EcolMon Statistics Review

Your name

Mon Sep 27 13:48:39 2021

Welcome to R! If you haven’t had the pleasure of using R already, it is simultaneously powerful enough to be the only data analysis package you’ll ever need, and complex enough that you may never feel like an expert no matter how much you use it (I don’t yet consider myself an expert after 21 years of using R in my classes and in my research). But, with a little guidance you can learn to do some pretty useful things with it without too many tears.

This document is formatted in R Markdown. R Studio understands R Markdown, and is able to feed commands written within “code chunks” to the R program, and report nicely formatted output below the chunk. I use R Markdown for teaching because the format makes it easy for me to give you instructions and background in text blocks like this one, have you write and execute your code right where the instructions tell you to, and answer questions below the R output. I use R Markdown in my research because it makes annotate my work, and communicating analyses and results with collaborators, much easier.

You’ll see as you move through the exercise that some of the text looks different - there are formatting characters that indicate to R Studio when **bold** or *italics* or ***bold italics*** should be used, and section headings are indicated with pound signs (the “#” characters). There are a few places where I ask you questions, which will be **bold face**, and the place for you to respond is below, next to a greater-than prompt, “>”. For example you will see things like:

**Question: is R easier to use with R Studio?**

You will write your answer next to the greater-than symbol, like so:

Why yes, yes it does.

Some of the formatting doesn’t change the appearance of the text much within R Studio, but when you push the “Knit” button at the top of this Rmd file window, R Studio reads the formatting commands, executes all the code chunks, and assembles the output together into a document using the output format of your choice (we are using MS Word format, but pdf and HTML web pages are also available options)- The formatting characters are used to set the format of section headings and text in the output - I have created a template document, called template.docx, which will be downloaded to your computer and used to specify some non-default formatting options, such as setting the answers you write to be red text in the Word file to make your work easier for me to find.

When you’re done with the exercise you can knit this file, and turn in the Word document with your nicely formatted work. Slick.

## Importing the data

R is impressive on its own, but what makes it truly amazing is that it can make use of contributed extensions (which R calls “libraries”) that add funcitonality not found in the base R package. For example, the ability to import Excel worksheets is provided by the readxl library. If you are working with R Studio installed on your own computer you will need to install it if it’s not already present - switch to the “Packages” tab to the right, and scroll down through the installed packages looking for “readxl”. If it’s not there, then click on “Install” and then type “readxl” (without the quotes) to find the package - readxl, along with any other packages it needs to run properly, will be installed from the CRAN repositories maintained by the R Project.

Once the library is installed, you still need to load it for use. If you are just using R interactively you can do this by checking the box next to readxl in the packages tab. But, you need to load the library “in code” for it to work when you knit the file, meaning that you need to write a command inside of a code chunk to load the library. I’ll give you this first command to get you started - click on the run button (the little green triangle pointing to the right at the top right corner of the grey code chunk below - if you hover over it a tooltip will pop up saying “Run Current Chunk”):

library(readxl)

You’ll see a green progress bar flash on the left edge of the command, from top to bottom. It won’t seem as though anything happened, but as long as you didn’t get a red color next to the command and an error message below the code chunk it worked - you can now use the import\_excel() function from the readxl library to import the first data set.

To import the worksheet you need, you will use the read\_excel() function from the readxl library. We will give read\_excel() takes two arguments: the name of the spreadsheet file (which is stat\_review.xlsx), and the name of the worksheet within it that has the data to import (which is IndepSamples). The command is:

read\_excel(path = “stat\_review.xlsx”, sheet = “IndepSamples”) -> indep.samples

Type this command into the code chunk below (at line 40). Please resist the temptation to copy and paste this command into the code chunk - it’s hard to learn R unless you type code yourself. Be careful, unlike Excel R is case sensitive, so if you use lower case leters for the I or S in IndepSamples you’ll get an error message.

read\_excel(path = "stat\_review.xlsx", sheet = "IndepSamples") -> indep.samples

In an R function, arguments can be used either by giving them in the default order, or by naming them and putting them in any order you like. Since we named the arguments as path = and sheet = we could have put them out of order and it would have been fine. Since they were in order, we could also have done without them entirely - using the command:

read\_excel(“stat\_review.xlsx”, “IndepSamples”) -> indep.samples

would have worked the same. As long as we name the arguments we could have changed the order:

read\_excel(sheet = “IndepSamples”, path = “stat\_review.xlsx”) -> indep.samples

and that too would have worked. However, without the labels, putting the arguments in the wrong order would have given us an error message - the command:

read\_excel(“IndepSamples”, “stat\_review.xlsx”) -> indep.samples

would have failed with an error message, because R would assume IndepSamples is the Excel file, and stat\_review.xlsx is the worksheet within it, and would have been unable to find a file called IndepSamples.

The last part of the command **assigns** the output of read\_excel() to an **object** we call indep.samples. The assignment operator in R is an arrow created by a dash followed by a greater-than symbol, ->, which assigns in the direction the arrow points, from left to right - thus, the -> arrow assigns the output of read\_excel() to a data set called indep.samples that is created by this command.

You can reverse the assignment order by using an arrow pointing left, <-, like so:

indep.samples <- read\_excel(path = “stat\_review.xlsx”, sheet = “IndepSamples”)

It is also possible to use the equal sign for assignment, but it only assigns from right to left, like the <- operator does, so, we would need to change the command to:

indep.samples = read\_excel(path = “stat\_review.xlsx”, sheet = “IndepSamples”)

Why am I telling you all of the different ways you could accomplish the same thing? Becuase it’s one of the things that makes R wonderful to use, but awful to learn - there are many ways to do the same thing, but there are still rules, and at first you’re not sure what you’re free to change and what absolutely has to be done a specific way. This gets better with practice.

Okay, to get back to the data, you should now see indep.samples in your Environment tab. You can open it by clicking on the name in the Environment tab, which will open it in a tab over this window - you’ll see the first column is Year, and the second column is NDVI.

## Linear regression

To begin, we will see whether NDVI is showing a consistent trend on average over time. NDVI is a numeric variable, and we can use Year as one as well. The stright-line relationship between two numeric variables is analyzed using simple linear regression.

In regression we make a distinction between a *predictor* variable and a \_response\_variable. The response variable is the one we think is changing (NDVI), and the predictor is the one we suspect is causing the change (Year).

### Visualizing a regression

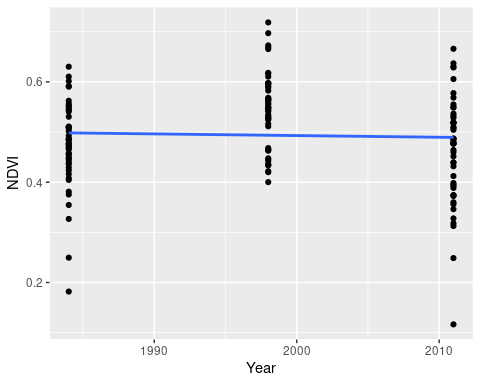
A good first step in analyzing any data set is to make some graphs. The typical graphical representation of a regression analysis is a **scatter plot**, which plots each data value on x,y axes, with the predictor variable assigned to the x-axis, and the response on the y-axis. The best fitting straight line is plotted through the data.

R has the ability to make graphs without having to install extra libraries, but these so-called “base” R graphs have some limitations. There is a very good graphing package, called ggplot2, that will allow us to make all of our graphs (relatively) simply.

If you’re working on your own computer, install ggplot2 now (along with any other packages it depends on, which will be installed automatically with your permission). If you’re working through CougarApps it is already installed. To use it, load the library, and then use the qplot() function to produce a quick plot of the data:

library(ggplot2) ggplot(indep.samples, aes(x = Year, y = NDVI)) + geom\_point() + geom\_smooth(method = “lm”, se = F)

library(ggplot2)  
ggplot(indep.samples, aes(x = Year, y = NDVI)) + geom\_point() + geom\_smooth(method = "lm", se = F)



ggplot() works a little differently from other R functions - the graph elements are built in steps. The first step is to use the ggplot() command to establish the data set to be plotted (indep.samples) as well as the “aesthetic mappings”, which are the roles played by the variables in the data set (the aes() arguments). Once the basics are established with the ggplot() function the geometric elements to be plotted are added, litereally witha + sign. We add points with geom\_point(), and add the line with geom\_smooth(method = “lm”, se = F).

You can see that the line is pretty flat, with a slope near 0. A slope of exactly 0 would represent a complete lack of relationship between NDVI and Year, because a slope of 0 is a perfectly flat line, with the same NDVI predicted no matter what the year is. If knowing the year has no effect on the predicted NDVI, then NDVI is *indepdendent* of year.

But, a slope that is close to 0 is not a slope of 0, because the slope is both a measure of the relationship between the predictor and the response varialbe, and it is unit converter - NDVI is a value between -1 and 1, and Year is a value in near 2000, so to convert between year and NDVI a slope will need to be a small number. We can’t tell if NDVI is independent of year just by looking at the size of the slope.

On the other hand, even if at the *population* level, across all of the pixels within public land in the FPA, there is no relationship between NDVI and year there will be some random variation in the slope of the line just due to the fact that we’re only looking at 50 points each year.

To deal with the sampling problem, we specify a **hypothesis** about the properties of the population that we can test with the data - specifically, we specify a ***null* hypothesis**, which in statistics is always the hypothesis of independence (or lack of pattern, or lack of difference, or randomness). In this case, since the line describing the relationship between NDVI and Year would have a slope of 0 if NDVI was independent of year, our null hypothesis is that at the population level the slope is 0. This is symbolized as Ho: = 0 (this symbol will render as the lower-case Greek letter beta = 0).

The null hypothesis in a linear regression is that there is no relationship between NDVI and Year - if this is true a flat line with a slope of 0 is the best fit line to the data, and it looks like this is very close to what we are seeing. We would not expect our linear regression to be statistically significant, but we can confirm that by running a linear regression on the data and obtaining the ANOvA table that tests the null hypothesis.

It looks like NDVI is about the same in 1984 and 2011, but was higher in 1998.

### Testing the null hypothesis of slope = 0

The line looks flat, but the slope for this sample of data is unlikely to be exactly equal to 0. We can get the slope by “fitting” a model to the data - this just means that we find the slope and intercept of the straight line that fits the data best. Fitting a straight line to data in R is done with the lm() function. If you remember, the equation for a straight line is:

y = mx + b

where y is the y variable, x is the x variable, and m is the slope. The constant, b, is the y-intercept, which means that it is the value of y when x is set to 0. In statistics, the regression line has the equation:

predicted mean of response = m (predictor variable value) + b

When we fit a straight line model in R we use a **model formula**, which tells the lm() function what the response variable is and what the predictor variable is - the slope and intercept are not specified. So, for this example our model formula is:

NDVI ~ Year

The tilde between NDVI and Year is taking the place of an equal sign, because = is used for other things in R (you can use it in place of <- as an assignment operator). We use this in the lm() function like so:

lm(NDVI ~ Year, data = indep.samples)

This lm() function says that NDVI is used as the response (becuase it’s on the left side of ~) and Year is used as the predictor (because it’s on the right side of ~), and uses data = indep.samples so that the function knows where to find the NDVI and Year variables.

Enter this in the code chunk below:

lm(NDVI ~ Year, data = indep.samples)

##   
## Call:  
## lm(formula = NDVI ~ Year, data = indep.samples)  
##   
## Coefficients:  
## (Intercept) Year   
## 1.1603683 -0.0003337

The output repeats the command you used (as the “Call”) and then gives the two coefficients - the intercept (as (Intercept)) and the slope (named for the predictor variable, Year). The null hypothesis is that the slope is equal to 0 for these variables, but this hypothesis refers to the population level - a sample of data like ours is extremely unlikely to have a slope of exactly 0. To test whether the slope of -0.0003337 represents a real relationship between NDVI and Year we need to get the ANOVA table for this model. We do this in R with two steps - we will fit the model again and assign it to an object, called ndvi.year.lm, and then we use the anova() function on it to obtain the ANOVA table. We will then type the name of the fitted model object, ndvi.year.lm, so that we can see the summary output.

lm(NDVI ~ Year, data = indep.samples) -> ndvi.year.lm anova(ndvi.year.lm) ndvi.year.lm

lm(NDVI ~ Year, data = indep.samples) -> ndvi.year.lm  
anova(ndvi.year.lm)

## Analysis of Variance Table  
##   
## Response: NDVI  
## Df Sum Sq Mean Sq F value Pr(>F)  
## Year 1 0.00203 0.0020306 0.2217 0.6384  
## Residuals 148 1.35559 0.0091594

ndvi.year.lm

##   
## Call:  
## lm(formula = NDVI ~ Year, data = indep.samples)  
##   
## Coefficients:  
## (Intercept) Year   
## 1.1603683 -0.0003337

Note that even though you ran the same lm() function here you didn’t get any output from it - that’s because you assigned it to the ndvi.year.lm object. Typing the name of the fitted model object we assigned the output to, ndvi.year.lm, allows us to see the output, after the anova() command’s ANOVA table is reported.

The p-value for the effect of Year on NDVI is 0.6384, which is very high - this means that the probability of getting a sample of data with a slope of -0.0003337 by random chance if the null hypothesis is true is 0.6384 (in other words, our slope is typical of slopes of random samples from a population with no relationship between NDVI and Year). To consider the slope to be atypical, and likely to be a real effect, the p-value would need to be 0.05 or less (0.05 is our alpha level for the test).

The most important assumptions of regression analysis are that the model fits the data (that is, the mean NDVI changes by a constant amount with each passing Year, in a straight line), that the data values are normally distributed around the line, and that the amount of vertical scatter around the line is the same as you move from low to high values of the predictor. The first assumption is easiest to assess with our scatter plot with a line through it that we already made - the mean seems to increase in 1998 and decrease back to about the same mean as 1984 between 1998 and 2011, so a straight line doesn’t fit very well. This is reason enough not to use linear regression for these data, but we’ll look at how we assess the other two assumptions for the sake of completeness.

The latter two assumptions are not so much about the data as they are about the **residuals**, which are vertical differences between data points and the line. We can calculate these differences, and then test to see if they are normally distributed using the Shapiro test. We can also test for any relationship between the square of the residuals and the predictor variable - if the residuals are bigger at one end of the x-axis than the other then the size of the predictor variable will predict the size of the squared residuals. This is called the Breusch-Pagan test, and we can use it to test for unequal variances in a regression.

We can assess the assumptions with an assumption test, which tests the hypothesis that the data meet the assumption we are testing. The test of normality of residuals is done with the shapiro.test() function - we can get the residuals for the model with residuals(ndvi.year.lm), and then use this function as the argument for shapiro.test() like so:

shapiro.test(residuals(ndvi.year.lm))

shapiro.test(residuals(ndvi.year.lm))

##   
## Shapiro-Wilk normality test  
##   
## data: residuals(ndvi.year.lm)  
## W = 0.97302, p-value = 0.004747

The null hypothesis for the Shapiro test of normality is that the data follow the normal distribution, so a p-value less than 0.05 means that the null hypothesis is rejected, and we fail the test.

We can test for lack of homogeneity of variance (HOV) using the bptest() function, which is in the lmtest library. If you don’t have it installed already, install lmtest now. The commands will be:

library(lmtest) bptest(ndvi.year.lm)

library(lmtest)

## Loading required package: zoo

##   
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':  
##   
## as.Date, as.Date.numeric

bptest(ndvi.year.lm)

##   
## studentized Breusch-Pagan test  
##   
## data: ndvi.year.lm  
## BP = 1.2946, df = 1, p-value = 0.2552

According to the BP test we have HOV - the null hypothesis is that the variance in the residuals is the same from left to right along the x-axis, so if p > 0.05 we retain that null, and “pass” the test.

If the data followed a straight line from year to year, and we passed the HOV test but failed normality, what would we have done? We probably would have proceeded to interpret the regression model - regression is **robust** to moderate deviations from normality, provided that we meet the HOV and linearity assumptions, meaning that we can still infer a linear chage in mean based on the slope of the regression even if the data are modestly non-normally distributed. But, since a straight line is a poor representation of how NDVI is changing on average over time, we shouldn’t use regression even if we passed both the normality and HOV tests - a poorly fitting model trumps all other considerations.

## Analysis of Variance

Treating Year as a numeric variable and using a linear regression to analyze it means that we are only looking for straight line changes in NDVI. If we had another pattern, such as an increase from 1984 to 1998 that reverses between 1998 and 2011, we could fail to detect it with a straight line.

If instead we treated Year as a grouping variable, and asked if there are differences on average between the years without requiring the difference between years to have any particular pattern, we could find that there are statistically different mean NDVI between years after all.

By default Year is imported as a numeric variable, which is what we need it to be when we run regression analysis. To use it for grouping purposes we also need it to be a **factor**, which is R’s categorical variable data type. Categorical variables are made up of levels, which are categories rather than numbers (if our variable was Color, we might have levels Red, Blue, and Green, for example). We can convert Year to a factor and assign it to a new column in indep.samples:

indep.samplesYear)

indep.samples$Year.factor <- factor(indep.samples$Year)

You now have three columns in indep.samples, the third one being Year.factor. You can confirm that Year and Year.factor are different by clicking the little blue circle with the white triangle inside it next to indep.samples, which displays a summary of the variables - Year is numeric (num), and Year.factor is a factor with three levels. Bear in mind that this means that R will consider 1984, 1998, and 2011 to be labels, but will not use their numeric values in any way when we use Year.factor as a predictor (they will be displayed in order because factor levels are created in alphabetical sort order by default, but their actual numeric values are not used).

### Graphing means for each year

ANOVA compares group means, and a good graph to illustrate what is being compared is a graph of means with 95% confidence intervals displayed as error bars.

Considering how common this type of graph is, it is a little surprising that R does not have a built-in function to make the table needed. There is a library, called Rmisc, that has such a function, so **if you are working on your own computer install Rmisc now - then in the get.summaryse.function code chunk below enter**:

library(Rmisc)

**If you’re running R Studio through CougarApps** you can add this function by copying the code below into the get.summaryse.function code chunk (copy everything between row 202 and 220, including the closing curly brace, “}”, and paste it into the code chunk):

summarySE <- function (df, measurevar, groups) { df <- data.frame(df) if (length(groups) == 1) { grp.list <- list(df[, groups]) names(grp.list) <- groups } else grp.list <- as.list(df[, groups]) summ.func <- function(x) c(N = length(x), mean = mean(x), sd = sd(x), se = sd(x)/sqrt(length(x)), ci = qt(0.975, length(x)-1)\*sd(x)/sqrt(length(x))) output <- do.call(data.frame, aggregate(df[, measurevar], by = grp.list, FUN = summ.func)) names(output)[(length(grp.list)+1):(length(grp.list)+5)] <- c(“N”,measurevar,“sd”,“se”,“ci”) return(output) }

summarySE <- function (df, measurevar, groups)   
{  
 df <- data.frame(df)  
 if (length(groups) == 1) {  
 grp.list <- list(df[, groups])  
 names(grp.list) <- groups  
 }  
 else grp.list <- as.list(df[, groups])  
 summ.func <- function(x) c(N = length(x), mean = mean(x), sd = sd(x), se = sd(x)/sqrt(length(x)), ci = qt(0.975, length(x)-1)\*sd(x)/sqrt(length(x)))  
 output <- do.call(data.frame, aggregate(df[, measurevar], by = grp.list, FUN = summ.func))  
 names(output)[(length(grp.list)+1):(length(grp.list)+5)] <- c("N",measurevar,"sd","se","ci")  
 return(output)  
}

Whether you installed Rmisc, or entered the function’s code in the code chunk, you should now be able to use it to make a table of means, with sample size, standard deviation, standard error, and uncertainty (called “ci” in the table).

We can now use summarySE() to get the mean NDVI each year - the command is:

summarySE(indep.samples, “NDVI”, “Year.factor”) -> ndvi.year.summ ndvi.year.summ

summarySE(indep.samples, "NDVI", "Year.factor") -> ndvi.year.summ  
ndvi.year.summ

## Year.factor N NDVI sd se ci  
## 1 1984 50 0.4746109 0.08474970 0.01198542 0.02408560  
## 2 1998 50 0.5427673 0.07564896 0.01069838 0.02149920  
## 3 2011 50 0.4637799 0.10516320 0.01487232 0.02988705

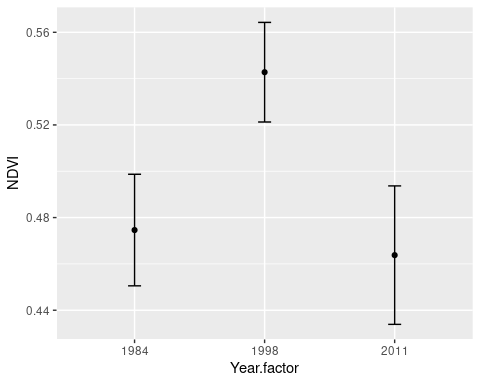
You should have created a summary table object, ndvi.year.summ, and since you entered the name below the summarySE() function the table’s contents is also displayed. We will plot the data from this summary table.

**Note how I am naming objects - having the ability to annotate your work is a big advantage of R Markdown, but it is still helpful to use names of objects that are “self documenting”, meaning that they give some information about what they contain. The summary table created for NDVI by year is called ndvi.year.summ, instead of “my.cool.table1”. Using a good naming convention helps you tremendously when you’re doing a complex analysis**

Now to graph the mean and 95% confidence intervals, we use ggplot() on this summary data:

ggplot(ndvi.year.summ, aes(x = Year.factor, y = NDVI)) + geom\_point() + geom\_errorbar(aes(ymin = NDVI - ci, ymax = NDVI + ci))

ggplot(ndvi.year.summ, aes(x = Year.factor, y = NDVI)) + geom\_point() + geom\_errorbar(aes(ymin = NDVI - ci, ymax = NDVI + ci), width = 0.1)



This ggplot() command is similar to the first, in that we identify a data set and assign roles to columns from that data set, and we use points to represent the means for each year. The error bars also come from the table, so we have another aes() statement within geom\_errorbar() that uses the NDVI (mean) and ci (uncertainty) columns to construct the error bars. The width = 0.1 argument is part of geom\_errorbar(), but is outside of aes(), so it is applied to all of the error bars - the devault width is way too big for my taste, and width = 0.1 makes the cross-bars look nicer.

Based on the graph, or impression from looking at the scatterplot is reinforced - it looks like there was in increase in mean NDVI in 1998, followed by a decrease in 2011 back to about the same level as in 1984.

But, the raw data isn’t visible in this graph, so it’s hard to tell if using means fits the data better than using a straight line. We can address that by adding the NDVI values from indep.samples to the graph:

ggplot(ndvi.year.summ, aes(x = Year.factor, y = NDVI)) + geom\_point() + geom\_errorbar(aes(ymin = NDVI - ci, ymax = NDVI + ci)) + geom\_point(data = indep.samples)

ggplot(ndvi.year.summ, aes(x = Year.factor, y = NDVI)) + geom\_point(data = indep.samples, color = "red") + geom\_point() + geom\_errorbar(aes(ymin = NDVI - ci, ymax = NDVI + ci), width = 0.1)



Because the column names are the same in indep.samples and ndvi.year.summ we didn’t have to tell this second geom\_point() which columns to use. To get the data points to plot behind the means and error bars this second geom\_point() had to be placed before the original one, and coloring the points red helps differentiate them from the mean and error bar symbols. You’ll see that using the mean for each year is doing a better job of representing how NDVI is changing each year, so ANOVA fits the data better than regression.

To test for differences among years, we will first fit a model that uses Year.factor as a predictor, and then ask for the ANOVA table.

lm(NDVI ~ Year.factor, data = indep.samples) -> ndvi.year.factor.lm anova(ndvi.year.factor.lm)

lm(NDVI ~ Year.factor, data = indep.samples) -> ndvi.year.factor.lm  
anova(ndvi.year.factor.lm)

## Analysis of Variance Table  
##   
## Response: NDVI  
## Df Sum Sq Mean Sq F value Pr(>F)   
## Year.factor 2 0.18336 0.091680 11.477 2.336e-05 \*\*\*  
## Residuals 147 1.17426 0.007988   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

You’ll see that there seems to be a difference between the means.

The ANOVA table’s test of Year.factor is an *omnibus* test, meaning that it tests for differences among the means for year overall, but doesn’t tell us which means are different. We can get Tukey post-hocs on these means using the TukeyHSD() function - it wants the model to be in a different format, so we will use the aov() function on the model to convert it to a form that TukeyHSD() can work on - we can “nest” functions inside of other functions to get the results we need in this way:

TukeyHSD(aov(ndvi.year.factor.lm))

TukeyHSD(aov(ndvi.year.factor.lm))

## Tukey multiple comparisons of means  
## 95% family-wise confidence level  
##   
## Fit: aov(formula = ndvi.year.factor.lm)  
##   
## $Year.factor  
## diff lwr upr p adj  
## 1998-1984 0.06815642 0.0258331 0.11047974 0.0005878  
## 2011-1984 -0.01083098 -0.0531543 0.03149233 0.8171137  
## 2011-1998 -0.07898740 -0.1213107 -0.03666409 0.0000567

The p-values are adjusted to compensate for the greater chances of false positives that comes from conducting multiple tests. Because these p-values are adjusted, you can consider them statistically significant if they are less than 0.05 (if they had not been adjusted, we would have had to adjust the alpha level instead, and only consider the differences significant if the un-adjusted p-values were below 0.05/3, for example).

As you will see, the difference between the years is not 0 for any pair of means, but the difference is only large enough to be confident that it is not just due to random chance for 1998 compared with 1984, and for 1998 compared with 2011.

We can do our tests of assumptions again - since we’re using residuals the assumption tests can change if the model changes, and now the residuals are based on differences of data values from group means.

shapiro.test(residuals(ndvi.year.factor.lm)) bptest(ndvi.year.factor.lm)

shapiro.test(residuals(ndvi.year.factor.lm))

##   
## Shapiro-Wilk normality test  
##   
## data: residuals(ndvi.year.factor.lm)  
## W = 0.9711, p-value = 0.002988

bptest(ndvi.year.factor.lm)

##   
## studentized Breusch-Pagan test  
##   
## data: ndvi.year.factor.lm  
## BP = 3.4951, df = 2, p-value = 0.1742

You’ll see that we still fail the normality test, but meet the HOV test. We can interpret these results, though, because a model that places a mean in the middle of each year’s data fits the data much better, and ANOVA is *robust* to violations of normality if HOV is met, and if the sample size is large (over about 35 observations per group, and we have 50).

So, to wrap up this section:

* It is possible to analyze these data using either regression or ANOVA
* Using the correct analysis is necessary to understand the patterns in the data - an analysis can only find patterns it is designed to find, and using regression on data that changes, but not along a straight line, can fail to detect patterns in the data.

## Using the same points repeatedly

The other option we’re considering for monitoring changes in NDVI is to use the same points each year. We will learn some new methods for analyzing “repeated measures” data like these next week, but for this week we will use the paired t-test, which you learned about in your introductory statistics classes.

First, let’s import the data so we can give it a look - use the read\_excel() function, but this time import the “PairedSamples” worksheet, and assign it to an object called paired.data.

read\_excel("stat\_review.xlsx", "PairedSamples") -> paired.data

If you open this data set by clicking on paired.data in the Environment tab, you’ll see there is a column identifying the point, and then one for each year of NDVI data: NDVI\_84, NDVI\_98, and NDVI\_11.

We can get means for each column using the summary() function, using paired.data as the argument (give it a try):

summary(paired.data)

## Point.number NDVI\_84 NDVI\_98 NDVI\_11   
## Min. : 1.00 Min. :0.1820 Min. :0.3362 Min. :0.1166   
## 1st Qu.:13.25 1st Qu.:0.4275 1st Qu.:0.5001 1st Qu.:0.4525   
## Median :25.50 Median :0.4683 Median :0.5668 Median :0.5044   
## Mean :25.50 Mean :0.4674 Mean :0.5602 Mean :0.4950   
## 3rd Qu.:37.75 3rd Qu.:0.5155 3rd Qu.:0.5954 3rd Qu.:0.5667   
## Max. :50.00 Max. :0.6563 Max. :0.8022 Max. :0.6659

The mean is reported for each year, and you’ll see that 1984 has a mean NDVI of -.4674, 1998 has a mean of 0.5602, and 2011 has a mean of 0.4950. This is similar to the pattern that we got when we used independent samples - increase between 84 and 98, followed by a decrease nearly back to the 1984 level.

But, since we have the same points measured repeatedly, we don’t expect the measurements from each year to be independent of one another - a point that is in a grassy field will have higher NDVI than one that is in a patch of chamise chapparal scrub, regardless of whether it’s a wet year or dry year. There are two issues with this sort of data:

* If we don’t account for the lack of independence between measurements, we are acting as though we have 150 independent data values, when we do not - this can make it more likely we’ll reject the null incorrectly
* The variation between points due to differences in the vegetation type they are in is statistical “noise”, meaning that it is adding random variation to the data that isn’t related to whether there are changes from year to year. This makes it less likely we will detect a real difference is there is one.

We can address both of these issues by analyzing the difference between the measurements at the same point in different years. This is what a paired t-test does - it subtracts two sets of paired measurements, and then compares them against a mean of 0 using a one-sample t-test.

To do a paired t-test with R’s t.test() function we just need to specify the two columns to be compared, and use the argument paired = T to get a paired test:

with(paired.data, t.test(NDVI\_84, NDVI\_98, paired = T))

with(paired.data, t.test(NDVI\_84, NDVI\_98, paired = T))

##   
## Paired t-test  
##   
## data: NDVI\_84 and NDVI\_98  
## t = -6.881, df = 49, p-value = 1.014e-08  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## -0.11986473 -0.06567767  
## sample estimates:  
## mean of the differences   
## -0.0927712

The with() command identifies the data set to use so that the t.test() function nested within it knows which data set holds the two columns of data. You’ll see that the p-value is much less than 0.05, so there has been a change in NDVI between these years. The df is equal to 49 because the test is working with 50 differences, rather than 100 data values.

In the next code chunk alter the command to compare NDVI\_84 to NDVI\_11, and then again to compare NDVI\_98 to NDVI\_11.

with(paired.data, t.test(NDVI\_84, NDVI\_11, paired = F))

##   
## Welch Two Sample t-test  
##   
## data: NDVI\_84 and NDVI\_11  
## t = -1.5224, df = 94.525, p-value = 0.1313  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## -0.063582765 0.008393093  
## sample estimates:  
## mean of x mean of y   
## 0.4674287 0.4950235

with(paired.data, t.test(NDVI\_98, NDVI\_11, paired = T))

##   
## Paired t-test  
##   
## data: NDVI\_98 and NDVI\_11  
## t = 4.1359, df = 49, p-value = 0.0001382  
## alternative hypothesis: true difference in means is not equal to 0  
## 95 percent confidence interval:  
## 0.03350795 0.09684477  
## sample estimates:  
## mean of the differences   
## 0.06517636

You’ll see that there is a slight difference between 1984 and 2011, and another big difference between 1998 and 2011. The difference between 1984 and 2011 is small enough that if you did not focus on the difference between paired measurements, and compared the two years as though they were two independent samples, the p-value is no longer less than 0.05 - you can confirm this by changing the paired = T argument to paired = F and running the analysis again. The advantage of tracking a fixed set of points over time, then, is that smaller differences are detectable than if independent samples are used - repeated measures of fixed locations are a very popular monitoring practice, and if they are analyzed properly this type of data can be very effective at detecting change.

That’s it! Knit your file and upload it to the class web site.