Geog 5050: Introduction to Applied Spatial Statistics in Geography
Exercise 02: Introduction to R

Value: 40 points
Due: Wednesday, 16 September, by 5:00 PM

Overview: The purpose of this exercise is to walk you through setting up R and help you gain some familiarity with the R environment. This is a huge exercise in terms of page space, but it moves pretty quickly. I've been very explicit in the directions in the first two parts and am only asking you to work through the book for Part 3 in this exercise, since it will be the first exposure to R for most of you.

Assignment Objectives:
-Install R
-Set up R and specify the work environment.
-Import a basic table.
-Get accustomed to the programming environment in R.
-Produce a variety of data plots.
-Learn how to install packages and install the spatial data packages we will use in the course.
-Learn how to perform basic mapping tasks, using your book as a guide.
-Start thinking about your final project.

Relevant Course Objectives:

- **C.** demonstrate an ability to critically interpret, evaluate, and communicate about scientific research in geography that uses both aspatial and spatial statistical methods.
- **D.** be able produce, interpret, and analyze exploratory and inferential problems on spatial data in geography.
- **E.** demonstrate proficiency in performing statistical work using modern GIS and statistical software.
- **G.** effectively communicate the results of spatial statistical work to a general audience using multiple means of presentation.

Sources:
-Components of this exercise were borrowed heavily from Dr. Patrick Bartlein, Professor of Geography, University of Oregon and some components from the textbook, *An Introduction to R for Spatial Analysis and Mapping* by Chris Brundson and Lex Comber. Dr. Bartlein graciously shared his course materials and granted me permission to use them for this course.

I had to copy the Mac instructions verbatim as I've no access to a Mac to check it; please let me know if you notice anything here that needs to be corrected.

-Election data by county: Federal Election Commission, downloaded at https://catalog.data.gov/dataset

Part 01: Setting Up R [5 points]

1A. Installing the Software

R can be downloaded from one of the "CRAN" (Comprehensive R Archive Network) sites. In the US, the main site is at http://cran.us.r-project.org/. To download R, go to a CRAN website, and look in the "Download and Install R" area. Click on the appropriate link.

For Windows 7 - 10

Depending on the age of your computer and version of Windows, you may be running either a "32-bit" or "64-bit" version of the Windows operating system. If you have the 64-bit version (most likely), R will install the appropriate version (R x64 4.0.2) and will also (for backwards compatibility) install the 32-bit version. You can run either.

1. On the "R for Windows" page, click either on the "base" or "install R for the first time links".
2. On the next page, click on "Download R 4.0.2 for Windows" link (4.0.2 was the current version on September 1, 2020 – this changes often; just download the latest), and save that file to your hard disk when prompted. Saving to the desktop is fine.
3. To install, double-click on the downloaded file, or open it from a downloads window. Don't be alarmed by unknown publisher warnings. Window's UAC (User Account Control) will also probably worry about an unidentified program wanting access to your computer. Click on "Run".
4. Select the proposed options in each part of the install dialog. When the "Select Components" screen appears, accept the standard choices. The default startup and other options also work fine.

Mac OS X

1. On the "R for Mac OS X" page (http://cran.us.r-project.org/bin/macosx/), there are multiple packages that could be downloaded, but the one you want is one of two topmost ones. If you are running Mavericks, Yosemite or El Capitan, download the R-4.0.2.pkg in the next step; if you are running an earlier version of OS X, download the R-4.0.2-snowleopard.pkg in the next step
2. To download the package click on the (e.g.) "R-4.0.2.pkg (notarized and signed)" link.
3. After the package finishes downloading, right-click in the downloads window of your browser, and click on "Show in Finder" (or just look in the Downloads folder). This will open a new Finder window with the installer package.
4. Then double-click on the installer package, and after a few screens, select a destination for the installation of the R framework (the program) and the R.app GUI. Note that you will have supply the Administrator's password. Close the window when the installation is done.
5. An application will appear in the Applications folder: R.app. You may want to drag R.app to the Dock to make R easier to start up.

There are three "FAQ" pages that contain additional information that may be useful for working out the kinks. These include:

- R Windows FAQ: http://cran.us.r-project.org/bin/windows/base/rw-FAQ.html
- R for Mac OS X FAQ: http://cran.us.r-project.org/bin/macosx/RMacOSX-FAQ.html, and
- R FAQ (general): http://cran.r-project.org/doc/FAQ/R-FAQ.html.

1B. Starting R

Start the R "GUI" (graphical user interface), the application you just downloaded. You should see something that starts with the following in your console window:

R version 4.0.2 (2020-06-22) -- "Taking Off Again"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

Make a directory you want to store your R files in and give it a simple but clear name. In Windows, you can set the directory that R is working with, called the "working directory," by clicking on [File → Change dir…] on the R menu.

On the Mac, you will probably be in a default folder. You can use the [Misc → Change Working Directory] menu command to browse to the folder you created above. To change the working directory that R starts with each time, you can use the [R → Preferences… → General, Startup] dialog.

The command window (or **"R Console"**) is where you type commands and view text (as opposed to graphics) results. The prompt is the character ">" (in red in Windows, usually) at the bottom of the text in the R Console window. If you've scrolled away from the prompt, typing anything in the window will bounce you back.

Most of the time when using R, you'll also want to use a word processor (such as *Word*), to collect text and images, and so you may want to start that too. The Windows and Mac versions each have built-in "script editors" which are simple text editors that allow commands to be entered, sent to R, and saved for later reuse.

On Windows, the script editor can be started using the File → New Script dialog, while on the Mac it's [File → New Document].

*You might want to save your work as you work through this exercise and other projects! To save the workspace image, type:

```
save.image()
```

You can load your workspace with:

```
load(".Rdata")
```

1C. Quitting R

There are several ways to quit R:
(1) click on the "close window" button,
(2) go to [File → Exit] from the RGui menu on Windows or clicking [R → Quit R] on the R GUI menu on the Mac (or clicking on the power switch,
(3) or type **quit()** at the command prompt (or simply **q()**). (*Note that you must type the parentheses.*)

R will ask if you want to save the current workspace image. In general, you'll want to do that, but there are cases when you might not want to (e.g., you've accidentally deleted some intermediate results).

1D. Getting Help

The first thing to do in learning *any* new software application is to figure out how to get help.

R has several approaches. A quick way to get help on a particular function or command, for example, the quit function described above, is to type a question mark plus the name of the function at the command line (e.g. **?quit**) you can also type **help(quit).** (*Note that typing* **?quit** *will be one of the few times in which a function* **(quit())** *is typed without the parentheses.)*

You can also get to a web page-based help system by typing **help.start()** at the command line or using the **Help → Html help** menu from the R GUI.

On the Mac, the Help menu has a search window as well as a link the Html help (**Help > R Help**).

1E. A Data Set

The Summit Creek geomorphic data consists of 88 observations of 11 variables along a 0.8-km stretch of Summit Creek in eastern Oregon. This data set was collected by Pat McDowell, Frank Magilligan and their students as part of their study of the effects of cattle "exclosures" on the morphology of stream channels. They divided this stretch of Summit Creek into individual "hydrologic units" (HU's) that were either pools, shallow "riffles," or straight "glides." The overall study area is divided into three sections: an upstream reach (reach A) in which cattle are permitted to graze, a middle reach (reach B) from which cattle have been excluded, and a downstream reach (reach C), in which cattle were again permitted to graze.

The dataset contains the following information:

Column	name	scale	R class	Definition
1	Location	alphanumeric	character	ID for a particular cross section
2	Reach	nominal	factor	Reach (A=upstream reach(grazed); B=exclosure (no cattle); C=downstream (grazed))
3	HU	nominal	factor	Hydrologic unit type (P=pool; R=riffle; G="glide" or straightwater stretch)
4	CumLen	ratio	numeric	cumulative distance downstream from the upstream end of the study area (m)
5	Length	ratio	numeric	length of a hydrologic unit (m)

6	DepthWS	ratio	numeric	depth of the channel from the water surface to the bottom (m)
7	WidthWS	ratio	numeric	width of the channel at the bankfull stage (m)
8	WidthBF	ratio	numeric	witdth of the channel at the bankfull stage (m)
9	HUAreaWS	ratio	numeric	area covered by the hydrologic unit at the water surface (sq m)
10	HUAreaBF	ratio	numeric	area covered by the hydrologic unit at the bankfull stage (sq m)
11	wsgrad	ratio	numeric	water-surface gradient (m/m, i.e. dimensionless

The above table is sometimes referred to as a "**codebook**" or "**data dictionary**" that provides an expanded definition for each variable. (There is often a tradeoff between short variable names, which are efficient to type and which tend to create fewer problems in database applications, and longer variable names that are more self-explanatory.)

1F. Importing a Data Set

Reading data

R can read data from different sources, including **text (ascii) data** and the **.csv (comma separated values)** format of Excel spreadsheets, as well as from an internal format, which is text-based, but not easily readable by humans. R stores the data, names of variables, etc., in an efficient form in its workspace (.Rdata) that can be saved and reloaded.

The most efficient way to open and import a new data set is in .csv format.

Here we will work with the file "sumcr.csv," included among the exercise files. Make a copy of the file to your working directory.

To ***read*** the Summit Creek data set into R, type the following:

```
sumcr <- read.csv("sumcr.csv")
```

You can verify that the file is in the right place by typing:

```
dir()
```

This will list the files in the working directory. Punctuation, spelling and case are important. R is case sensitive; in other words, **Sumcr** is not the same thing as **sumcr**, and **Read.csv** is not the same as **read.csv**.

As noted above, if you're not in the working directory, you can use [File → Change dir] (Windows) **or [Misc→ Change Working Directory]** (Mac). You can also "strong-arm" the change of the working directory using the **setwd()** function. (Note that a lot of the commands are shorthand for English: "setwd" refers to "set working directory." Being aware of that logic behind function names can help you remember them…) I might use the following command for my computer:

```
setwd("c:\\Anth\\geog5050\\work01\\") # Windows
```

```
setwd("/Users/Anth/Documents/geog5050/work01") # Mac
```

Note the use of the double-backslash "\\" in specifying the folder paths in Windows. (R uses a single backslash "\" as an operator, and so the first backslash "escapes" the second, telling R to treat the combination like a single backslash).

The "#" refers to comments; this tells the computer to ignore what follows, but it's helpful for taking notes for us humans.

The **read.csv()** function creates a data frame "object" called "sumcr" that contains the data from the .csv file. Note that the data frame object doesn't need to have the same name as the file, but by convention it usually does. The "<-" arrow is called the "assignment operator", which, as it sounds, assigns whatever object is to its right to whatever object is to its left, sometimes creating a new object in the process. In reading a line of text, the operator is usually spoken as "gets" as in: "the dataframe sumcr gets the contents of the sumcr.csv file." In newer versions of R, the equals (=) sign can be used, but most existing texts and .pdf files use the <- version.

The advantage of the download-first-then-read-in approach is that you have an Excel-editable copy of the data set in your working folder.

An alternative approach for reading data is to use the **file.choose()** function to browse to a particular file:

```
sumcr <- read.csv(file.choose())
```

This will open a "Select file..." dialog box. There's a disadvantage to this approach in that it is not "reproducible"—at some later time, you may not be able to recall what file was read in to produce a particular result.

Looking at the data
The next thing to do is to check to see that R indeed has the Summit Creek data frame in its workspace. This can be done by typing **ls()** (the list function) at the command line, or (Windows) clicking on **Misc → List objects** on the RGUI menu.

The data frame can be examined by simply typing the name of the data frame at the command line (e.g., **sumcr**, which will create a lot of output, or by typing **head (sumcr)**, which lists the first five lines. You should be able to guess what **tail(sumcr)** does…

The **names()** function can be used to get a list of the variables in a data frame, e.g.,:

```
names(sumcr)
```

The individual variables are referred to by a "compound" name consisting of the data frame name and the variable name, joined by a dollar sign ($), e.g. **sumcr$WidthWS.** Note that variable names are case-sensitive too (e.g., the name "sumcr$WidthWS" is not the same as "sumcr$widthws"). This manner of

referring to variables can be made less cumbersome by using the **attach()** function. For example, try typing the following (there is no need type the comments!)

```
sumcr$WidthWS   # (works ok)
WidthWS   # (produces the error message Object "WidthWS" not found)
```

Then try typing **attach(sumcr)**, press Enter, and now type **WidthWS** on the next line (should work ok now).

1H. Printing

Use the **summary()** function to produce a quick summarization of the data set:

```
summary(sumcr)
```

To print the summary out, select the text, and click on the "print" icon, or use File > Print. For this assignment, you should copy and paste the text into a txt document. In windows, go to any location in the file explorer, right-click, and select **new→text document**. Paste the summary into the new document and name it **x02_RSummary.txt**.

>>>

What to turn in

-Upload a text document with the requested output: **Ex02_RSummary.txt**.

Part 02: Producing and Reading Plots in R [20 points]

The objective of this part of the exercise is to demonstrate some of the basic univariate plots for displaying data. This exercise will not involve much manipulation of R code; the individual "commands" (**`text in this courier font in the gray boxes`**) can be easily copied and pasted into the R Console window. I recommend that you take the time to type out the commands in these exercises. While this may not be the case for you, I find that adding a little tactile interaction to my own learning really helps… I also recommend that you skim through this part before you start working on it.

The plotting and display functions in R are impressive. If you want to see a tour with some of the examples of the kinds of things you can do in R, type:

```
demo(graphics)
```

…and then click through the demo in the graphics window.

When you're done, you can reset the graphical parameters with this command. The graphical parameters are settings for the graphical display and these "stick," so it's nearly always a good idea to reset them before you start to run a script.

```
dev.off()
```

2A. Univariate scatter plot (scatter diagrams) -- plot() version

The univariate "scatter diagram" is a very simple plot of the values of a variable, plotted vs the observation number, or row number in a rectangular data set (labeled "Index" on the plot). (We'll touch upon regular scatter diagrams or scatterplots later in the exercise). The data set here are arranged in downstream order, so there actually is some meaning to the observation number. That won't always be the case, however.

Check to see if the "sumcr" data set ("data frame") is still in your workspace using the ls() or "list" function:

```
ls()
```

If not, read it into your working directory again as you did in part 01. You can find the working directory after starting R by typing:

```
getwd()
```

Make sure that the sumcr data are attached if they are not:

```
attach(sumcr)
```

To create a univariate scatter diagram for the variable "Length", type

```
plot(Length)
```

Remember that you can refer to the data dictionary at the beginning of the assignment to see what the variables mean. "Length" contains data on the length of the hydrologic unit in meters…

Q1: What is the typical length of a hydrologic unit along Summit Cr.? What is the shortest, and what is the longest? What is the range of hydrologic unit lengths (difference between the smallest and largest)? (Just estimate from causally examining the graph to get these answers.)

Note that there is no information lost in this display. The original values of the variables, and even the order of the observations, could be reconstructed using a ruler.

It's sometimes helpful to be able to return to previously generated plots. The obvious way to do that is to just re-issue the commands again, but the individual plots can also be saved.

Using Windows, after creating a plot, make sure the R Graphics window is active (by clicking on it), and then use the [RGui History → Recording] menu to turn recording on.

On the Mac, the Quartz window has history turned on by default. Subsequent plots will then be added, and can be viewed again by using the PgUp and PgDn keys on Windows, or Command-left arrow on the Mac, when the RGraphics window is active.

2B. Univariate scatter Diagram -- stripchart() version

(This plot is also known as a "strip plot" or "dot plot".) Type the following:

```
stripchart(Length)
```

Here's an alternative version with the points with identical values "stacked":

```
stripchart(Length, method="stack")
```

Q2: Describe what this plot looks like, in comparison with the first one for length. Has any information been lost? Is there any particular pattern to the points in this plot? Was that pattern also evident on the first plot?

The two plots each have one point for each observation, and each point is plotted using the particular value of Length, but each gives a slightly different view of the data. That's a good practice: one view may enable you to see a pattern that is not evident in the other.

2C. Dotplots (dotcharts)

Another way to examine a single variable and gain some insight into its variations across the observations in a data is through Cleveland's dotplots (called "dotchart" in R). In the simple version here, the plot again shows individual observations. By default, the observations are arranged in the row-order of the data frame (i.e., the first observation on the bottom of the plot, last on the top). Try

```
dotchart(WidthWS)
```

Here's an alternative, this time with each line labeled by the Location variable

```
dotchart(WidthWS, labels=as.character(Location), cex=0.5)
```

The **as.character()** function converts the Location variable back to a character string (it was read in as a factor by the read.csv() function. The cex=0.5 parameter makes the characters smaller for legibility).

Here's a version in which the observations are sorted by the values of WidthWS. The creation of an index using the order() function is used to rearrange the values of WidthWS and the corresponding values of the Location character-string label:

```
index <- order(WidthWS)

dotchart(WidthWS[index], labels=as.character(Location[index]), cex=0.5)
```

2D. Boxplots

A *boxplot* contains an object (a box) and some decorations (lines, etc.) that are drawn to illustrate certain aspects of a variable. The box is drawn in such a way that the box itself encloses half of the data points. The top edge of the box is drawn so that 1/4 of the observations have values greater than that value, the bottom edge is drawn so that 1/4 of the observations have values that are less than that value, and the line in the middle of the box is drawn at the median. Try:

```
boxplot(Length)
```

(the "whiskers", by default, extend to 1.5 times the interquartile range). Here's an alternative version where the "whiskers" of the plot extend to the extremes of the data:

```
boxplot(Length, range = 0)
```

Q3: Compare it to the univariate scatter diagram and strip plot for Length. What are the advantages of a bloxplot over the scatterplot?

At this point we're done with the Summit Cr. data set, and so it's a good practice to "detach" using the **detach()** function. This removes the shorthand way of referring to the variables in that data frame, which will avoid possible collisions with the variables in another data frame that could have variables with the same names as those in the Summit Cr. data frame--a data frame from another study site for example. **detach()** doesn't remove the data frame from the workspace, which you can verify using the ls() function. To detach the sumcr data frame, type:

```
detach(sumcr)
```

2E. Histograms

A histogram essentially is a bar chart of a frequency table, where the heights of the bars reflect the relative (or absolute) proportion of observations that fall within particular class intervals of the variable of interest. The shape of the histogram reveals the distribution of the individual observations.

Open the data on the presidential election results by state in 2012 (PrVt2012.csv). The data include state names (STATE), the percentage of votes for Obama (PctObm), total votes (TotVote), and the region (Region). Put the file in your working directory.

The command to read the .csv file is a little different than was used for reading the Summit Creek. data:

```
usvote <- read.csv("PrVt2012.csv", as.is=1)
```

The as.is=1 parameter prevents R from turning the state name (STATE) in column 1 into a "factor," leaving it as a text label.

Here's the alternative approach, using **file.choose()**

```
usvote <- read.csv(file.choose(), as.is=1)
```

You can view the entire table by simply typing the object ("usvote") at the command prompt.

The nature of the individual variables in a data frame, i.e., whether they are continuous numeric variables "factors" that indicate group membership, or character-string labels can be seen using the str() function, which shows the "structure" of the data frame, and also prints out a little data:

```
str(usvote)
```

The listing produced should indicate that the STATE variable is a character string (chr), the vote variables are integer (int) or numerical (num) variables, and region is a factor (Factor).

Attach the data frame by typing:

```
attach(usvote)
```

Now, get histograms for the variable PctObm; the portion of voters who cast a vote for Barack Obama in the 2012 presidential election. To get a basic histogram, type:

```
hist(PctObm)
```

Q4: Describe the distribution of PctObm. What range of values occur the most frequently? What is the overall range of PctObm values in the data set (looking at the figure)?

Q5: Produce a stripchart of the PctObm votes using the stripchart() function described above. How well does each plot type describe the distribution of the PctObm values? (it's easy to see and difficult to articulate, but give it a shot.)

Experiment with the number of bars in the histogram: Type the following:

```
hist(PctObm, breaks=20)
```

Q6: Describe what the histogram looks like now. How does the shape of the histogram change as the number of bars increases? What does it look like if breaks=5?

2F. Density Plots

Evidently, the shape of the histogram (and consequently what it may imply about the distribution of the data) can vary considerably depending on the bin widths that are used to summarize the data or the number of bars (bins) used. An alternative plot type is the "kernel density smoother plot" This plot is produced by first using the density() function to estimate the number of data points in the vicinity of different values of the Obama vote percent values, and then plotting these. To produce the plot, type the following two lines at the command prompt:

```
PctObm.density <- density(PctObm)
plot(PctObm.density)
```

Q7: Describe the different views that the histogram and density lines give of the data. Which view seems less dependent on the particular way the plot is generated? Which view loses the least information about the individual values of the variable?

2G. A Composite Plot

The views of the data provided by the different plotting methods vary quite a bit. Some retain a lot of information, but may be hard to interpret (particularly if there are a lot of data), while others are very simple appearing, but lose information. One strategy for dealing with this is to produce a plot that superimposes several different plots. Type the following, one line at a time:

```
PctObm.density <- density(PctObm)
hist(PctObm, breaks=20, probability=TRUE)
lines(PctObm.density)
rug(PctObm)
```

Q8: What does the resulting plot contain? What does the rug() function appear to do? Does this plot offer any advantages over the individual plots, or is it too cluttered?

>>>

What to turn in

-A word document containing a paragraph with answers to the eight questions. You should be able to fit your responses on a single page: **Ex02Questions.docx.**

Part 03: Working with Spatial Data in R [10 points]

Chapter 02 of the text (Brundson and Comber) will take some time to work through and think about, but it begins to teach you some of the way that you can essentially use R. Because it's important that you work through this text, I want to award some points this week for going through the process of doing it.

For this part of the exercise, you need to send in:

A. An image (a .pdf or image file) of the map of Appling County you produced by the end of section 2.4.1.
B. An image (a .pdf or image file) of the plot of rectangles you produced, similar to Figure 2.7, in the text. For this exercise, ***alter the plots*** so that they are two overlapping rectangles (rather than two overlapping squares).

Part 04: Planning your Final Project [5 points]

It makes a lot of sense to start planning the work you will do on your final project early(!) The purpose of this part of the assignment is to get you to start doing that. Take a little time to review the final assignment for this class.

Report on the basic question you would like to ask in your analysis. Write a short paragraph to explain the state of the field you are interested in examining (for example, if you are interested in examining deforestation as it relates to protected areas, provide a couple of general statements about the kinds of issues being investigated in the field).

Write down, in explicit form, a draft of the ***research question*** you would like to ask. By this, I literally mean "write down a question," with all the trimmings, such as a question mark!

Finally, list the types of data you will need for your analysis. You don't have to actually produce the data or a link (I'll probably ask you to do that in a future exercise), but you should start thinking about what you will need to perform your analysis.

The write-up for this only needs to include as many words as it takes to address the questions; please limit your response to a **maximum** of two pages (~500 words).