the text portion to emphasize or highlight important points.

See the R Markdown Cheat Sheet - <https://rstudio.com/resources/cheatsheets/> (scroll down)

Edit the markdown document to include all of the steps used in the sand\_studio.R script.

And Install and load the knitr package.

\*\*\* if you get stuck open start\_Rmarkdown\_wills.rmd to use as a guide

Save R markdown file as start\_Rmarkdown\_name.rmd

Use knit to create an html file -

\*\*\*\*note that errors in the Rmarkdown file will reference the chunk the error is in. Run the entire chunk cntrl-shift-R to send the commands to the console and locate the error.

Edit to add a red histogram of depths in sand2

Use knit to create a new html

Open in browser if you like – the file will be saved to your working directory

Send html to your mentor – this is your pre-course assignment.

8. R projects – not covered in this class

This is a handy way to save all related material in one place. We won’t cover these in detail, but they provide a way to organize data and scripts in one place for easy sharing.