Applied Economic Statistics Using R: Lecture Notes

Scott W. Hegerty, Ph.D.
S-Hegerty@neiu.edu

April 2022

Associated videos, code and data, and other content available at
https://github.com/hegerty/ECON346

Introduction

These notes summarize some of the lecture material from my course, which I designed and offered in Fall 2019. I may expand or improve them before I offer it again. Most of the class was "hands-on"; half the class meetings were lab sessions, and even "lectures" involved demonstrations of real code. I have some corresponding .R code files, as well as links to explanatory videos and other material, on my website at *www.scotthegerty.com* or *https://github.com/hegerty/ECON346*.

This is just a basic outline of how the class is structured. Since it draws upon students' prior knowledge in Economics and Statistics, I don't provide much in the way of detailed explanation here. There are also much better R tutorials than I can write myself, so I mention some good (and free!) ones below. I also cover such important topics as data collection, report presentation, and mathematical formulas in another course, "Macroeconomic Data Analysis." But since that course (which primarily uses Excel) introduces some R applications, everything overlaps. Here, I mostly have references to other references.

Whenever you take a course, it helps to think about what it's about. Sometimes, college classes keep an old title for years(!), but here, the title contains three components, which define what we cover:

1) **Applied Statistics**: This is opposed to *theoretical* statistics. Here, there are no mathematical proofs, just finding useful conclusions with real data.
2) **Economics**: As opposed to biostatistics, for example; here we focus on two main data examples: Real GDP and inflation, and an application of the "Rank-Size Rule" in urban analysis.
3) **R** software: You could do a similar course using Excel, Python, Eviews, Stata, or any number of packages. Here, the main goal is to gain highly useful skills with R in particular.

These three components can be combined in a number of different ways. That can be thought of with the following formula:

$$\gamma_1 ECON + \gamma_2 STATS + \gamma_3 R = 100\%$$

Based on a poll of the class—which contained mostly majors with a mixture of backgrounds, ranging from complete beginners to having programming experience in other software—the best combination was to have $\gamma_1$at about 5% and the rest equally weighted. Stats review was combined with R practice, with assignments calculating and testing statistical concepts and arriving at economic conclusions.

While not much time is spent on actual Economics content (other than perhaps mentioning what "inflation" is and why it is important), it helps to look at the **nature of economic data**. A lot of time series have one observation per year, quarter, etc. (and T observations in total), while "cross-sectional" data have N observations and need not have a time component. Panel data combine both, with $N \times T$ observations in all. One quick example to make the distinction could be house prices: You could have a time-series with a single variable: An index that combines all Chicago house prices each month. Or, you could have citywide price data from last October. A panel could have all houses, every month. Econometric techniques can be used to uncover relationships that drive house prices, over time or across neighborhoods.

Learning R can take a lot of up-front effort (it has a high fixed cost), that pays off much later. My best advice for learning it is to **use it exclusively on a project that interests you**.

- Look up what you don't know (you can use a search engine, which might point you to a forum such as Stack Exchange, if you have questions). Just watch out for overly elaborate answers.
- Feel free to "borrow" code, but don't just copy it if you don't know what needs to be changed. Some students keep the original author's variable names, for example!
- Don't give up and switch to other software you know better. Once you finally complete your project, review what you have done—later on, you may be surprised at what you learned (and perhaps forgot).

The six topics covered here are:

1. Basics of R for Economic Statistics
2. Summary Statistics
3. Data Visualization
4. Statistical Distributions
5. Hypothesis Testing
6. Advanced Topics: Regression and Time-Series Analysis

Here are the **readings** that I assigned for my course:

W. N. Venables and D. M. Smith, *An Introduction to R*
G. Grolemund and H. Wickam, *R for Data Science*
Both are freely available online.

1. Basics of R for Economic Statistics

In any class, some students are transitioning from a different software package (which might have a more user-friendly interface), while others are brand-new to any type of statistical software. While some have more experience than others, R (or any coding language) often requires that students do more of the thinking themselves. There is no pre-written drop-down menu, for example.

A key point is that software can do more than you can do yourself, but it **cannot read your mind**. You have to speak its language. It is good at repetitive tasks that you don't want to do, if you train it properly. For example, if I gave you a list of 1,000 U.S. cities and asked you to find the ones with 200,000 people or more, you could sit down with a pen and paper and spend a while circling names. You have to think about each one, doing the math in your head. Or, you could use R and be done in seconds. Plus, if I changed my mind and wanted cities with 100,000 people or more, you could change a single character in R and be done in one second.

Working with R involves a lot of using and re-using the same coding principles until they become familiar. For that reason, I am not going to go over an exhaustive tutorial. Instead, I have a set of .R files that show the code in action. The listed texts are also a great reference.

I am not that great at writing code. But it works when I run it! Generally, code has to strike a balance between **efficiency** and **practicality**. Some of my code is broken down into sections to make it clearer. Other times, I found a clumsy way to do something but never updated my methods. But I have a few suggestions:

1) Start simple, then improve as your skills improve. At first, you might be happy to make a simple graph, but later on you might change things like point size and axes.
2) Try to avoid "hard coding" things like numbers. You might have 48 observations in your vector *x*, so measuring from 1:48 will work, you will have to go back and change your code if you drop an observation. Instead, `1:length(x)` will work no matter how many observations *x* has.
3) Try to have the computer think for you as much as possible. Instead of looking up "how long is *x*?," remembering the answer, and typing it in, including `1:length(x)` in your code looks it up and enters it automatically, while you sit back and enjoy.

4) Thinking about the computer's language is a challenge that is hard to "book learn." You have to *do it* to learn it, so just paging through tutorials will make your eyes glaze over. I am also not a fan of "cookbook"-style tutorials; I think it is best to have an end goal (question or project) and find ways to meet it. Otherwise, you are blowing through step after step to get to the end.
5) You have to communicate in a specific way so that the software does what you want—no more, no less. Make sure that your result is correct by checking and re-checking. Sometimes I do "cheat" and check my answer in other software to avoid a costly mistake.
6) Once you know one language, others are relatively easy to learn. This is because much of the work is in *thinking* and *problem-solving*, not syntax. That said, there are differences. The first object in a Python list is numbered 0, while it is 1 in R.

Here are a few very basic concepts to cover before beginning. R is an **object-oriented** language. You can actually create a named item with your calculated values, which you can recall and use for further calculations. Data are organized in **dataframes**, as well as matrices and vectors. You can check the type of an object, or convert between types.

Commands in R are nested; they "telescope" outward. Each command is inside parentheses. For example, the log change in a variable can be calculated as `dlnx<-diff(log(x))`. The natural logarithm is calculated first, and then the difference is taken of that value. (Note how the variable is assigned using a left-pointing arrow.) Square brackets often define subsets of some type. `matrix[2,3]` represents the element in the second row and third column, for example.

There are many, many more things that I could cover here, but an example might be better. Suppose I have four countries' growth rates, and want to know which of the first three has the highest average. I could calculate each with Excel (or a calculator, or by hand!), look at them, decide myself, and write down the answer. Or, I could code:

```
print(colnames(dataset)[which.max(colMeans(dataset[,1:3]))])
```

Data in my dataset (which I named "dataset") are, as is typical, arranged in vertical columns. The first calculation in this line of code is for the "column means," but here I only want columns 1 to 3 (represented by the range 1:3). The subset that I want is inside the parentheses). Then, I isolate which of those three column means is the maximum. R would return a number from 1 to 3 if I stopped there. But I can have the software do even more for me. I can return the specific column name that has that maximum value. But instead of `(colnames(dataset)[3]` (if it is the third column), the calculation that gets that number is put in there. Finally, the answer is “printed” so that it is visible onscreen.

The notation also involves matched parentheses, matched brackets, specific capitalization, specific command spelling and syntax, and a range of columns. All of this will make more sense as you practice. But one way to think of it is by putting it in a logical order:

```
 5           4           3           2           1

print(colnames(dataset)[which.max(colMeans(dataset[,1:3]))])
```

The steps are as follows:

1) Subset the dataset by choosing columns 1 to 3. You could choose 1 and 3 (and not 2) by making a new object: `c(1,3)`. Notice that the “row” part—before the comma—is empty. This means “all rows.” If you had a 2 there, it would just mean “row 2.”
2) Take the column means. This gives you three numbers, one for each column. You could also do row means if you want. The “apply” function can do a similar calculation, for more than just means. If you did the full dataset, you would have four numbers.
3) Choose which of the three numbers is the largest. This gives you a column number, not the growth rate itself (`max()` would do that).
4) Feed that integer into the list of column names for the dataset. These could be country names, for example. This finds that number’s item.
5) Return the country name that matches the maximum growth rate. Sometimes, you don’t need this command, but sometimes when you run R it doesn’t show every step.

Having this code, what if you decided to include that fourth country? You could delete the 1:3, or change 3 to 4. With one click, you might have a new country name as your answer. Having done all the preliminary work, this step is much easier.

Hopefully, seeing the examples (**Lec01_Intro.R** and **Lec02_GDPgrowth.R**), and working with your own data will help this syntax make more sense. We now move to the five main topics in the course.

2. Summary Statistics

Most students have taken at least one statistics course, but you might need to review. While I don't do any proofs here, I find it to be a useful assignment to do these statistics by hand and then compare the measures with the R commands. For example, students can sum variable *x* and divide that number by *N*; this should equal mean(x).

The main goal here is to create a table of basic summary statistics for a set of variables. I focus mainly on the **mean** and **standard deviation**, as well as **median**, **minimum**, and **maximum**. These can be calculated with `mean(), sd(), median(), min(), and max()`.
R's "summary" function also gives some **quartile** values that are also useful.

Try to do as much as you can to produce a table automatically. This includes:

- Calculating statistics for a number of variables using the "apply" function
- Concatenating (binding) table rows using the "rbind" function (you can do the same for columns using "cbind")
- Rounding to a set number of decimal places using `round(x, #)`
- Assigning row and column names using `colnames(x)<-c("x","x","x")` (Note that the text you put in is in quotes).

Setting up code this way makes it applicable for any dataset, or allows for easy updating of results with minimal changes.

These are *univariate*, or one-variable, measures. The main **bivariate** measure that I use here is the **correlation**, or the measure of association between two variables. The "traditional" correlation measure is related to the **covariance**, which is adjusted for each variable's standard deviation (or the square root of the variance). I also introduce the Spearman correlation, which is based on rankings rather than deviations from the mean; this nonparametric measure is insensitive to outliers. Correlations range from -1 (a perfectly negative relationship) to +1 (a perfectly positive relationship). The R command is `cor(x,y)`, with Pearson as a default, or `cor(x,y,method = "spearman")`.

For some students, being able to calculate these measures is an excellent achievement for this course. Others might want to use these for hypothesis testing or econometric analysis. **Lec03_GrowthRates.R** and **Lec04_Inflation.R** show some further examples of these statistics.

3. Data Visualization

Visualizing your data properly will help both you and your reader understand the point you are trying to make. Before beginning a project, you will want to look at any patterns in your data; this does not have to look perfect. But once you have something ready to print or present, you want it to look good. This involves 1) accurately conveying the "story" you want to tell; 2) presenting information efficiently and avoiding common mistakes; and in Economics, 3) "Signaling" that you are proficient with software and data analysis. In class, I talk about how academic papers are written in LaTeX, while many journals only accept submissions in Word. It is inefficient but often professionally necessary.

In my "Macroeconomic Data Analysis" class, I spend a lot of time on formatting reports and presentations. I also discuss common mistakes to avoid when preparing charts and tables. Here, I focus on the R components, but I do want to stress:

1) Know your audience—their goals and academic background.
2) Know when to use a chart and when to use a table; don't use one when the other is better.
3) Avoid common errors such as incorporating visual distortions—pie charts and 3D graphs are common offenders here.

Here, I focus on "base R" graphics, since this is an intro class. You might wish to use the package *ggplot* or *ggplot2*. There is an excellent book by Kieran Healy called *Data Visualization* (Princeton University Press, 2018) that shows both how to think about your data and how to present them well. Most of my academic work uses base R anyway.

I suggest looking at others' work to see how they use various techniques in their visualization. Some news or political publications (like *FiveThirtyEight* or *The Economist*) have such unique styles that they can be easily recognized. Many academic papers look much simpler. But all well-presented data use a lot of common tools. In my course examples, I apply a lot of them in the context of addressing other questions. This can be seen in the "Statistical Distributions" section.

Some useful data visualization functions in R include:

- **Histograms**: `hist(x,breaks = #)`; note that the number of breaks can be chosen, resulting in thicker or thinner bars.
- **Bar graphs**: `barplot(x)`; you can choose the height, width, etc. Don't forget that RStudio shows menu options as you type the functions; try experimenting with these.

- While I do not like pie charts [`pie(x)`], I think the “stacked bar” is better. Still, usually a table presents the information most efficiently.
- **Boxplots**: `boxplot(x)`; these give a good representation of the data distribution, such as median and other quantiles. I don’t see them used so much in Economics, but maybe that is just me.
- I personally like the “**stripchart**”: `stripchart(x)`, which lays each point out in a line. High-density areas are depicted as clusters.
- I use the line graph a lot with **time series**. This can either be done as `plot(x,type=”l”)`, explicitly choosing a “line” as a type as opposed to the default setting of points. I prefer `ts.plot(x)` for time series, and this method also better incorporates multiple lines.
- Two series can be plotted as a **scatterplot**: `plot(x,y)` as well.

You might wish to overlay graphs on a single axis using `par(new=TRUE)`. I’m not really going into it here, as you should refer to other, more detailed sources if you need to. I do have examples in the .R files. I also sometimes “print to file” (usually as a .jpg), with 300dpi or higher resolution.

When I make a graph, I usually remove the default x- and y-labels `(xlab=””, ylab=””)`, but add a header `(main =””)`. You can also add a legend to your graph; this is included in some of the .R files. Make sure that every element of your lines are matched in the legend.

Here are some examples of the customizations you can use. You can do a web search for charts and lists for all the options; eventually you might find out that you have memorized them!

Object: line/point	`type=“l”`	`type=”p”`
Point type: circle/triangle	`pch=20`	`pch=2`
Point size	`cex=3`	
Lines:		
Type: solid/dots	`lty=1`	`lty=2`
Width	`lwd=2`	
Color	`col=“red”`	`col=”#FF0000”`
Axis labels	`xlab=“” , ylab=“Inflation”`	
Axis range: (0-100)	`ylim=c(0,100)`	
Title	`main=“My Chart”`	
Multiple charts	`par(mfrow=c(2,2))`	
Margins	`par(mar=c(1,1,1,1))`	
Additional lines (horizontal)	`abline(h=0,lty=2,col=“grey”)`	

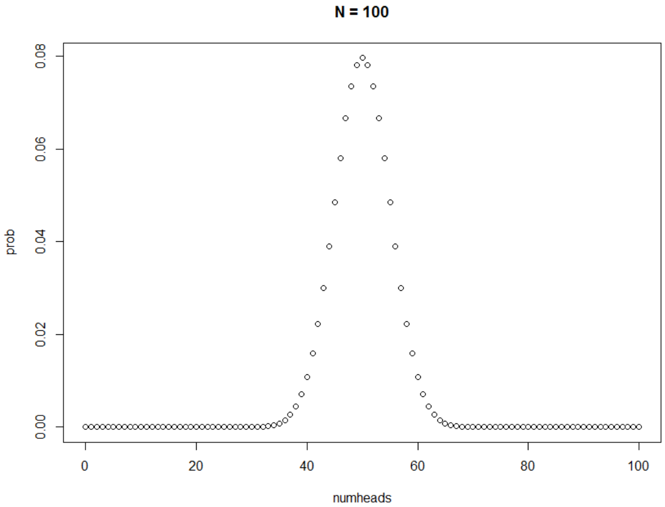
Colors can either be depicted with a list of names or with **hexadecimal** (Base 16) code. Rather than going from 0 to 9, hexadecimal numbers go further: [0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F] (so F = 15). Just like base 10 stops at 9 before starting over, base 16 stops at 15. And just like the highest 2-digit base-10 number is $10 * 10 - 1 = 99$, the highest 2-digit base-16 number is $16 * 16 - 1 = 255$. The base-16 equivalent to 100 is 256, so some computer systems display 256 colors.

Hexadecimal colors are presented as RGB, or red, green, blue, with 256 shades each. In R, green might be #00FF00, or another combination of colors. Black is #000000 and white is #FFFFFF. One of the many shades of gray could be #AAAAAA, for example. You may wish to look online for a color picker that allows you to click on your desired color to get the corresponding code.

Two .R files, **Lec05_Visualization.R** and **Lec06_Visualization2.R**, show the concepts discussed here.

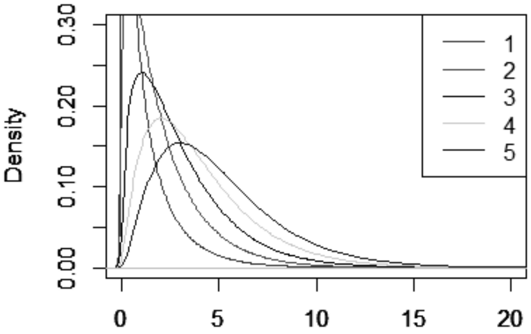
4. Statistical Distributions

A statistical distribution plots the various outcomes of some event on the x-axis and the probability of such an event on the y-axis. Not every event is equally likely (unless you have a *uniform distribution*). For the well-known Normal distribution, values close to the mean are far more likely than are "outliers." I used R to plot the outcome of 100 independent coin tosses (with $p(heads) = 0.5$). The most likely outcome is 50 heads, but the variance $= np(1-p) = 25$, so the standard deviation = 5. More than 60 heads would be unlikely. We can use these moments (mean and variance, plus skewness and kurtosis) to make statistical inferences. The larger N gets in this binomial distribution, the more it represents a normal distribution.

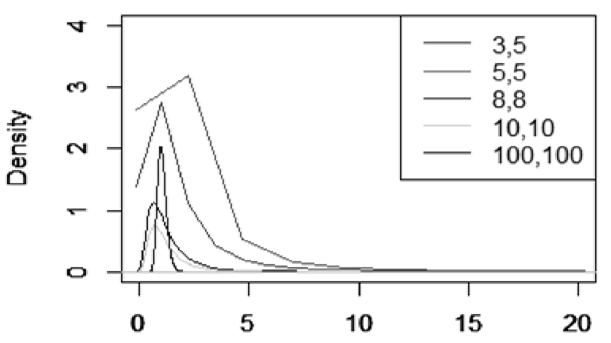


There are a huge number of these distributions—I like to show Michael McLaughlin's *Compendium of Common Probability Distributions*—but in Economics, the most-used are:

- The **t-distribution**. This is related to the Normal distribution, but is used for hypothesis testing in small samples ($N < 30$). You need to know the **degrees of freedom**, which are related to sample size. The statistic's critical values depend on them too.
- The **Chi-square distribution.** This is based on the sum of squared normal distributions. The degrees of freedom are the number of these distributions. The shape—and the critical values—also depend on the degrees of freedom.



- The **F-distribution.** Like the Chi-square, this is used for **joint tests** (where multiple conditions may hold). One application is whether the ratio of two variances equals 1. This distribution has two sets of degrees of freedom, and its properties depend on both.



The main use of these distributions is in **hypothesis testing**. In regression analysis, in particular, one key test is whether the coefficient $\beta = 0$, this is done via a t-test of $\frac{\beta=0}{s.e.}$ using the coefficient's standard error.

As part of this section, I also use a **loop** to generate distributions with different degrees of freedom. I use "for loops," which repeat a set of commands for a specified list or range of values, much more often than I use "while loops," which run as long as a particular condition is met. I use loops to create columns of numbers, each element of which is calculated by the commands in the loop. The trick is to start by creating a null object, and then appending new values at the end.

If you want information about a statistical distribution, there are four related *R* commands. These examples are for the Normal distribution, but you can do them for others (such as *unif*, *chisq*, *t,* or *f*). Some of these need extra parameters, so make sure you include them):

`dnorm`	gives the height of the probability distribution function
`pnorm`	gives the conditional distribution function (*quantile, mean, sd*)
`qnorm`	gives the inverse CDF (*p-val, mean, sd*)
`rnorm`	generates random observations *(# obs, mean, sd*)

For example, `rt(300,20)` returns 300 random observations drawn from a t-distribution with 30 degrees of freedom, and `qf(.95,10,15)` gives the 95% critical value from an F-distribution with (10,15) degrees of freedom. If you enter `qnorm(.05)`, you get -1.644854, which is the lower 10% tail of the Normal distribution. Typing `dunif(1,0,5)` shows the height of a Uniform [0,5] distribution at the value 1, or 0.2.

Many of these concepts (including t-, Chi Square, and F-distributions) are shown with the exercises **Lec07_Distributions.R** and **Lec07b_MoreDistributions.R**.

5. Hypothesis Testing

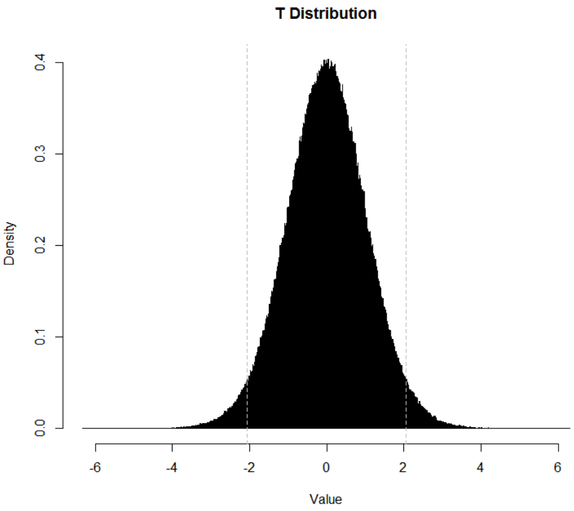
Much of economic analysis involves **hypothesis testing**, or examining whether sufficient evidence exists to show that some event has occurred or that some relationship exists. If there is enough evidence, you reject the **null hypothesis**, which is literally that nothing has occurred, in favor of your chosen **alternative hypothesis**. Note the double negative: you "reject the null": This is "not nothing." (But you never "accept the null.") If you don't find the necessary evidence, you "fail to reject the null," which is a triple negative.

One example to think about could be some type of cold medicine. The null hypothesis is that it is ineffective, and the alternative is that it helps people get better more quickly. You would need significant evidence that people are recovering more rapidly than normal with the medicine. You need evidence that exceeds random chance, or random variation—some people get over a cold quickly anyway, so how do you know the medicine made the difference?

This "evidence" and "random variation" can be shown on a statistical distribution. In the coin-toss example in Section 4, the number of heads (if the number of tosses is large) resembles a normal distribution. "Rare," or low-probability, events, can be found far from the mean. This distance is often normalized so that the distribution is presented as number of standard deviations from the mean, with zero at the center. About 68% of observations lie within one standard deviation from the mean. With 100 tosses, you would expect 50 to turn up heads, but 45 to 55 are pretty typical.

The "rule of thumb" for this distribution is that "extreme events" can be found about 2 (actually 1.96) standard deviations from the mean. If the entire distribution represents 100% of the possibilities, the "tips" at the end represent 2.5% on each side. Together, they represent the 5% that is too rare to attribute to chance. If you flip a coin 100 times, and 39 turn up heads, something might be wrong with the coin. This 5% represents a p-value of 0.05 or 95% significance. It is harder to find proof at the 1% level (99% significance), since this even further (2.576σ) from the mean. But you need less proof at the 10% level, since 1.645 standard deviations is now extreme. With my coin toss, if I really wanted to think I was being cheated, I could get angry with 41 heads at 10%; if I wanted more proof, I would need 37 heads at 1% before I started making accusations.

I combine some R concepts to generate random data that follow a t-distribution, and then plotting the corresponding histogram (with so many bins it is a solid figure). It is not perfectly smooth, since the data are random. I also add some vertical lines at the 5% **critical values**, beyond which an event is deemed to be significant. I can look at the t-statistic (> 2 in absolute value) or the p-value (< 0.05) to make my determination.



A **t-test** is often conducted to compare the difference in means between two samples or series. For example, $\bar{X}_1 = 3.2$ and $\bar{X}_2 = 3.1$, but is this difference *significant*? You would need to know the standard deviations (and the sample sizes). In essence, you have to "beat" random variation to know that there's something really there. One exercise that I do in class is to use data series and conduct the t-test manually, by using the formula $t = \frac{\bar{X}_2 - \bar{X}_1}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$, and comparing the result to the R output from `t.test(x1,x2)`.

The F- and Chi-Square distributions also have similar properties. If the **test statistic** exceeds the **critical value** at a chosen significance level, the **p-value** is low as well. These exact numbers depend on the distribution, though. People used to have to look up tables for specific degrees of freedom and significance levels to find the critical values; now software will give you everything you need, including precise p-values.

Regression analysis often tests the hypothesis that a coefficient equals zero. If you reject the null, a significant relationship between variables exists.

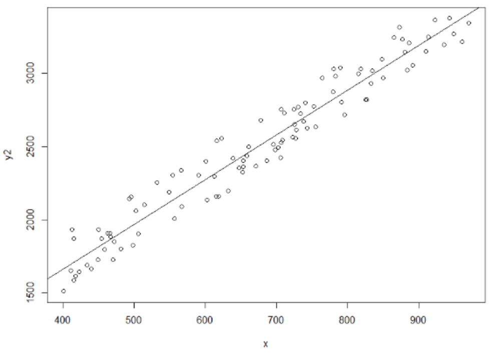
The file **Lec08_ttest.R** shows an example of the t-test.

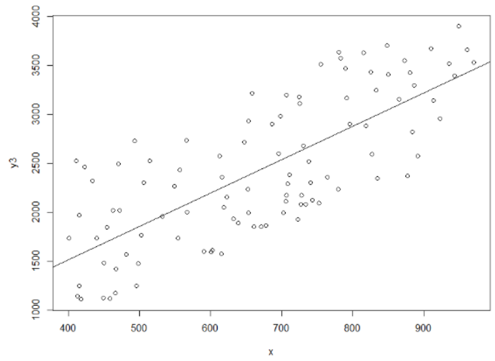
6. Advanced Topics: Regression and Time-Series Analysis

I cover these final two topics in a separate econometrics course (with equal time given to regression and time series). Oftentimes, regression analysis is covered during the final week or two of an introductory statistics course. I don't do much more than that here.

The word "**regression**" implies the concept of "regression toward the mean," where values should revert to some sort of predicted value. Here, we focus on predicting some sort of linear relationship between variables. In the bivariate case, we can think of an economic relationship, such as one where larger houses in a neighborhood should cost more than smaller ones. We might find a positive relationship between the two. But since there is more to a houses' price than its square footage, we would need to introduce additional variables (such as school quality, for example) in a **multivariate** regression.

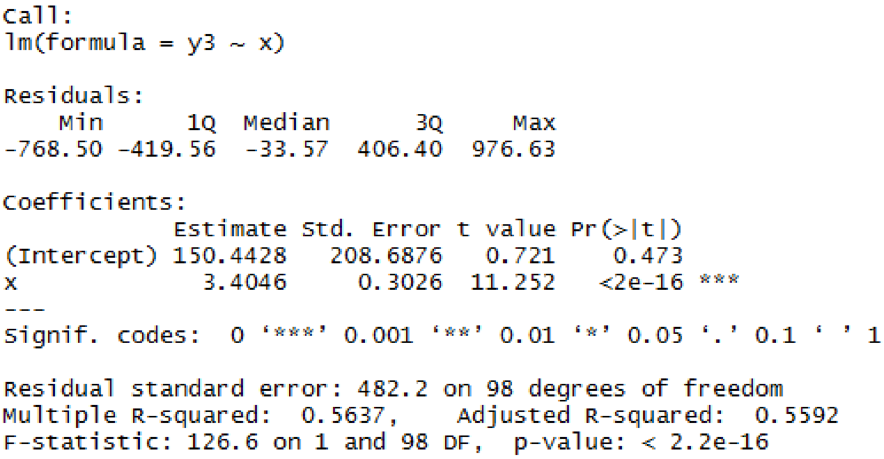
Another way to think of regression is to uncover the slope and intercept of a line: $y = mx + b$. But rather than drawing the line, you use real data to find it. And since there is a lot of variation in real-life data, the data points appear "fuzzy." A regression equation is written as $y = \alpha + \beta x + \varepsilon$. These two graphs show the same original equation, with different amounts of error.





Finding the slope and intercept can be done via **Ordinary Least Squares** (OLS). This essentially finds the "best fit line" through the points. There is a gap (the "error" or the "residual") between each point and the line; all these distances can be squared and summed together. The OLS line minimizes the sum of squared residuals. Using squared values means positive and negative errors do not cancel out, and it also penalizes large errors. In practice, the OLS coefficients are found by minimizing a matrix operation; advanced students might recognize that $\hat{\beta} = (X'X)^{-1}(X'Y)$.

The R formula for the "linear model" is `lm(y~x)`; you might wish to assign the object a name. For a multivariate model such as $y = \alpha + \beta x + \partial z + \varepsilon$ it would be `reg1<-lm(y~x+z)`, for example. Then look at `summary(reg1)`. The results for such a regression performed in R are:



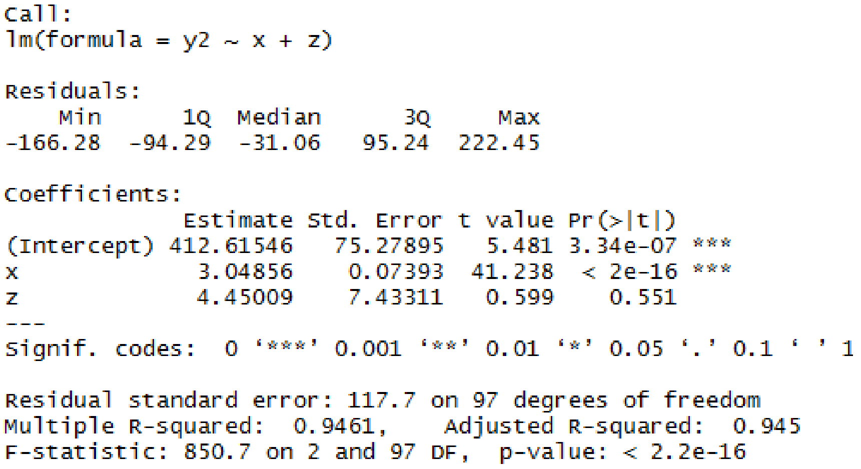
```
Call:
lm(formula = y3 ~ x)

Residuals:
    Min      1Q  Median      3Q     Max 
-768.50 -419.56  -33.57  406.40  976.63 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 150.4428   208.6876   0.721    0.473    
x             3.4046     0.3026  11.252   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 482.2 on 98 degrees of freedom
Multiple R-squared:  0.5637,    Adjusted R-squared:  0.5592 
F-statistic: 126.6 on 1 and 98 DF,  p-value: < 2.2e-16
```

The most important elements of this output are the **coefficients** (the α and βestimates) and their **standard errors** (which are conceptually similar to, but not exactly the same as, standard deviations). The **t-value** is for a hypothesis test that $\beta = 0$, or $\frac{\beta - 0}{s.e.}$; you will notice that this value is the first number divided by the second. The fourth column is the p-value associated with the t-statistic.

We reject the null hypothesis that $\beta = 0$ for the variable x, since $t > 2$ and $p = 0$ (it is written in exponential form, so basically it is a decimal followed by 16 zeros). But we reject the null hypothesis that the intercept $\alpha = 0$. The t-value is below the critical value, and it is not significant at 5 percent.

For an introductory class, the other number I focus on is R^2. Here, it is 0.56, which is fairly high (again, this is randomly-generated data). This statistic is the "explained sum of squares" divided by the "total sum of squares": $R^2 = \frac{ESS}{TSS}$. TSS is the sum of the "residual sum of squares" (RSS, mentioned above) plus ESS. In essence, R^2 captures how well the model works: What fraction of the variance it explains out of the total. It can range from zero to one. Many of the other statistics here get omitted from a final regression table that gets published. You might wish to extract certain elements from your regression results, for example using `reg1$coefficients`.

Here is some output from a slightly different regression, which also includes a second explanatory variable. It is not significant, but the intercept is. It has good explanatory power, as R^2 is high.



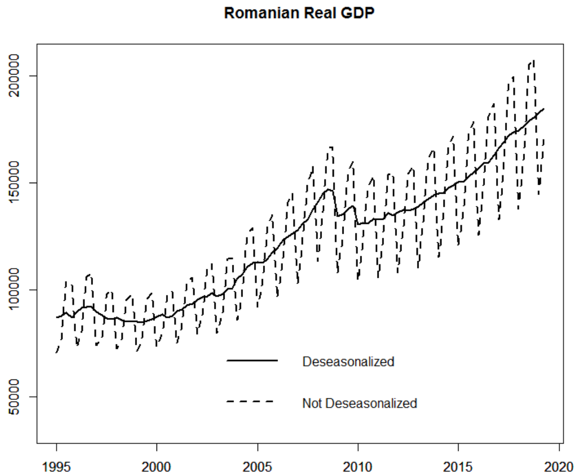
```
Call:
lm(formula = y2 ~ x + z)

Residuals:
    Min      1Q  Median      3Q     Max 
-166.28  -94.29  -31.06   95.24  222.45 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept) 412.61546   75.27895   5.481 3.34e-07 ***
x             3.04856    0.07393  41.238  < 2e-16 ***
z             4.45009    7.43311   0.599    0.551    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 117.7 on 97 degrees of freedom
Multiple R-squared:  0.9461,    Adjusted R-squared:  0.945 
F-statistic: 850.7 on 2 and 97 DF,  p-value: < 2.2e-16
```

After performing regression analysis, it is a good idea to plot residuals and to conduct diagnostic tests. Good econometric work involves a lot of testing, re-estimating, and re-testing. An Econometrics course will also spend a lot of time on **heteroskedasticity** (unequal variance caused by a relationship between the error term and some variable) and **autocorrelation** ("self correlation," often between a variable and its past values). These can be corrected easily by using **robust standard errors**: `summary(reg1, robust = TRUE)`. I talk much more about autocorrelation in time-series analysis.

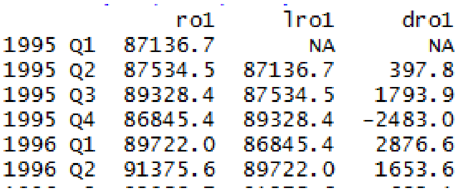
After an overview with **Lec09_Regression.R**, I test the "Rank-Size Rule" in Urban Economics with **Lec10Regression-RankSize.R**, and then incorporate a loop to test this relationship in **Lec10bLoops-Regression.R**. In Fall 2021, I will also show an alternative estimator with **Lec09b_nonparbeta.R**.

The topic of **Time Series** can easily take whole course by itself, at any level. Here, I talk about basic time-series properties and show some common issues and some statistical procedures and tests. This overlaps with my “Macroeconomic Data Analysis” class, where I used R as an optional software for a subset of interested students. I use some of that material here. You can generate a time series using `ts()`. You will need to know its start or end point (year and quarter, month, etc), as well as its frequency (how many observations per year). Doing so allows these formulas to run and for the series to plot better. Note that (year, quarter) are in a group together: `x<-ts(x, start=c(1995,1), frequency=4)`.

Time series can be plotted like any other variable, using `ts.plot()`. This allows for multiple lines in a single graph. You can differentiate each one through colors, line type, and line width. Because I don’t want to replicate my other class, I use a different variable for discussion: Romanian GDP. For whatever reason, it has a lot of seasonality.



A **lagged variable** is simply shifted backward in time, so that last period’s value x_{t-1} corresponds with this period’s value x_t. A **differenced variable** is created as Δx or $x_t - x_{t-1}$. Typically one period is used, but not always. You can second-difference a variable by differing a first-differenced variable $\Delta x_t - \Delta x_{t-1}$: For the Romanian real GDP data, we can see the lagged and differenced variables. Note the *NA*s—there is nothing to subtract from the first observation. Sometimes the lagged variable is “shoved up” one period, creating an extra observation at the end. You can make a lag using syntax such as `lro1<-lag(ro1,-1)`. Note that lags are negative, and leads are positive. You could do 4 lags by changing the last number, often termed k. Differences can be calculated as in `dlro1<-diff(ro1)`.



```
            ro1     lro1     dro1
1995 Q1 87136.7       NA       NA
1995 Q2 87534.5  87136.7    397.8
1995 Q3 89328.4  87534.5   1793.9
1995 Q4 86845.4  89328.4  -2483.0
1996 Q1 89722.0  86845.4   2876.6
1996 Q2 91375.6  89722.0   1653.6
```

You can perform an **autoregression** by regressing *x* on lagged *x*. This might show that a variable, such as a stock price, is explained by its own past values. You might also do a **cross-correlation**, relating one variable's current value to another variable's past value. (Or, you can do this for future variables by generating a **lead variable**).

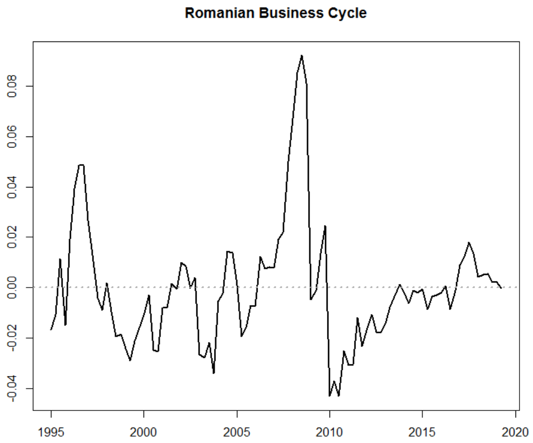
Differences are also important when a time-series variable is not **stationary**. If the variable's mean (or variance) changes with time, traditional statistical inference will not be valid. Differencing the variable might render it stationary. The number of times that a variable need to be differenced is its *order of integration*: If it is I(2), it needs to be differenced twice. If it is I(0), it is stationary already.

Stationarity tests involve looking for a "unit root," where $X_t = X_{t-1} + \varepsilon_t$ and the coefficient is 1. The most well-known is the Augmented Dickey-Fuller (ADF) test, but I use the mostly similar Phillips-Perron (PP) test: `PP.test(x)` Its null hypothesis is that a unit root exists and the variable is nonstationary. If it is rejected, the variable is stationary. If not, you should difference it.

Time series can also be decomposed into their component parts. In general, they can be written as $T\ x\ C\ x\ S\ x\ E$, with a **trend**, **cycle**, and **seasonal** component. The fourth is the error component. While there are a lot of classic methods to do this, I generally do this in R by removing seasonality (if I do not already have seasonally-adjusted data) and then extracting the cyclical component. These steps can be using R **packages**.

I am introducing packages for the first time here, but that does not mean you might not need or want them sooner. I try to utilize multiple concepts at once, and here is a good place to add this one. Packages are basically mini-programs that do something that base R does not. They are submitted by users and developers, tested, and made available for download and installation. Here, I will use the *seasonal* package, which applies the Census-X13 method to deseasonalize a time series. (I have used the *stl* function in the past, but I like this package a little better).

You may need to install the package as in: `install.packages("seasonal")` and then load it with `library(seasonal)`. Then you can run `seas(ro1)`, for example. Next, the package *mFilter* allows you to filter the data and extract the trend and the cycle. The best-known of the "miscellaneous" filters available is the Hodrick-Prescott filter. I sometimes calculate business-cycle correlations for various country pairs by calculating *cross-correlation functions* between one country's filtered log real GDP and the lead and lag values of its neighbors. High correlation coefficients at the same time indicate that the countries' economies are synchronized.



Two more advance topics that I cover mainly in Econometrics class are ARIMA analysis, which stands for Autoregressive Integrated Moving Average. It is a univariate method of analysis that forecasts a variable based only on its own past values. Unlike a typical regression, it needs no additional explanatory variables. We have seen the terms *AR* and *I* before; *MA* uses present and past errors to forecast the variable. I then incorporate volatility modeling with a GARCH (generalized Autoregressive Conditional Heteroskedasticity) process, which treats ε in the regression as time-varying rather than a pattern-free "white noise" process. While I show how do to this in some videos, I usually don't cover it in class.

A number of videos from my Macroeconomic Data Analysis course cover time-series topics: **343_Lab1_R.R**, **Deseas_CPI.R**, **R_Cycles.R**, **Autoregressions_R.R**, and **GARCH_JPY.R** are a start. These are available at bit.ly/2SlGXU0. Obviously, R can be used for many more advanced projects. This course is just a start; hopefully the skills used here can be used well into the future.