Module 3: Assembling Data

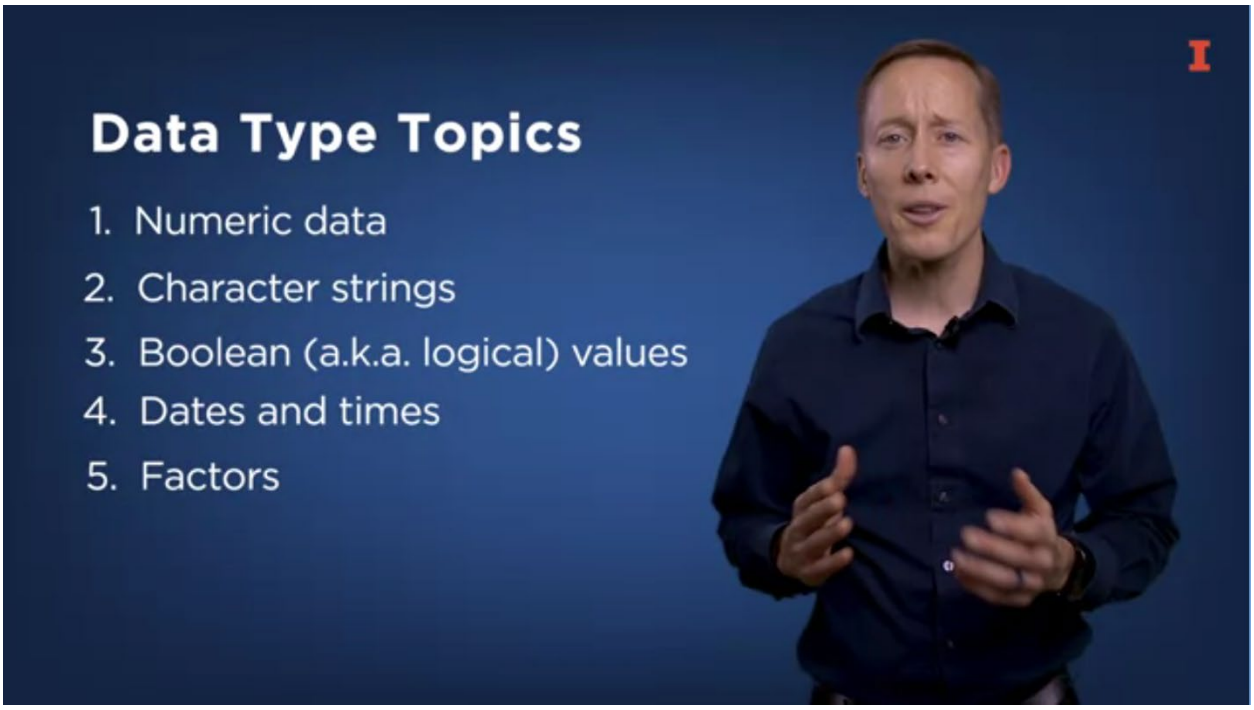
Table of Contents

Module 3 Overview

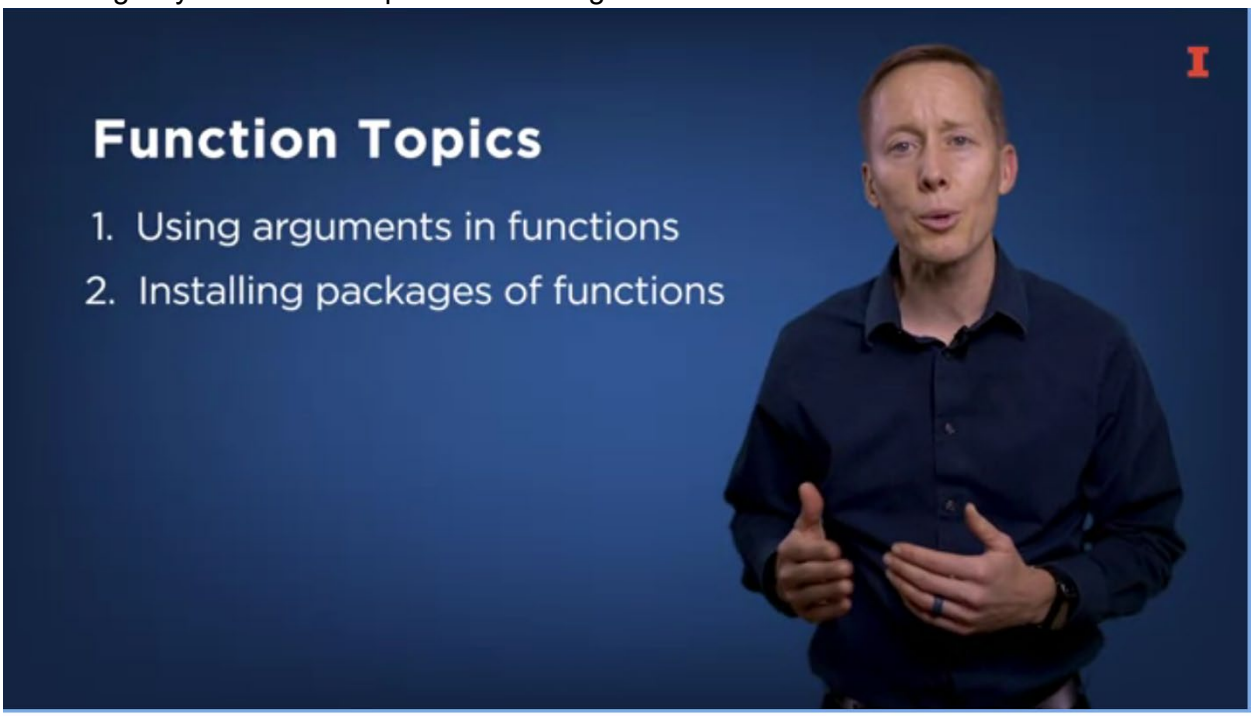
Module 3: Introduction

[MUSIC] What stands out to you when you see this photo studio? To me, it's the lights. By my count, there are at least 16 lights placed in strategic places around the white backdrop. It also looks like there are at least three different types of lights, umbrella panel and some other kind. The point is that photography is a whole lot more than pushing a button on a camera. Specifically, before photos are taken, a lot of time is spent setting up the lights and creating the right environment. Well, just as photography deals with a whole lot more than pressing a button on a camera, business analytics deals with a whole lot more than just running an algorithm.

One of the key elements that takes place before the algorithm is run is assembling the data. In this module, we will begin by covering some general aspects of assembling data for business analytics, such as where to find data and different approaches for organizing it. There are many ways to organize data, but for our purposes we will stick with tabular data in which an observation is a row and a characteristic is a column. After discussing some general issues, associated with assembling data, we will then focus on the conceptual topic of data types.



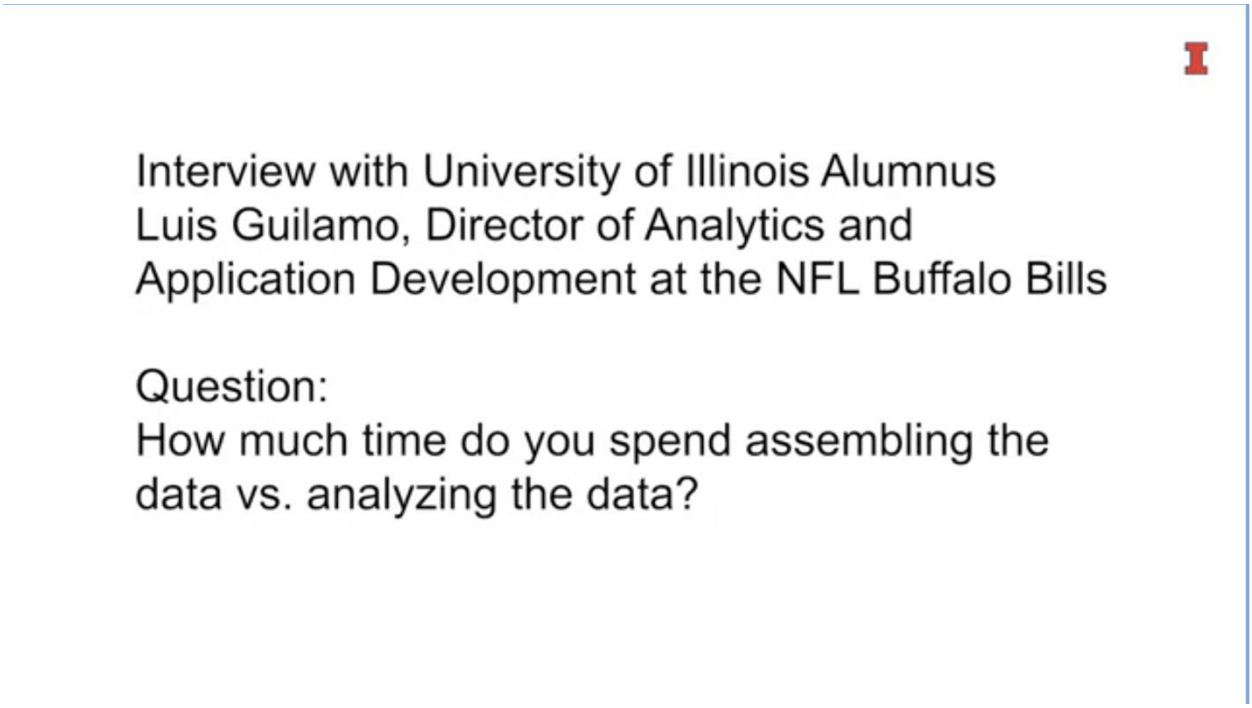
Numeric data is probably the most obvious data type. Numeric data includes data that is stored as an integer or a fraction. You see this a lot in business data. However, it's important to recognize that other data types like character strings, Boolean or logical values, dates and times and factors. These other data types allow you to aggregate data in different ways, such as by product line or, by a time period, like a month or a year. Moreover, Text Analytics is an important topic in business in and of itself. Text Analytics makes it possible to capture trends in social media, business news and managers reports. Text Analytics is also used for creating chatbots. In short, it's important to understand some issues with non numeric data types, so we will also give you firsthand experience working with them.



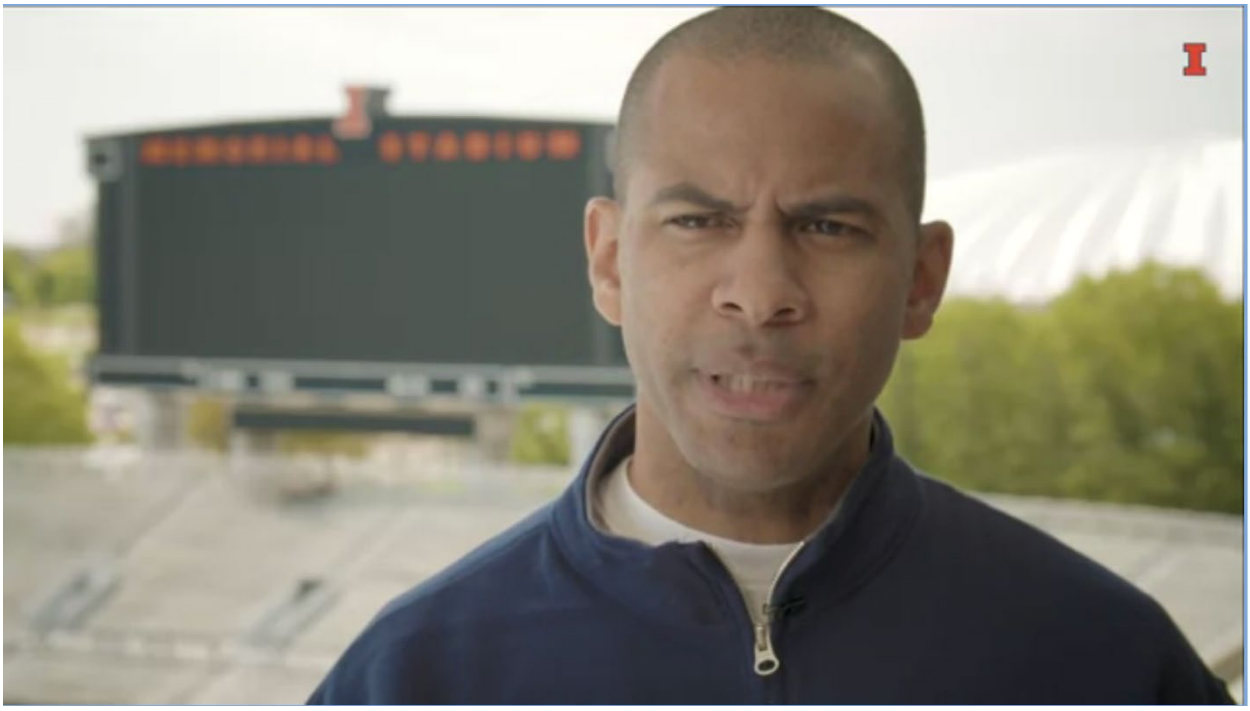
Finally, we will take time to take a detailed look at functions, because they will make it easier to process numeric and non numeric data types. Just like an Excel, functions in a data analytic language allow you to perform a task by typing the function name and then entering the argument values. We will also show you how to install and use packages or modules, which contain functions that are used for performing interrelated tasks. For example, there's a package that contains functions to process date and time data, and there's another package that contains functions that are used for processing character strings. As you get into the data assembly process, remember that it's not unusual to spend time preparing data before it can be analyzed. In fact, sometimes great insights come as a result of the effort that goes into assembling the data. [MUSIC]

Lesson 3-1 Assembling Data

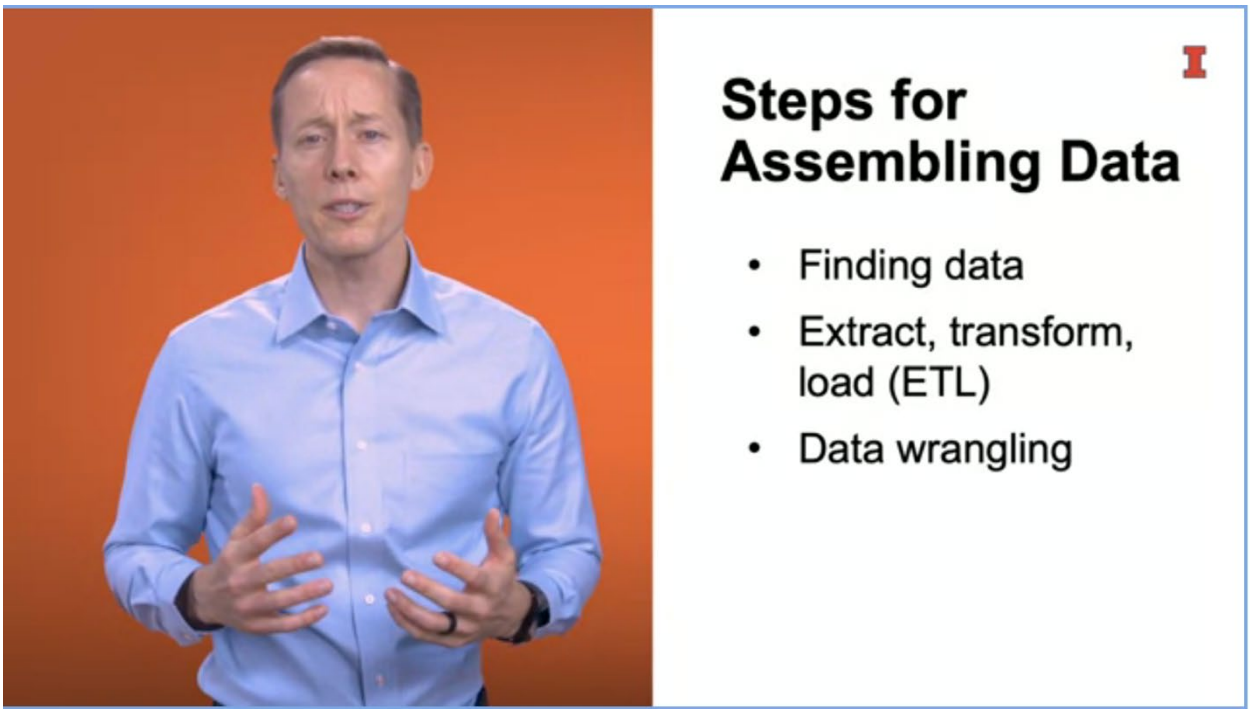
Lesson 3-1.1 Assembling Data



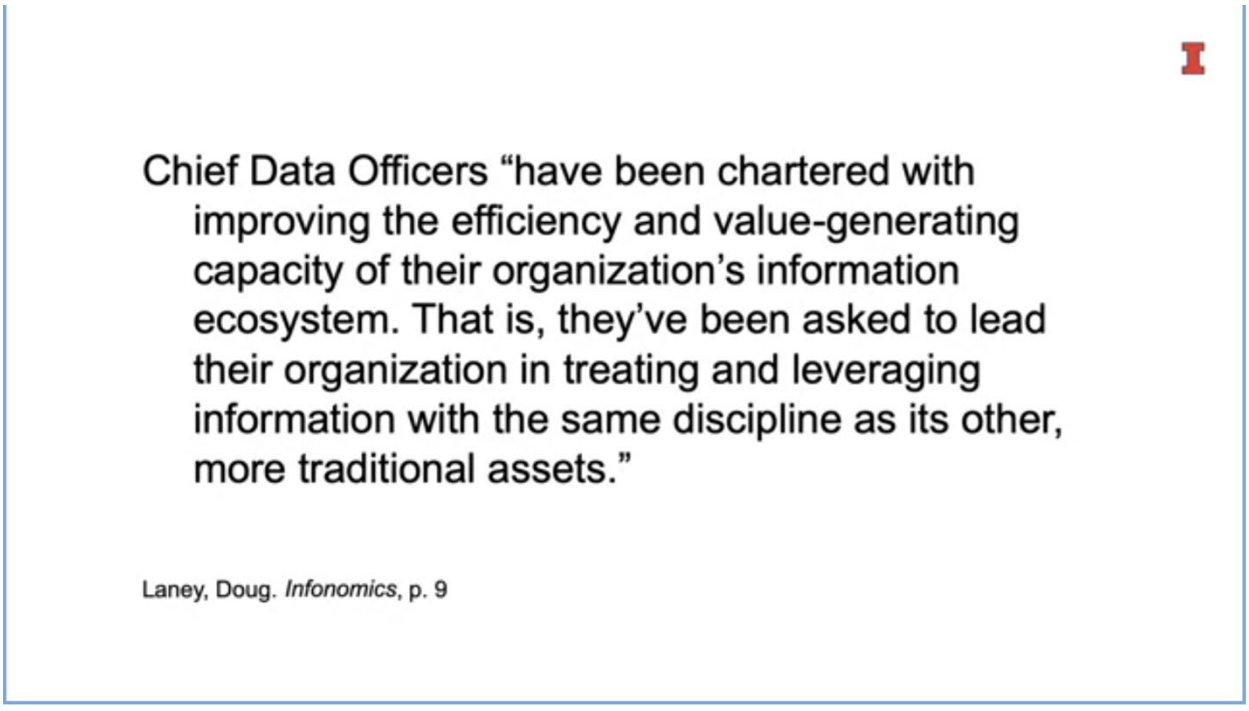
In this lesson, you'll learn more details about assembling data. The second step in the fact framework. Assembling data is critical because it often has a direct impact on how effectively the question can be answered. Interestingly, it often requires more time than the actual calculation of the results. I asked Luis Guillermo how much time he spends assembling the data versus analyzing the data. Here's his response.



Yeah, the amount of time that I spend and data preparation is about 80 percent of the time where 20 percent is dedicated to actually doing calculations. Obviously that can vary from time to time. But the main problem is data quality, data accuracy, making sure that it's correct, that it's shaped in the way that you need it to be able to do your analytics, ensuring that you have everything that you need essentially. That's really the key part. Once you have everything in the form that you need it, putting a model on top of it is very simple. If you're adept at interpreting those models, then you'll go quite far, quite quickly. I think it's clear, at least from Louis perspective, the assembling data takes a lot of time. Let's talk about why that's the case.

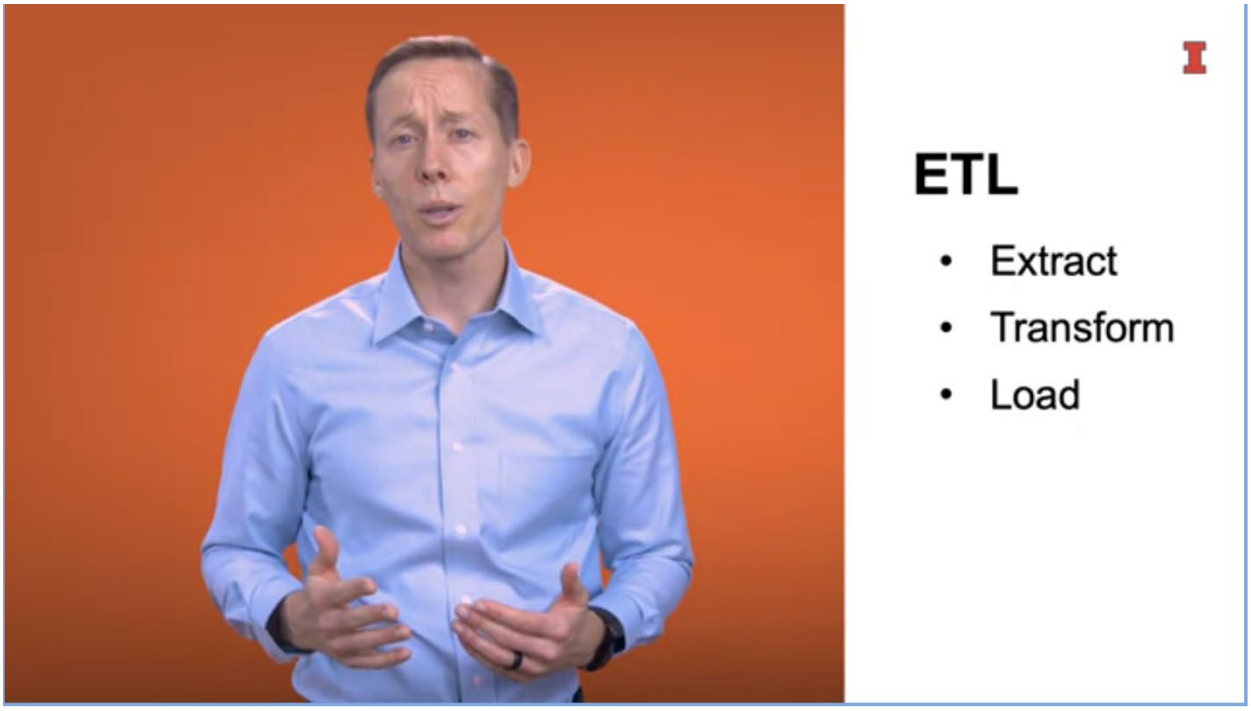


Assembling data typically involves several steps. Finding data, extracting the data, transforming the data, and loading the data, or ETL and data wrangling. The importance of finding relevant data can't be overstated. Data is often protected for safety and regulatory reasons, as well as to protect trade secrets. Oftentimes people don't know what data sets exist, even within their organization. Or even if employees know that a data set exists and may be hard to access due to security and privacy concerns. For instance, it's unlikely that the HR department will share employee salary information with the marketing department. Many companies now employ chief data officers or CDOs.

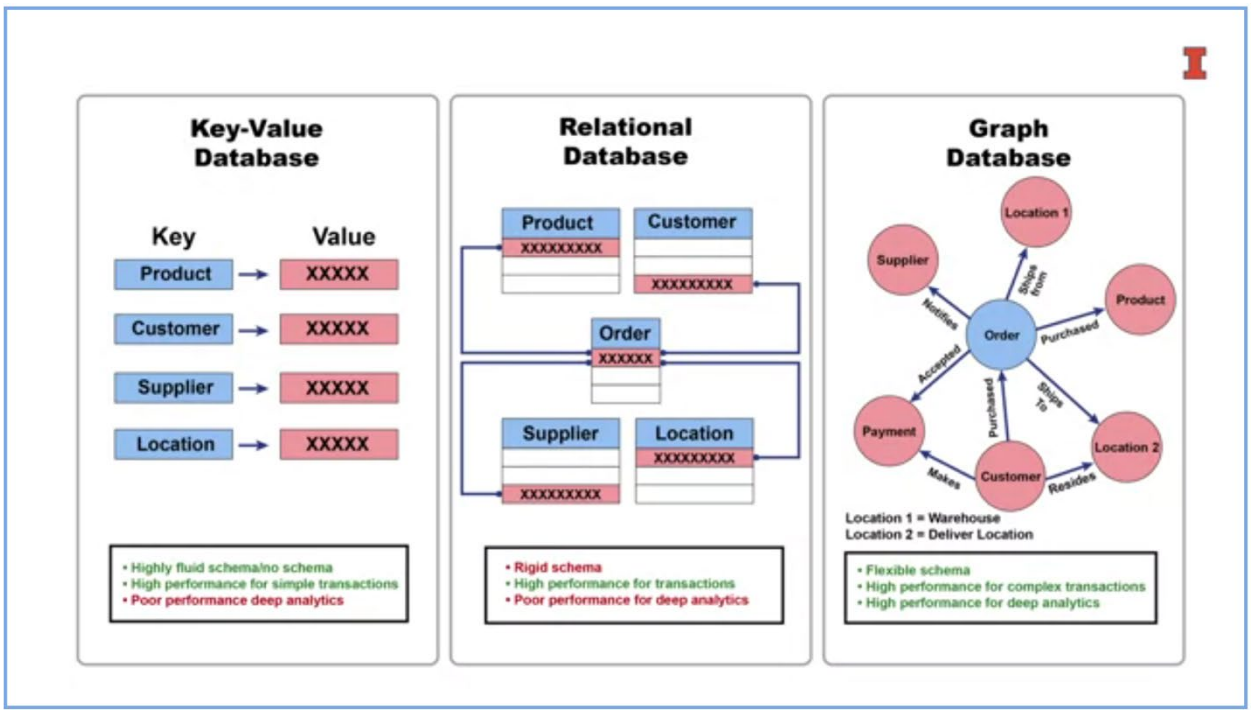


In his book in phenomics, university of Illinois Alumni, Doug Laney suggests that most chief data officers have been chartered with improving the efficiency and value generating capacity of their organization's information ecosystem. That is, they've been asked to lead the organization in treating and leveraging information with the same discipline as its other more traditional assets. Thus, CDOs should not only keep track of data that's available to employees of the organization, but they should also make sure that the right employees know how to find out what data is available, as well as how to get access to it. Data sources may include internal and external data. For instance, if you're a manager who's trying to find out the factors that caused a decrease in sales for various regions. Then you'd probably want to make sure that you have information related to sales and location. But that's not all. You'd also want to gather data about potential factors that could influence sales for each region. Perhaps you'd want to find wheather data and data related to the types of homes, like single-family homes two family homes, and number of rooms per home that are in each region.

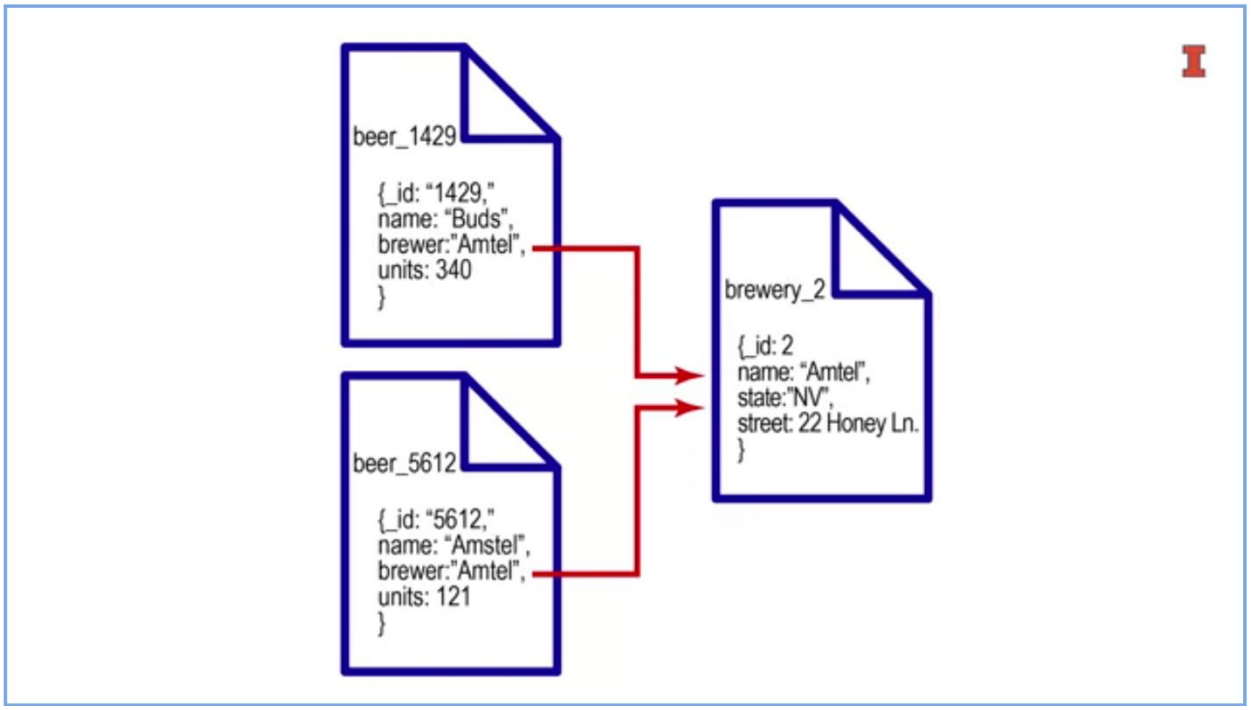
There are thousands of publicly available data sets that could be considered. Governmental agencies, at least in the United States, make many data sets available to the public, including weather data, financial data for companies whose stocks are traded on US exchanges, census data, tax data, and health data. You can browse thousands of data sets on www.data.gov. There are many other interesting data sets available for many other websites as well. If the question is important enough, you'll want to gather your own data. Gathering your own data can be done in many ways. For instance, you could create a survey to gather your own data. Or you can create a web crawler to gather data that is dispersed on multiple websites. Or you can start measuring something that previously hadn't been measured. Once you've identified a data set, you'll want to extract data from where it resides, transform it into a structure that meets your needs, and then load it into a data processing tool.



This is frequently abbreviated as ETL for extract, transform, and load. This is important because data repository store data in a variety of formats and then return data to the user in a different variety of formats. Typically, you'll want to transform the data to a table format so that you can perform the visualization or analysis. However, other formats also exist, such as graph database format. If you wanted to analyze graph data and you'd want to transform the data accordingly.

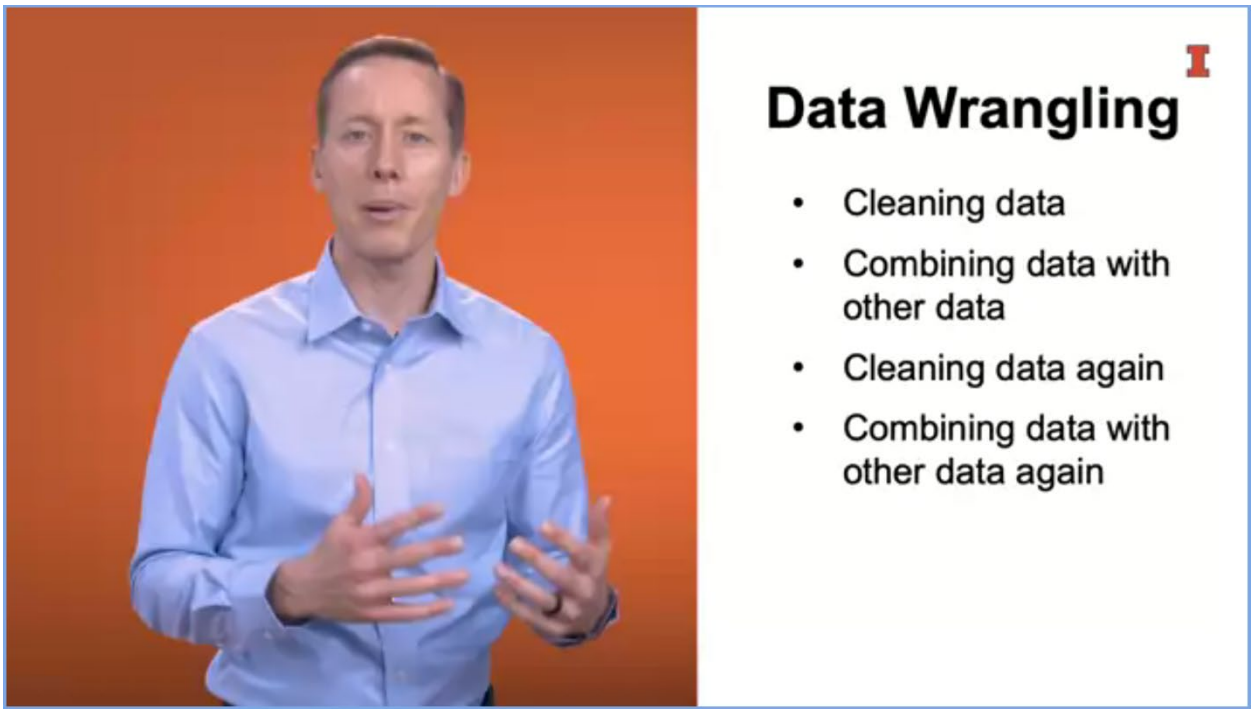


The most commonly known data repository is a relational database, which stores data in multiple tables. A Structured Query Language, or SQL, is often used to extract the desired subset of data which is then returned in a table format. This is nice because the data is already in a table format and it's ready to use. Here's some other ways in which data is stored and extracted. If you're gathering data from websites, then you'll need to gather HTML or XML data, which has lots of tags to identify the different pieces of data.



You'll then have to extract the key pieces of information and then transform it to a table format. Sometimes data is stored in a relational database, but it's accessed using an Application Programming Interface or API, which has a structure like a website rather than a SQL query. The data is often returned in JavaScript Object Notation or JSON format. This format is a series of embedded lists separated by curly braces, colons, and commas. As with HTML and XML data, the key pieces of data would need to be converted into a table format before it can be analyzed. Once the data is in a table format, you'll most likely need to then go through a data wrangling or data munging process.

This process includes cleaning data, combining data with other data, cleaning it again, perhaps combining it again, and then cleaning it again and then changing its shape. While I realized that you probably don't appreciate many of these terms and this process right now, I hope you get the idea that it's not a straightforward linear process.

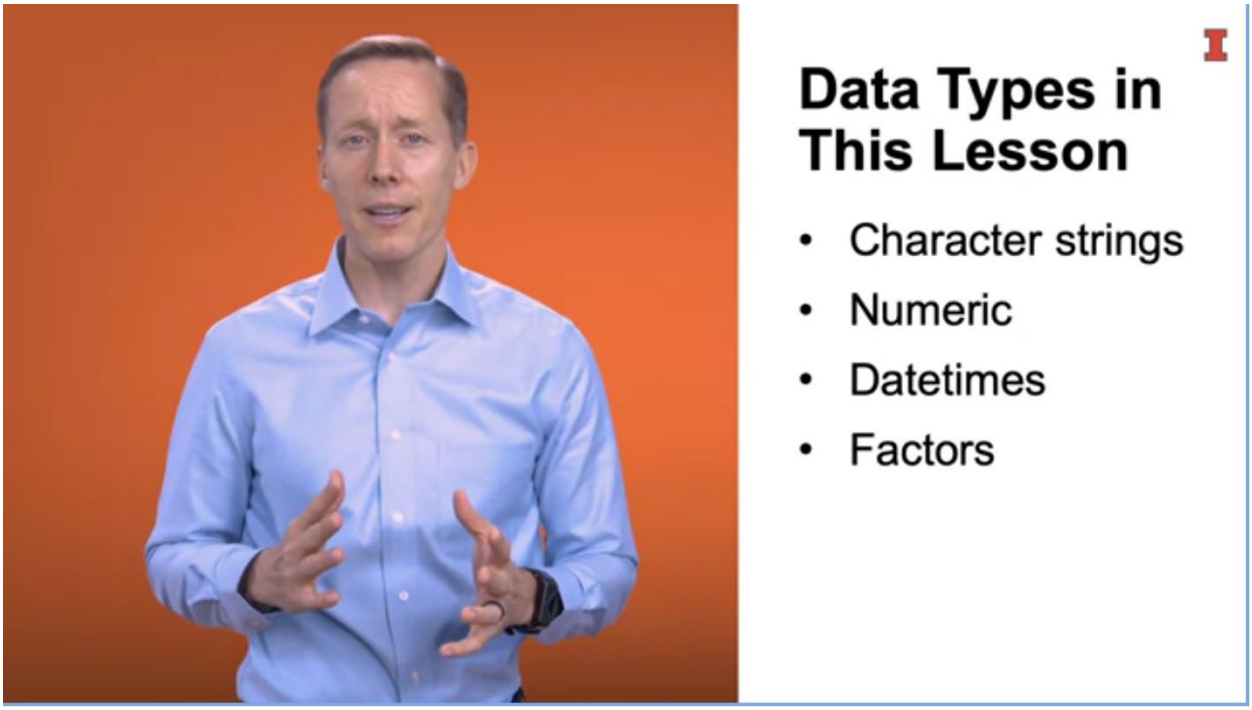


These steps associated with assembling Data are often iterated. During the data wrangling process, you may realize that some of the data you found has too many errors. You may decide

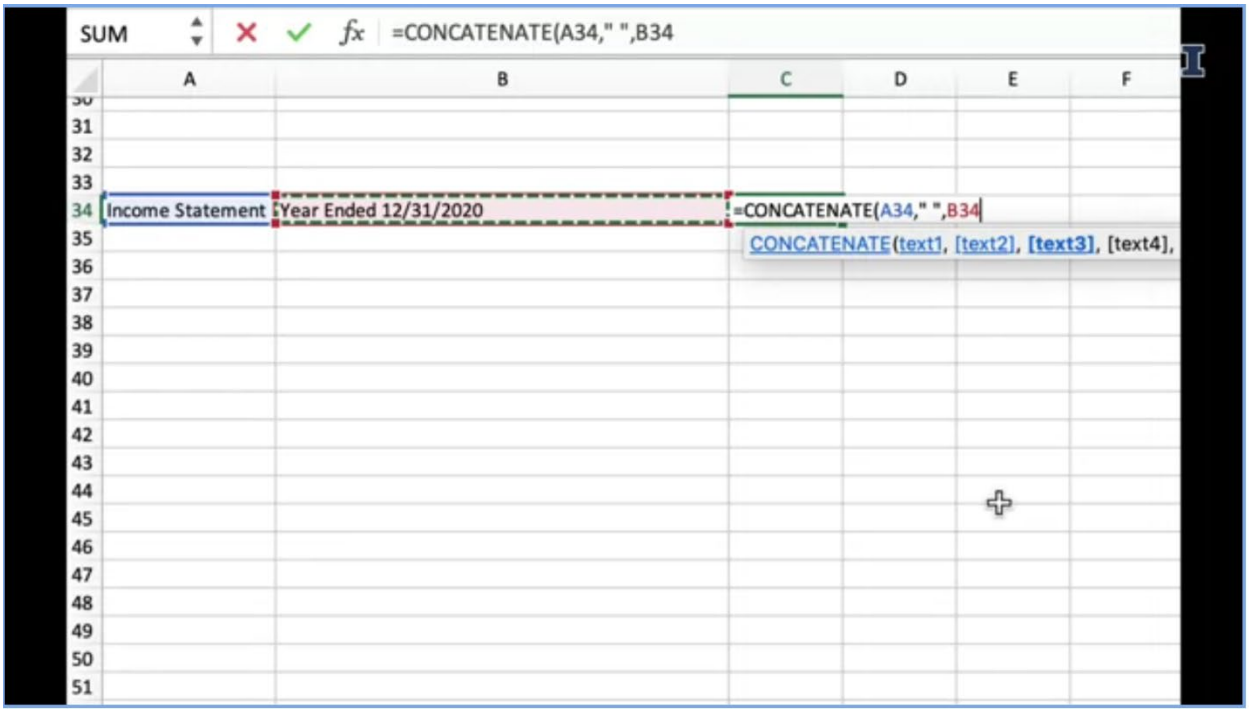
that you need to find a different data set and then you start the process over. The underlying point is that assembling the right data is so important and it often requires a lot of work.

Lesson 3-2 Data Types

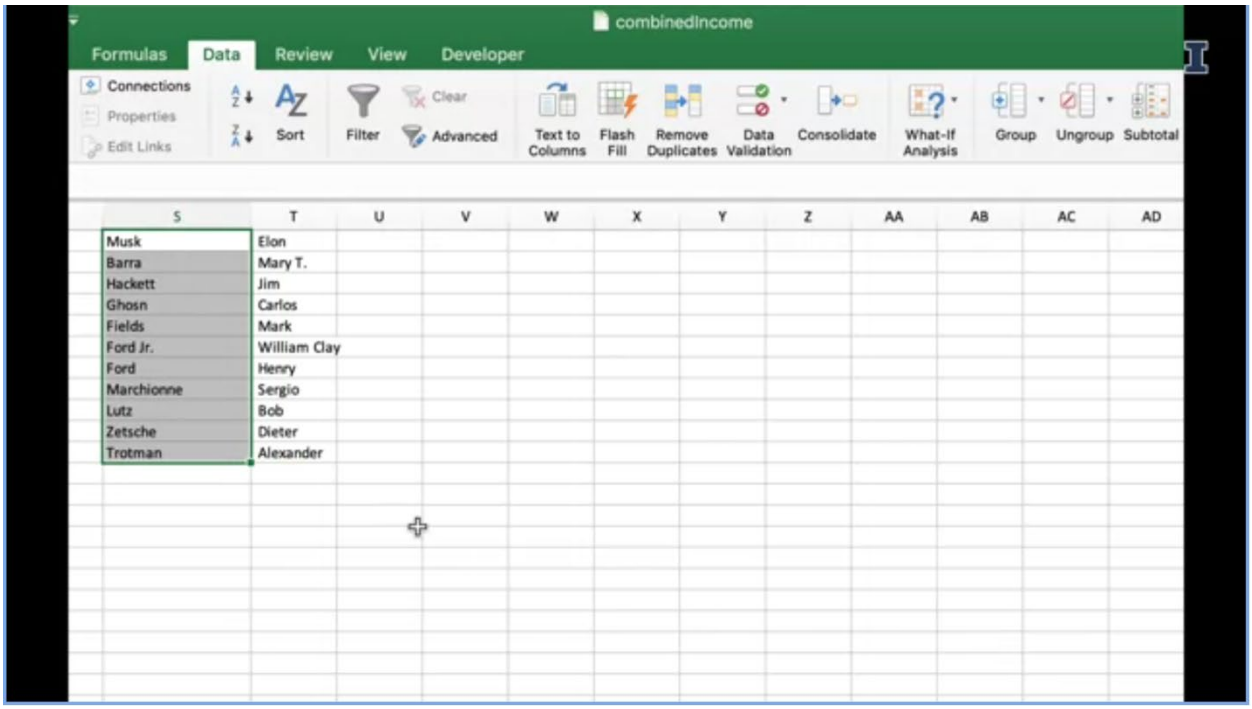
Lesson 3-2.1 Data Types



As you've probably noticed, it's imperative that all values within a column of a data frame are all the same type. There are a variety of data types such as character strings, booleans, factors, numeric, integers, and dates. In this lesson, we'll talk about four different data types; character strings, numeric, dates, and factors. Let's start with character strings. Character strings are simply text characters that are strung together. In the English language, strings are most often made up of one or more alphabetic characters. However, digits can also be represented as a string. For instance, the word income is a string, and so is income statement for the year ended 12/31/2020 even though it has numeric characters in it.

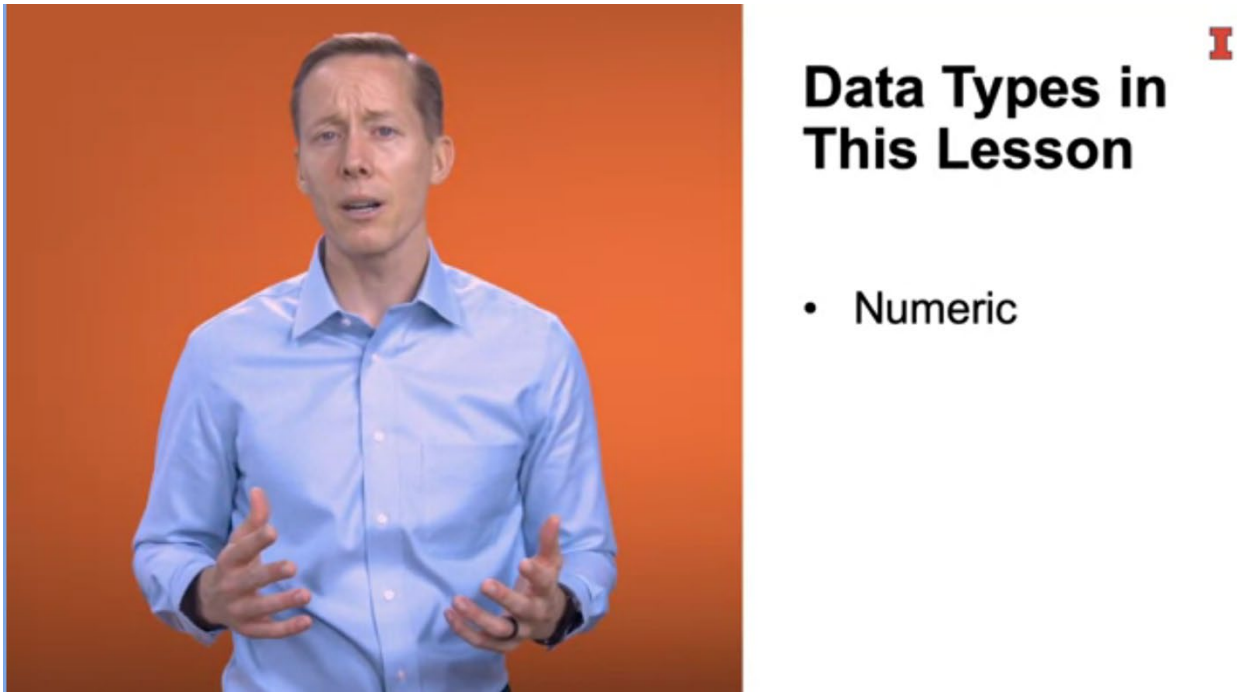


Character strings are often manipulated. For instance, they're often combined to make larger character strings. You may have done this in Excel using the concatenate function. Other times, character strings are broken apart or are parsed in a number of ways. Ideally a delimiter like the pipe symbol or a comma can be used to create sub-strings.

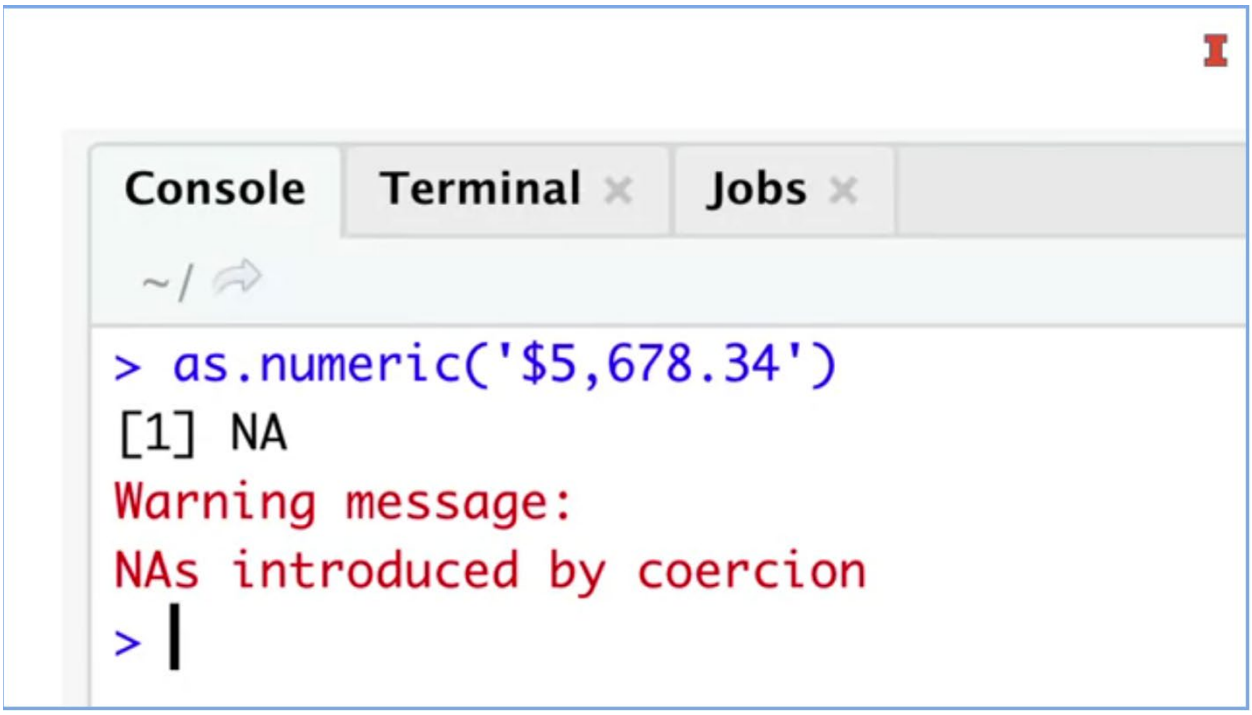


You may have used other functions in Excel to extract the first or the last characters or to extract certain patterns or sub-strings. Now let's talk about numeric types. You're already

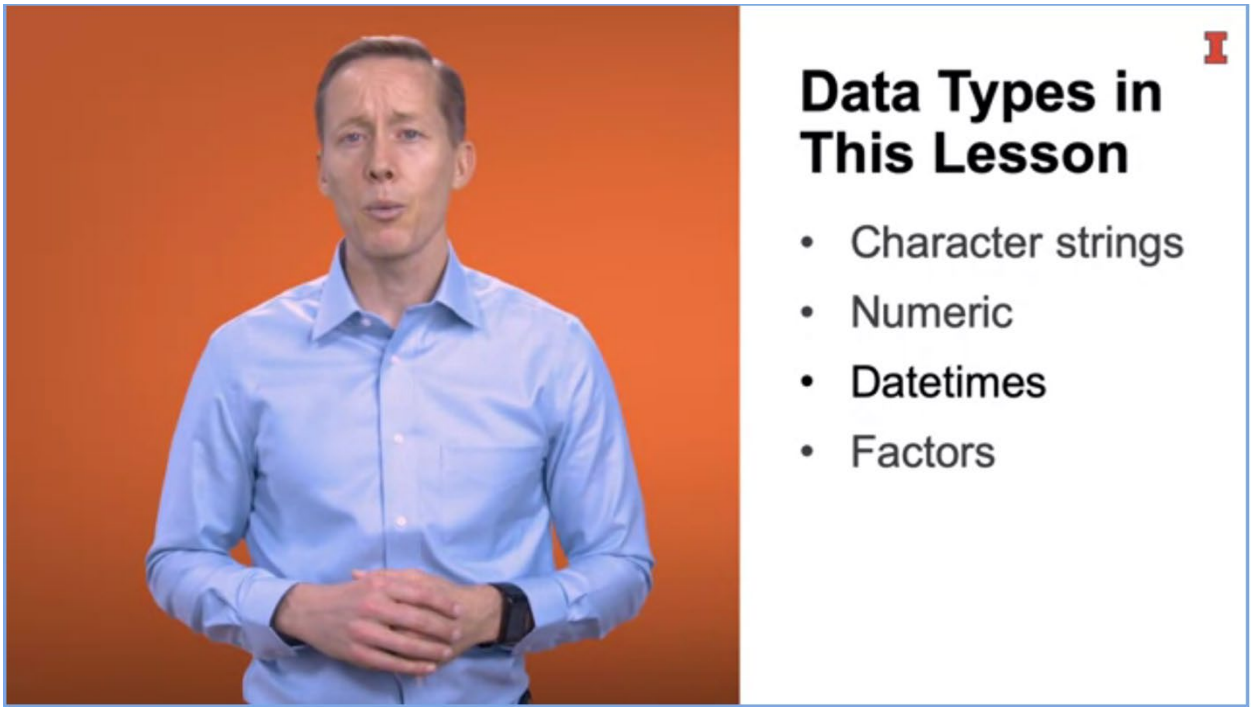
familiar with this data type. This is probably the most important data type because all data science is based on numeric data, including image detection and natural language processing.



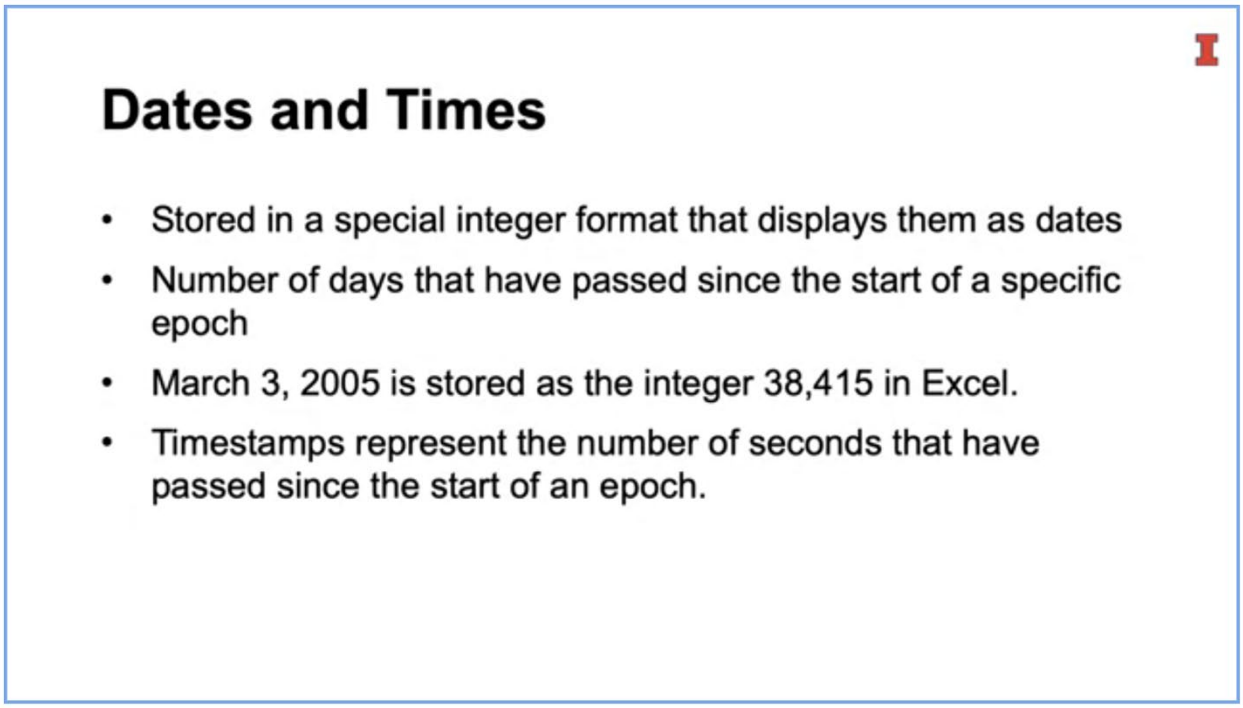
Essentially, all data needs to be converted to a numeric value before it can be used in an algorithm. Business data obviously has lots of numbers in it. However, one issue that often occurs is that numeric data can be read in as character strings.



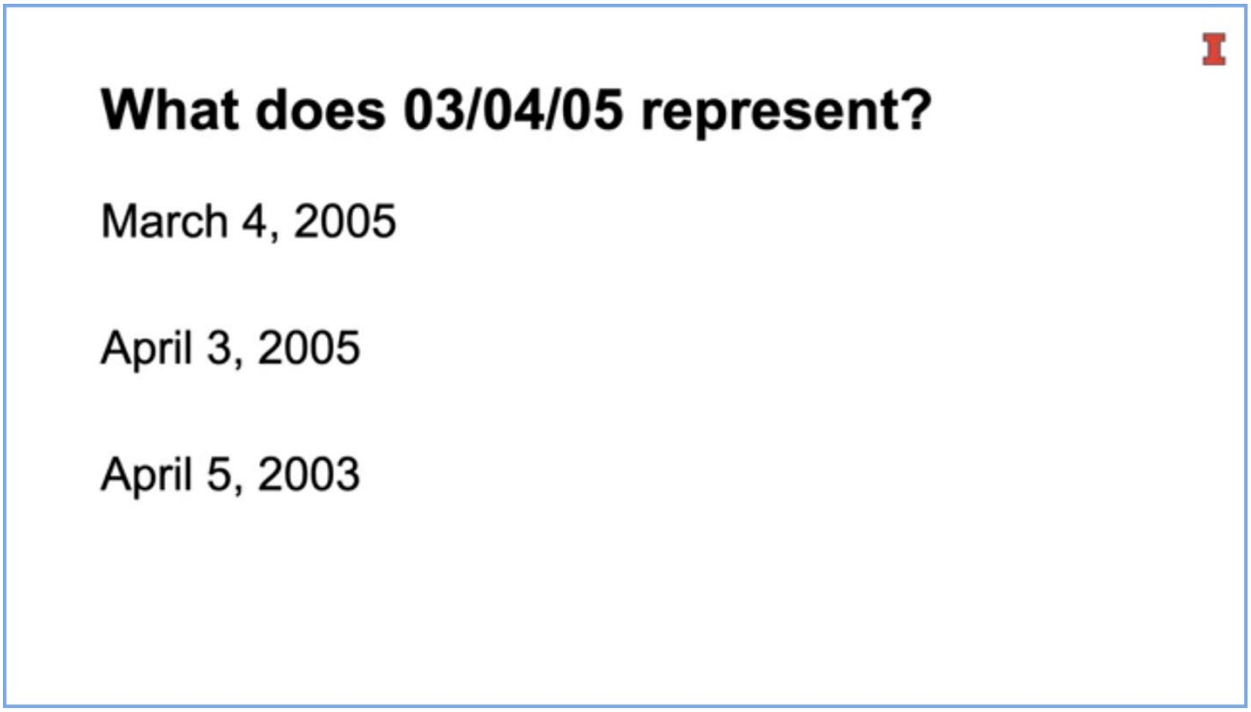
This is because we often display numeric data in the form of currency, which includes a currency sign as well as commas after every three digits. This obviously makes the data easy for humans to read and process. However, machines can't perform mathematical functions until those strings and commas are removed and they're converted to a numeric type.



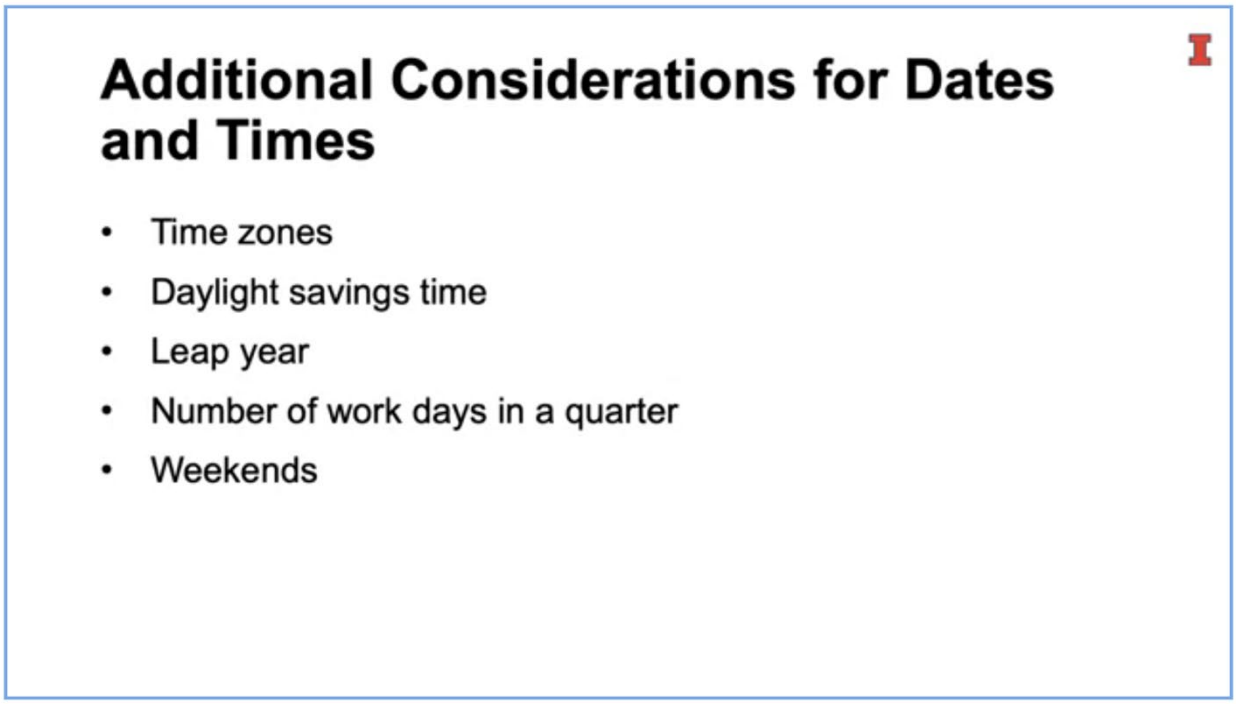
The third data type that I want to discuss are date-time types. Dates and times are stored in a special integer format, even though they're often displayed as strings. Specifically, dates are stored as the number of days that have passed since a specific reference state or epoch. Excel uses the beginning of the 20th century as the epoch. It would encode the date of March 3rd, 2005 as 38,415 because that's how many days have passed since the 20th century began at midnight on January 1st, 1900. In R, and I believe most programming languages, the reference point or epoch is January 1st, 1970.



Timestamps are recorded in a similar way as dates, except that they are based on the number of minutes, seconds, or milliseconds that have passed with respect to the epoch. At any rate, if you want to be able to do any calculations with dates, like calculate how many days are between them, or what the date will be in x number of days, then you'll need to make sure that they're stored in a date format. Another related issue with dates is that they can be displayed and recorded using a variety of patterns.



For instance, 03, 04, 05 could be referring to at least three different dates. March 4th, 2005, April 3rd, 2005, or April 5th, 2003 among others. As long as dates are recorded using a consistent pattern, then you can use built-in functions that quickly and easily convert date strings to a date-time format. Problems often arise, however, when merging data sets that use two different date patterns. When this happens, it takes creativity to find an elegant way to convert all the dates to a date type. Dates and times have many other issues. For instance, it gets more complex when you have to do with time zones, daylight savings time, and leap year.



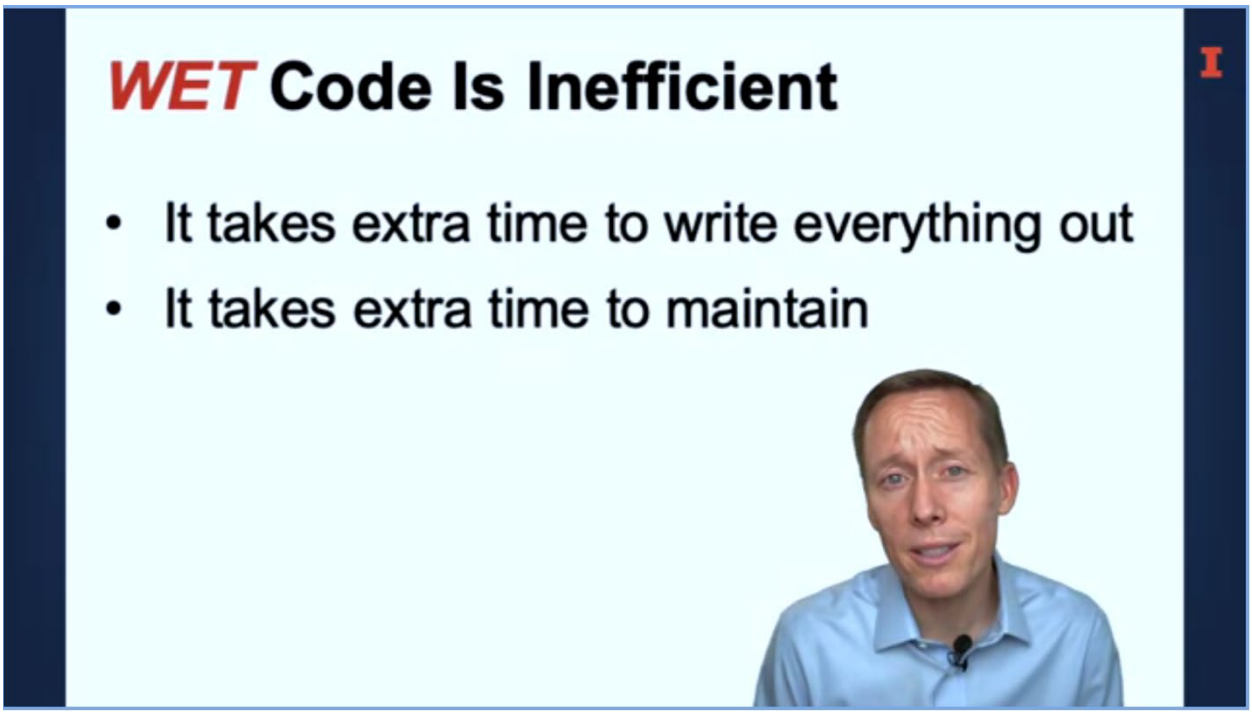
The months in the Gregorian calendar often make things tricky as well, because the number of days changes per month. For example, consider calculations relating to earnings per workday for each quarter during the year 2019. The first quarter has 90 days, the second has 91, and the last two have 92. Then you have to factor in weekends and holidays. Dates and times are complex. The last data type to discuss in this lesson is the factor type. Remember the strings as factors option when reading in data from an external file. That's the datatype that I'm talking about now, when I talked about numeric data some minutes ago, I mentioned that all data needs to be converted to a numeric value for data science algorithms. Since R is intended primarily for statistical analysis, it aggressively strives to convert data to a format that can be used in these algorithms. Thus, when our encounters a categorical variable like gender, country, or hemisphere, the default is to convert it to a special numeric type that can be used in the algorithms. How is a factor type different than a string type and a numeric type?

Well, it's like a date-time format in the sense that factors are stored as numeric types but they're displayed as a string. If a vector of strings has two categories, like male and female, then it assigns an integer to each category based on the order in which they appear in the data. For instance, if male were to appear before female than male would be assigned the number 1 and female would be assigned the number 2. But it would display the character strings of male and female, or M and F rather than the numbers. R would also report that the vector is a factor datatype with two levels. You can see then that the number of levels tells you the number of different categories in a column of data. In conclusion, datatype is often different from the way it's displayed. It's important to understand the different data types so that you can process it in a meaningful way. You already know how to deal with numeric data types. We'll spend separate lessons on dates, factors, and strings, and how to deal with them in R

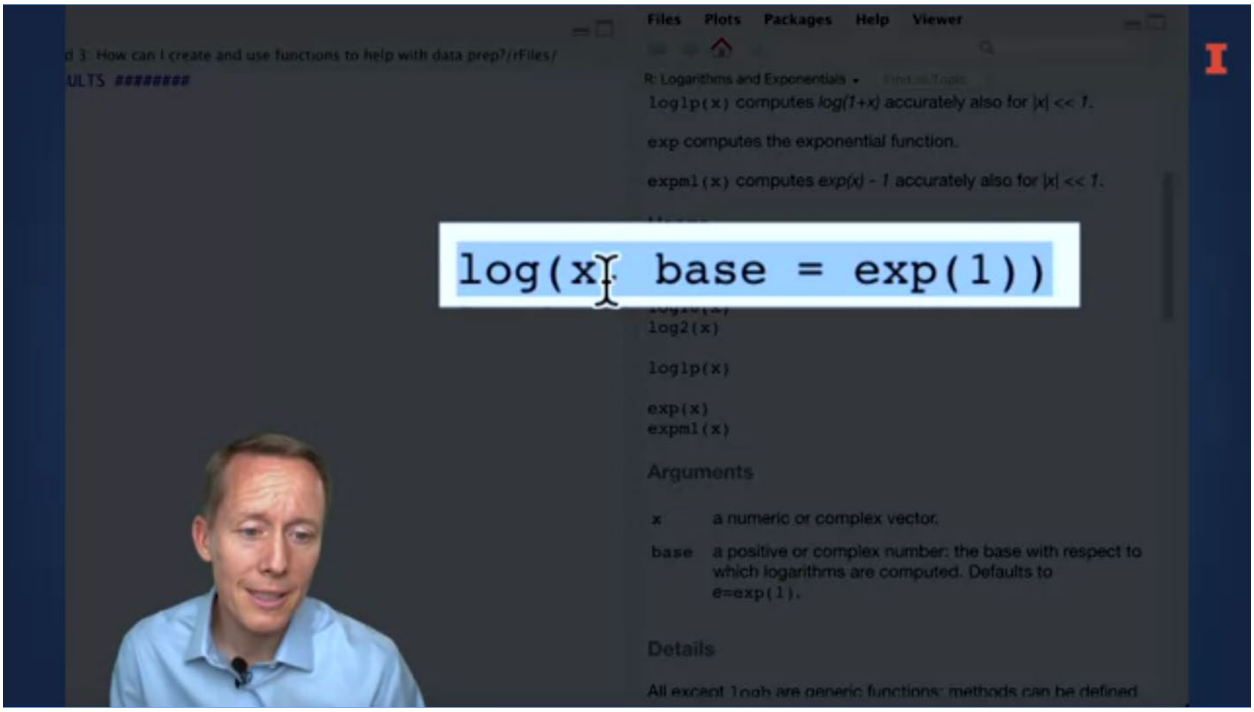
Lesson 3-3 More on Functions

Lesson 3-3.1 More on Functions

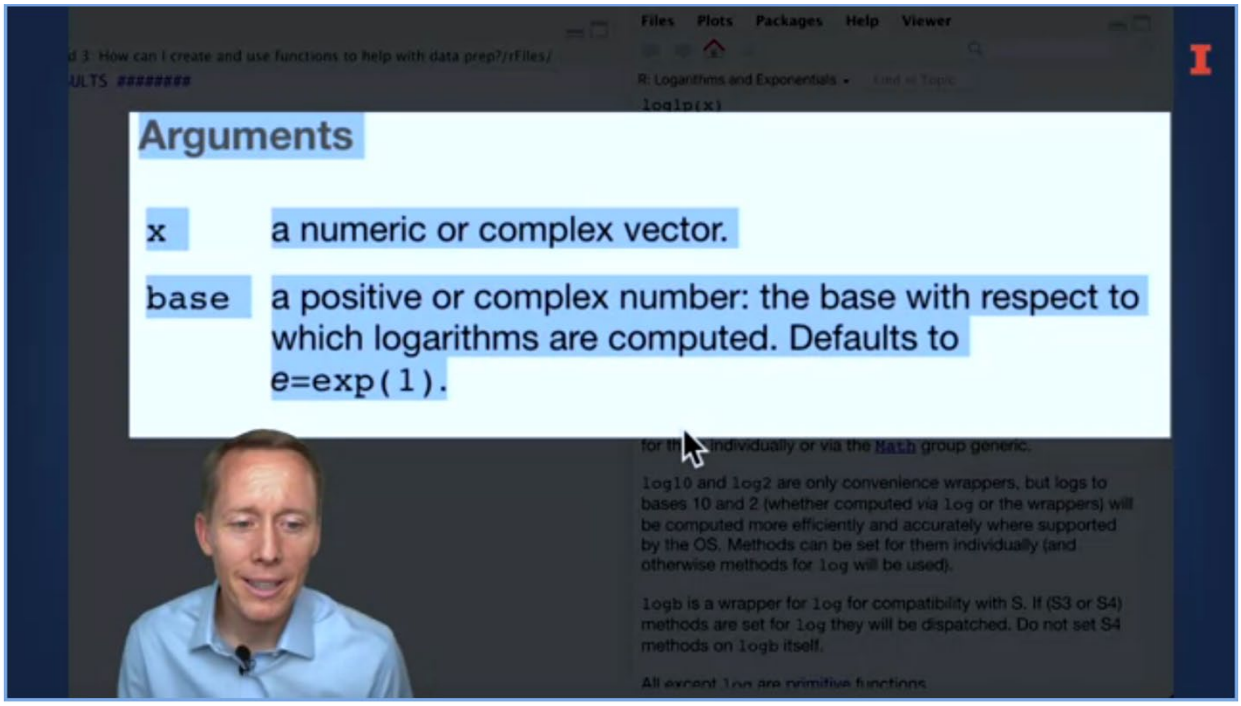
Whether you're doing with data types, reading, and data, or some other analytic process. A fundamental principle when using a data analytic language and really programming, in general, is that you should not repeat your code. This concept is often known as DRY, which is an acronym,which stands for Don't Repeat Yourself. The opposite of dry code is WET code, WET is also an acronym, which stands for Write Every Time.



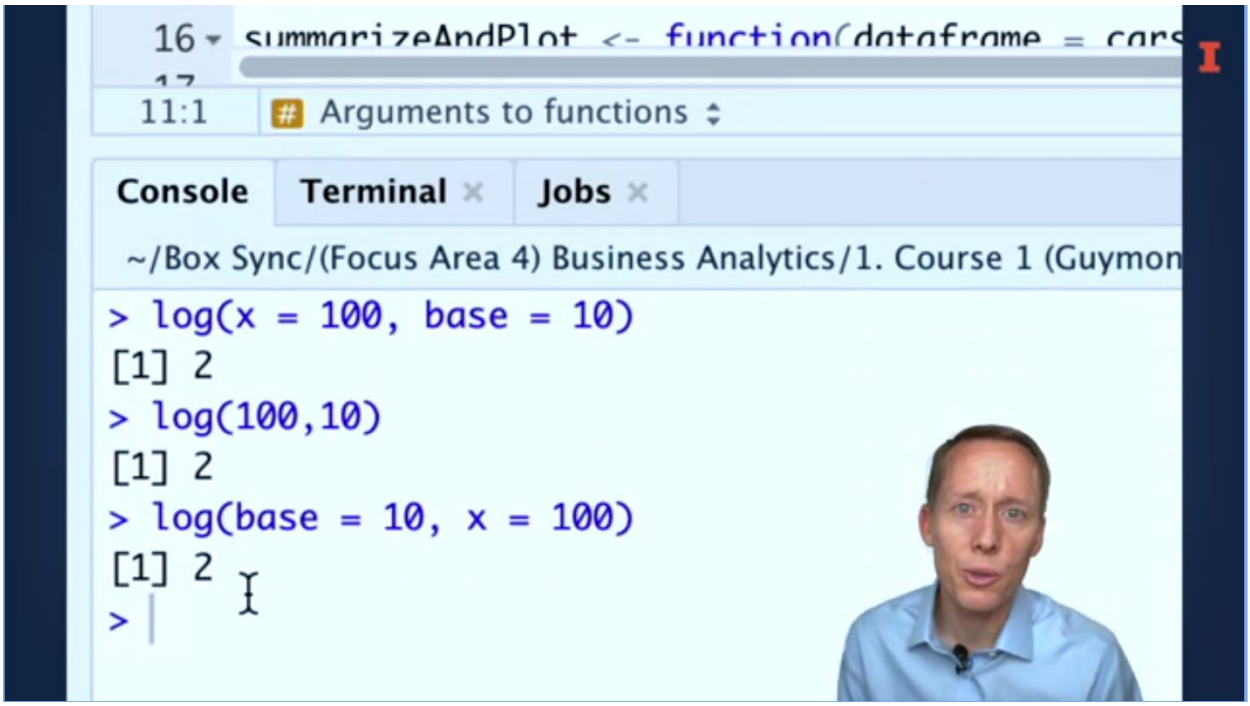
Wet code is inefficient, first of all, when you create the code, it takes a lot of time to manually type out or even copy and paste the code over and over again. The second reason why it's inefficient is because if there's a problem in your code, we have to update it for some reason in that process. Then you have to go and find every time that you've used that process and update it or fix it.



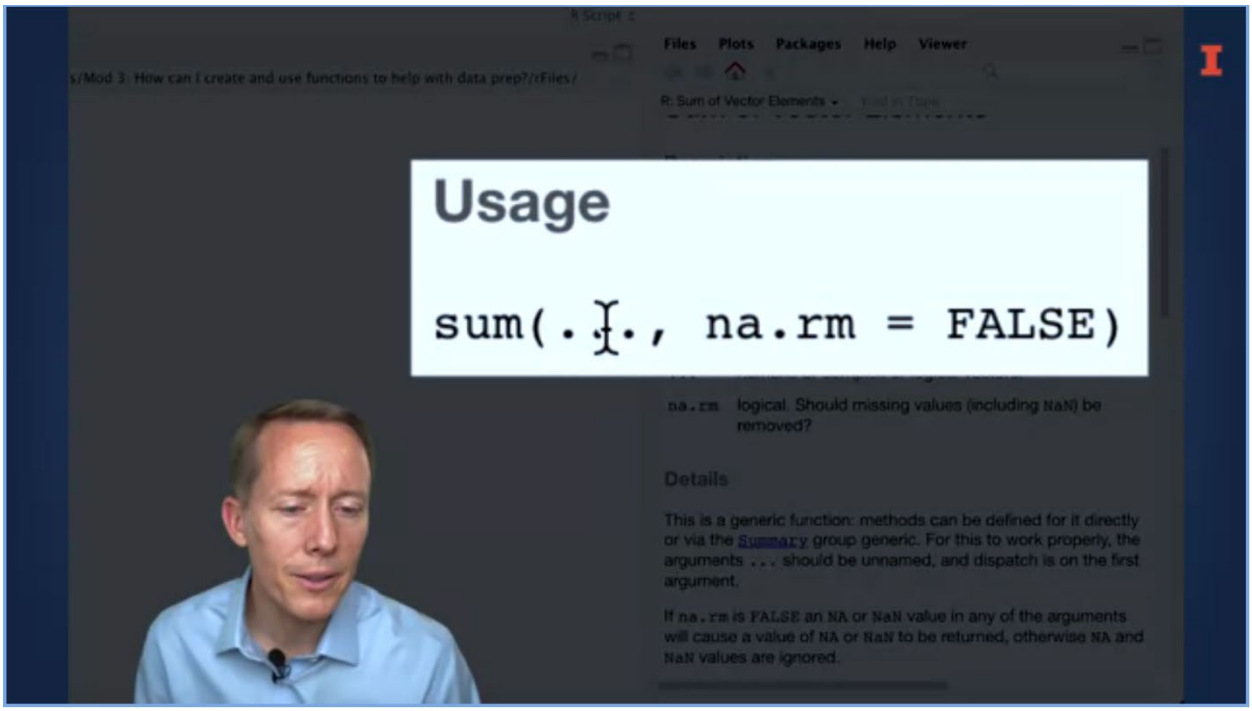
In this lesson, I want to talk at a deeper level about functions, functions you should already be a little bit familiar with, but I want to talk about the arguments in a function and how those can vary, and then even create a function of our own. Let's first read the help documentation for the log function. As you know, in the help documentation, there's a section that first described intuitively what this function can do and in this case, it gives variations of this function. Then there's a section on usage and that shows how this function is used and lets for a minute pause and look at this first example here it says log and then in parentheses, the first argument is x, the second argument is the base and it says the default value for the base is Euler's constant, basically E.



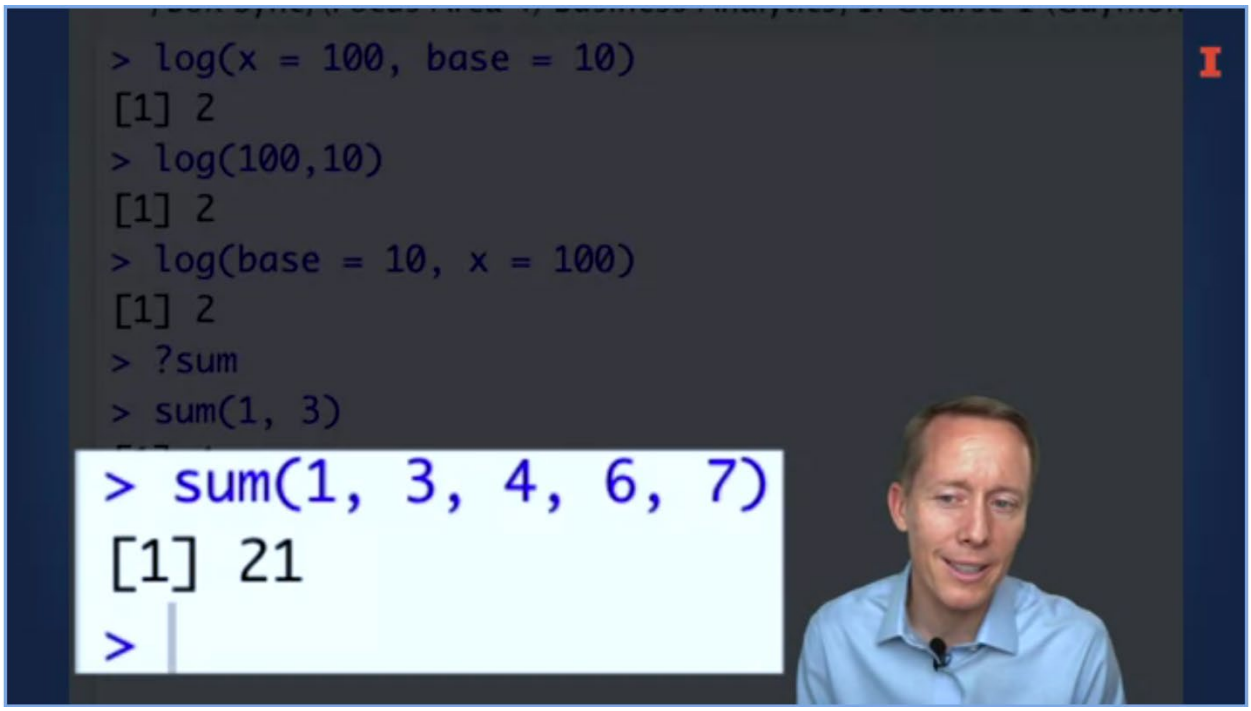
Now if you use other versions of this function, you don't necessarily need to identify the base because it is identified in the name of the function itself. Another key aspect of the documentation is the argument section, so in this section, it tells us more details about what those arguments can be. For instance, x is a numeric or complex vector, so it tells us the data type and then that it can be a vector. Base is a positive or complex number and then it describes that in more detail.



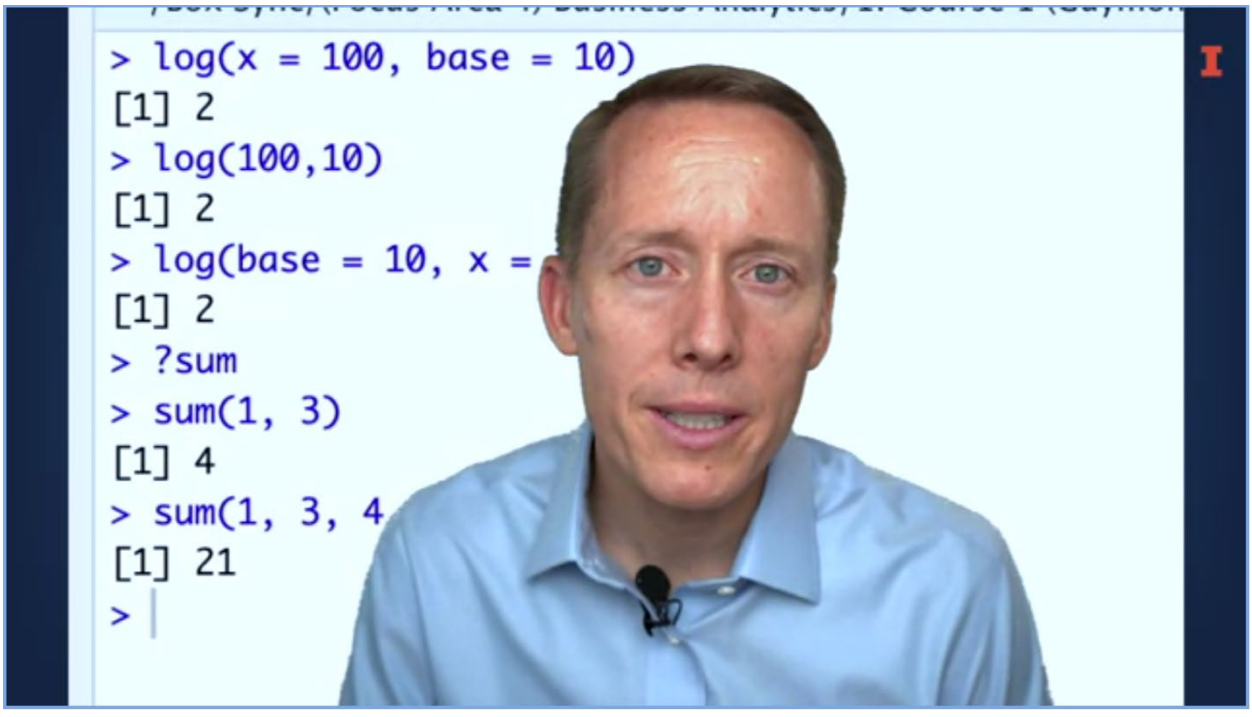
Let's test out the log function, I'll type out log open parenthesis, and then we'll explicitly say that x is going to equal 100 and the base will equal 10, so in other words, 10 to what power equals 100? If I run this, it tells me the answer is two, so 10 squared raised to the second power equals 100. Now, a neat thing about functions is that you don't have to explicitly identify what the values mean if you put them in the order that the arguments are taken. I can say log of 100, 10, and that also returns the same value of two. If I do want to put them out of order or skip an argument then I can do so if I just explicitly define what the argument values are, so in this case I can say the base and want to be 10 and the x, I want to be 100 and again we get the same value too.



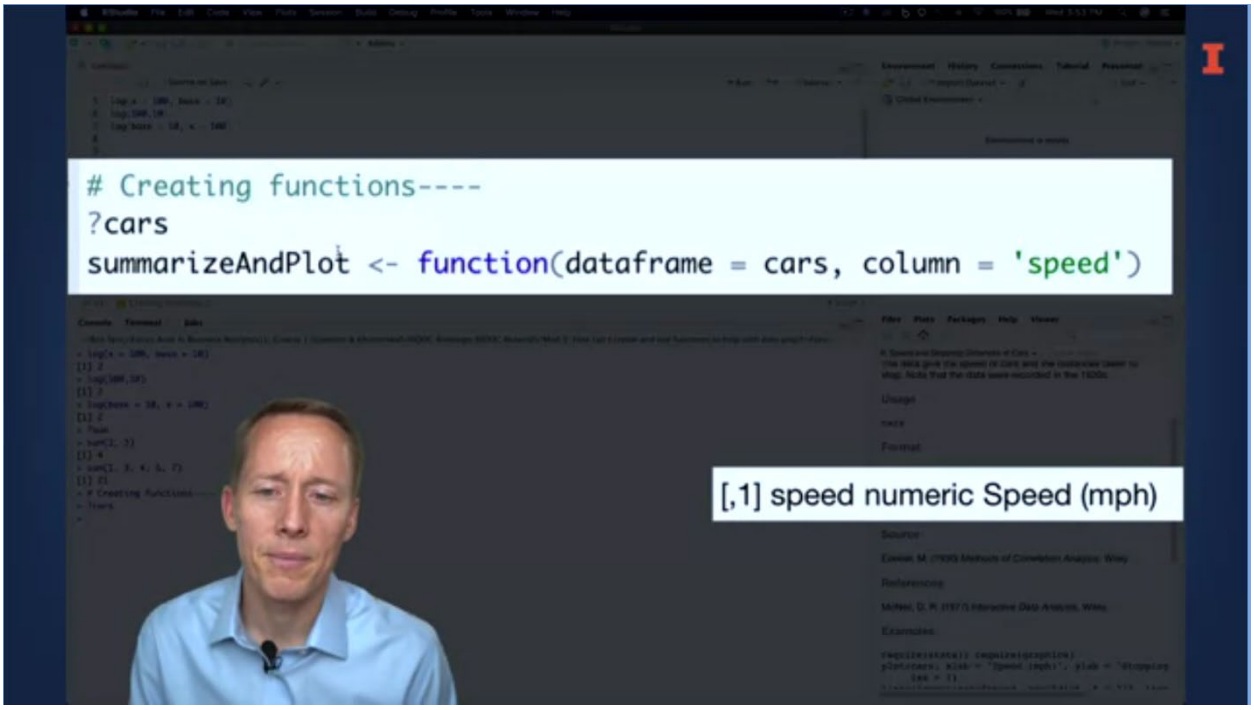
Now let's read the help documentation for the sum function. Now this function gives us an intuitive description and then it describes how it should be used and it says sum and then open close parentheses, and then it has dot-dot-dots for one argument and a.rm as a second argument. Now let's look at this dot-dot-dot, that's an ellipses. In this case, ellipses and actually in almost every programming case means there can be more than one argument here, so this needs to be numeric, complex, or logical vector. Then any dot Rm is a logical value which indicates whether missing data should be included.



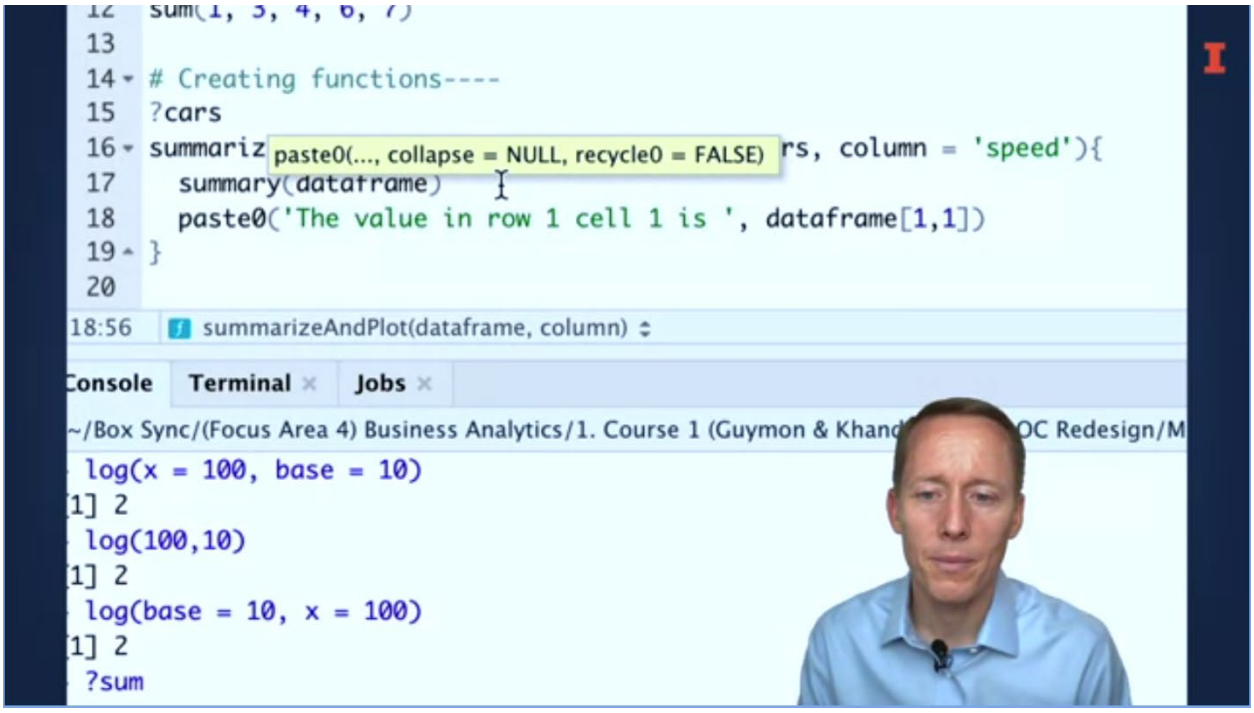
For instance, I can use the sum function with just one and three and I get four, or I can add in four again, six, seven add in a lot of other numbers and it still works, so those ellipses just indicate you can use more than one variable. Now, that word I just said, variable is important with Functions and that's the whole reason why Functions are useful. They take a process, perform the same steps, but they use different values for the arguments, that's why functions make code DRY, I don't have to go in and do whatever processes is being taken to create the sum and the log. Now we can create our own functions.



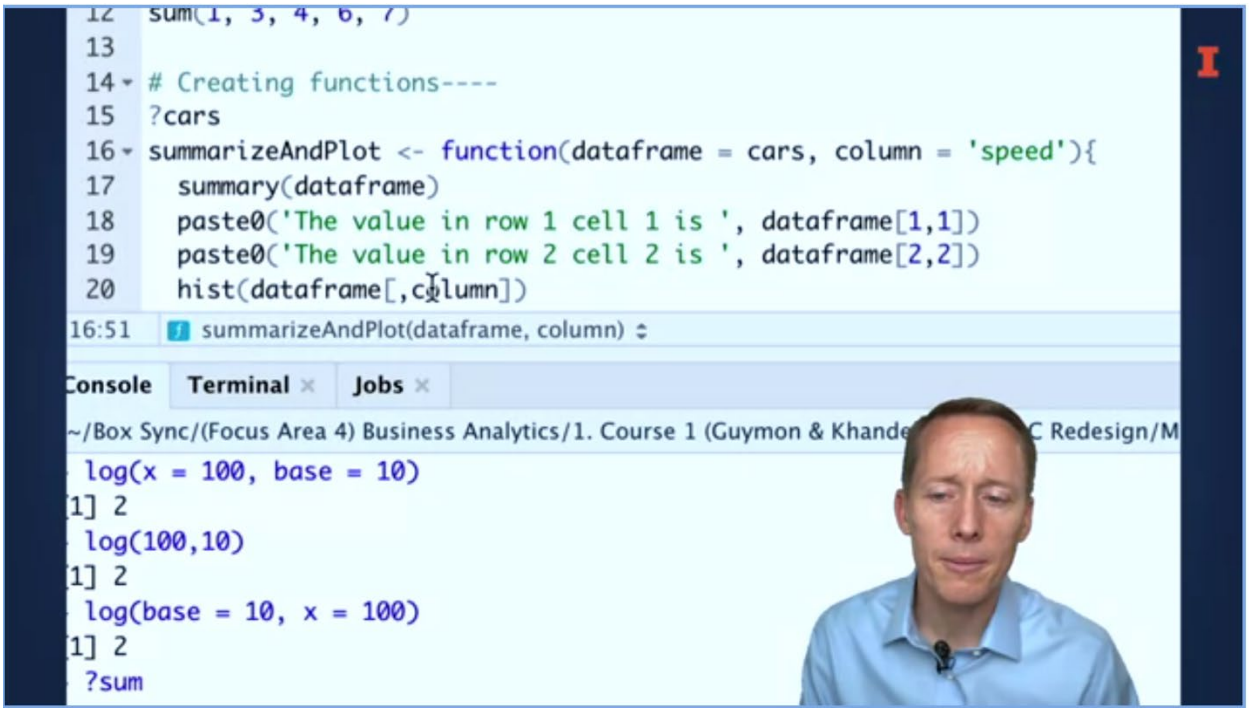
Let's create our own function that has a data analytic purpose for whatever reason, let's say that with a lot of different datasets, look at the descriptive statistics for each column, print out a value for two different cells within the data frame, and then create a histogram for a certain column of the data frame that may vary each time I use that function. That's a process I often go through and rather than typing out each of those steps every time, I can package them together into one function. I'm going to use this to start with on the built-in cars dataset. If you want to see what that dataset is, you can type out, "Question mark cars" and in the Help pane it tells us a description of this data, it's recorded from the 1920s, and there are two columns, speed, and distance, which is the stopping distance.



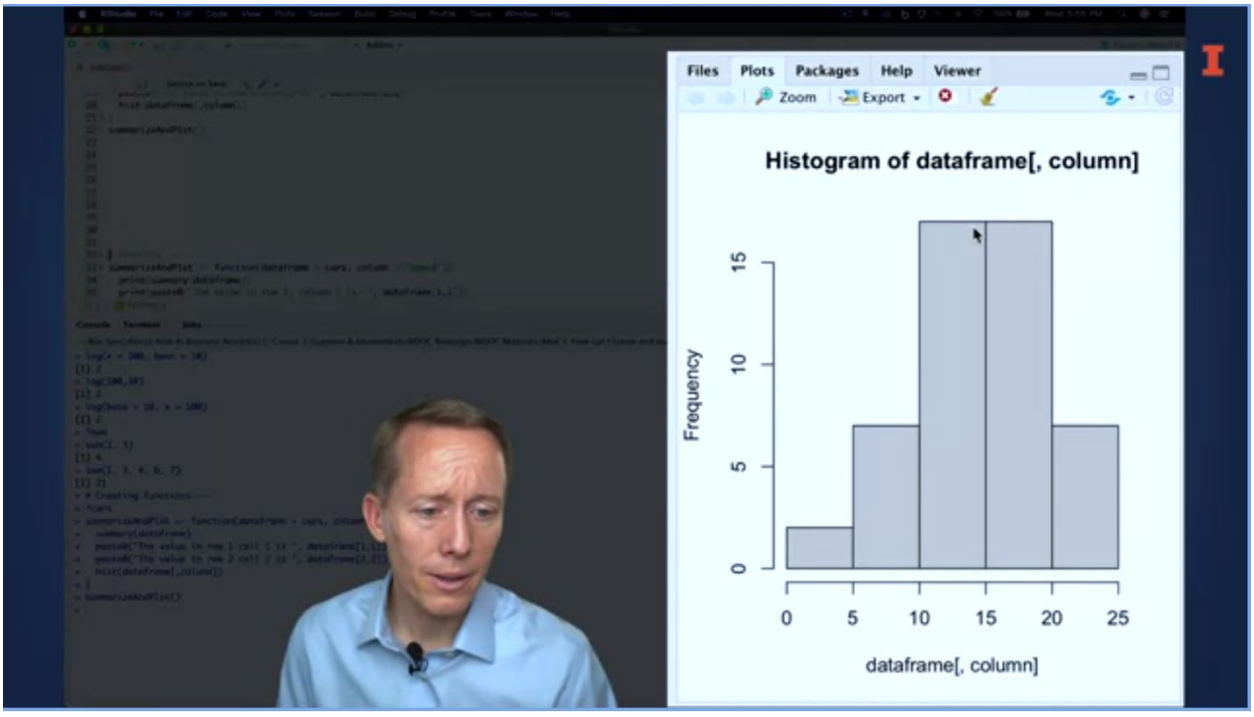
Now let's go ahead and create the function, let's call this the summarize and plot function and to indicate that it's a function, I need to type out the word function and then I'll type parenthesis and within these parentheses, I will indicate what the arguments are. The first argument, in this case, will be data frame, I want this to be generalizable to any data frame and I'm going to say that the default value will be the cars data frame. The other argument is going to be the column, so I will type out column and I will set it to be the default value of speed.



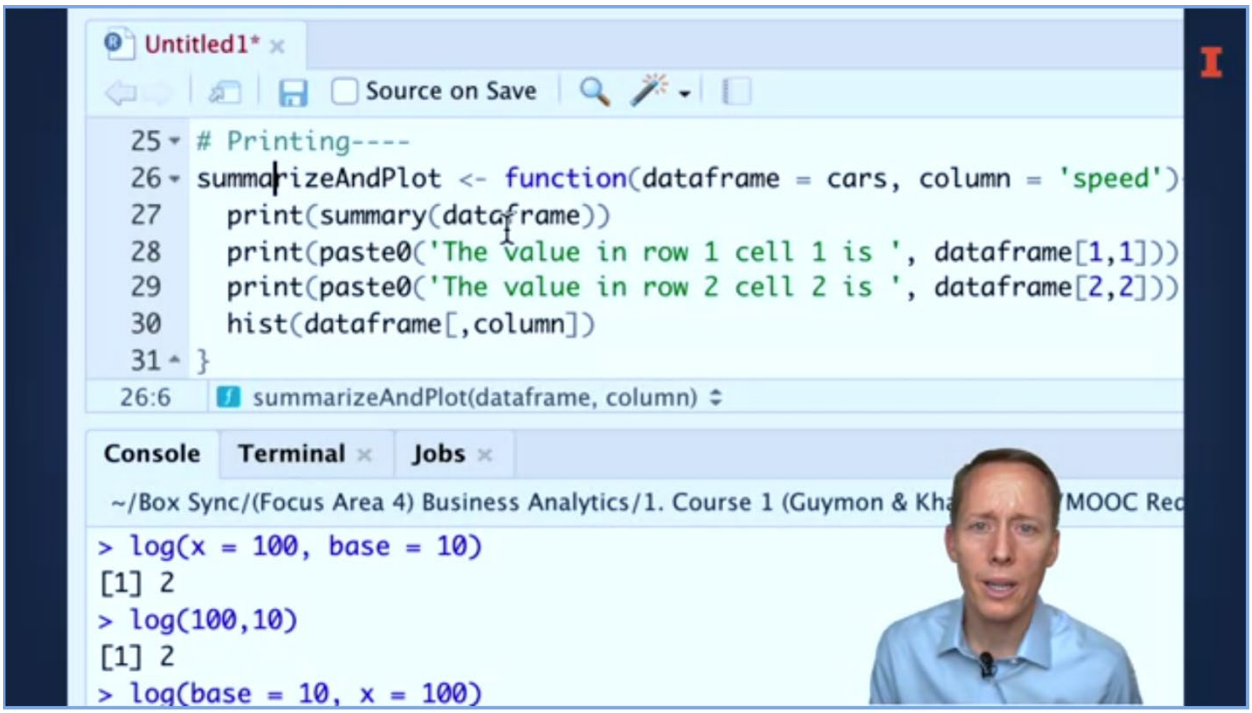
At this point, I need to now indicate what steps I want to be taken when I run this function. I'll type out curly braces, and everything that comes in-between these curly braces, will be performed by the function. The first thing I want this function to do, is to calculate descriptive statistics for every column. Notice that I'm typing out dataframe there because it's not always going to be the curly's dataframe.



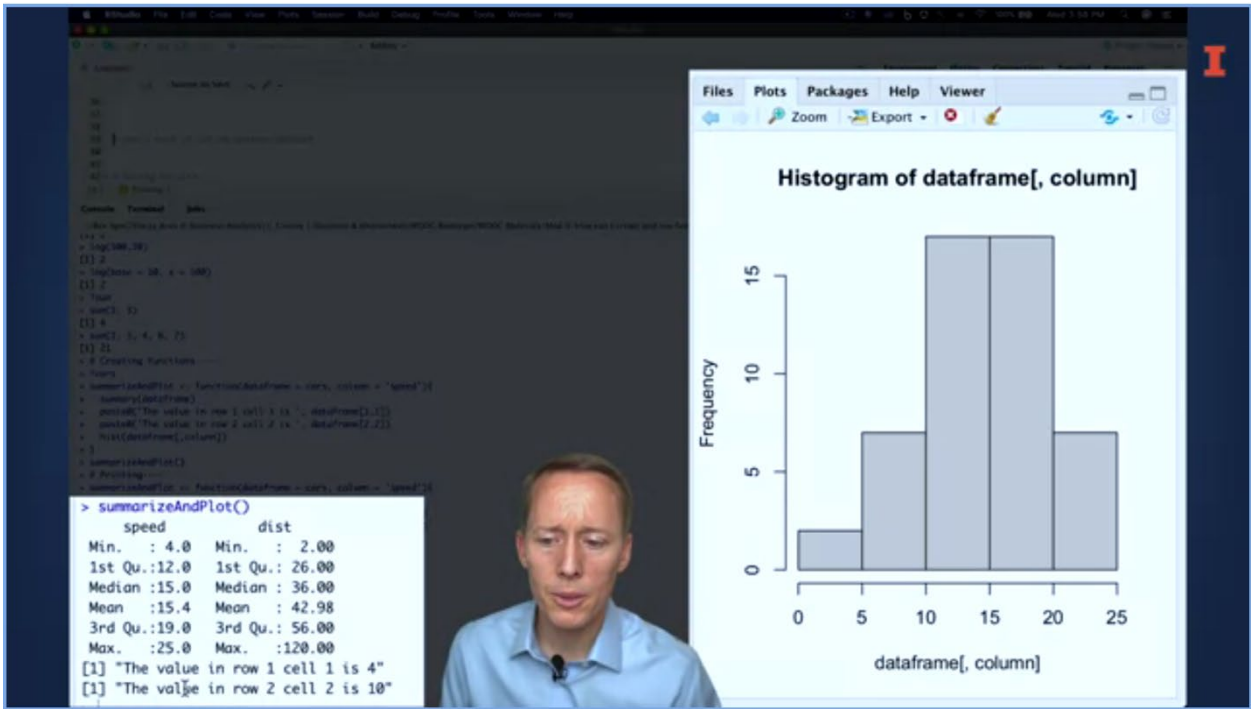
I'm going to use the summary function on whatever the dataframe is that I input into the function. The next thing I want to do, is to return a value from a cell on the dataframe. I will indicate this by saying, paste the value in row one, cell one is, and I'll say dataframe and then [1,1] it's taking first row, first column. Let's say I want to do this for another cell in the dataframe. I'll say for row two cell two. I'll change that to two and two. Now I am copying and pasting here. There is a more efficient way to do this, but for our purposes, this will be fine. Finally, I want to create a histogram of a certain column. I'll say hist, and then I'll indicate the DataFrame, and I want to use all the rows, so I'll just put a comma to indicate will take all the rows, and then for the column, I will say, I'm going to take whatever value has been assigned to this column variable here, and use that, so that will be my function. I can create this now, and I have this new object in the global environment under the function section. Let's go ahead and run it.



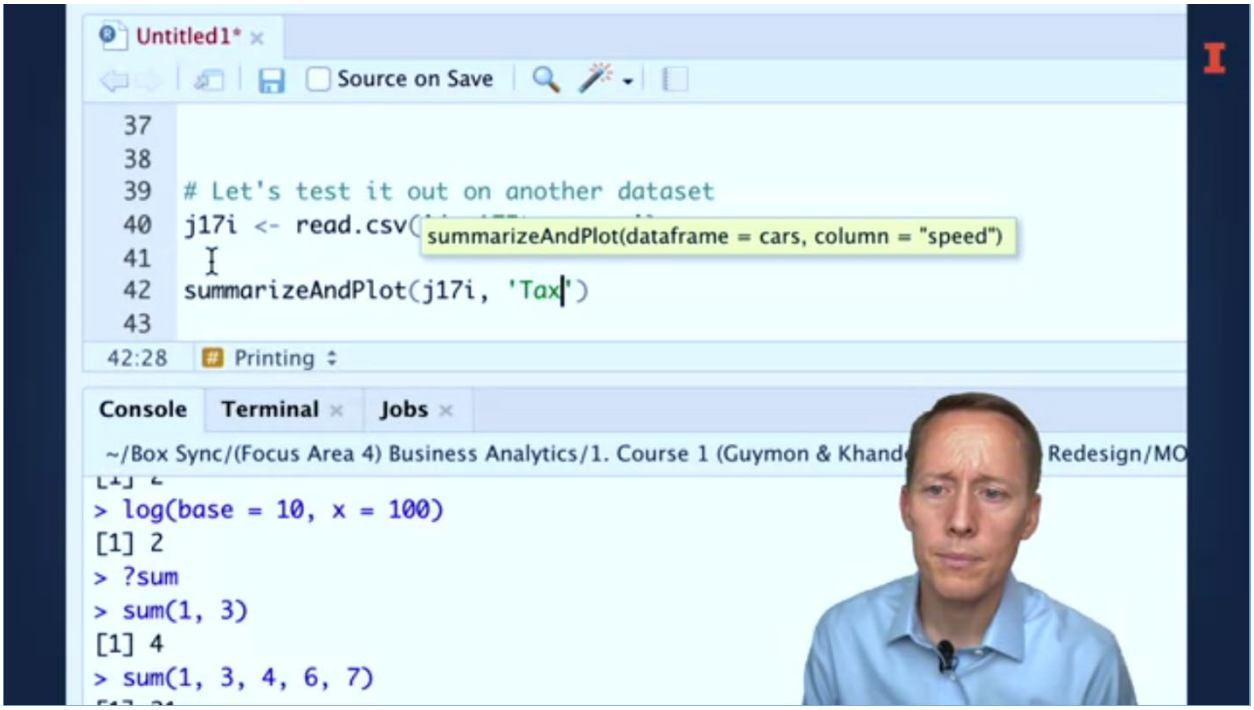
Notice that it printed out a histogram of the speed column there, but it did not print out the text that I wanted it to print out the descriptive statistics, and the values of those two cells. That's because it didn't know that we wanted to print those values out to the console. We need to explicitly indicate that these values should be printed.



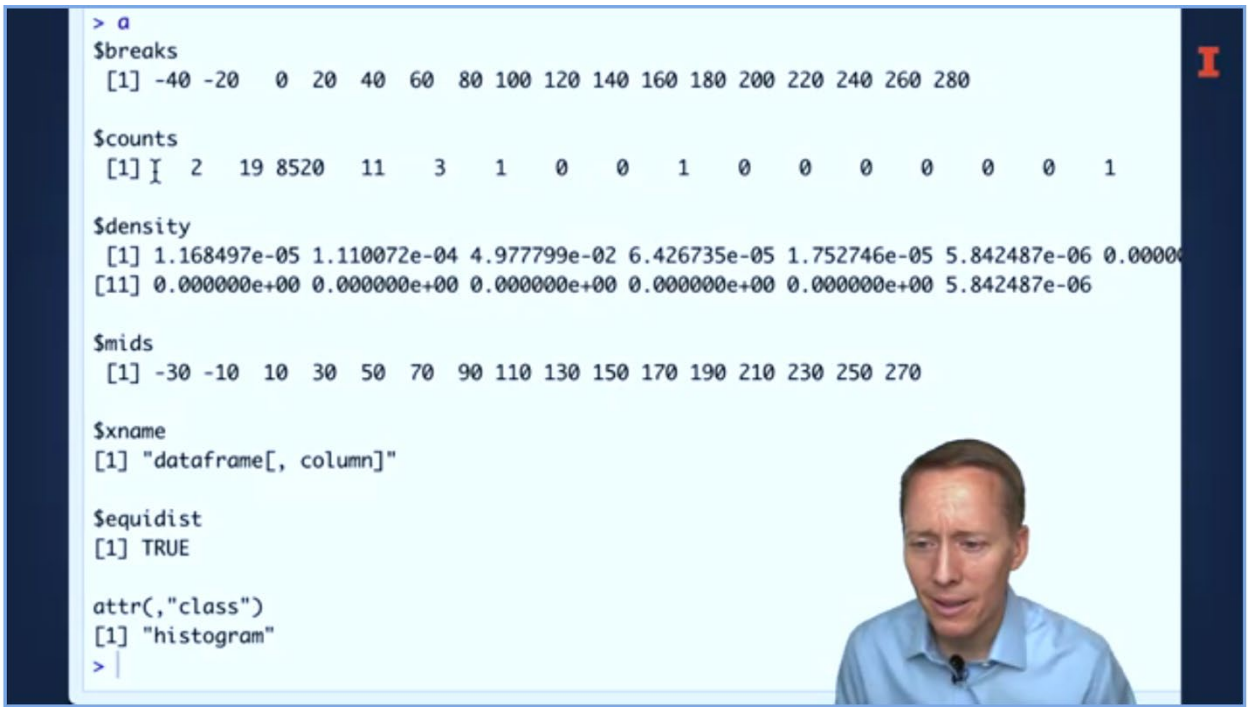
Let's update our function, by indicating that we do want all of these values to be printed. I will copy and paste this function, and I will wrap each of these first three processes of the function within the print function. I'll go ahead and create this, and then I will run this function.



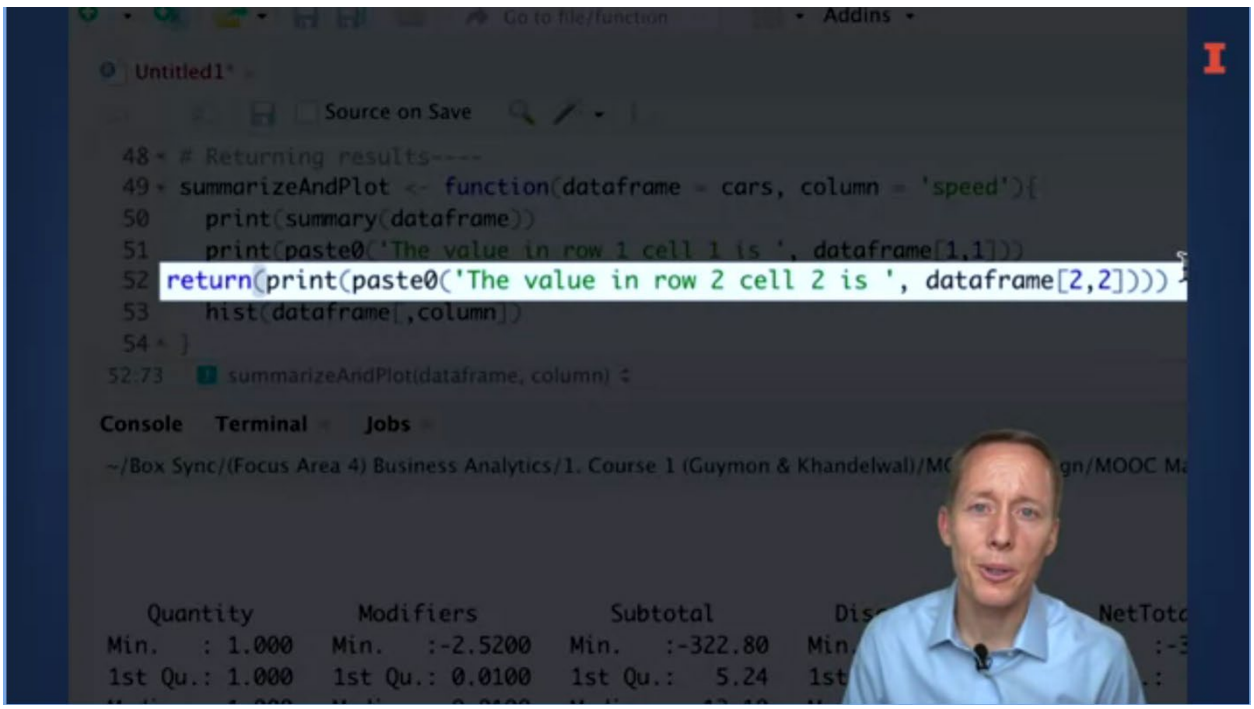
This time, it did print out descriptive statistics for both columns as well as the values in those two cells, and the histogram is there again. Let's test this out on another dataset. You can see in my working directory here, that I've got the Jan 17 items.CSV file. Let's go ahead and read that in as j17i, and let's make sure we know what the columns are in this DataFrame.



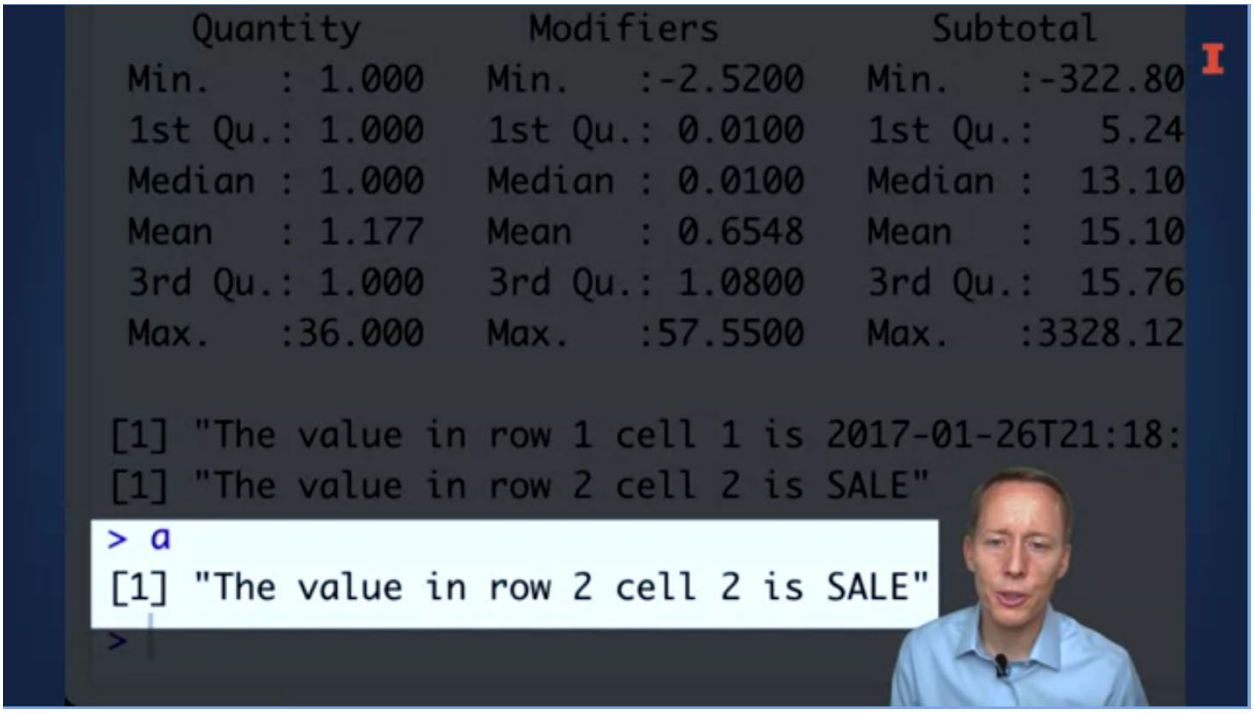
There's all the columns. Let's go ahead and run the summarize and plot function on the j17i DataFrame, and let's use the tax column. You can see that it has printed out descriptive statistics for each column there, as well as the values in those two cells that we indicated, and the histogram of the tax column. We created a function that allows us to perform all those steps with basically a single line of code. You can take the output of a function, and save it as a new object. In this case, I'm going to save it to the A object.



Let's go ahead and take a look at what is in that A object. It's not very clear here, but these are basically the elements that are used to create the histogram. Now, rather than returning the elements that make up the histogram, let's say that I want to return the second sentence there that tells us what's in row two, column two.



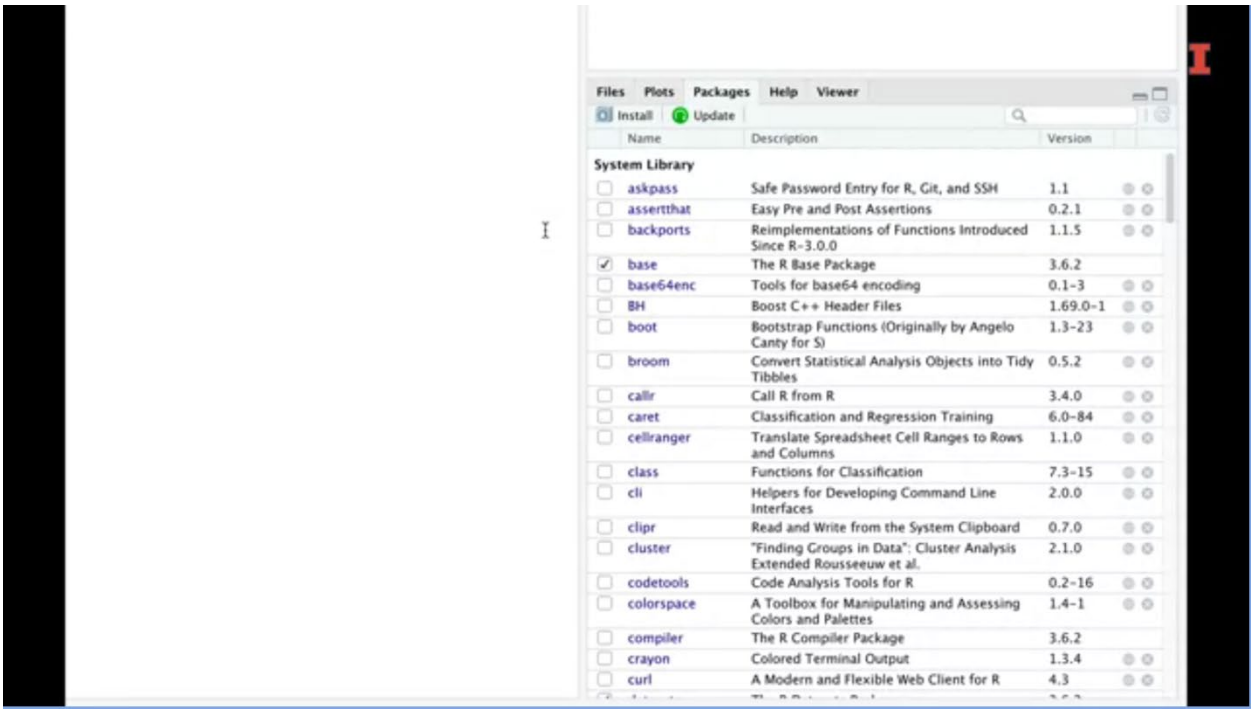
I'll copy and paste that function, and then I can wrap whatever I want to be saved within a function, which is the return function. I will recreate this, summarize and plot function, and I'll rerun it on the j17i.



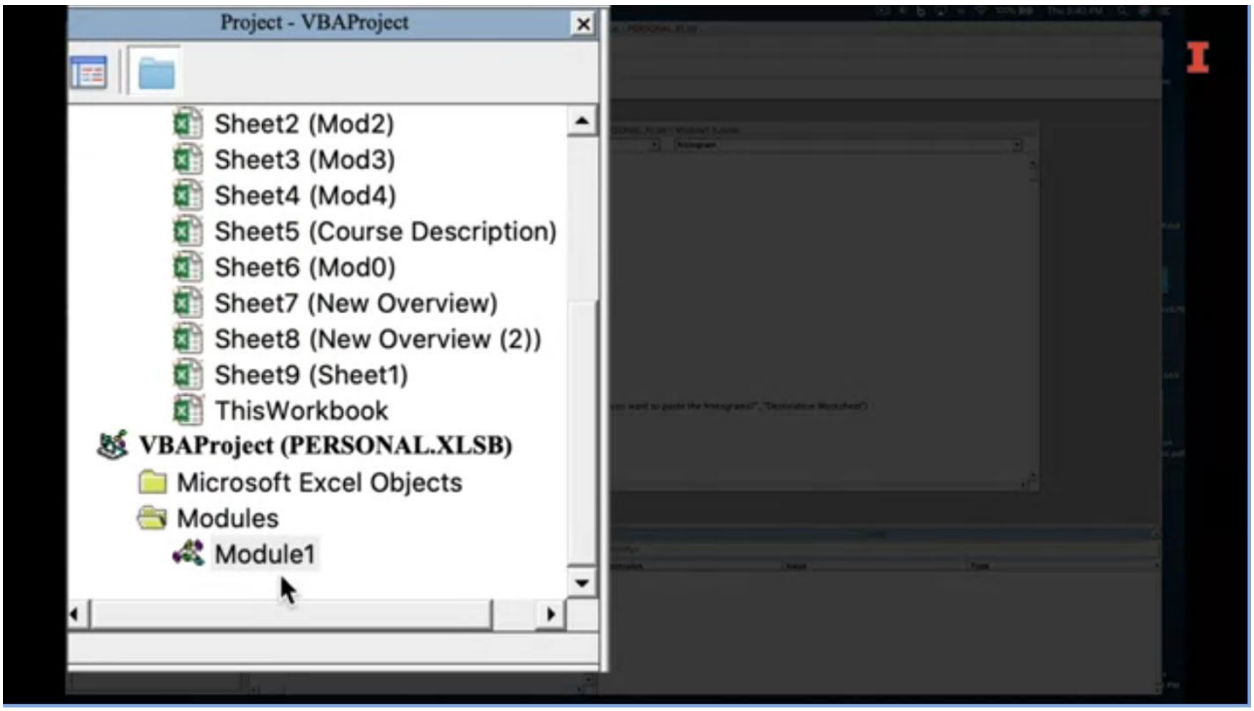
Just to mix things up, rather than use the tax column, let's use the subtotal column. You can see that the histogram, you can't actually tell, because it looks very similar, but the numbers are different here, and if I run now A, see what's stored in A, you can see that it's the second sentence here. The value in row two, cell two is sale. Now you can return more than one object from a function, and the way you do that is by creating lists, but that's beyond the scope of this lesson here. There's a little bit deeper dive into functions. I hope that make sense how they can be useful for creating dry code.

Lesson 3-4 Packages

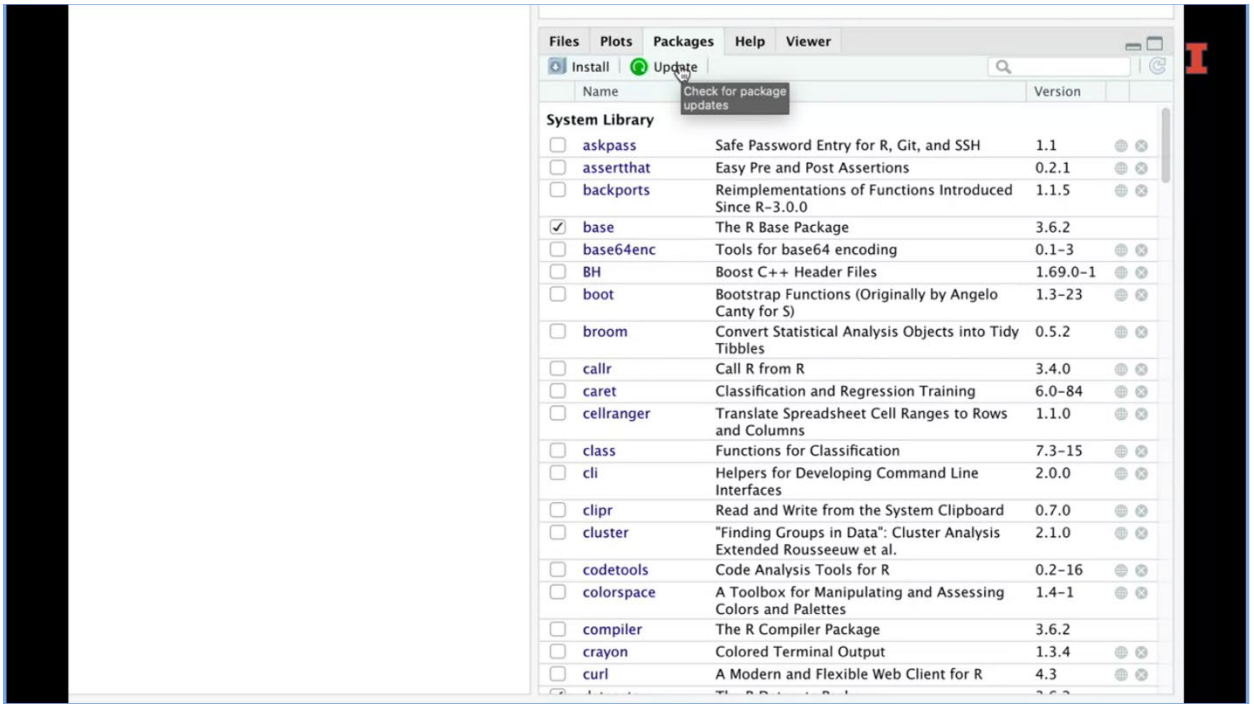
Lesson 3-4.1 Packages



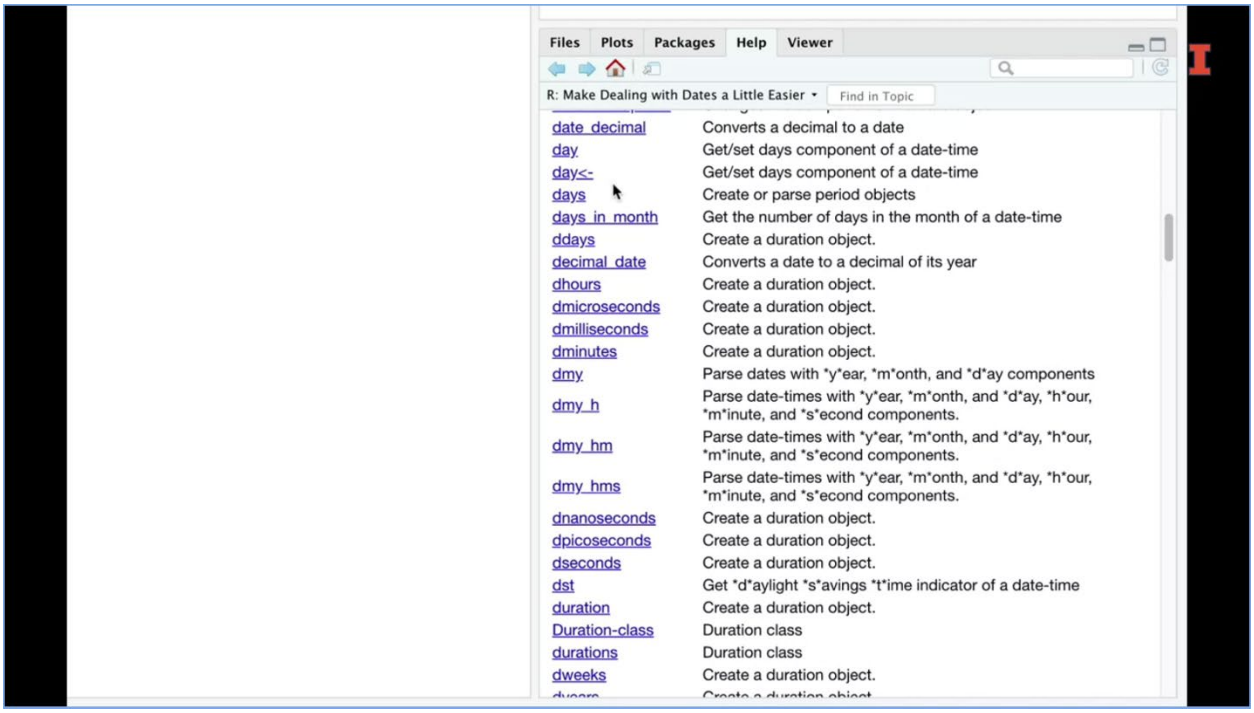
This section of the course is about data types. But before jumping into different data types and learning how to deal with those, it's worth your while to learn how to manage packages. Because doing so will make it much easier for you to complete some common tasks associated with data types, as well as many other aspects of data analysis. Now, as the name implies, a package you can think of really as a gift that someone has created for you.



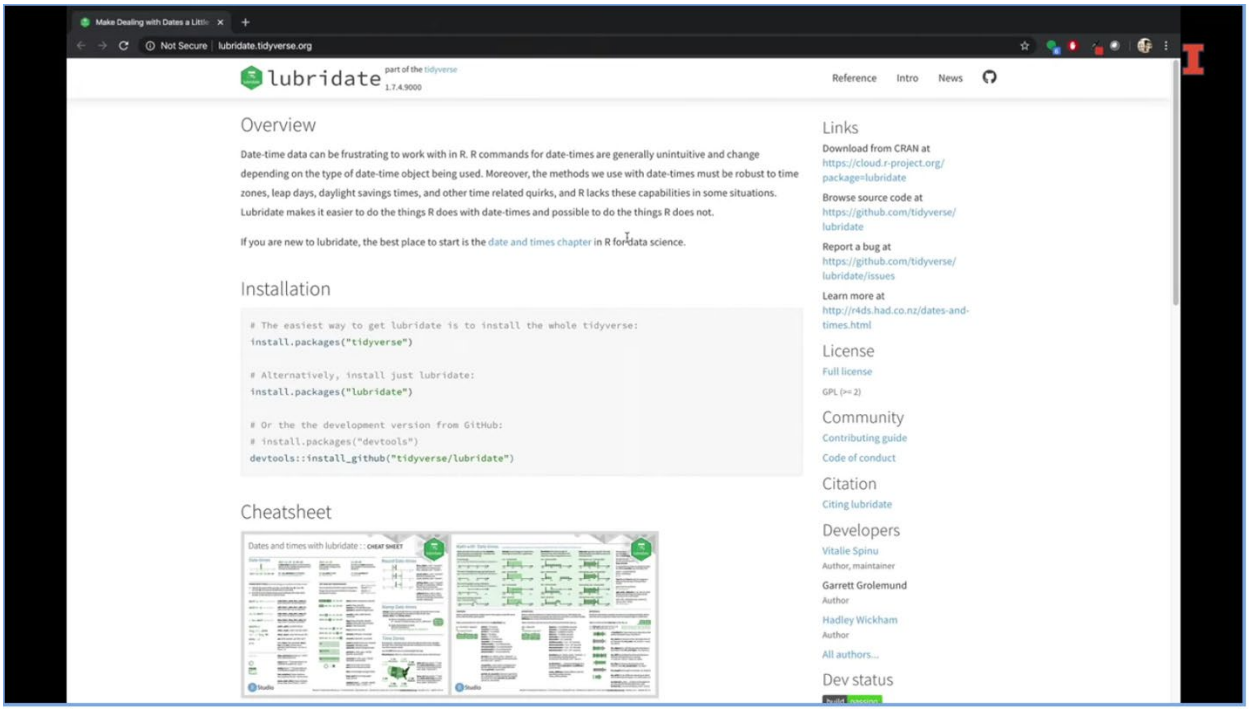
And within that package, there is a list of functions that perform some related tasks. In relation to Excel, if you've ever created a macro that has saved you a lot of time, it really can feel like a gift. And if you've ever stepped into the code for that macro and looked at it in the visual basic editor, you may notice that in the project pane, there is a folder called Modules, and then a variety of scripts.



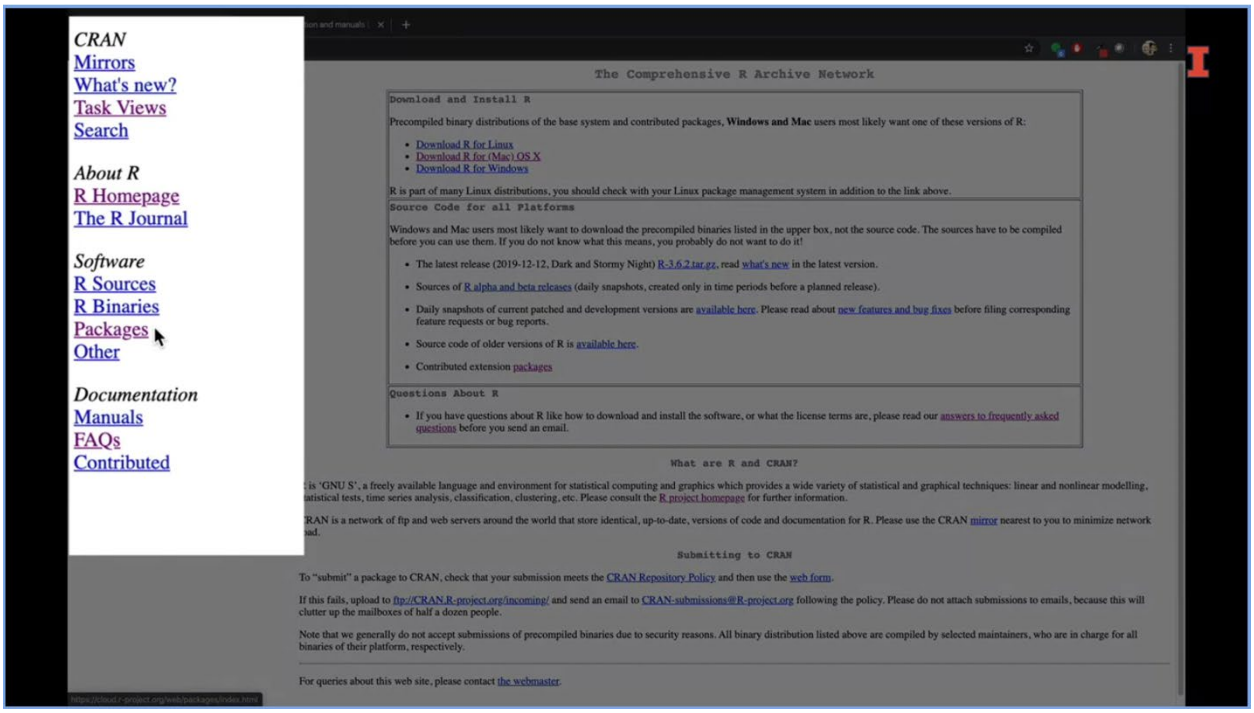
And in these scripts is the code for the different macros, so that same idea applies to packages in R. Let's talk about how to use this packages' pane. You'll see at the top that there are tools to install, update, and search for packages. And then there's a list of packages by name, a description, the version, and a couple of other things. R comes with some packages that are pre-loaded in it. And then you can install more packages based on what tasks you need to perform. If you want to learn more about the packages that are already installed in your project or on your machine, you can simply either scroll down to a certain package. Or search for a package by name or by function.



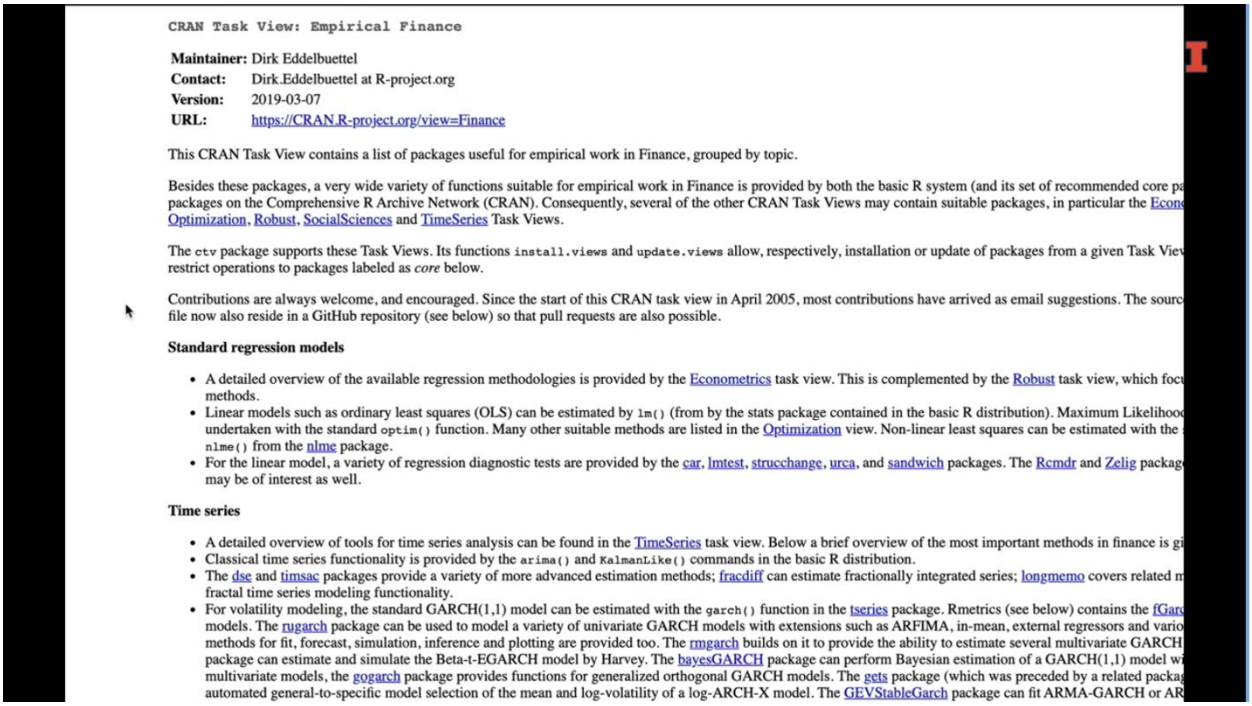
So there's a text or I can search for date, we're going to be talking about some date functions in a little while here. And I could click on the name of that package, so I'll click on lubridate. And here's some documentation for that, and it starts by listing all the different functions within that package. And then if I want detailed help with a certain function, I can go ahead and just click on the name of the function, and I get all the help associated with that function. I'm going to go out of the help pane back into the Packages pane and let's talk about these icons on the far right.



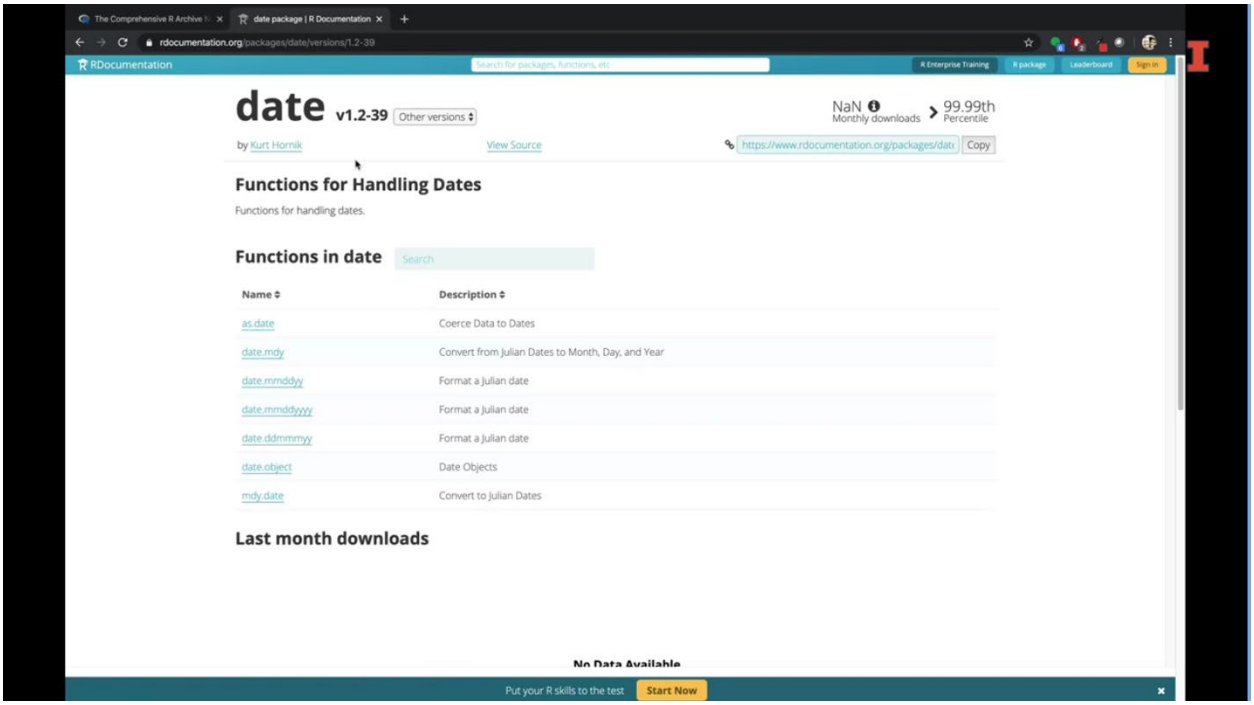
The furthest to the right allows you to remove a package if you don't want it anymore. This one next to it tells you where you can get additional help related topackages. So if I click on that icon, it will open up in a browser a website that has additional documentation. And some documentation is better than others, this one happens to have a cheat sheet associated with it. Anyway, there's another way that you can get help with learning how to use the functions in that package.



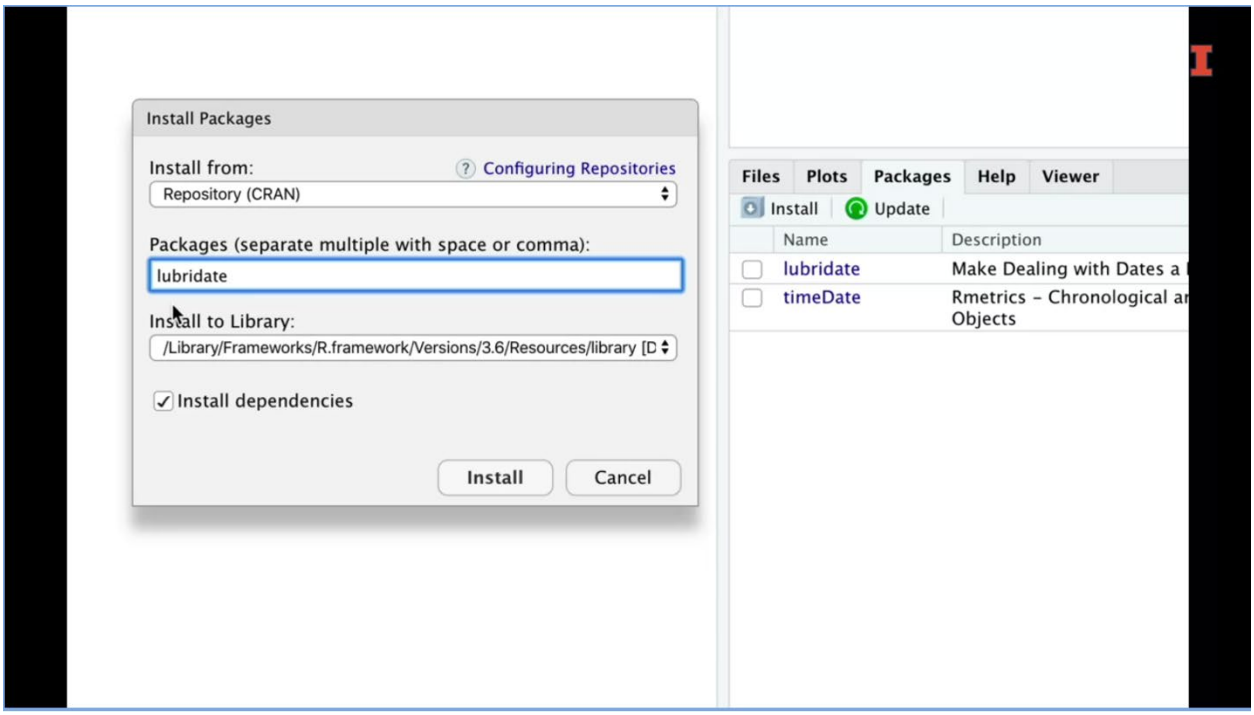
What if you want to perform a data analytic task, but you don't know how to do it and you'd like some help. And you want to see if someone else has made a package for you? I'd like to show you a couple of websites to search for packages to perform a task. The first website is the r-project.org website. This is where we went to install R in the first place. So if I follow the same path initially to install R, go to CRAN, then click the 0-Cloud server. And now, instead of downloading R, I'm going to go to the left, sidebar here and click on Packages. On this page, it tells me about available packages and how to install packages.



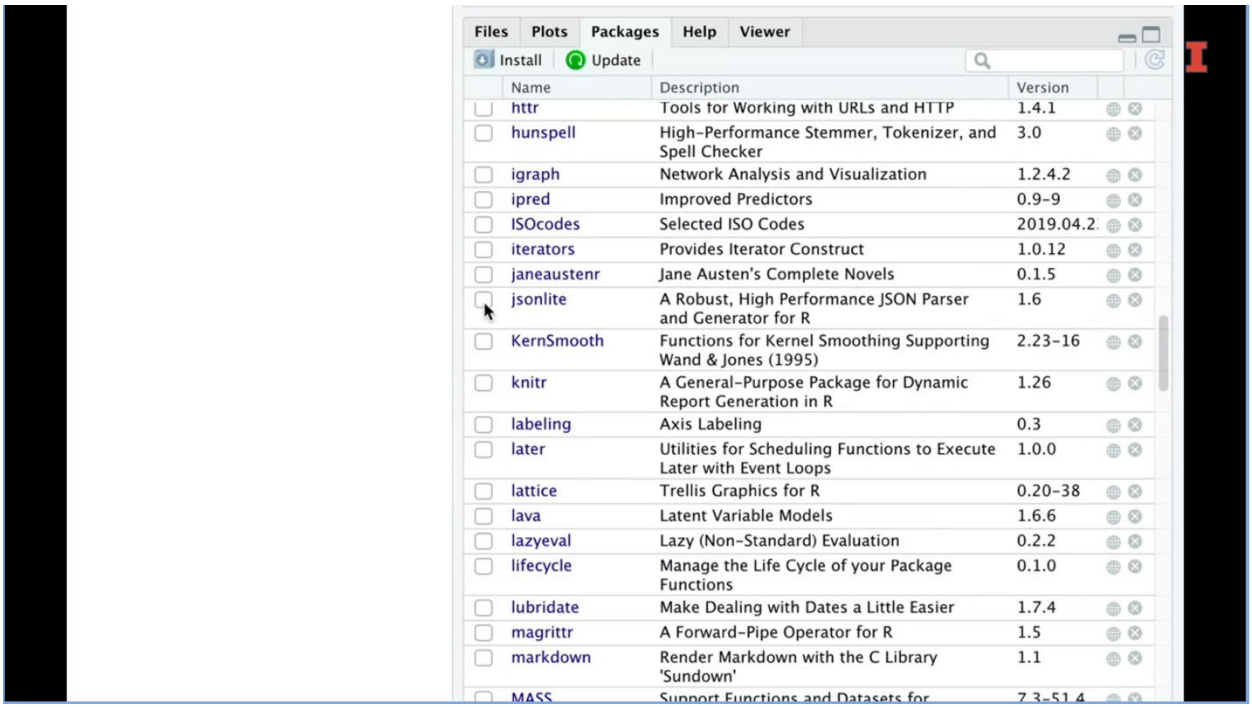
And you can see that you can search for packages by date of publication or by name. But what I find to be most helpful is to search for packages based on the functionality, which is what this link does. So if I click on that, there are about 41 different topics here, general topics. And let's say I want to learn more about finance, I can click on the finance link. And this takes me to some additional details and a link to some other packages related to financial applications, all right? So there's a whole bunch there. So that's one website.



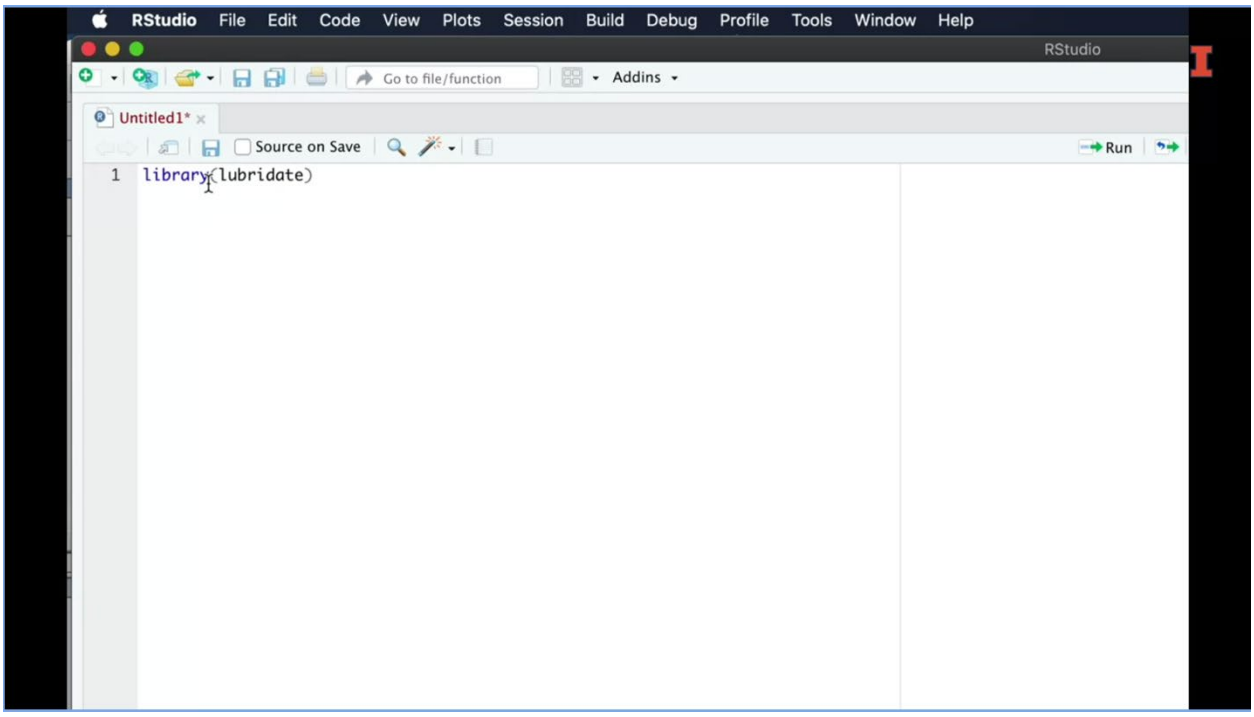
Another website that we've already looked at once before when we talked about functions is rdocumentation.org website. And if I want to search for a package to help me deal with dates, I can start typing in date. And up comes some package names as well as function names, and I can click on a package name here. And it tells me information about this package, how often it's downloaded or the percentile, so this is downloaded quite a bit. And then it tells me what the functions are in this package, as well as some additional information. And I could click on a link for one of those functions, and it gives me the documentation related to that package on this website.



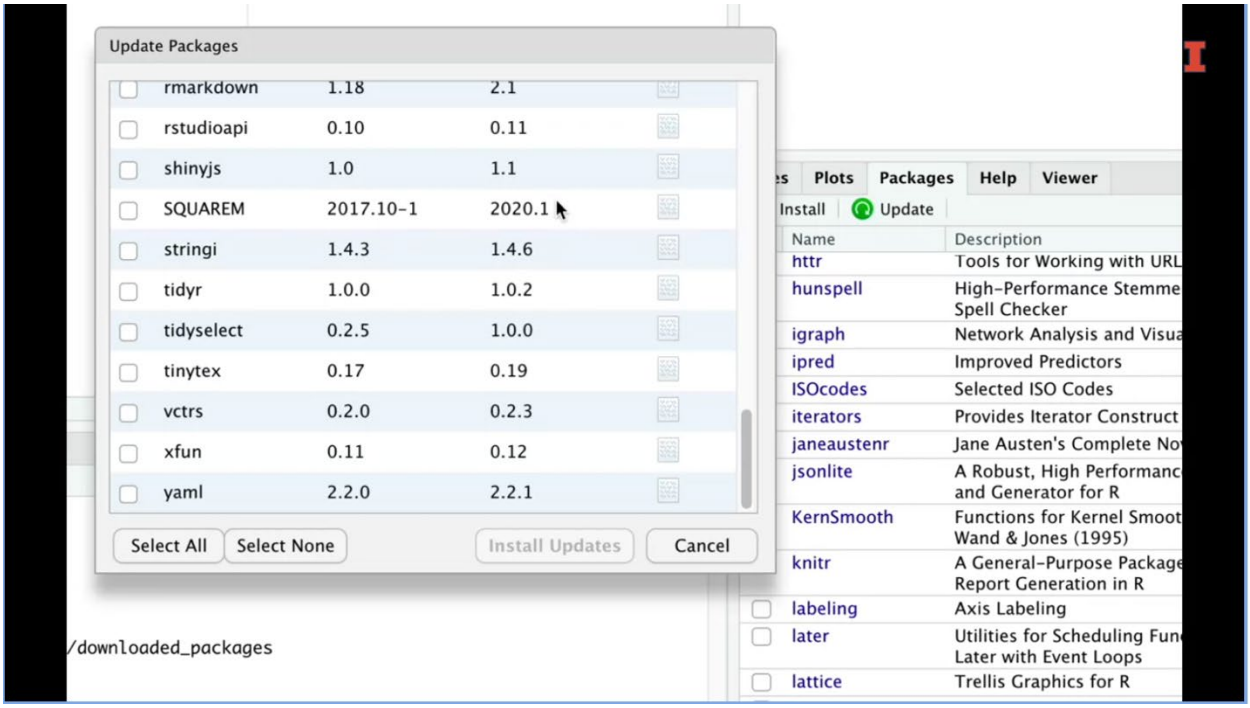
Now let's say I want to install a particular package. Let's say, for example, I know I've heard about the lubridate package and I read about it, it looks awesome and I want to install it. I could go back over to the r-project.org, and this tells me how to install packages, it's very easy to do programmatically. I could just type out install that package is and then type out the package name. But in most cases, what I do is I used the utilities built into R Studio. So you can click on this Install button in the top left hand corner of the Packages pane. And you can type the name of a package, so I could type date that I saw there and there's date. I could type lubridate, and you can see that lubridate pops up. And let's say I do want to go ahead and install it, then I just click the Install button.



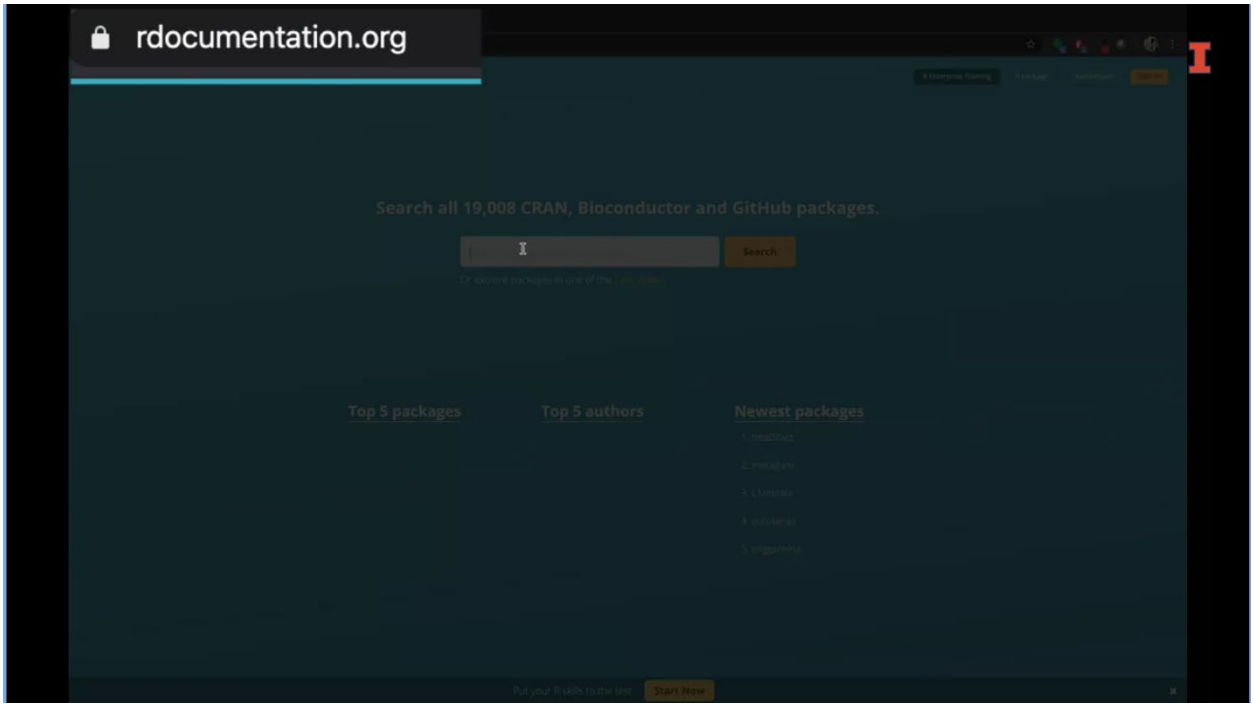
And you can see what's happening over here in the console as it is installing the package. And now it's installed and I can start using it. Let me X out of that subset, and I'll just scroll down. And as I scroll, notice that most of these packages do not have a check mark next to it. So a check mark indicates whether or not a package is loaded for use. So you can't really use the functions in a package until you load those functions into the environment.



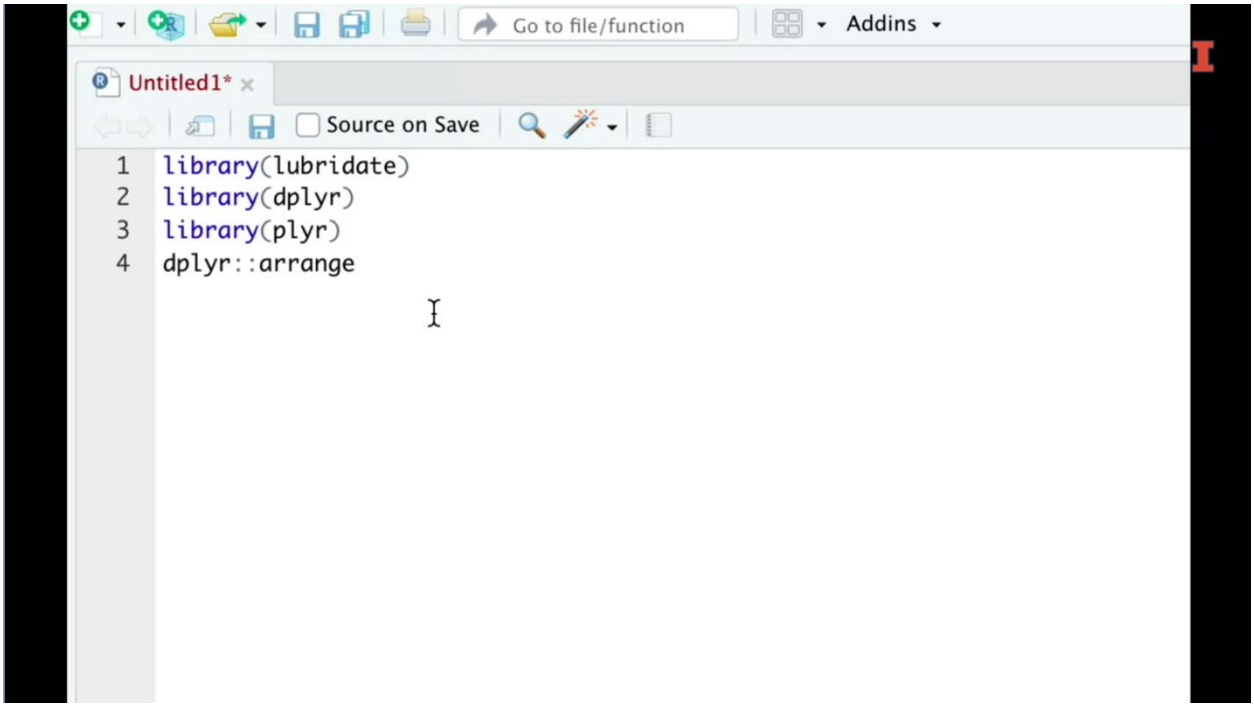
Let me show you how to do that. I'll create a new script, and if I want to load the functions in the lubridate package, I can type out library. And you can see it turns blue, and then it gives me a list of the different packages, and there's lubridate. So I'll select that, press Command+Enter. And you can see that in the console, it told me it has attached lubridate and now over in the Packages pane, I have a check mark next to lubricate. I can also just click the box, and that would load a package as well. So that loaded the lifecycle package, and I can click it again to unload it or detach it.



These packages do not stay the same for very long. R is continually updated, and other packages that some of these packages depend on are also updated. If you want to update your packages, you can click on the Update button in the Package pane, and it will give you a list of all the different packages that need to be updated. And you could simply select the ones you want or select all of them and then install the updates. Now, why wouldn't you ever want to keep them all updated?



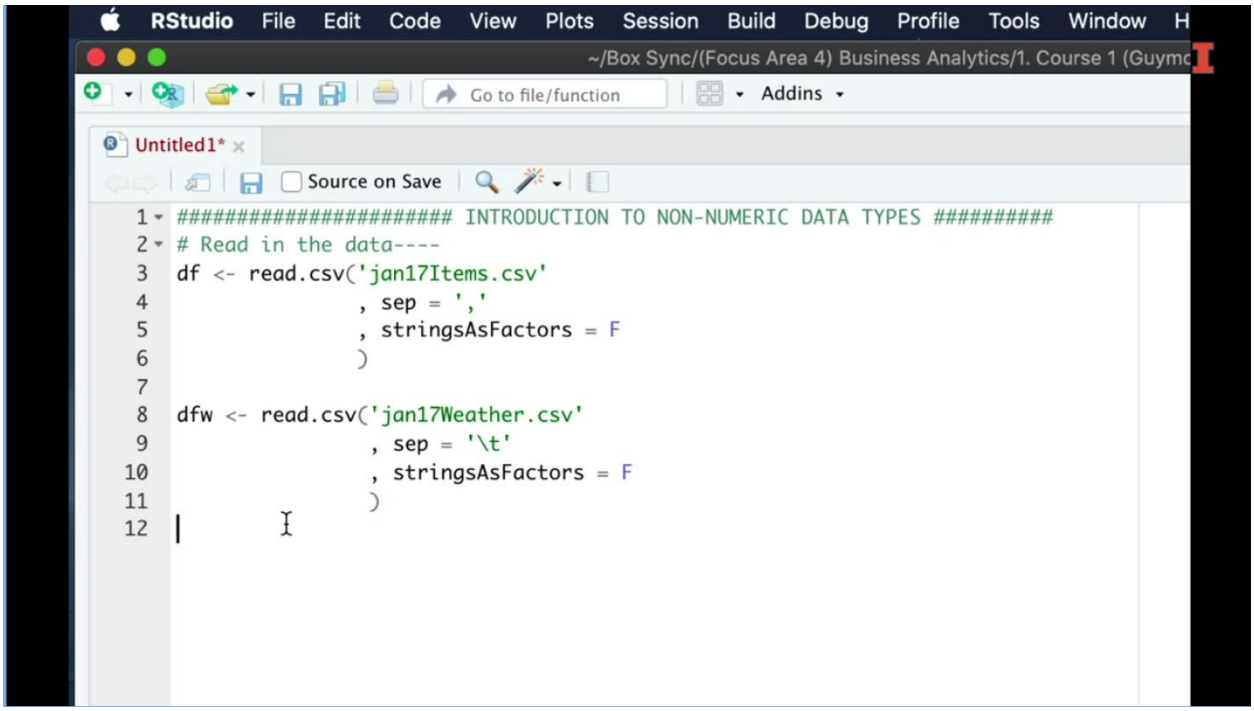
Well, sometimes, when a package is updated, it may change how it works with other functions. And sometimes unintentionally, it prevents certain functions from working. Or sometimes, it stops using certain functions and replaces it with better functions. So whenever you update your package, you need to make sure that all of your scripts that use that package are also updated. Or you need to test them to make sure that they work as you expect them to. That's especially important if you're putting a script into a process that happens automatically. Another question is, why don't you just load all of the packages for use? Well, it takes a while to load packages, and that's just a lot more that has to be retained in the memory of the machine that you're using. So you really only want to load the package is that you currently need. Before I conclude there's one more thing about packages that I want to talk to you about and that is name space. And to illustrate this, I'm going to go back to rdocumentation.org website and I will search for a function, arrange.



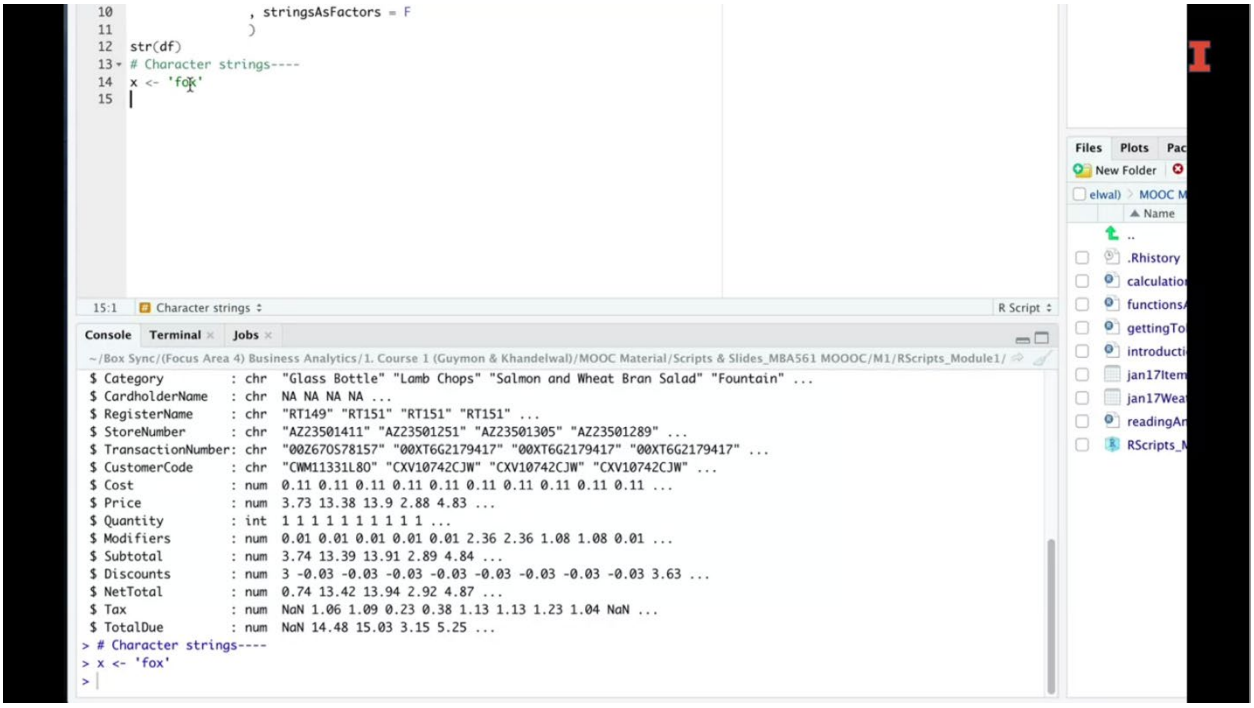
And notice that there are several arrange functions and that makes sense, it's a common name. And with over 19,000 packages, it's bound to be used in more than one package. And let's say I've got two packages dplyr and plyr, and they're both installed and loaded on my machine. If I want to use the arrange function, I need to tell R which version of arrange I want to use. The way I do that is, I type the name of the package and then two colons. And so this is telling R to use the arrange function from the dplyr package instead of the plyr package. So in conclusion, packages are really helpful. They allow us to focus on data analytic tasks and not worry as much about the low level programming.

Lesson 3-5 Introduction to Other Data Types

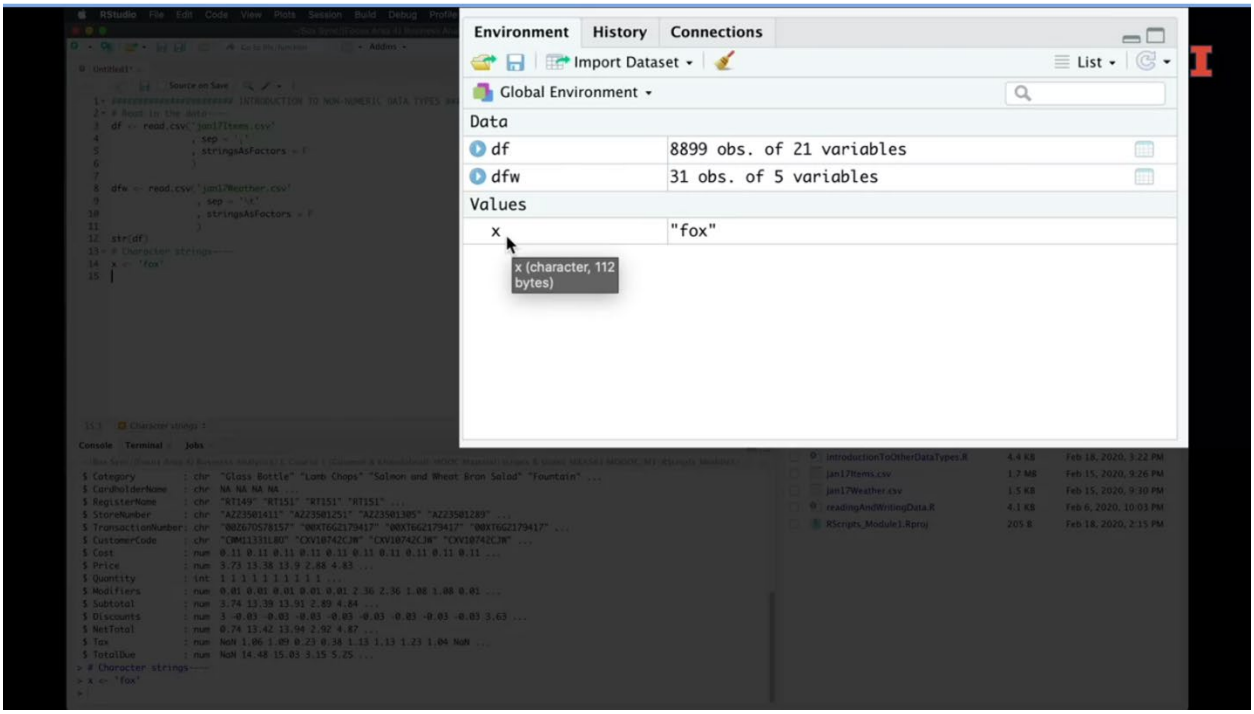
Lesson 3-5.1 Introduction to Other Data Types



To this point, we've talked a lot about numeric objects, vectors and columns in the data frame. In this lesson, I want to give you a brief introduction to other data types that you see in a data frame. So to start out with, let's open a new script. And give it a title in subheading and read in some data. Okay, I've given the title of introduction to non numeric data types and I've got a subheading for reading in the data and you can see that the data that I'm going to read in is the jan17Items.csv and jan17Weather.csv files and I'll store those as df and dfw data frame objects, respectively. So I'll go ahead and run that.

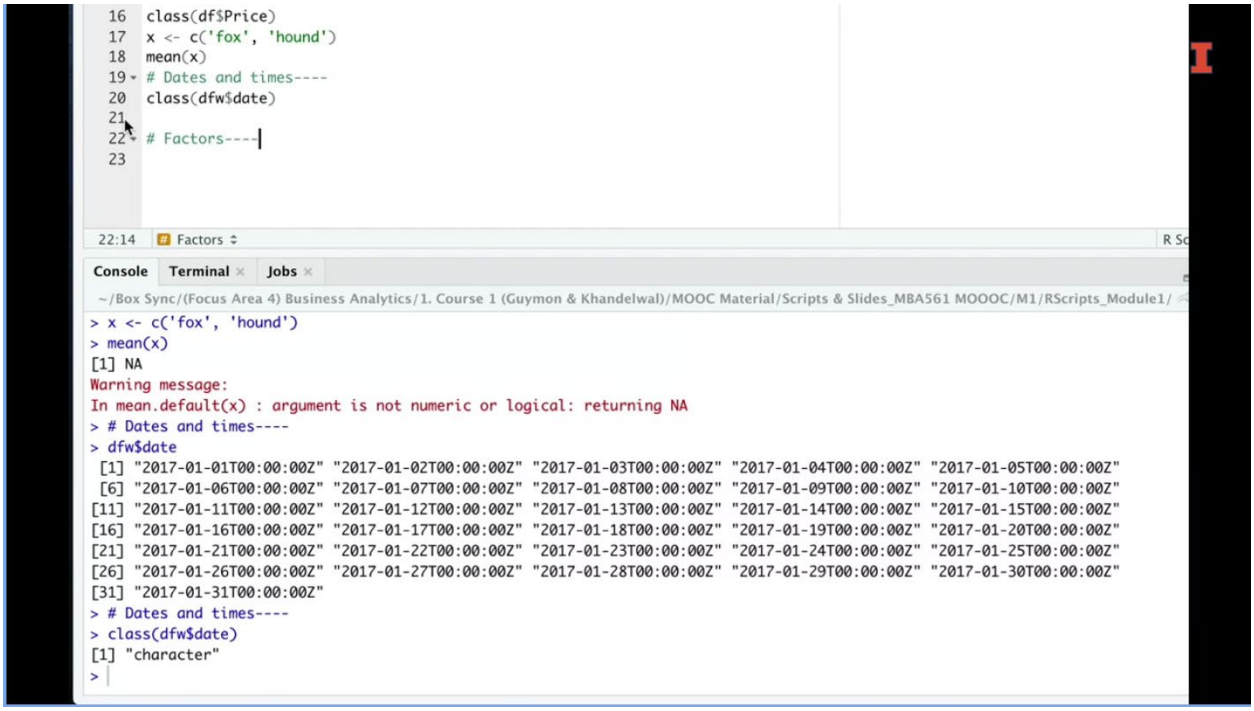


And a quick overview in the environment. Pain shows me that those were read in and it looks like they were read in correctly. So perfect. Now, let's go ahead and look at the column types of the df data frame and I will use the structure command str and df. And let's focus on this column here that tells us the type so we can see that for all of the first. 12 or so columns. They are character string types chr, whereas the last miner. So columns are numeric types, so num for numerical, int for integer. So those numeric columns are the types of vectors that we've been focusing on. At this point, I want to talk to you just briefly about these other data types so that you have an idea of what to do with them and what to expect. And then we'll spend other lessons focusing specifically on each data type and how you can manipulate data in those types of vectors. So let's start by talking about character strings.

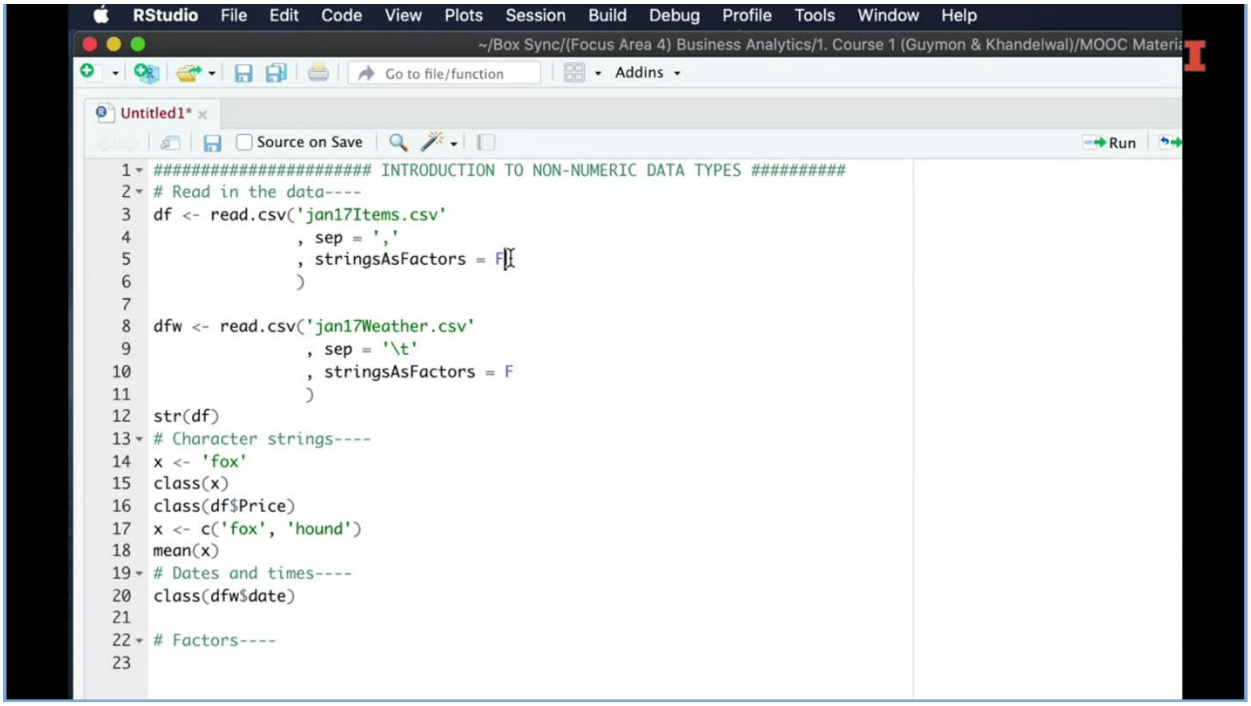


And let's start by creating a simple character string object. We'll create the object x and assign it the value of fox, the word fox. And you can see that I put in quotation marks to indicate that I'm not assigning it the value from another object, but just this character string of fox. Now, if you look over in the environment, you see that I've got this X. If I hover over a little, it says it's a character and tells me how many bites. It is. I can see it's fox. I can also find out the type of an object by using the class function. So I do.

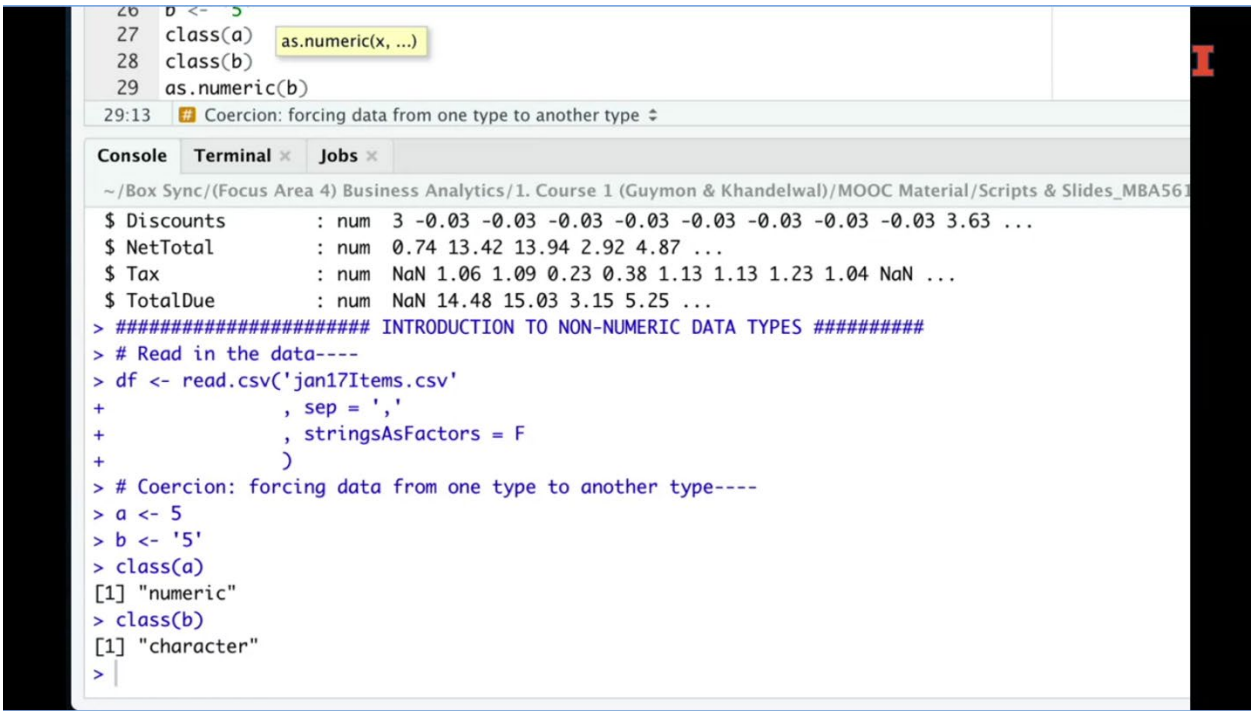
Class x down here in the consulate tells me it's a character. I can do that on a column of data as well. So if I do df and I'll just do time. That's also a character. If I do it on a different column, say price tells me it's numeric. Alright, I can create another character string vector by using the combined function, and I can enter two character strings. I'll do fox and hound, run that. And now you can see that x is a character. It does tell me now what type of object is. It's a character vector, and it has two values and fox and hound.



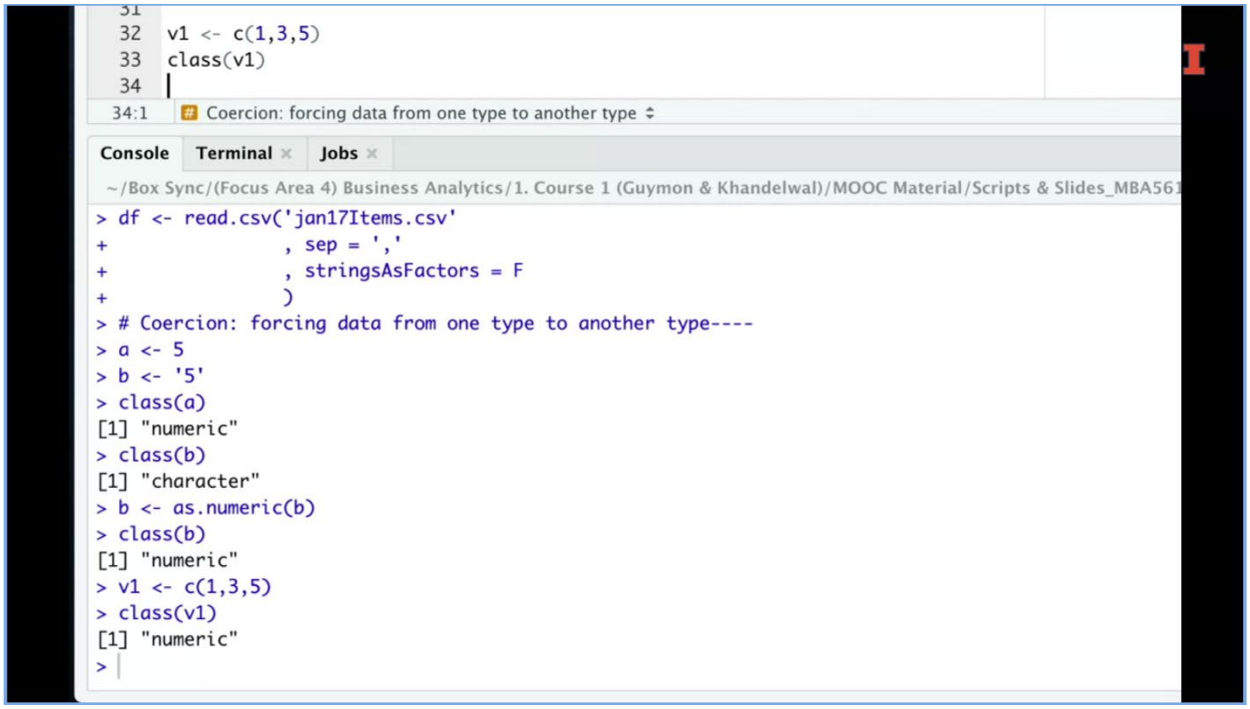
All right, so those are character strings. The important thing to recognize is that you cannot perform mathematical calculations on character strings. For example, if I tried to find out the mean of x, it would give me an error, because that's not a logical computation you can do with that. So that's important because sometimes you'll see that columns are read in as a character type instead of a numeric type. Now let's talk about dates and times briefly. [SOUND] If we look at the dfw, the weather data frame and look at the date column, we can see that it's obviously a date, and it also has a time stamp in there. So it obviously has a date and time format, but it's not a date and time type. In fact, if I look at the class of this column tells me it's a character, all right, so just keep that in mind, will spend another lesson working on dates and times. But at this point, just recognize that because it's a character vector, we cannot calculate, for instance, how far apart two different dates are or what the date will be 362 days in advance.



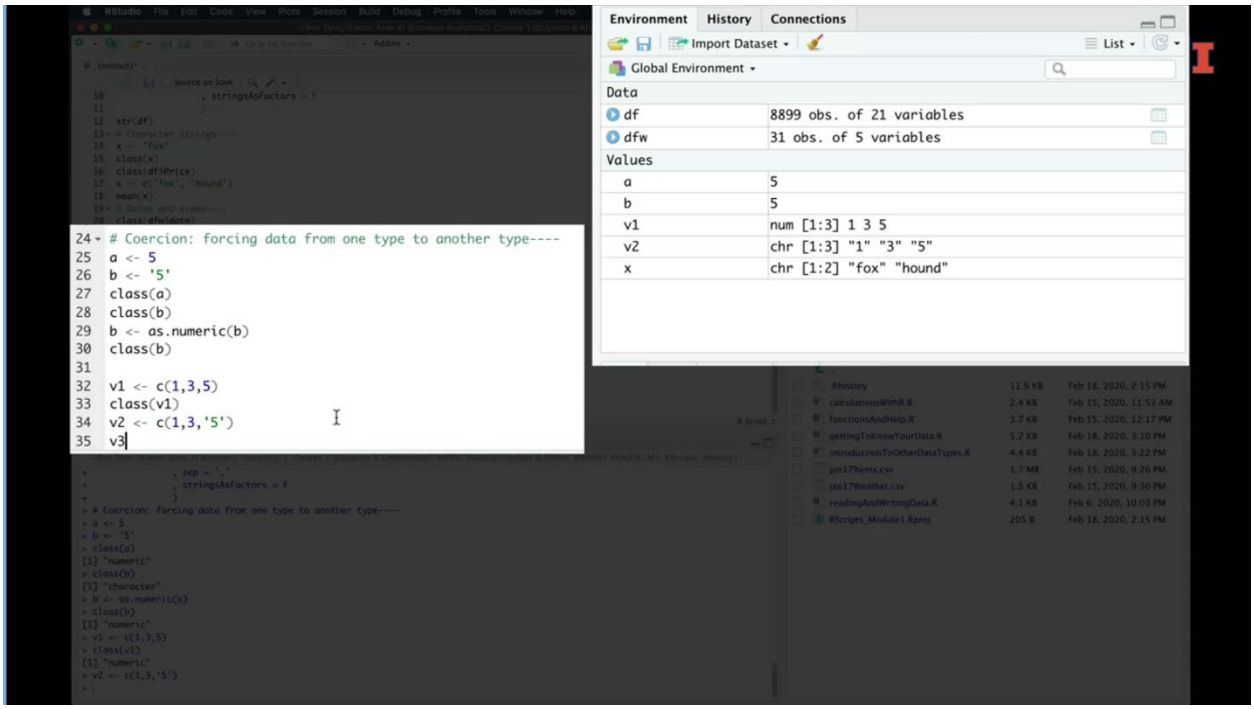
Now let's talk about factors. Remember that when we read in csv data, we have this argument where we indicate manually that strings as factors are false. What happens if we convert that to true and then run that and then look at the structure of df. Notice what happens to the data type for these first 12 or so columns. They change from character strings, to factors. So at this point, just recognize that our assumes that you want to make a conversion from character strings to numeric data types and factor is a special numeric data type, and we will talk about that later. So I'll leave that as strings as factors as false for now and read in the data.



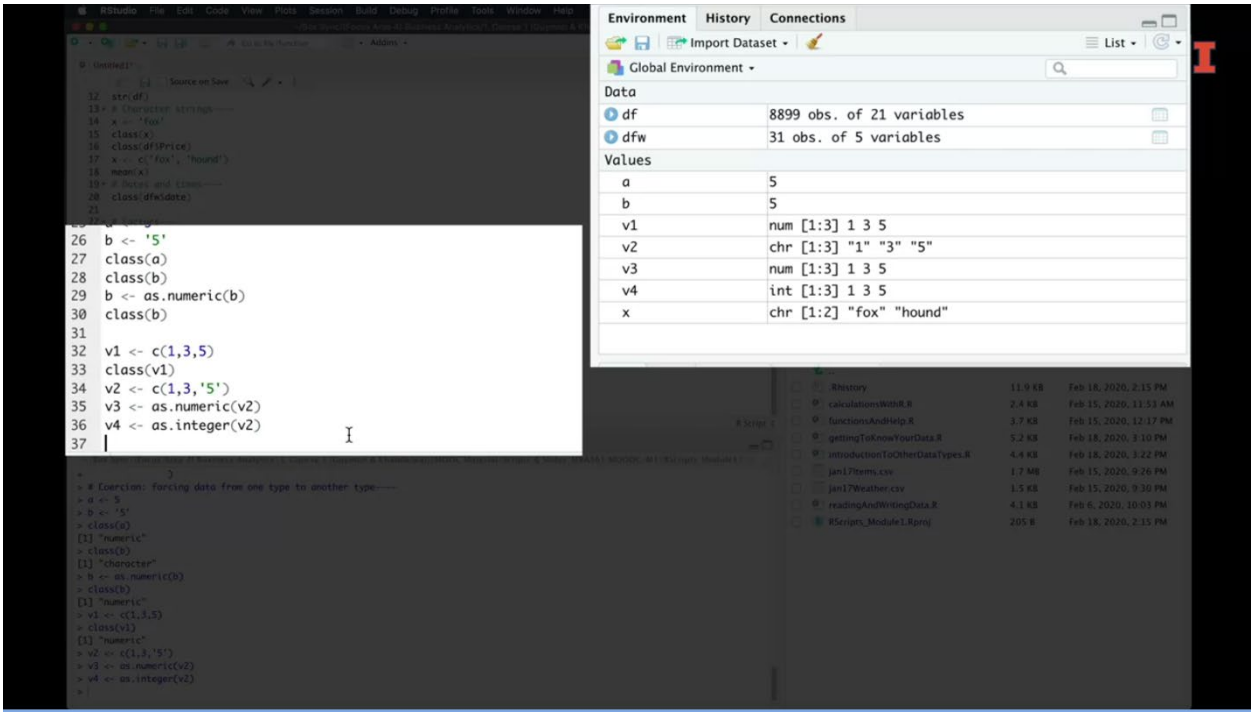
Okay, now let's talk about coercion, and what I mean by coercion is forcing data from one type to another type. [SOUND] This is important because, as I mentioned, sometimes data is read in and a column of numeric data is read in as a character string, and so you want to convert it to a numeric type. So let's start this discussion by talking about two simple objects. A will be numeric value of 5, b will be a character string of 5. All right, if I look at the type of A it is numeric and of B, it is character. So if I want to convert B to numeric type, I can use this as dot numeric function, and it will make b it will coerce b to be a numeric type so I'll assign that to itself again.



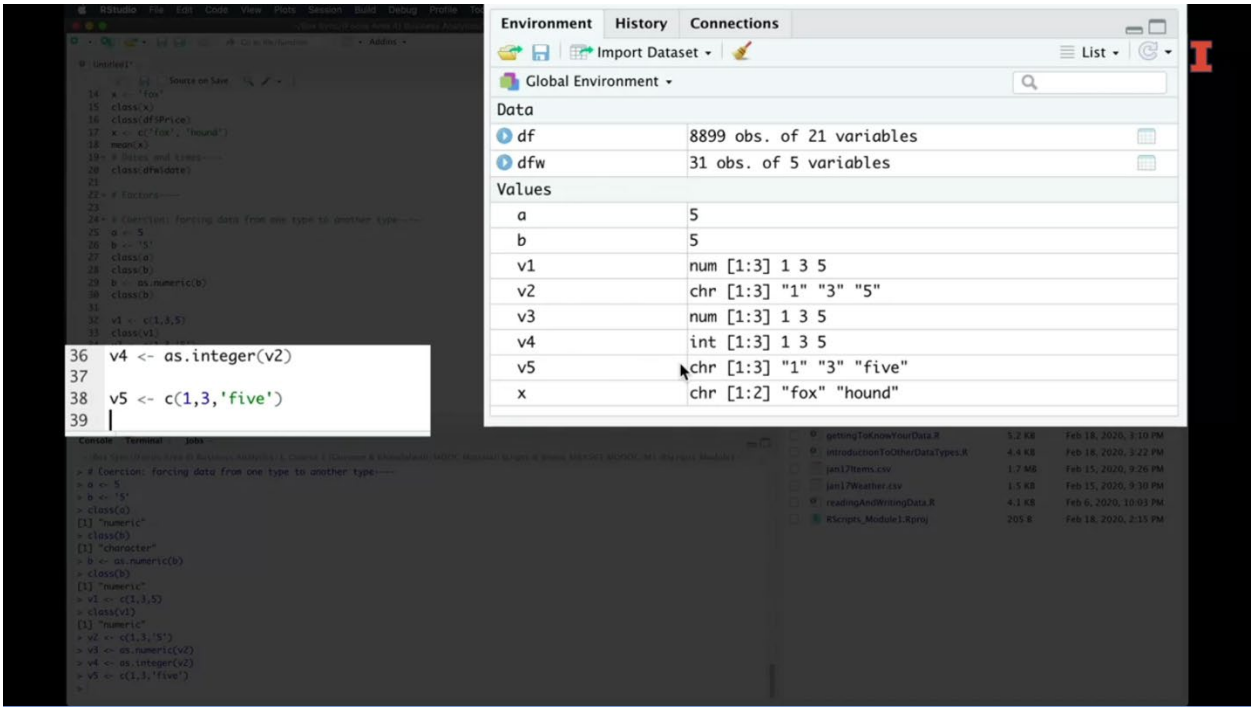
And now if I look at the class of B, it is numeric, so you can explicitly force objects to be a certain data type. So why are numeric vector sometimes read in as character vectors? Well, let's look at what happens if we create a numeric vector. Let's say of 1, 3, 5 and we look at the class, it is numeric.



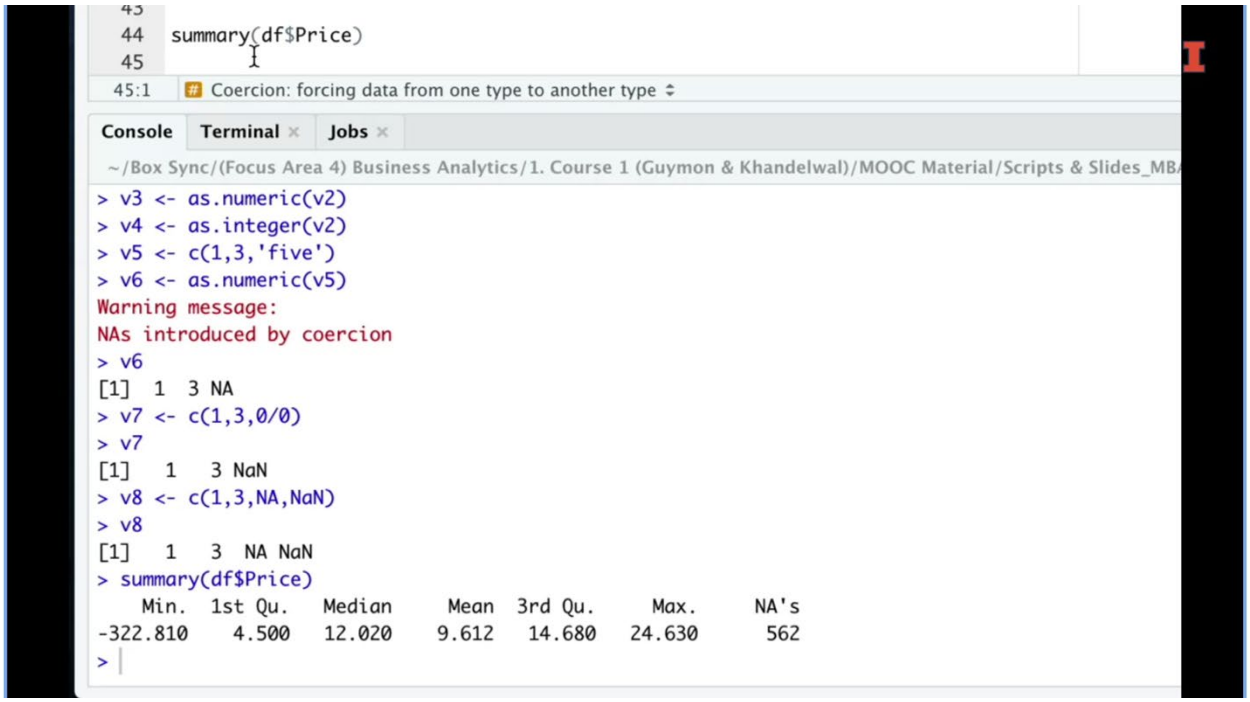
And now let's compare that to a similar vector that has mixed types in there. So I'll do 1, 3 and then in quotation marks 5. And if I look at the data type for that, it is character.
So whenever you mix together numeric and character data types, it will coerce all of the data to be a character data type. And that's typically the case for all data types.



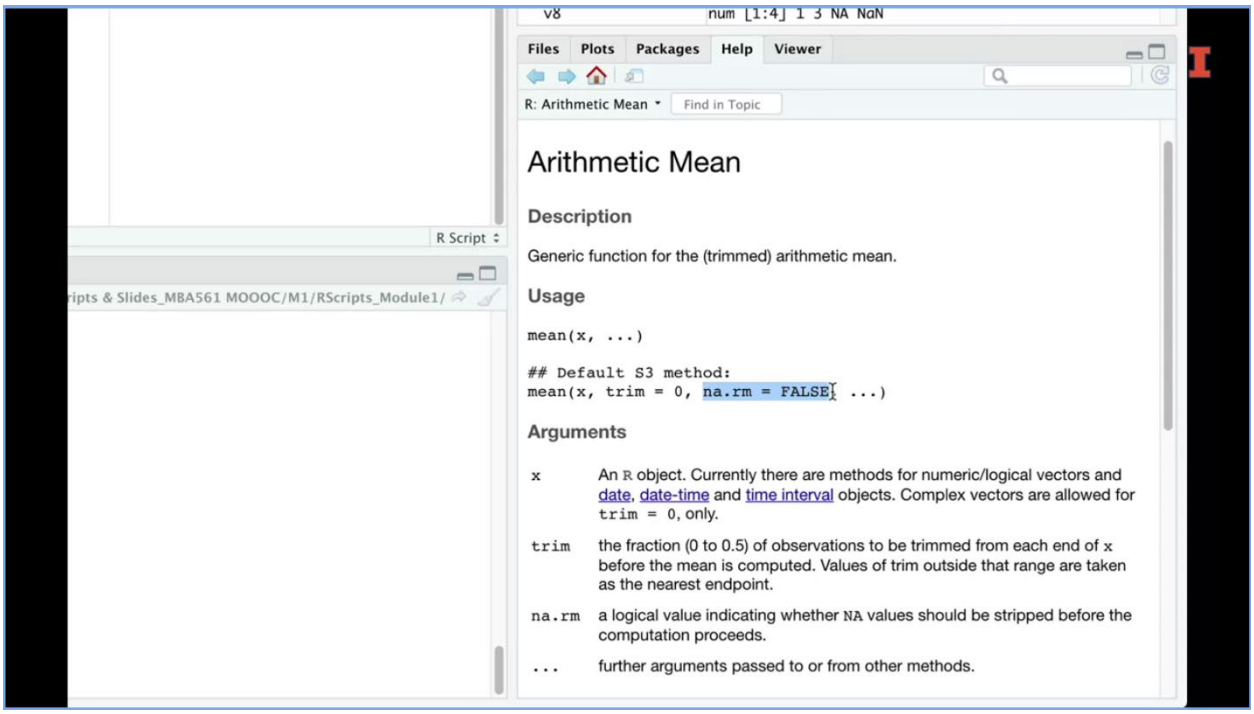
Characters like the default data type. We could coerce v to to be a numeric data type by using this as numeric function for v2. And now if we look at v3, it has the same values in it. But there are numeric values instead of character values, all right, we can also coerce those to be integer, and if we look at v4, we can see that those are integers, now. The difference between integer and numeric really isn't important at this point, so we won't worry about that right now. So one other thing that I want to mention with respect to coercion is what happens if you have a vector that includes numeric values and strings that can't be converted into numeric types.



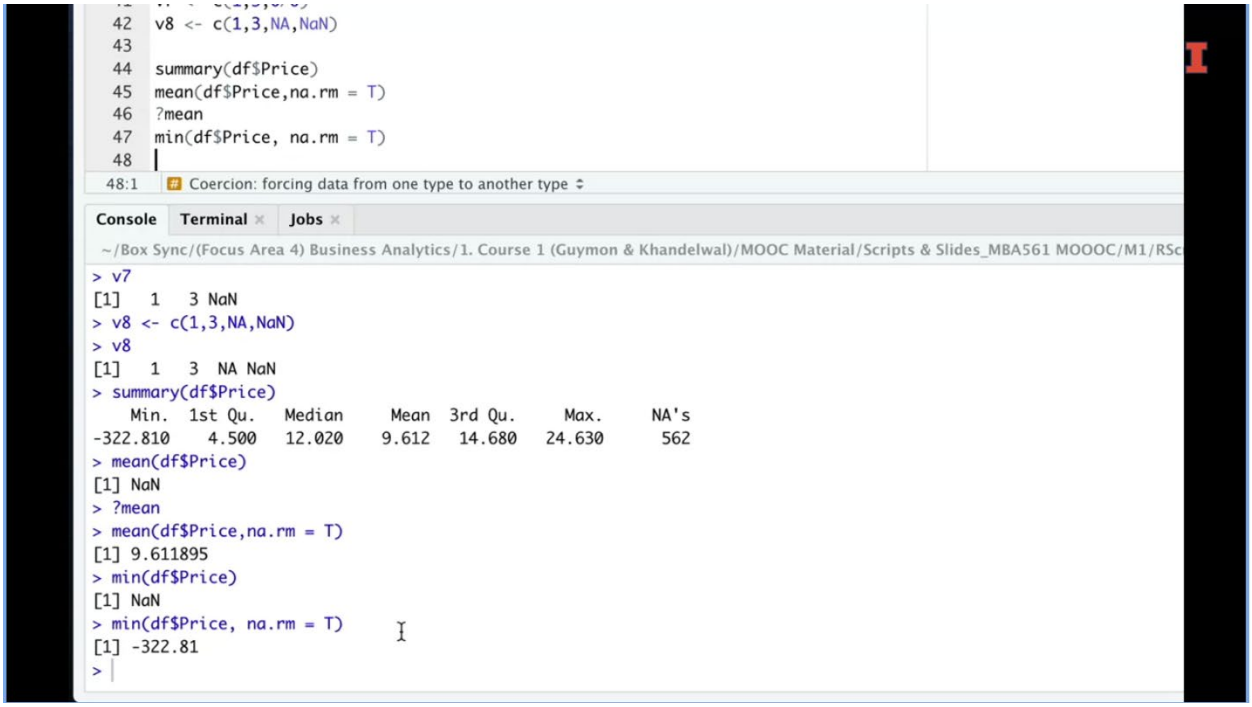
So let's say, for instance, I've got vector 5 and that is equal to 1, 3. And then the word 5, you can look and see that, that is a character vector. If I try to coerce that to be numeric, I get this warning at the bottom that, says NAs are introduced by coercion, and I can look over at v6 or print out v6, and I can see that it is 1, 3 and then that word 5 was converted to NA. Now NA stands for not applicable, meaning it couldn't really coerce the word 5 into a numeric value. So it just puts in NA, now. Sometimes you'll also see something similar to NA, which is NaN, and that happens if you have a value that is not a numeric value.



If, for instance, you have 1, 3 and let's say 0 divided by 0. All right. And then we print out v7. You can see that it changes that 0/0, which would be undefined to NaN. For not a number, you can even manually, you can create vectors that have NAs and then a ends in them just by using NA and NaN, and you can see that they turn blue because they are special words, reserved words that have a special meaning to them. And anyway, so those are important because when you see those in a character or sorry in a numeric vector, it's important that you know what to do with them. Let's say, for instance, I want to calculate the summary statistics for the price column of the items data frame. I can do that, and it tells me that I've got 562 NAs in there, now by default.



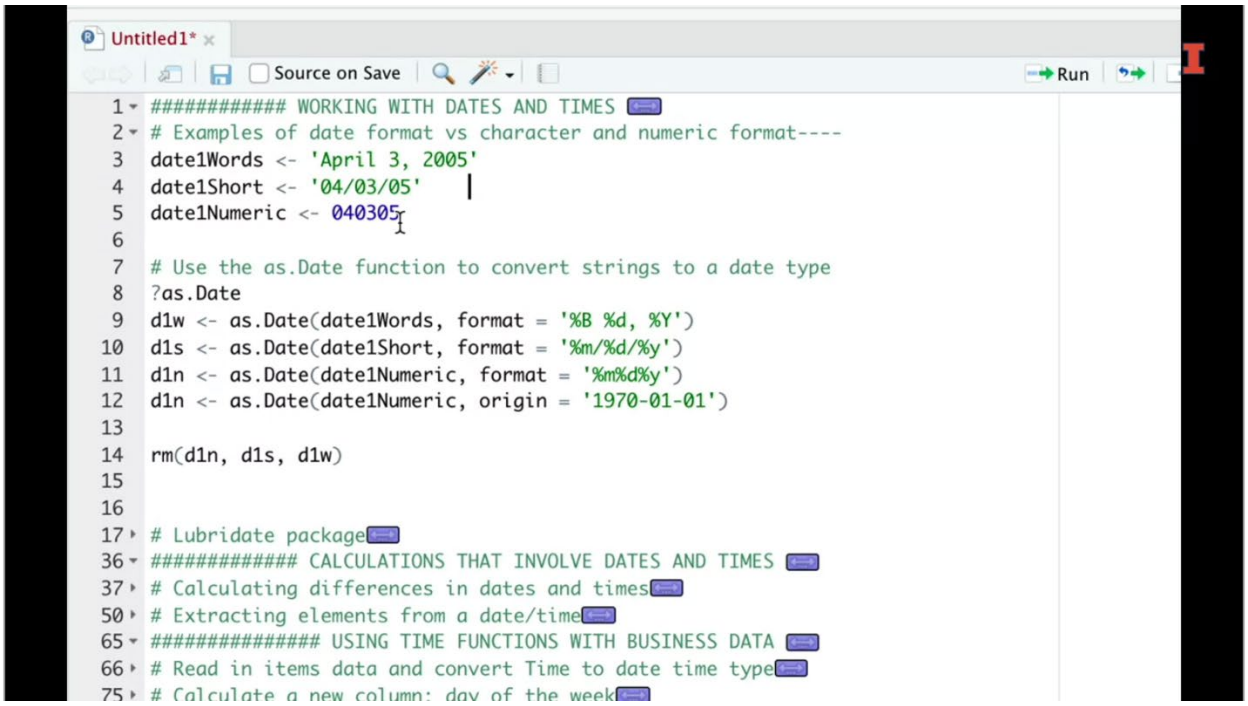
If you try to perform a numeric calculation on this vector, this price column, it will try to include NAs and NaNs in that mean calculation, and it can't do it. And so it returns NaN because it doesn't know what to do with NAs. So if you look at the help for these calculations, you can see that one of the arguments is na.rm, and it is defaulted to false. Now what is na.rm stand for? It says it's a logical value, indicating whether NA values should be stripped before the computation proceeds. So if it's false, they will not be stripped. If it's true, then they will be stripped. So we need to make sure that we include explicitly this attribute na.rm, and set it equal to true meaning. We want to remove the NA values.



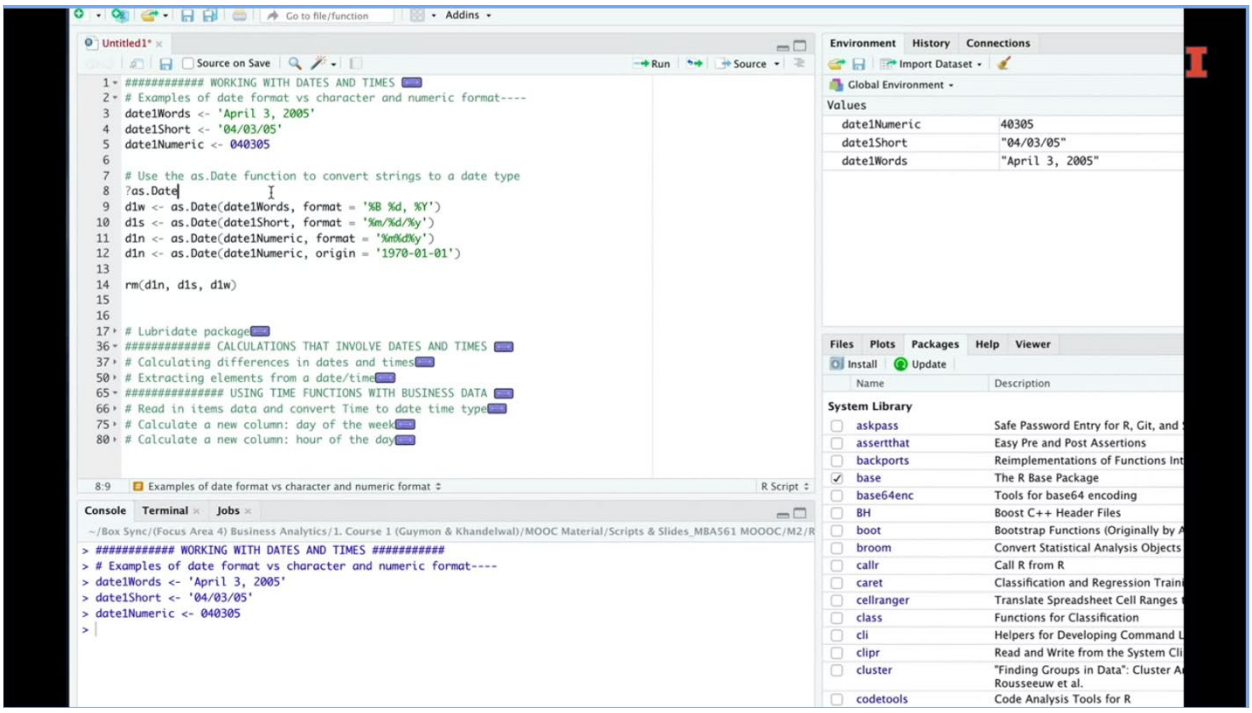
And now, if I run that it will tell me the value. The mean value of that price column in the df data frame. And that is the case with many other numeric functions. So, for instance, if I tried the minimum of the price column, it will return NaN unless I explicitly indicate na.rm is equal to true. And then it gives me the minimum value, so there's an introduction to a few other non numeric data types as well as what to do when you have NA and NaN in numeric vectors.

Lesson 3-6 Creating Date Types

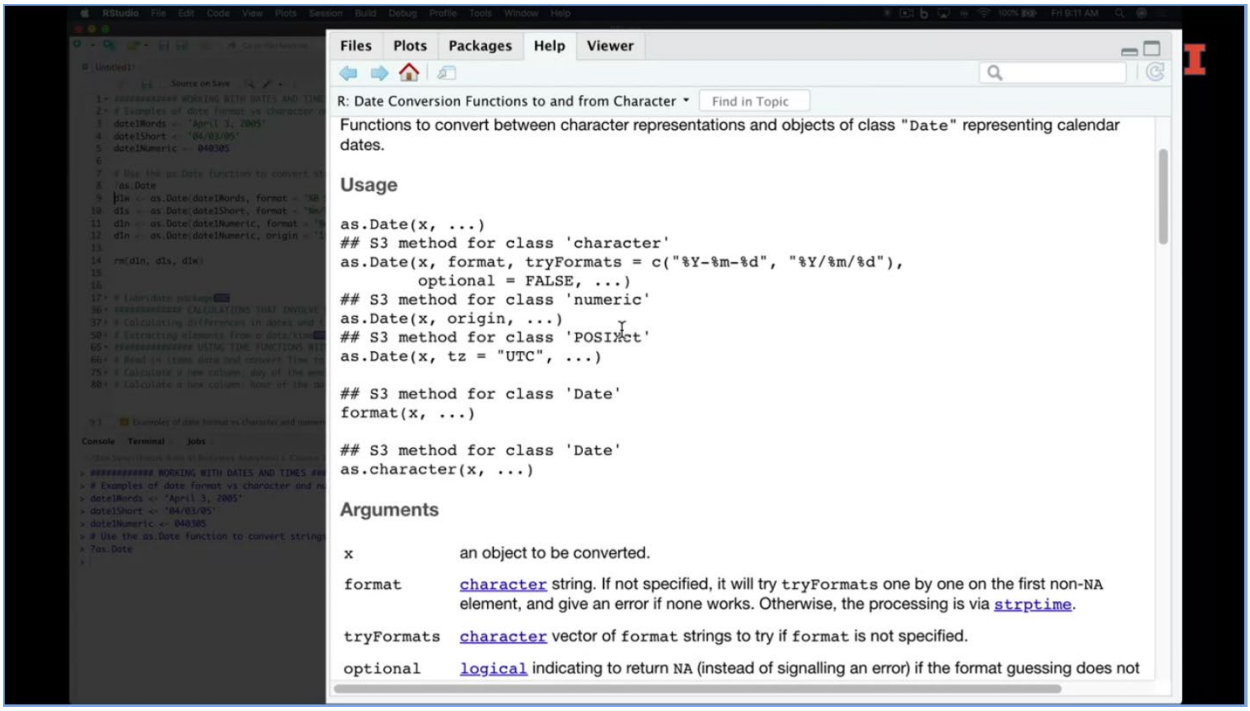
Lesson 3-6.1 Creating Date Types



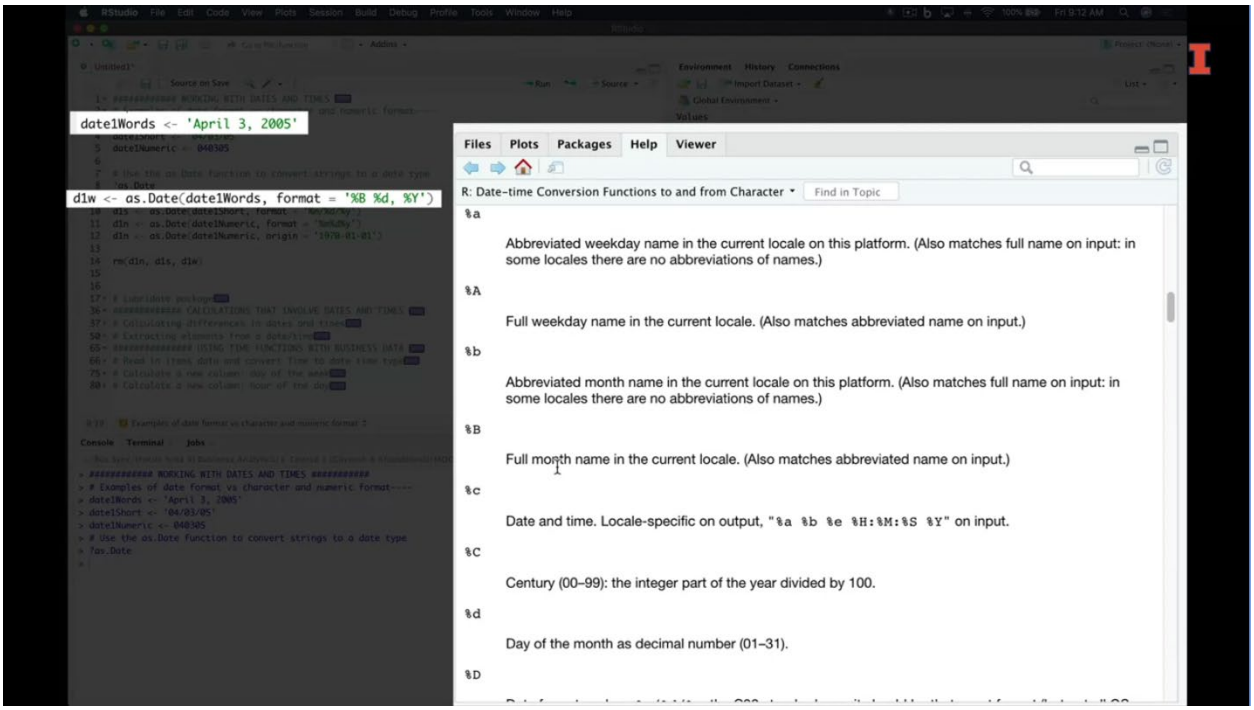
In this lesson, we're going to focus on understanding the date format and how to convert strings and numeric values to a date format. Dates are a big topic, we won't be able to cover everything in this lesson or this course, even on dates, but hopefully, we'll give you enough to get you started so that you can accomplish many of the tasks that are required in business analytics. Now, business data often has dates in it. One of the problems with dates though is that they recorded in many different ways. For example, you can record the date of April 3rd, 2005 in this manner, or what is often the case where I live is it's abbreviated as a two-digit month forward slash two-digit day forward slash two-digit year.



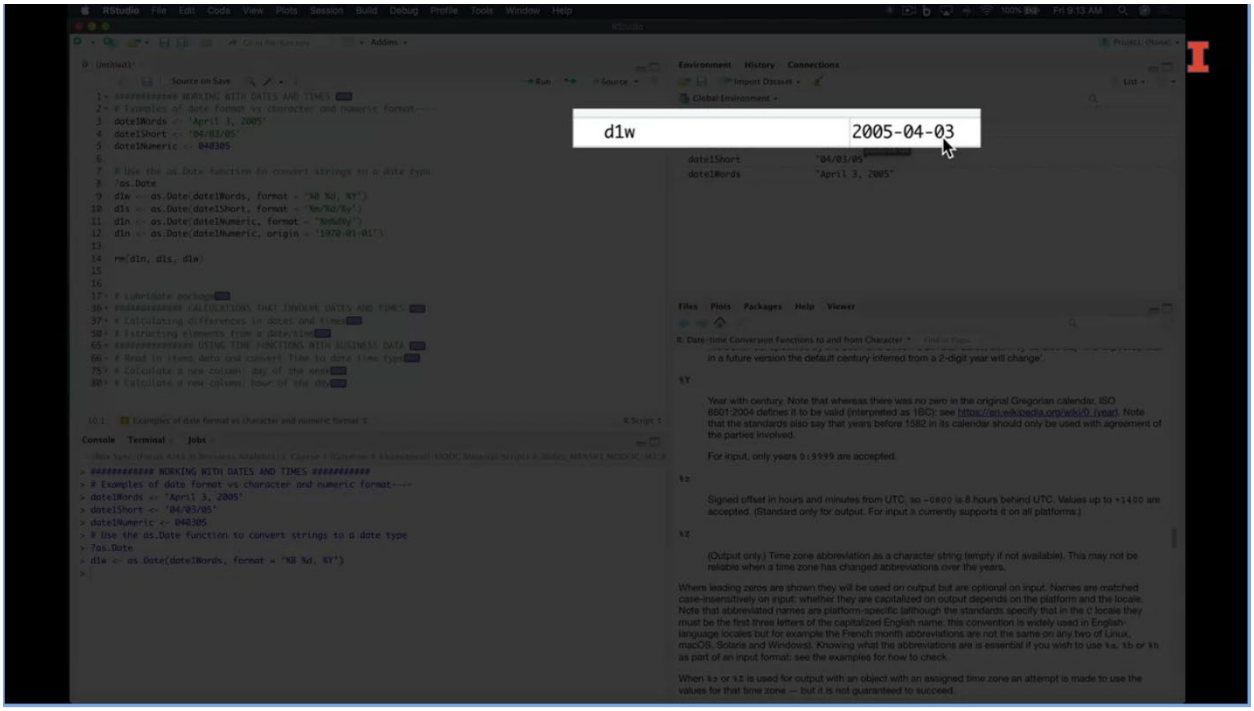
Then sometimes that can be recorded just as a numeric sequence of numbers, like so. I'm going to create these three objects here. The key with dates is that we want to both be able to read what the date is, and also we want to be able to make calculations with date, such as calculating the difference between two dates, how much time has passed. Let's start by learning about some of the base functions in R for dealing with dates. The key function that I'd like to teach you about is the as date function.



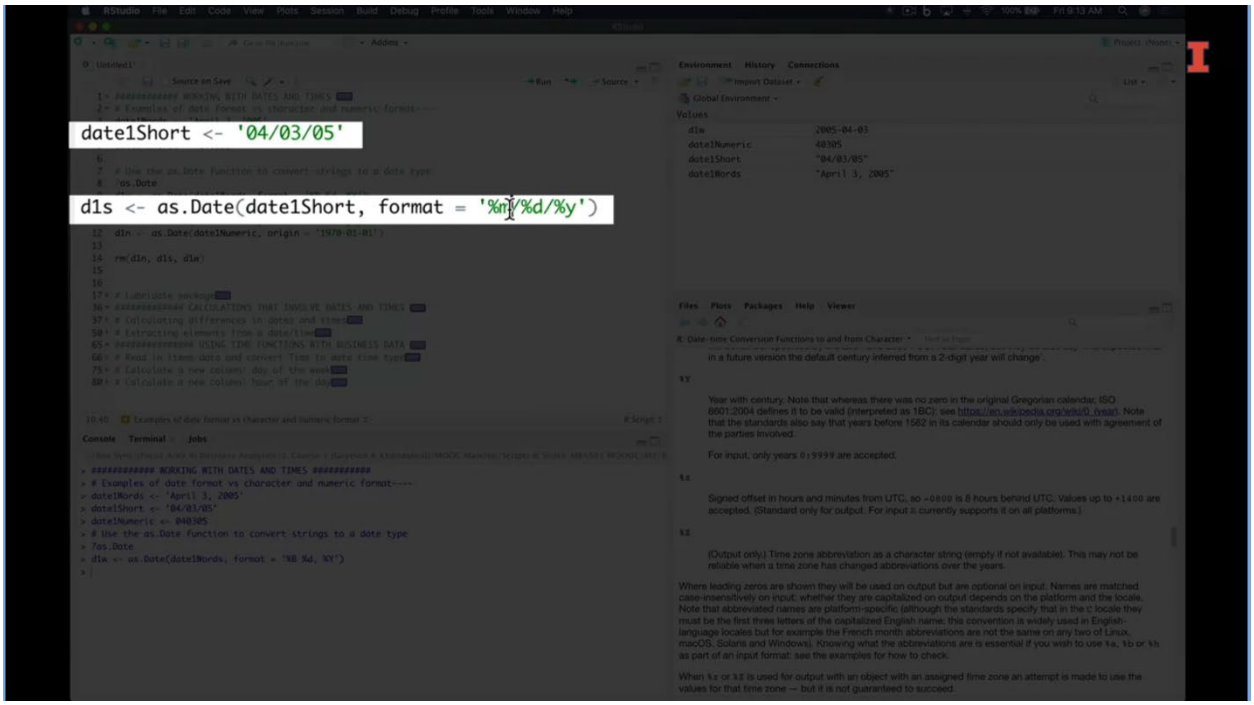
Let's look at the hope documentation for this function. If we look at this, there are not a lot of arguments associated with it, but one thing that we do have to specify when using this function is the format in which the string is coming in. If we click on that strip time and look for the help in that, it takes us to a list of the abbreviations for indicating what type of format the date is recorded in. When we use this function, the way we do it is we call the function and then we add our date object, and then we have to specify the format.



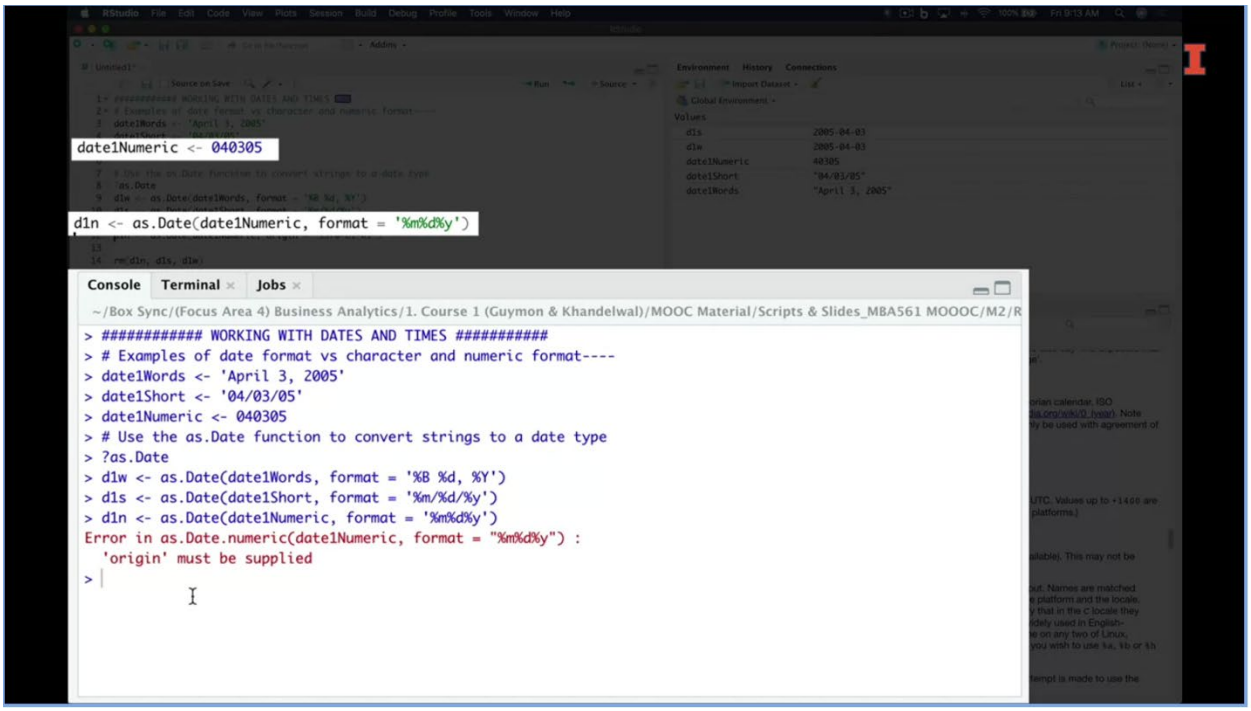
This date one word is the April 3rd, 2005. I have to indicate with a percentage sign in a capital B that this is the full month's name. Then there's a space, and then with the percentage d, that indicates the day of the month as a decimal number, and there's a comma, and then percent capital Y, and that capital Y is the four-digit year.



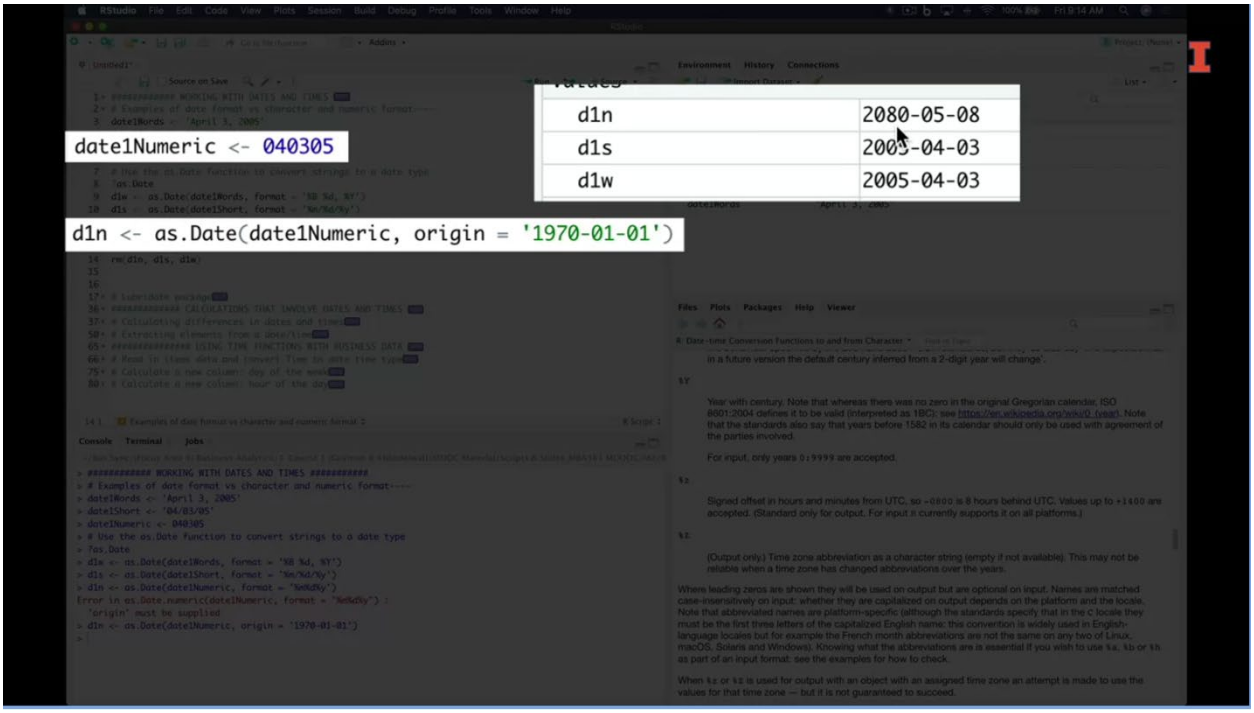
Let's go ahead and run that, and it creates this new object, d1 w. You can see that the format is the four-digit year dash, two-digit month dash, two-digit day. This is a standardized format known as ISO 8601 format. This is the format that is most often used when programming.



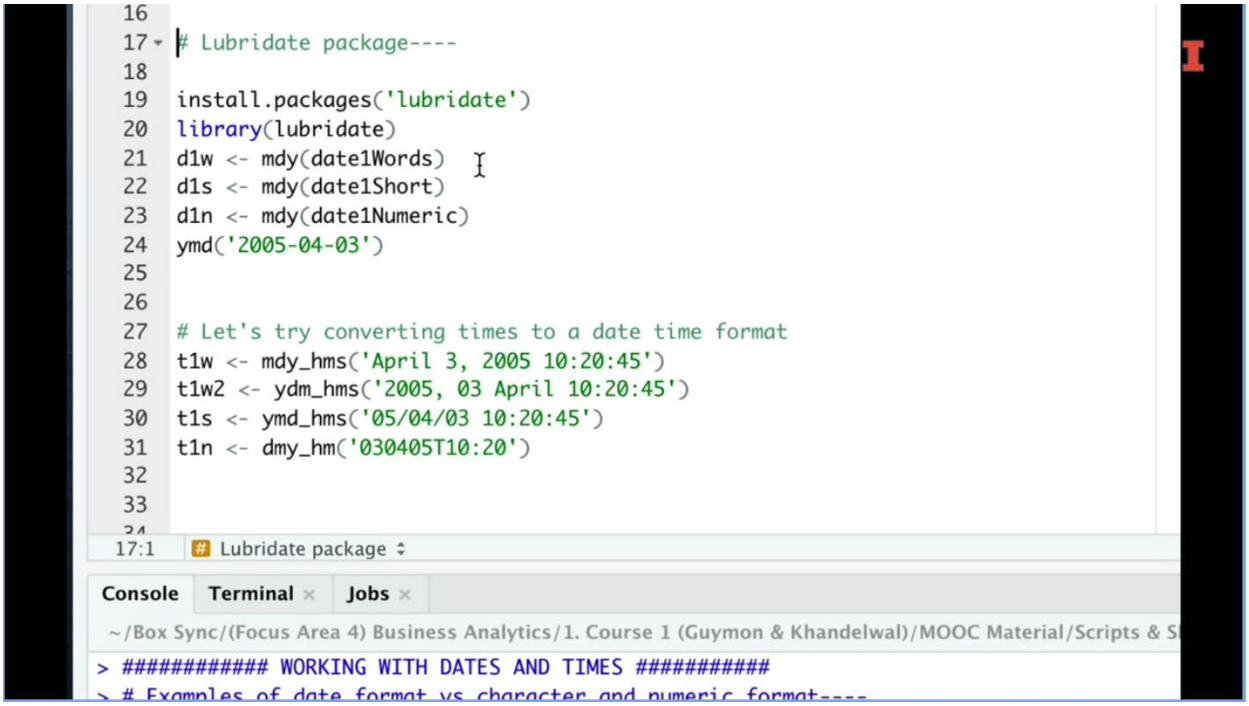
Now let's go ahead and illustrate what you would need to do to use the update function to convert the same date but in a different shorthand notation here. We'd have to use a lower percent, lowercase m forward slash, and so forth, but if I run that, it gives us this new object and it is the same date so it worked. What about this numeric version down here?



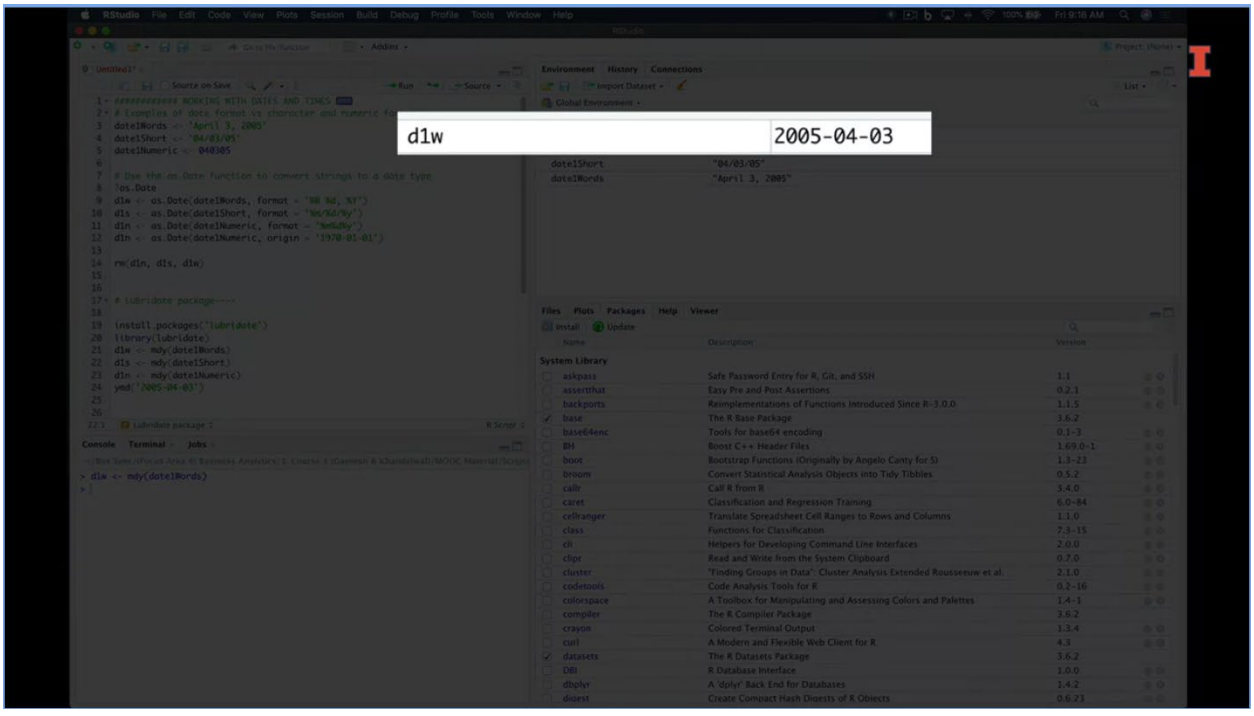
If I run that and I tried to indicate using the symbols what it means, I get an error. This is because it's expecting a string and instead I gave it a number. When I give it a number, it says, he must be telling me that you want me to calculate how many days it has been since the epic. I have to supply that epic.



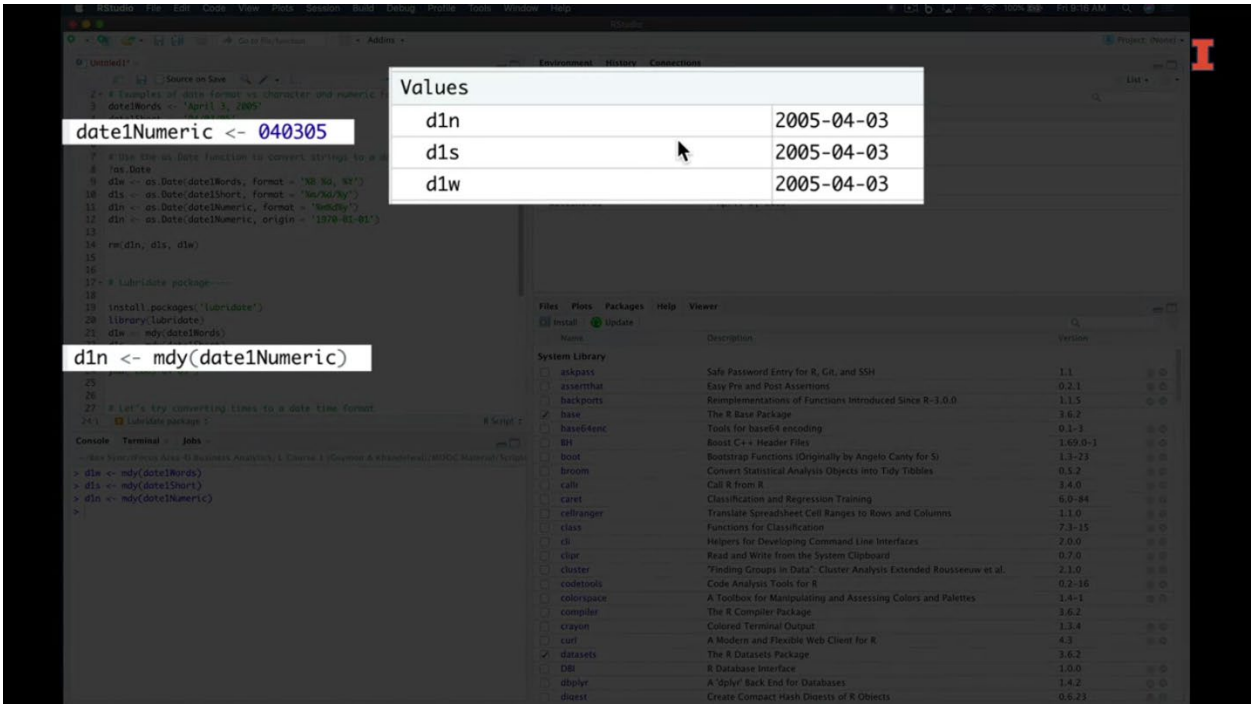
This next version here, I supplied the origin of January 1st of 1970. If I run that, I don't get an error and I get this new object. It worked, however, it is not the right date. Apparently, 40,305 days after January 1st 1970 is May 8th, 2080. It didn't convert it to the right date, even though it is a date format. You can see that there are some important considerations that you need to be aware of when converting the way a date is recorded into a date type. Let's go ahead and remove those objects that I just created.



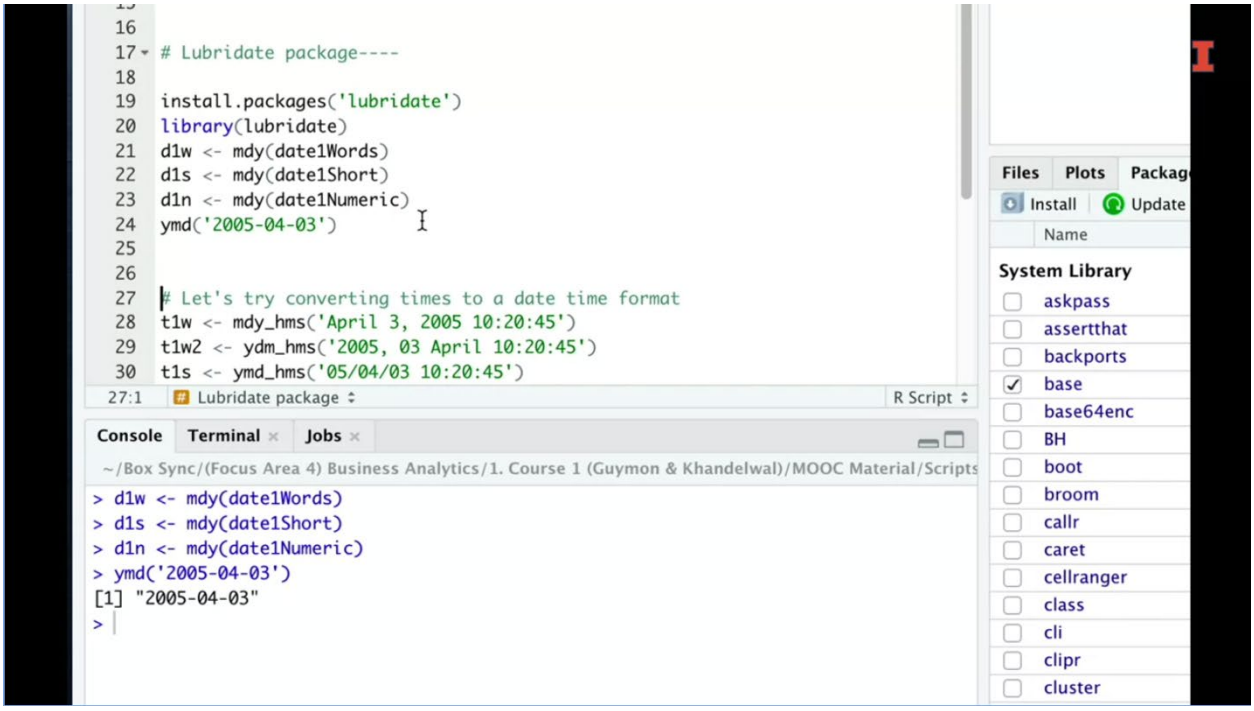
Now let's talk about the Lubridate package. The Lubridate package makes dealing with dates a whole lot easier. If you haven't installed the Lubridate package, go ahead and install it by using the install.packages function, or you can go ahead and go over to the packages pane and click on the Install utility and type in Lubridate and install that. Now I've already installed it. The next thing I need to do is make sure it's loaded, so I'll go ahead and load the package. I'll run that library Lubridate.



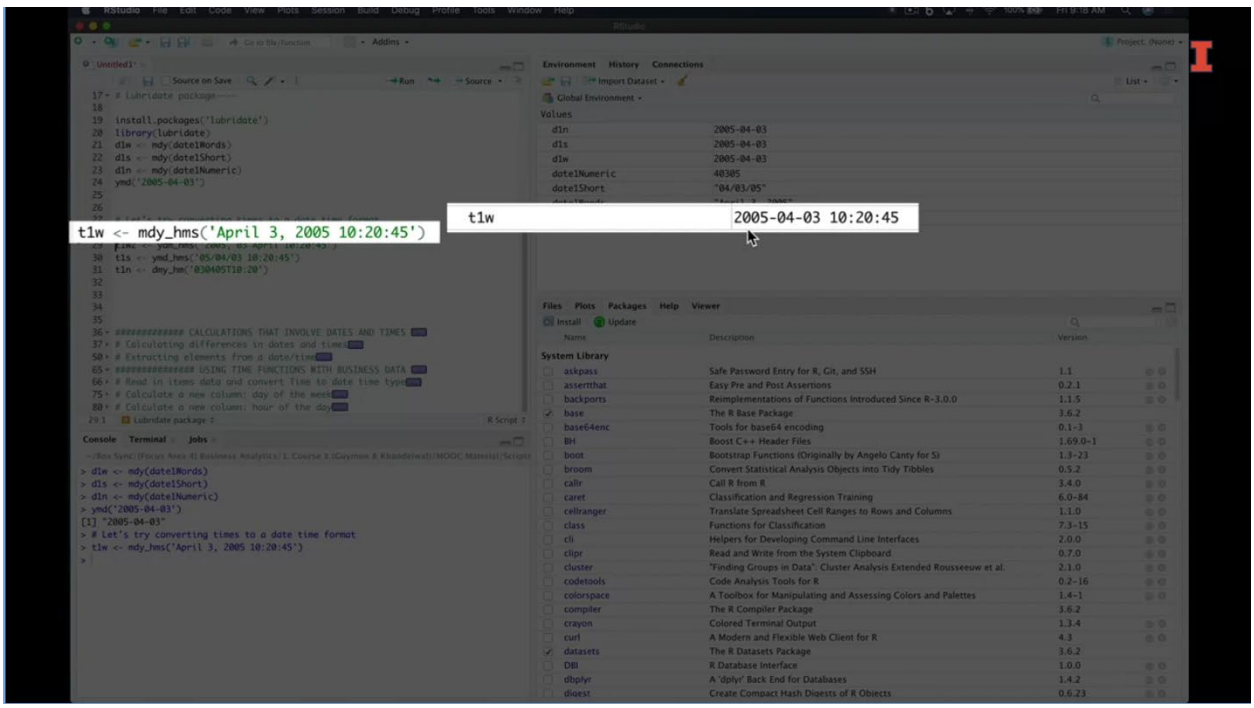
Now I can access the functions in the Lubridate package. Now I'd like to illustrate how we can make these conversions on these strings in numeric representations of date into the appropriate date type using Lubridate. Let's start with the date one words, which is this April 3rd, 2005. The function I'm going to use as MDY, and what that stands for is month, day, year. Basically I'm saying first is the month, then as the day, then it's the year. I'll run that, and I can see in the environment that this object was created and it definitely looks like the right date, so very good.



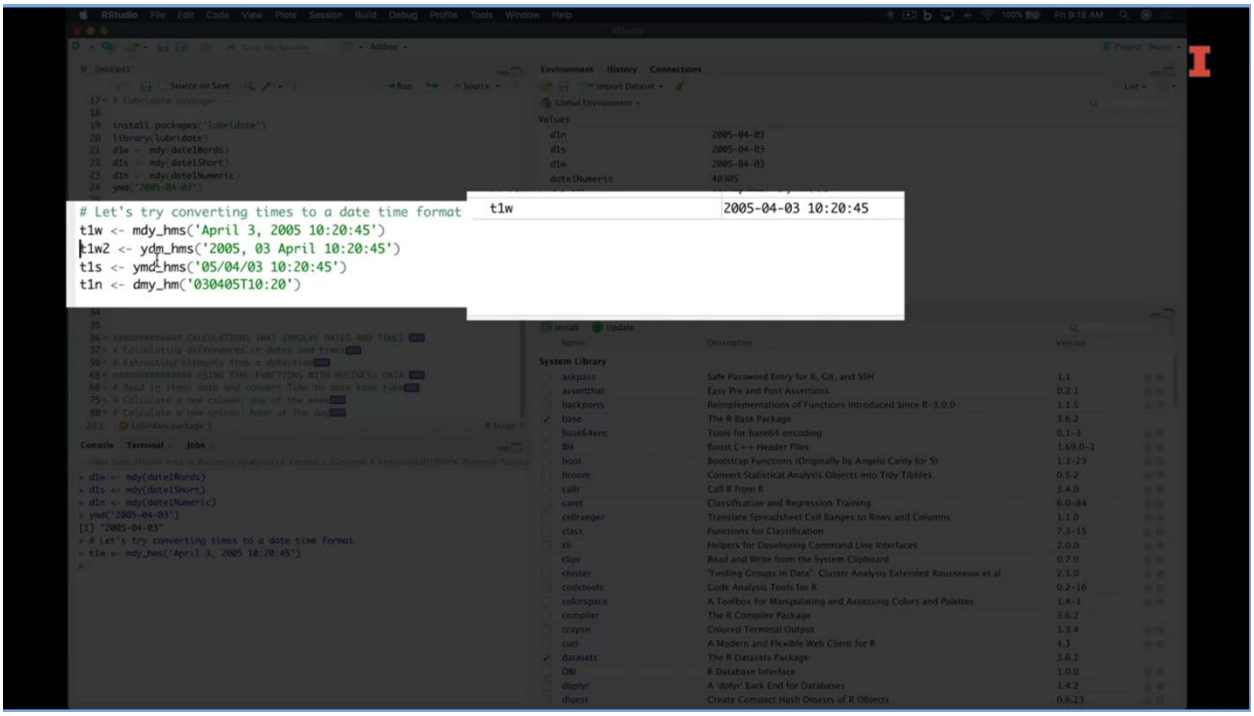
What about if I use the MDY on this date one short, go ahead and run that, and awesome. It converted it to the same date, exactly what we want. Even though it was a different format, since it was the month, day, year, Lubridate was able to interpret that 04 meant the month and converted it to April. What about the numeric version? We'll go ahead and run that. Wonderful. This converted it to the right data as well, even though it was a number and not a string. This is awesome because it's just one function and it's taking into consideration the different ways in which the date could have been recorded and making the appropriate transformation to the date that we want and the date format.



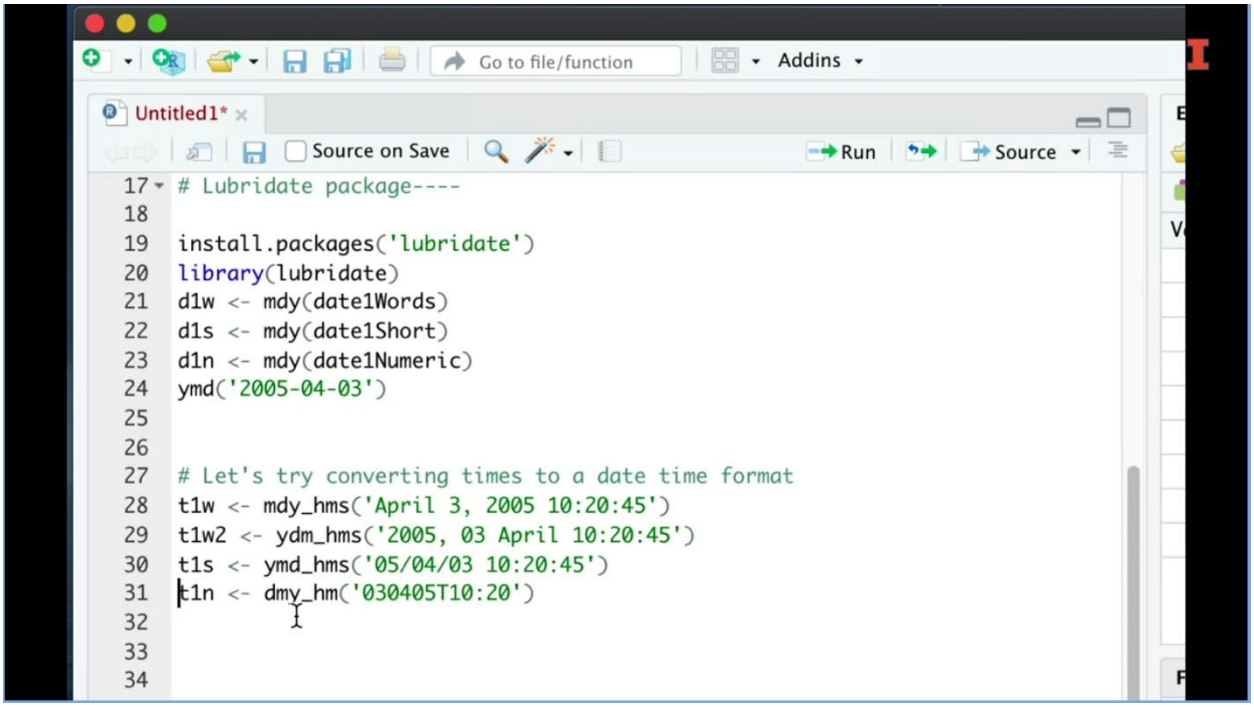
Now, what if the date is recorded in the ISO 8601 format? Well, I can use a different function. This is YMD, for year, month, day. Let's just run that and look in the console, and sure enough, it converted it to the right date. Hopefully this helps you appreciate the beauty of Lubridate and how much easier it is to work with dates using this Lubridate package.



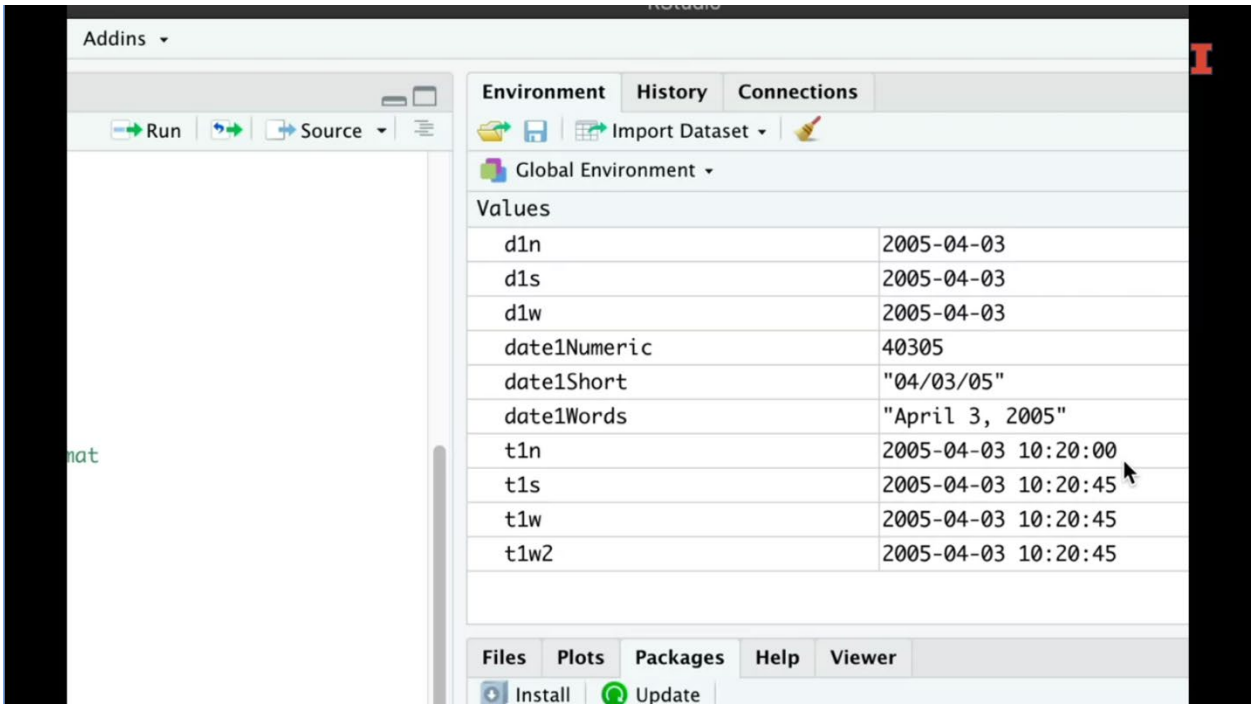
Now let's also talk about converting times to date time objects. Here is April 3rd, 2005, and then a random time of 10: 20: 45 seconds. There's a function in Lubridate that similar to the MDY function, but in this case we're going to put an underscore and add on HMS for hours, minutes, seconds. If I run that, I can look over in my environment and I can see that this object was created and it shows the ISO 8601 format of the date. Wonderful it worked.



What if I have the year, then the day, and then the month, and then the time, well, I use this YDM, easy to remember year, day, month and then underscore hour, minute, second. Let's look over and they Environment. Sure enough, it created the right tag, the right datatype. Very good.



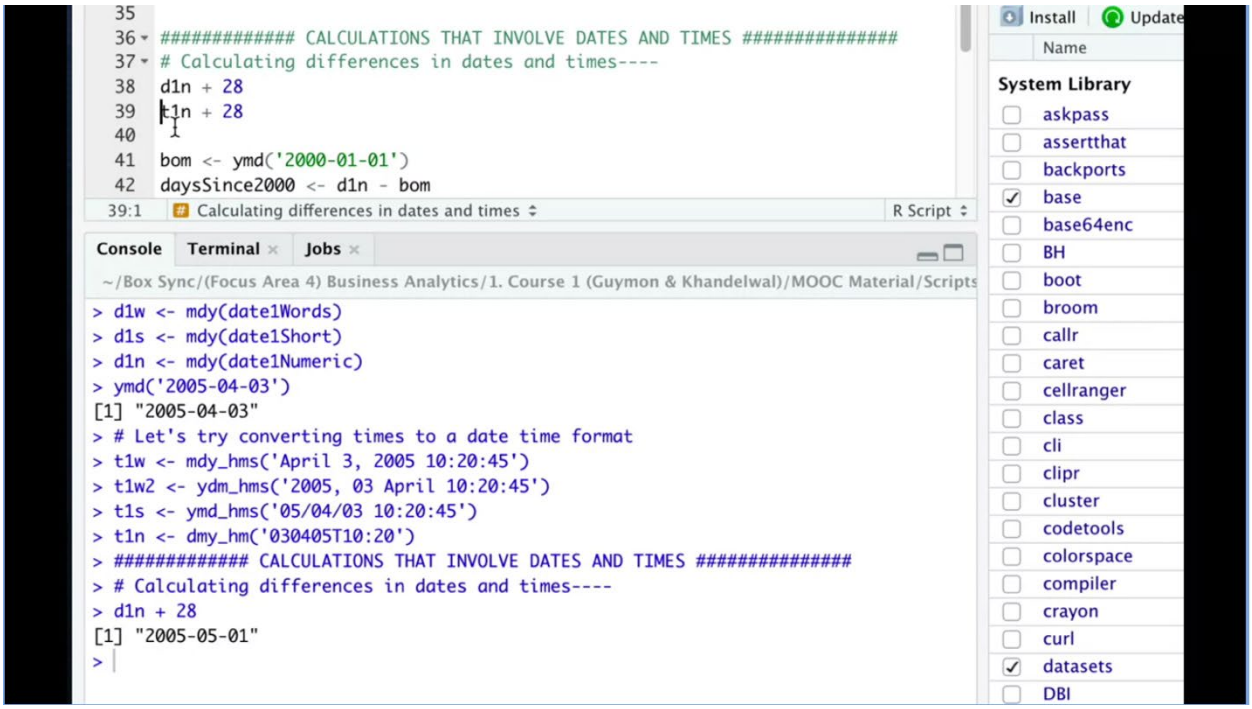
Let's see what happens if I have the year, month, day, and then hour, minute, second, I can use a different function, but it easy to remember that run that. Once again, it converts it to the right data type. What if I have a situation where I have the day, the month, and the year, and then a t to indicate time, and then I just have the hours and the minutes and now seconds. You can probably guess at this point what the function will look like. There'll be DMY for day, month, year underscore, HM, for hour, minutes.



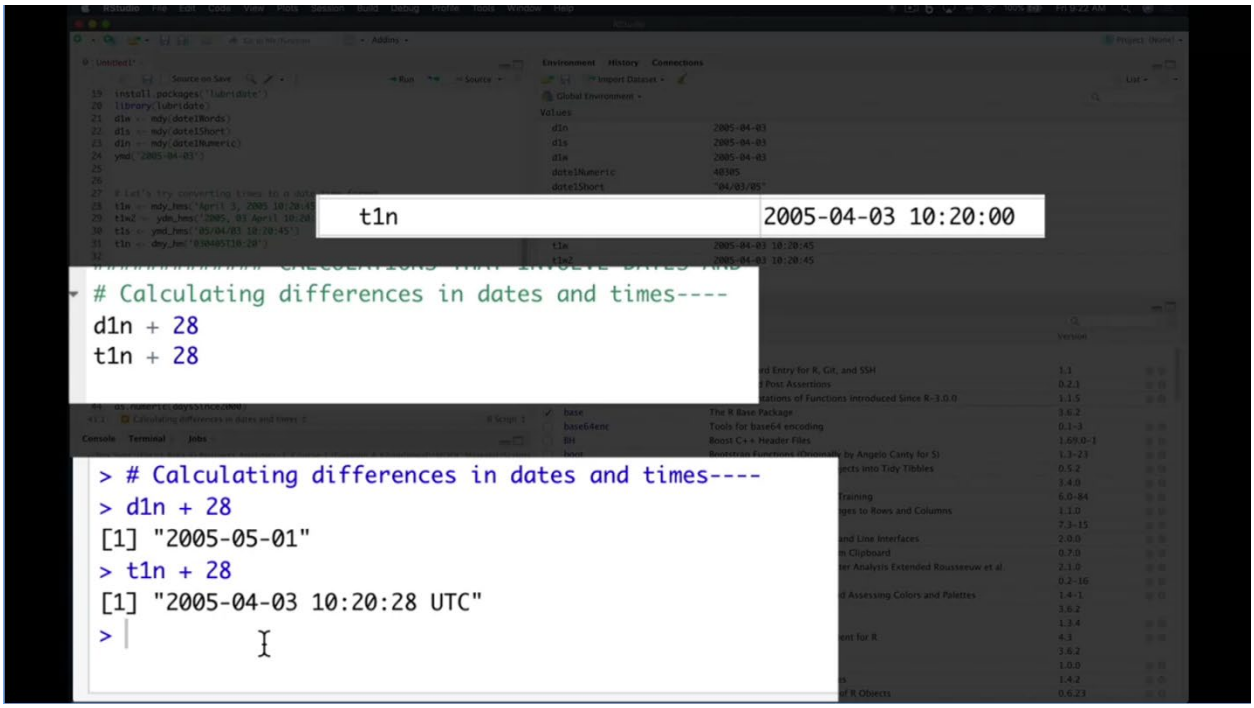
If I run that and look over in the environment, it creates a datetime object that has the date and the seconds are just rounded to the beginning of the minute. That's a brief introduction to converting strings and numeric values to date objects using the base package and the Lubridate package.

Lesson 3-7 Calculations with Dates

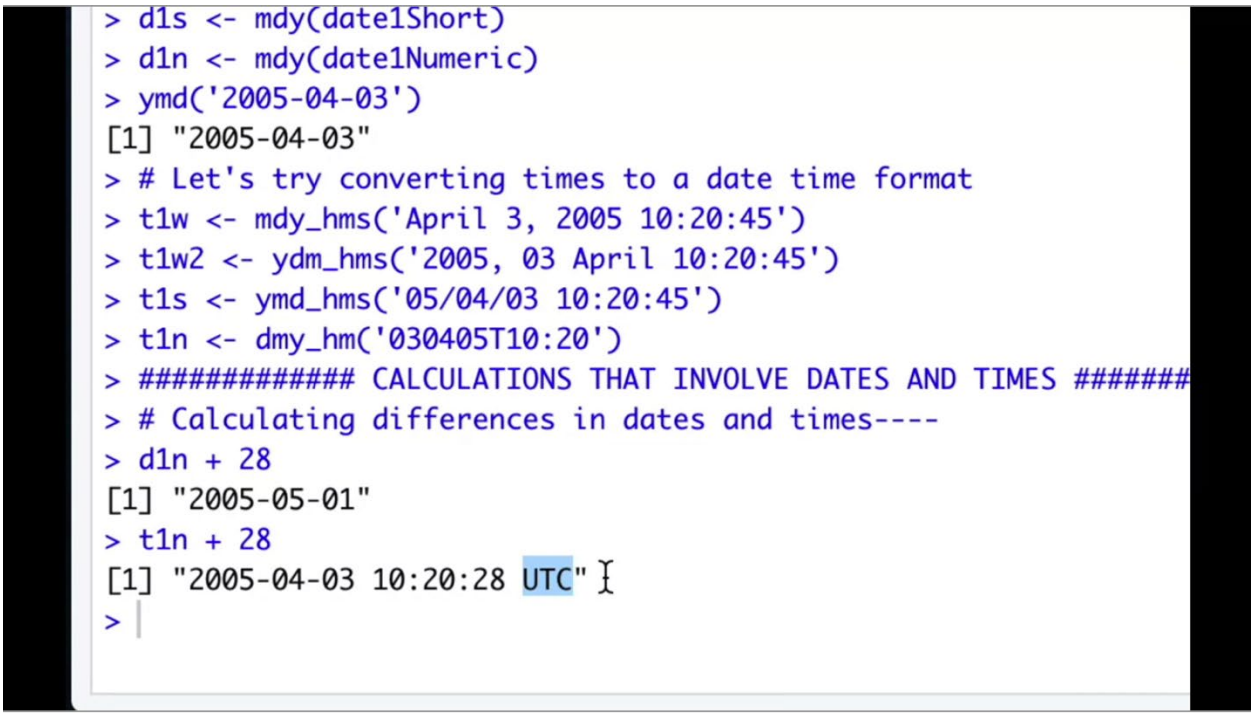
Lesson 3-7.1 Calculations with Dates



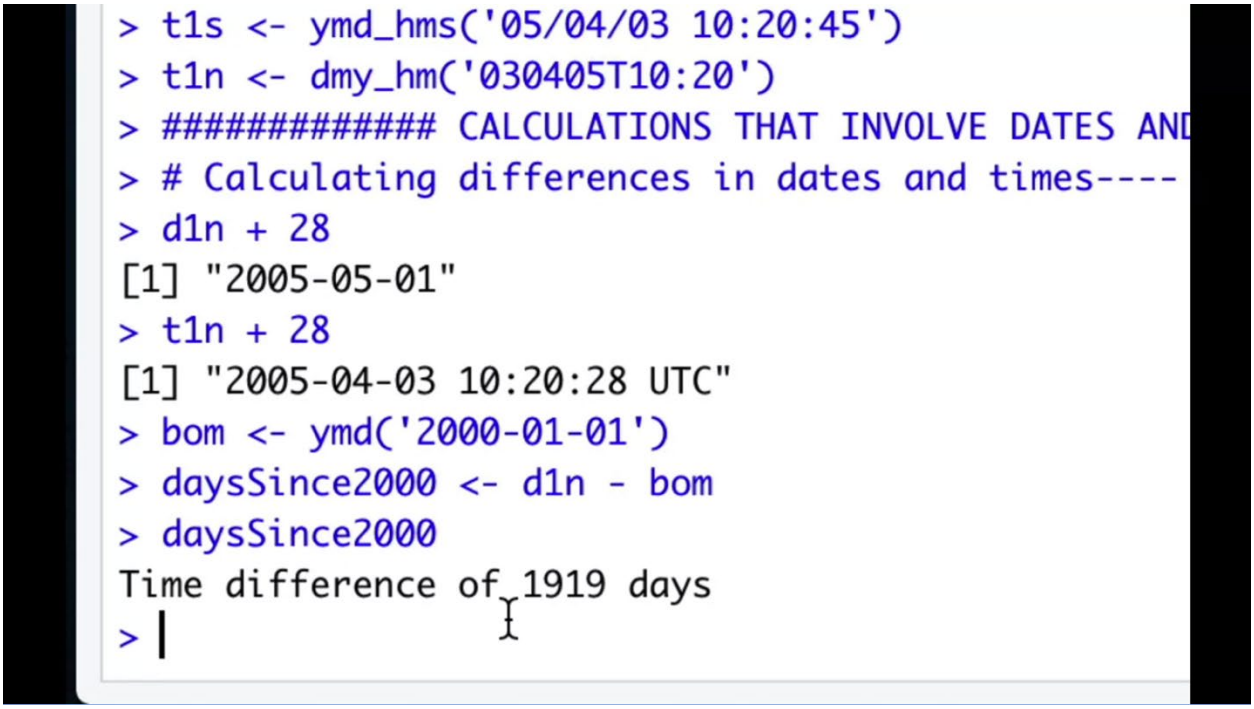
In this lesson, we'll focus on making calculations that involve dates and times and then applying what we've learned about dates to business data. So, let's go ahead and take this date object, d1n which, if we look in the environment, this is a date object that represents April 3rd, 2005, and we just add a number to it. Add 28. You can see that in the consulate prints out a date, May 1st 2005 that is 28 days after that original date of April 3rd, 2005.



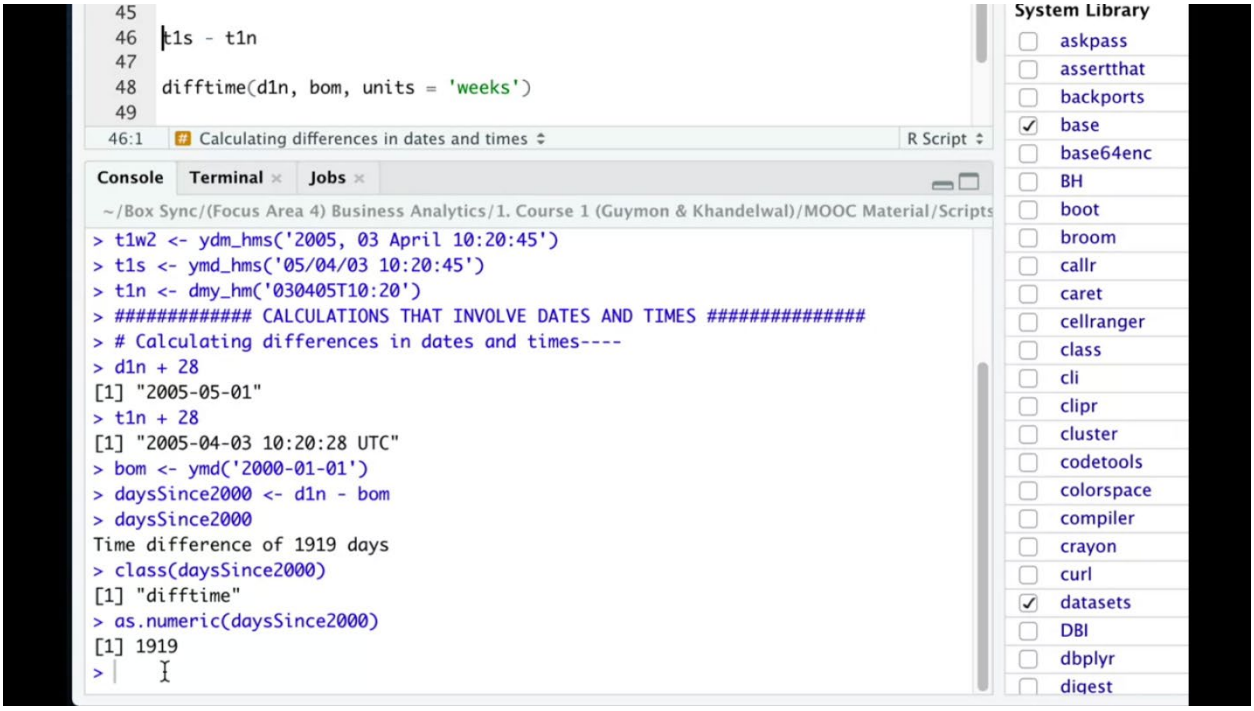
What if I take this t1n object, which is a time object, and I add 28 to it. Down on the console, that tells us some interesting things, and now it's the same day of the year, April 3rd 2005, but it has added 28 seconds to it, and so this is reflecting the difference in the way dates are stored versus times.



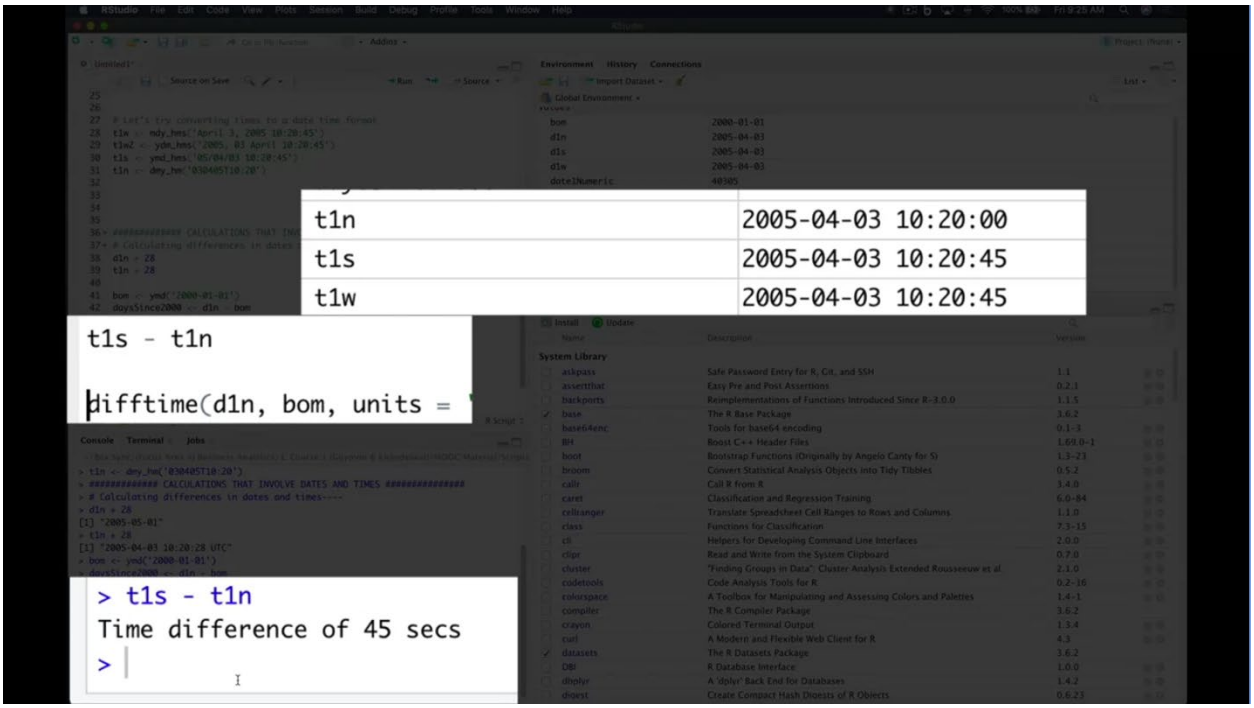
Dates are stored as an integer, which indicates the number of days that have passed since the epic began, whereas time is recorded as the number of seconds that have passed. And so If you had 28 to it, then you're just adding 28 seconds versus days. Now, at times, we also have to worry about time zones, and so, UTC stands for coordinated universal time, and that represents what the time is at zero longitude. So, unless you specify time zones explicitly, it will assume that you're referring to zero longitude. It's beyond the scope of this lesson to talk about dealing with time zones, but that's something that you could investigate further if you're interested in doing so.



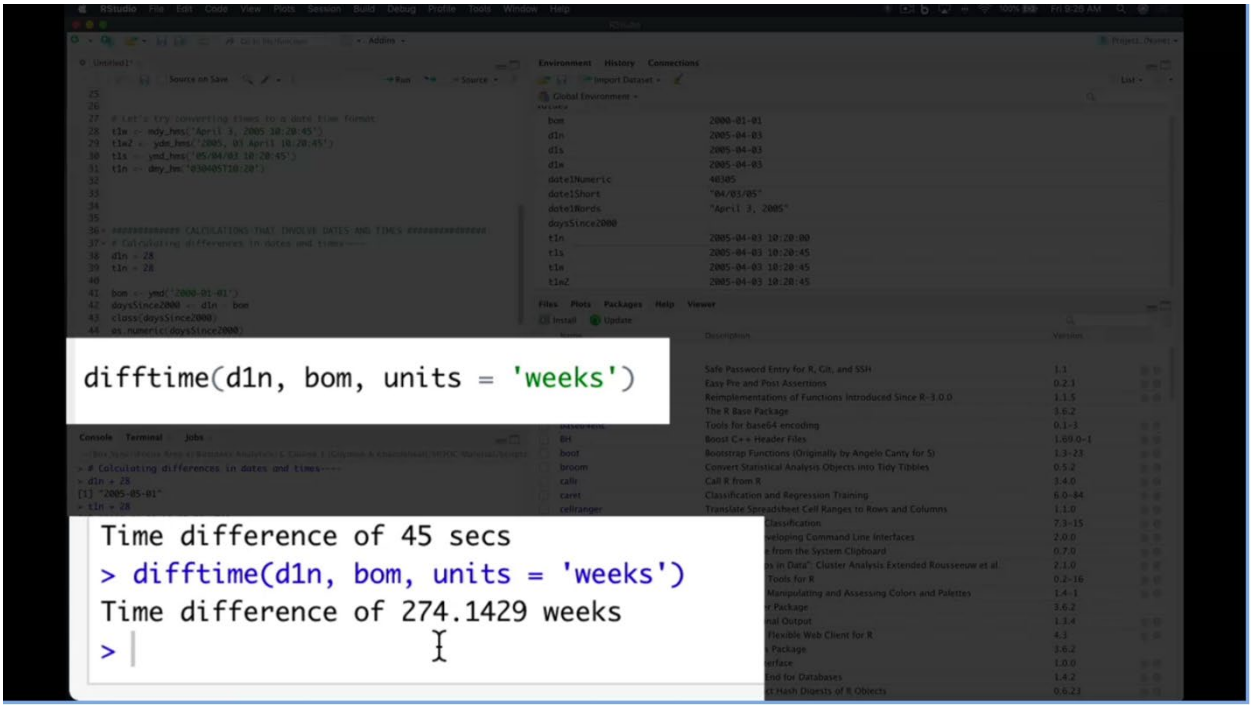
Let's talk about other calculations that you might want to perform that involved dates. Let's create a new date object and will use the ymd function and convert the string here to January 1st, 2000, and we'll call that the beginning of the millennium. Then, let's make a calculation to find out the number of days that have passed since April 3rd, 2005 and January 1st, 2000. If I run that and then in the console look at daysSince 2000, it tells me a time difference of 1919 days, so that's pretty clear. If I look at the class. That is a difftime object.



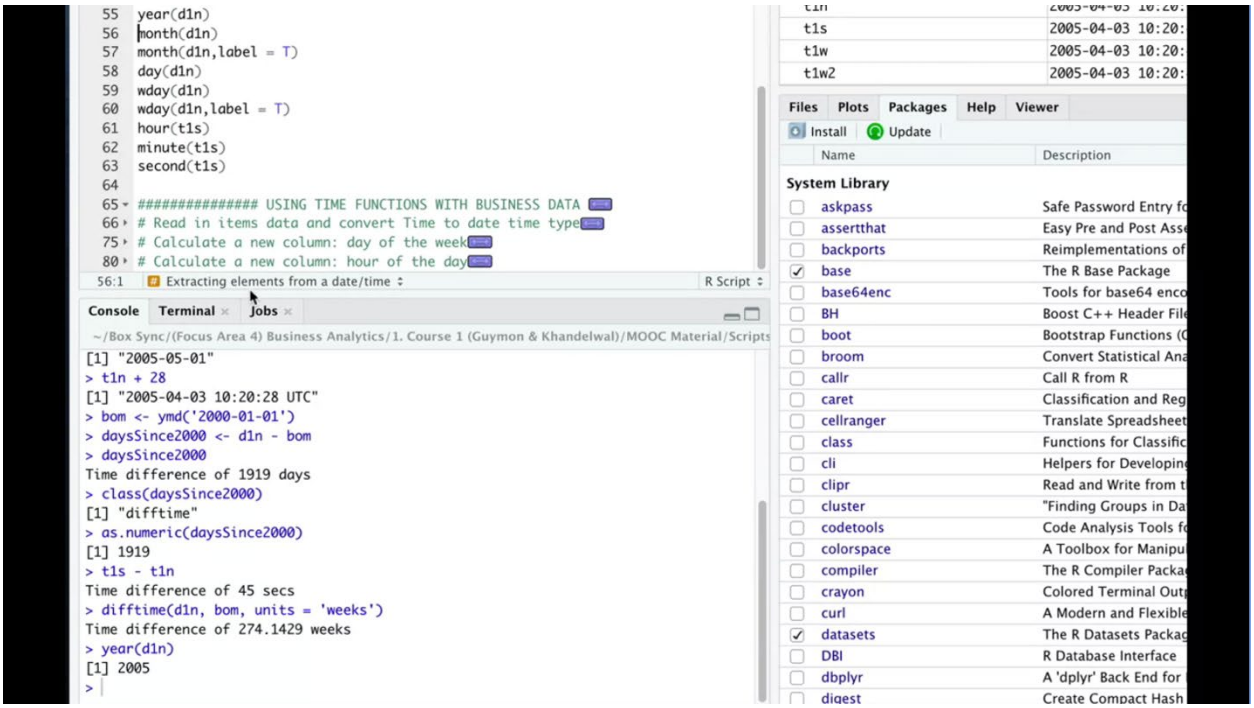
So, it's not a numeric data type, and oftentimes will want to take the number of days that have passed and convert it into years or months or weeks. Or compare that to some other quantity of time. So, we can coerce that date, that difftime class to a numeric data type by using the as dot numeric functions. So if I do that, you can see in the console that it prints out just the integer of 1919.



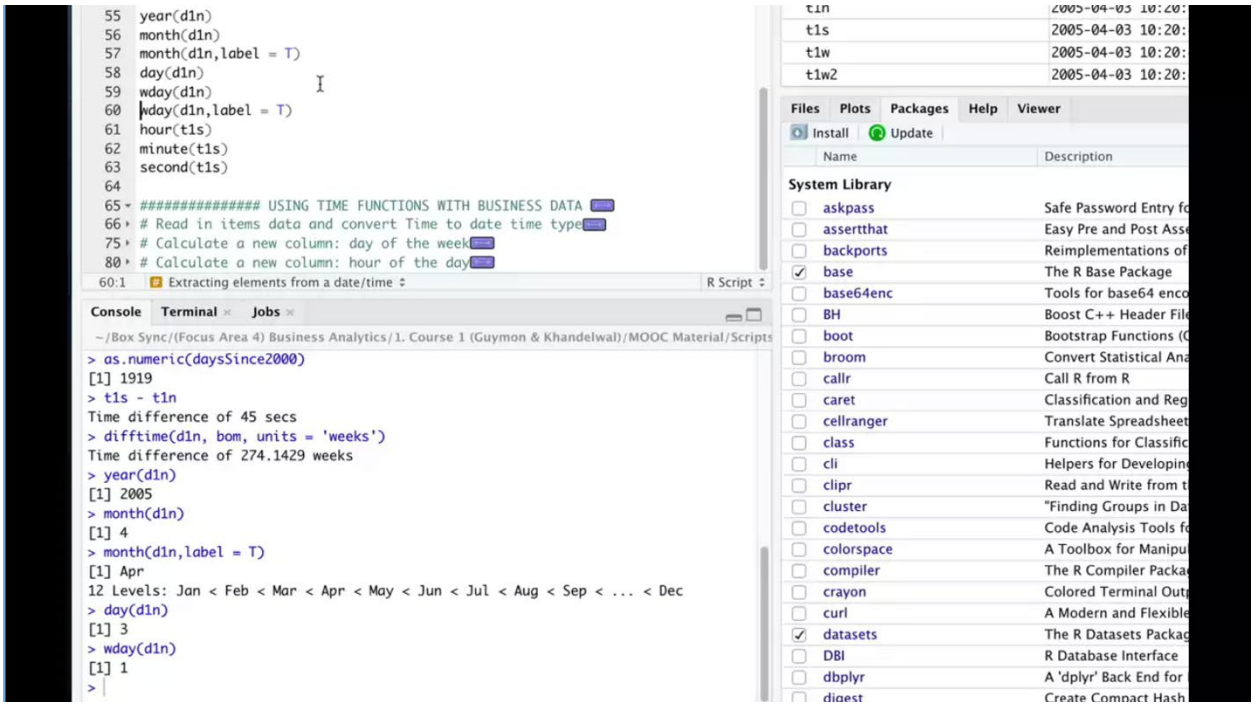
All right, let's go ahead and look at what happens if we take t1s, which is a time stamp of 10:20:45 seconds. And compare that to a time stamp of 10:20:00 seconds, both on April 3rd, 2005. If I run that, it tells me a difference of 45 seconds. So once again, this is reflecting the way that times are stored as the number of seconds that have passed, and so it automatically results to telling me the number of seconds.



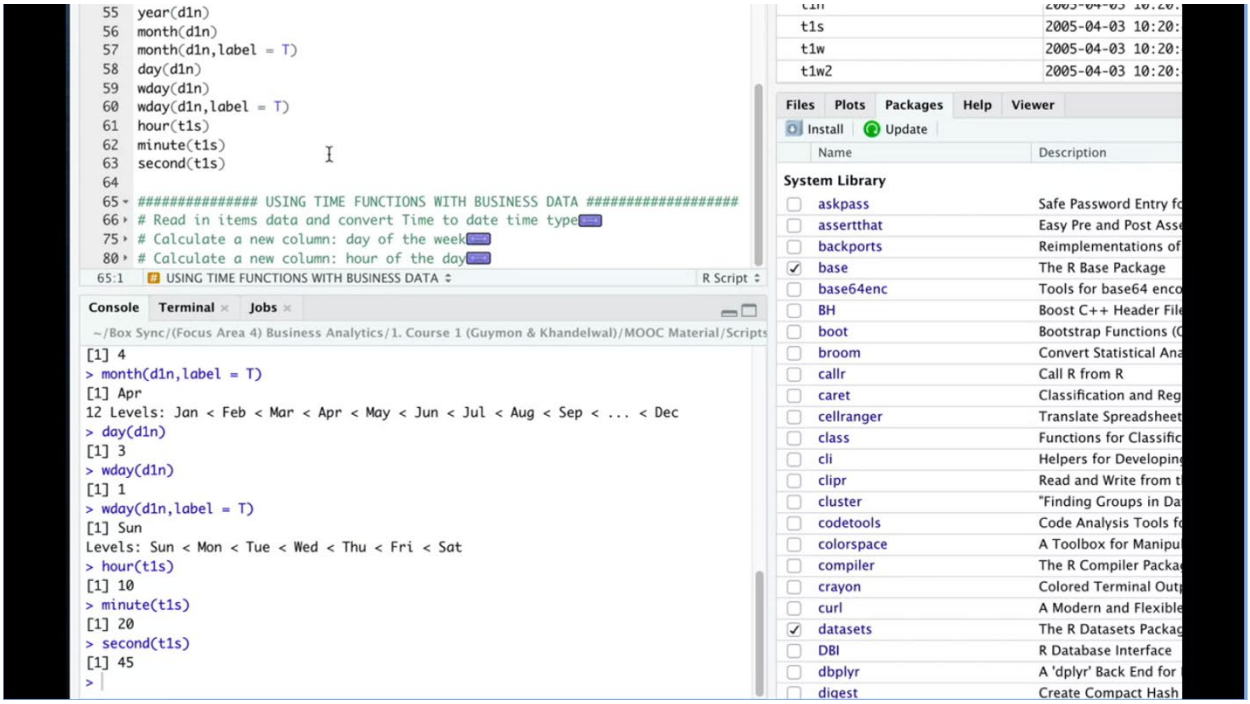
Now, if I want to make a quick conversion, I can use a base function called difftime, and I can enter a beginning time and an end time, and it'll take this first time or date and subtract the second date from it. So in this case, I've got April 3rd, 2005 and January 1st, 2000, and we're going to calculate the difference in weeks. So, I'll go ahead and run that, and it tells me the difference is 274.1429 weeks. So, there are some calculations that involve dates and times that are often performed.



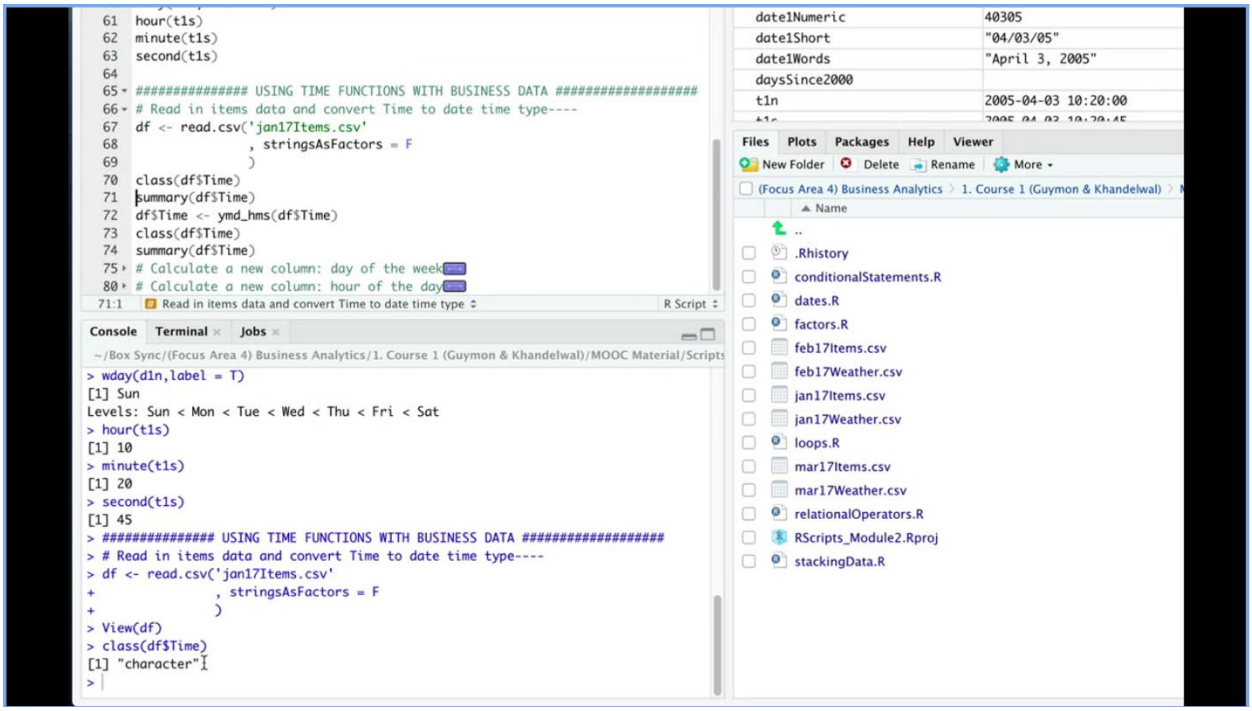
Now, another thing that is often done with dates and times is we'd like to extract certain elements from it. For instance, we've got this date of April 3, 2005 and perhaps we want to extract the year or the month through the day. Or if we have a time stamp, we may want to extract just the hour or the minute or the seconds. So, the lubricate package makes that very simple. We can simply call the year function and then enter a date object in there, and it returns the year 2005 or if I use the month function, it returns 4 for April.



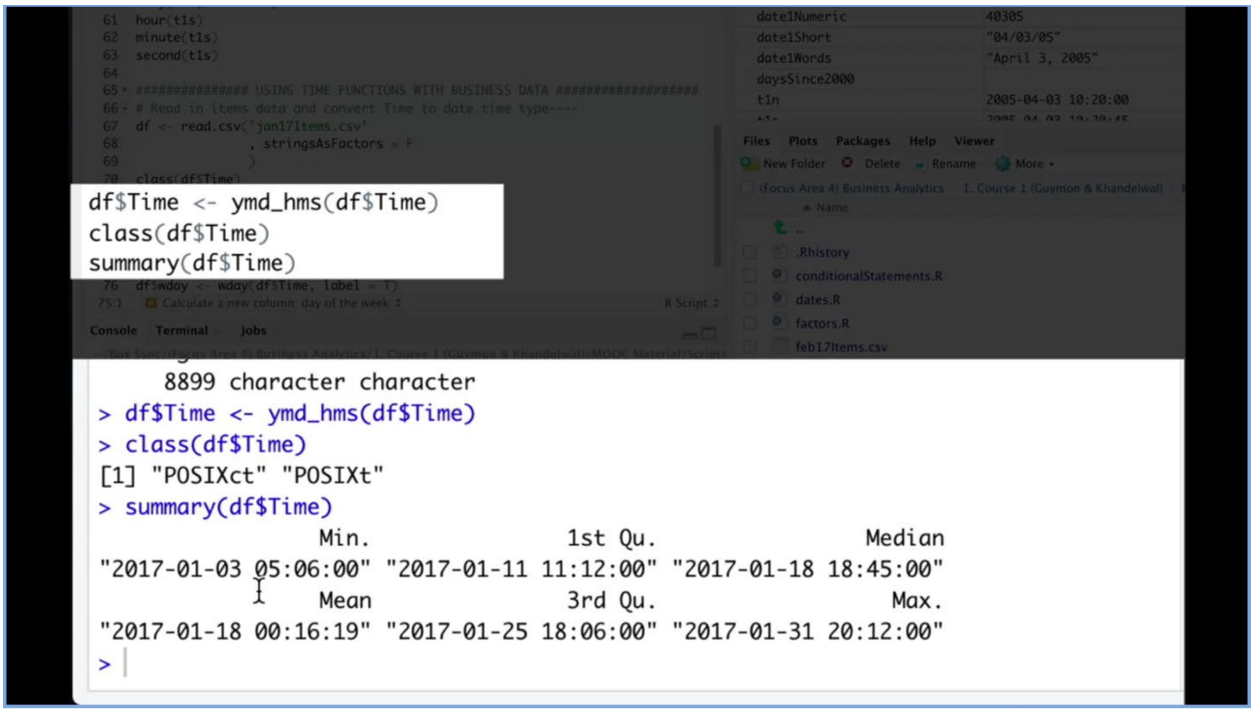
If I want to give it a label of April, I can use this label argument run that, and it returns an abbreviated month name there. If you look into that function more, you can tell it to return the full name if you want. And then it also tells us that this is a special data type now, which we'll talk about later. But it's an ordered factor data type. All right, I can extract the day by using the day function that tells me the day of the month. Three.



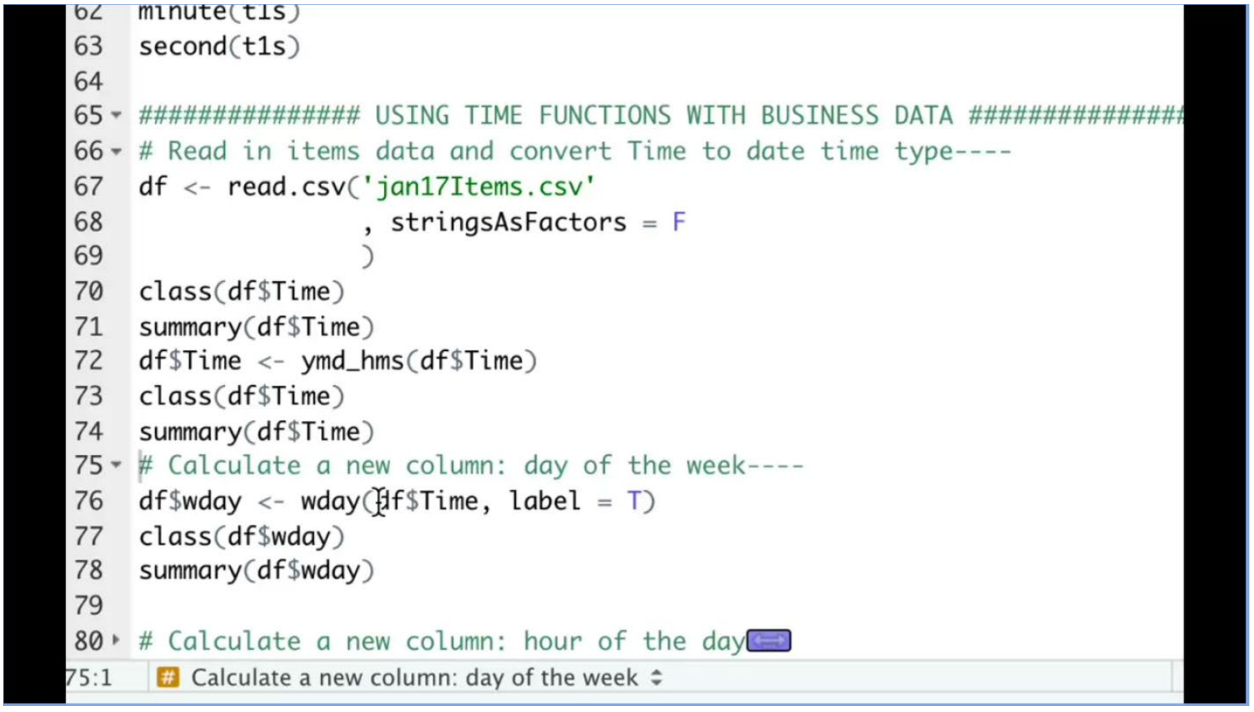
If I want the day of the week, I can use the wday function, and that's one. And if I'm not sure if that's a Sunday or on Monday, I can add a label in there, and that tells me that it's a Sunday, and again there are more options here. You can go in and set a parameter that sets Monday as the first day of the week if you want, but the default is to use Sunday as the first day of the week. All right, what if I've got a time stamp and I want to extract the hour. Well there's an hour function from want to extract the minute, you can use the minute function, and second, to extract the seconds. So, lubricate makes it very intuitive for dealing with dates and extracting components of a time or date that would be useful.



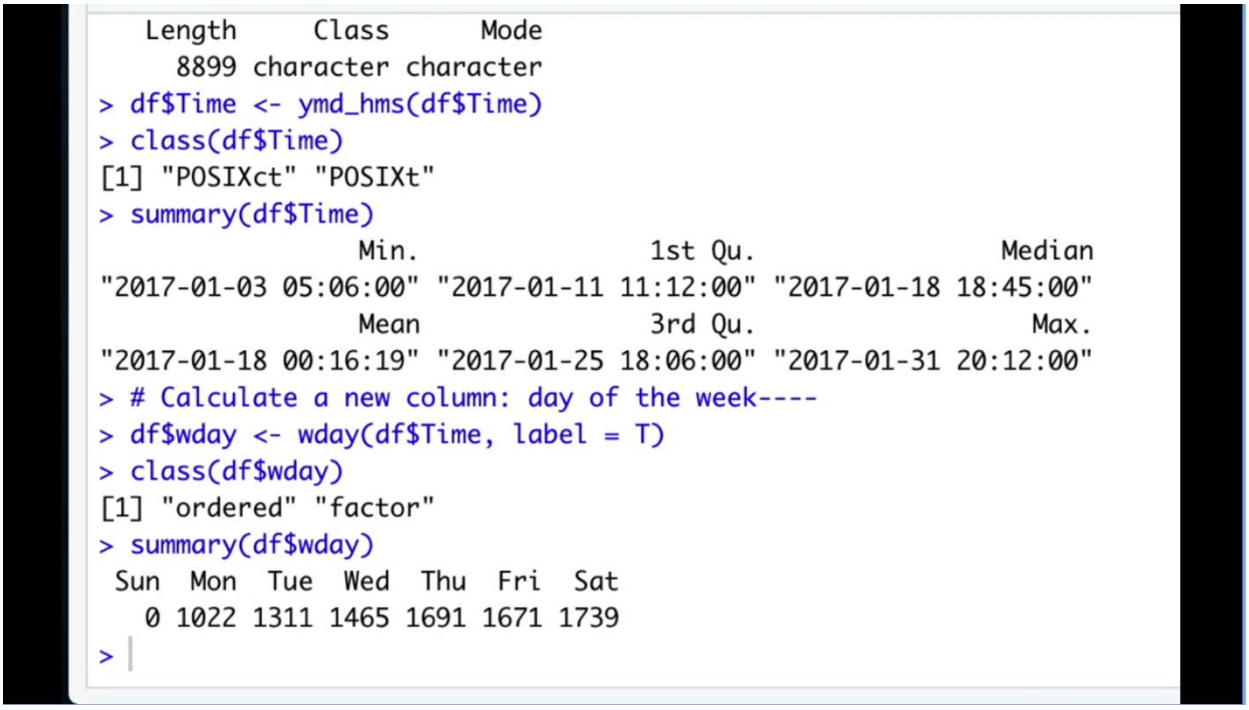
Let's take what we've learned about dates and illustrate how to apply it to business data. So, let's first read in the data. We're going to use the jan17Items data. We'll read that in as we normally do, and let's go ahead and investigate this data set. If I open it, I can see the first column is time, and it's obviously a time stamp, right? It's got a date and then a time stamp in there. But if I look at the type, it is currently a character type, and I like the way it's displayed. But as a character type, I wouldn't be able to make any calculations or extract the day of the week or the year and so forth. Also, if I try to perform a summary calculation to get some additional insight, it doesn't really tell me much detail, just tells me how many observations, there are basically 8899 and then tells me it's all character.



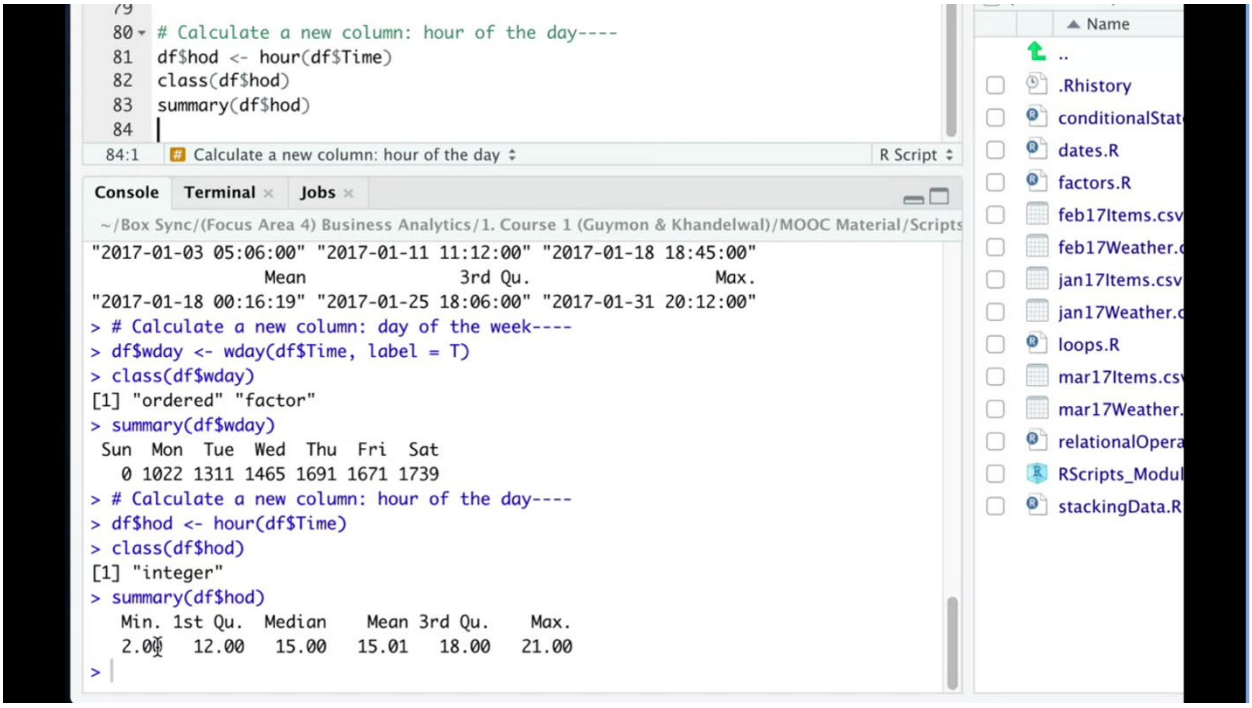
So, let's go ahead and convert that time column to time type. And so, I'll use the ymd_ hms function from the lubricate package and refer to that column, and I'll sign it back to itself. So, if I run that and now look at the class of that column or the type, it's a POSIXct which basically means a time type. And now, if I look at the summary, it tells me the minimum time, which in this case is January 3, 2017 at 5:06am. And I can see different parts of the distribution. But the max time is January 31st, 2017 at 10:12 pm. So, it uses that 24 hour time convention.



All right, so again as a date, time object. In this case, we can do a lot more interesting calculations. All right, well, now, let's say maybe we want to investigate business fluctuations and trends based on day of the week. Let's calculate a new column day of the week and we can do it based on that Time column. So, I'm going to use the wday function from the lubricate package, and I'll tell it to perform this function on every observation in the Time column of the df dataset.



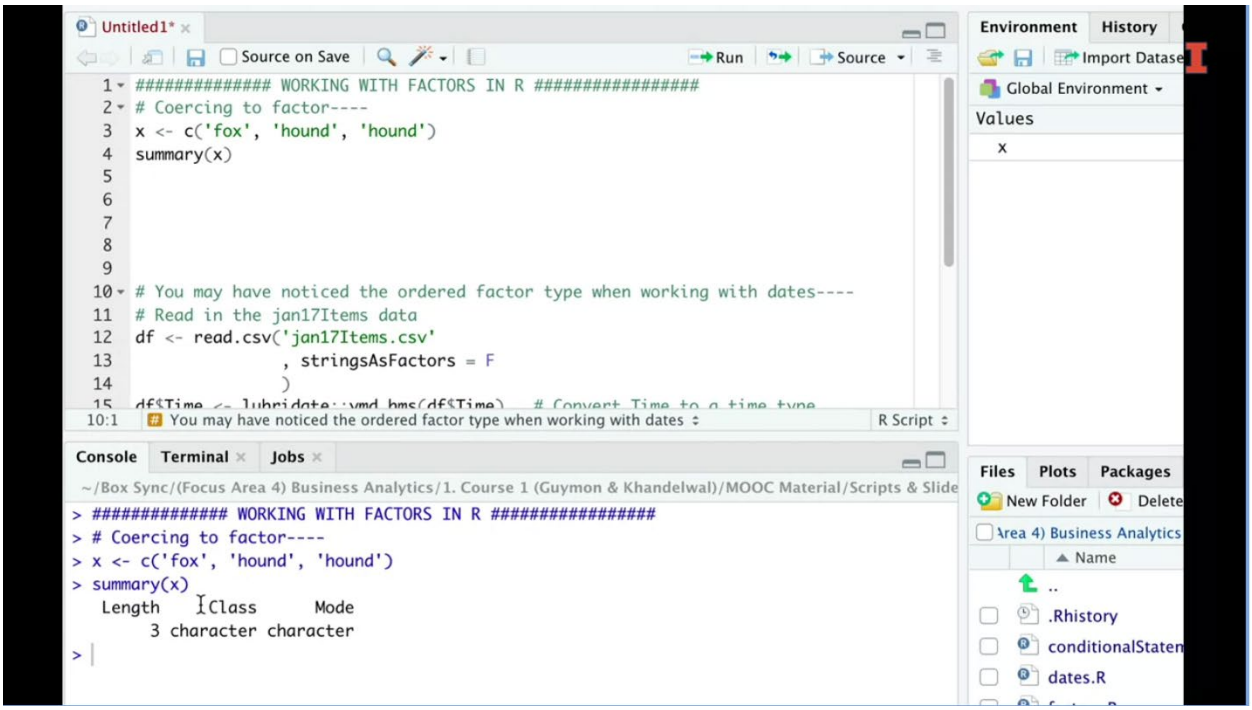
Let's go ahead and add the label to it, so I'll run that. And now let's look at the type so we use the class function tells me it's an ordered factor type. Now, let's go ahead and look at a summary of this column so I'll run that. And wonderful, this is now telling me how many observations there are in this data set for each of these days of the week and notice that it orders those in a meaningful order. It's not alphabetic order, but it's as the days go, which is really helpful. All right, so again, that's a benefit of having a date time format in our Time column and using the lubricate package. It makes it really easy to extract this information.



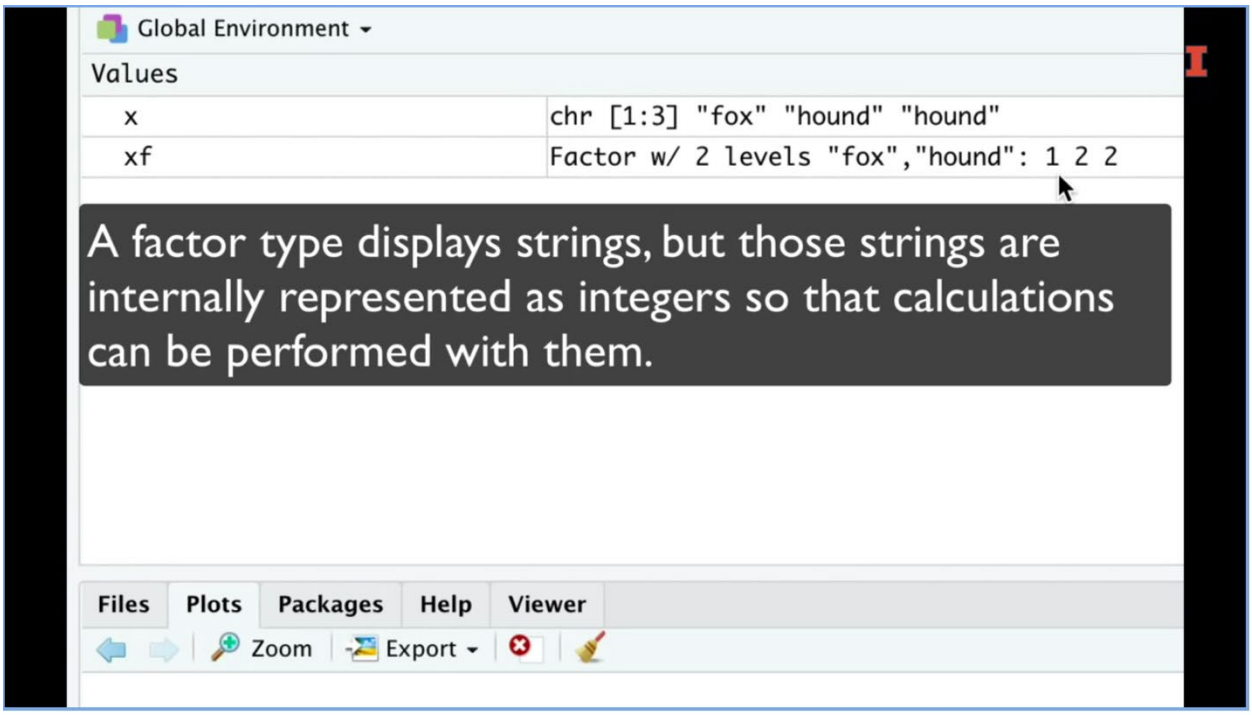
How about let's calculate a new column? Maybe we're interested in looking at sales trends based on hour of the day. So, I can create a new column. I'll call it hod for hour of the day, and I'll use the hour function in the lubricate and I apply it to the Time column. And let's look at the data type here. This is an integer, and if I look at a summary of the hour of the day, it tells me the minimum hour is two. So, 2 a.m. The maximum hour is 21 or 11 p.m. So with that, you should be able to get started performing a lot of date time calculations using business data.

Lesson 3-8 Factors

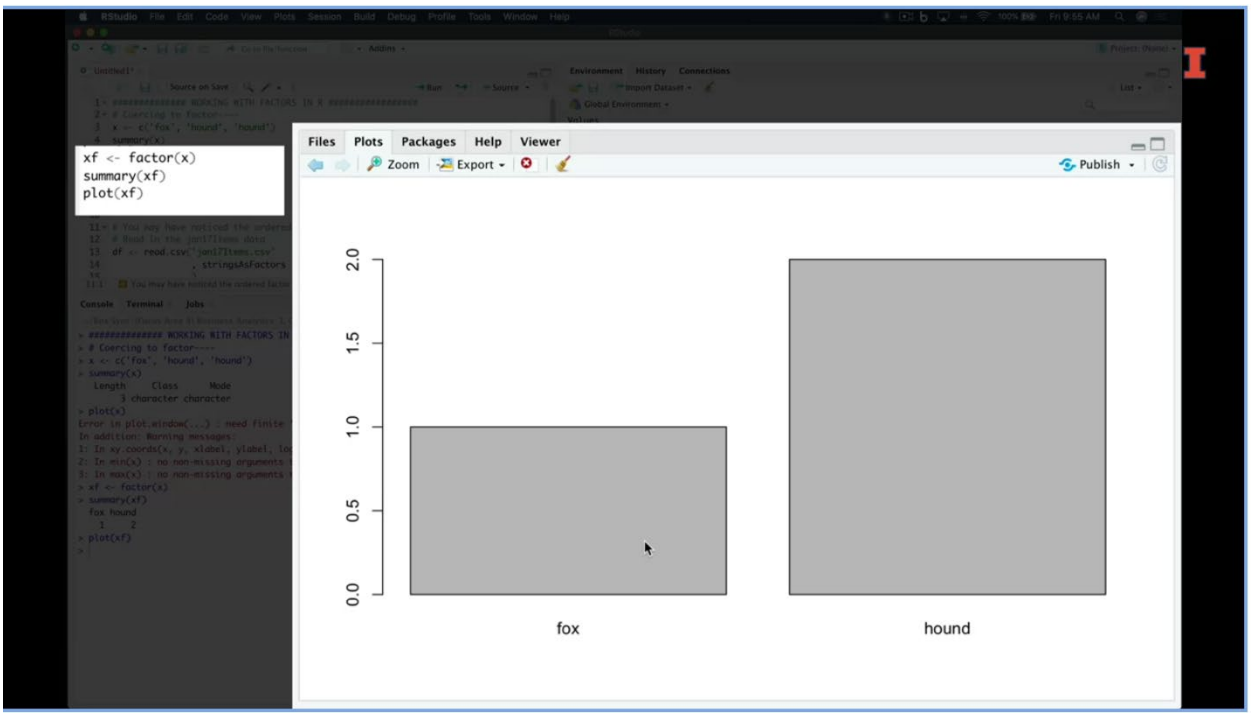
Lesson 3-8.1 Factors



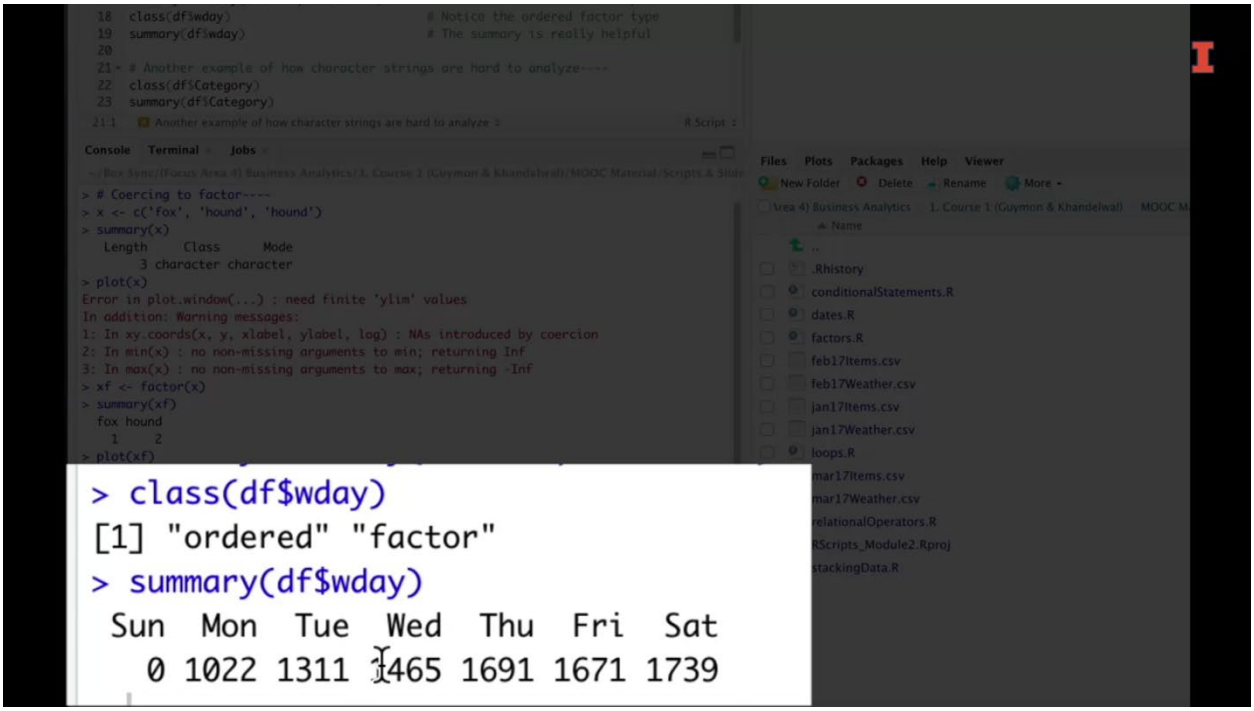
In this lesson, you're going to learn about the factor data type now before we talk about the factor data type using business data. Let's just look at it in terms of a simple vector. So let's say I have this vector here X, which is a character vector that has the word fox and hound in it, and Hound is in there twice. Now, if this were a much longer vector, I might be interested in knowing how many times each word appears. And I may also be interested in knowing what the different words are in this character strength. So it would be nice if I could use the summary function and if it would tell me that information.



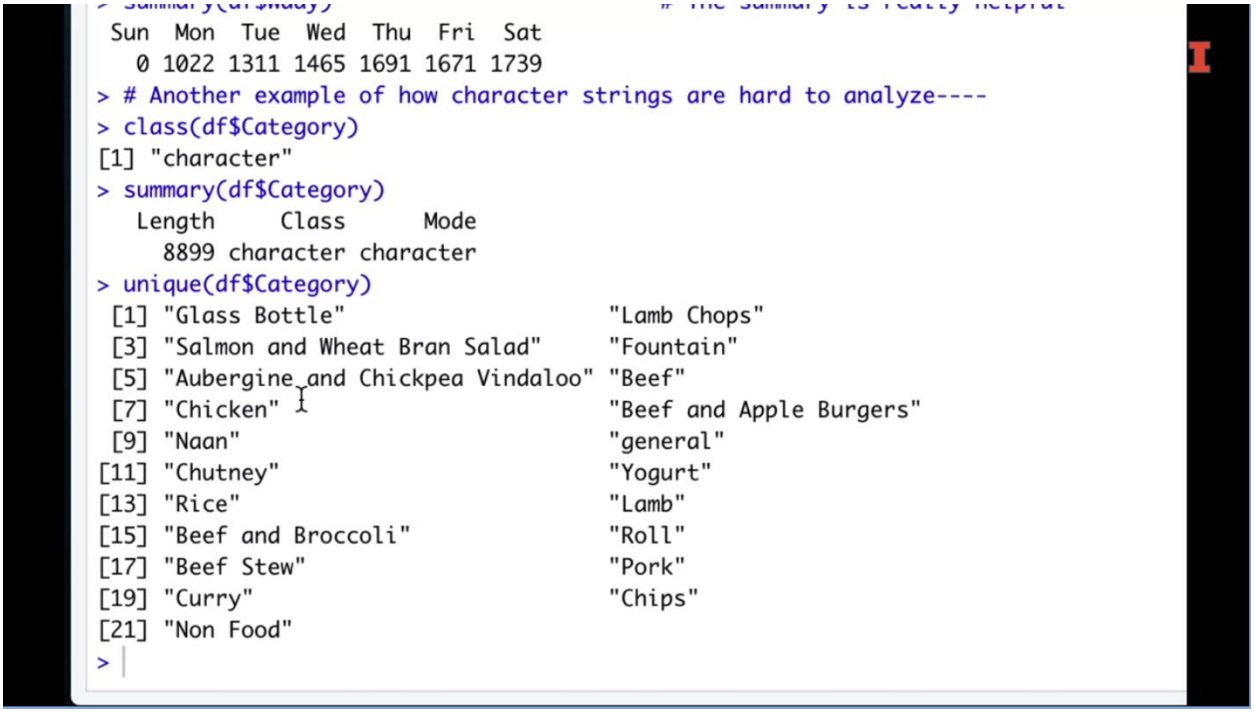
But for a character string, it only tells me how many observations there are. Also, if I were to try and plot this vector, it couldn't do it because it is a character. String doesn't know what to plot. So let's go ahead and convert this to a factor, and we can coerce it to a factor data type by using the factor function. And then I'll put X in there, and now you can see I've got this XF object, and it tells me it is a factor. And it has two levels, fox and Hound. And then it also gives me integers indicating which level each of these words means. So that's what a factor is.



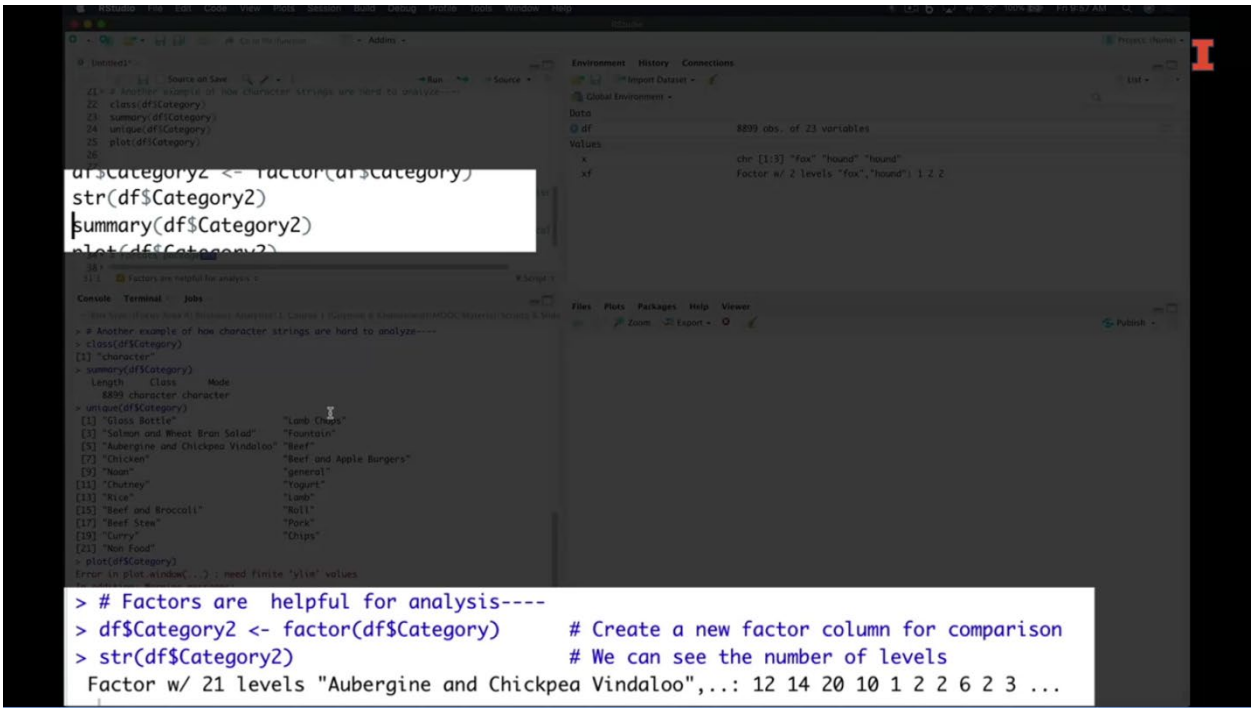
It's a combination of a character string for display purposes so that we can interpret it as humans as well as a numeric integer value, so that we can do some analysis with it. So now if I look at a summary of this ex FDA object, it tells me that foxes in their wants and hound is in there twice, and I can even plot it. And it gives me a history Graham that indicates the frequency with which each word or each level of the factor appears. All right, so now let's talk about factors in the context of business data. So let's go ahead and read.



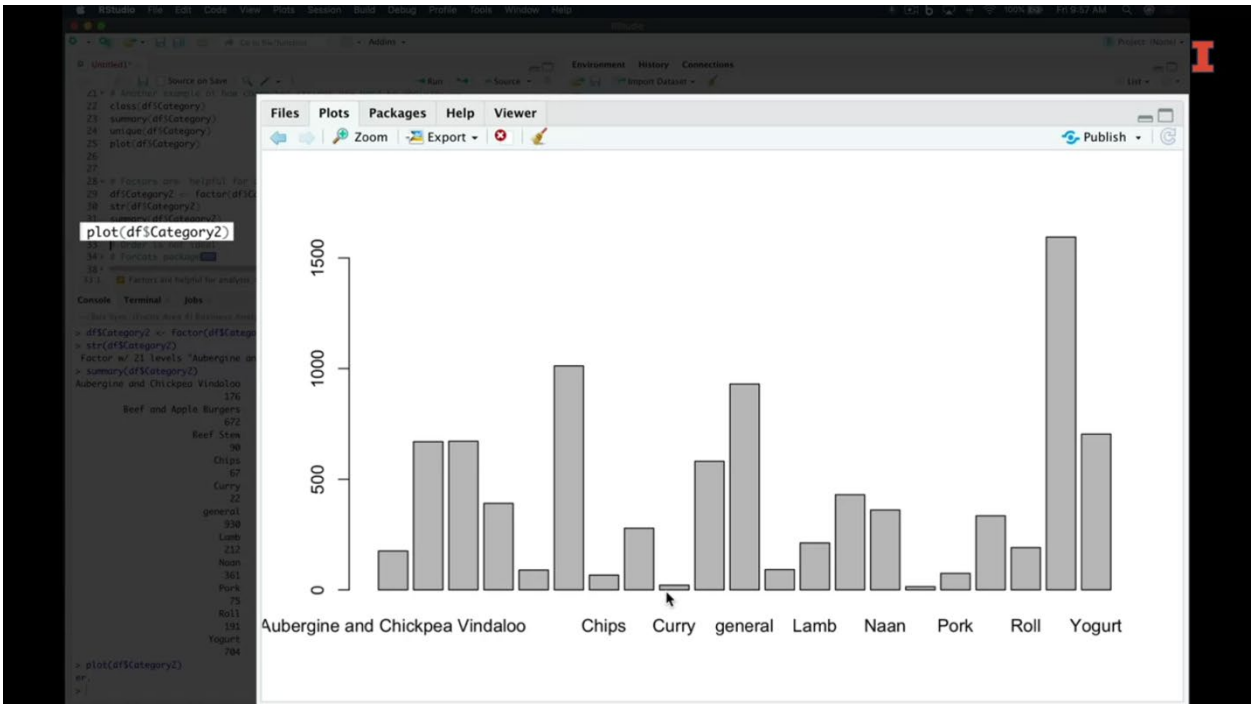
In this January 17 items C S v file, and you may have already noticed in the lesson on dates that if we convert time the time column to a date and then create a new column W day for weekday, it will be an ordered factor. All right, so look at the class. It tells me it's an ordered factor, and if I do a summary of that, it tells me how many times each day of the week appears and it puts it in the right order for me. Now let's turn again to this January 17 items data set and let's look at the category column.



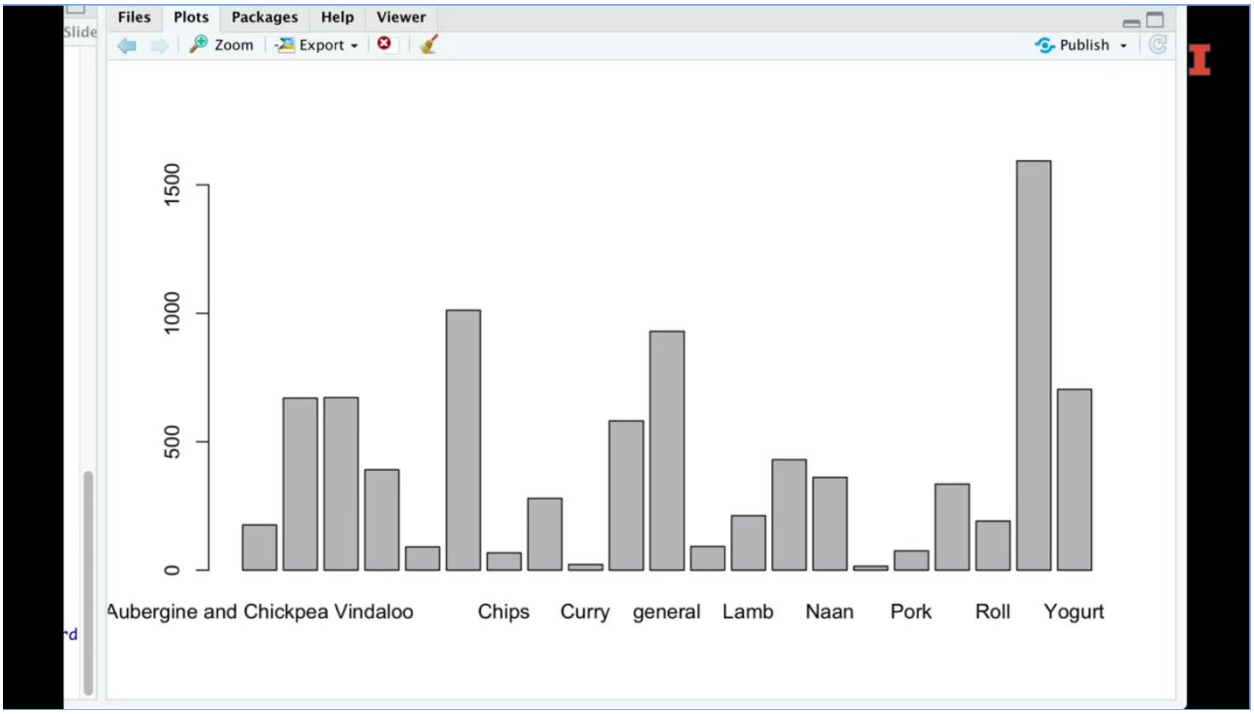
If we look at the class or the type of this column, it is a character string. And so if I try to look at the different categories in my data, it's not going to tell me anything. I can use this unique function, and it prints out a list of the different unique categories that show up in this data set. And so that's helpful, and I can see that there are 21 there, but if I try to plot it, nothing will happen.



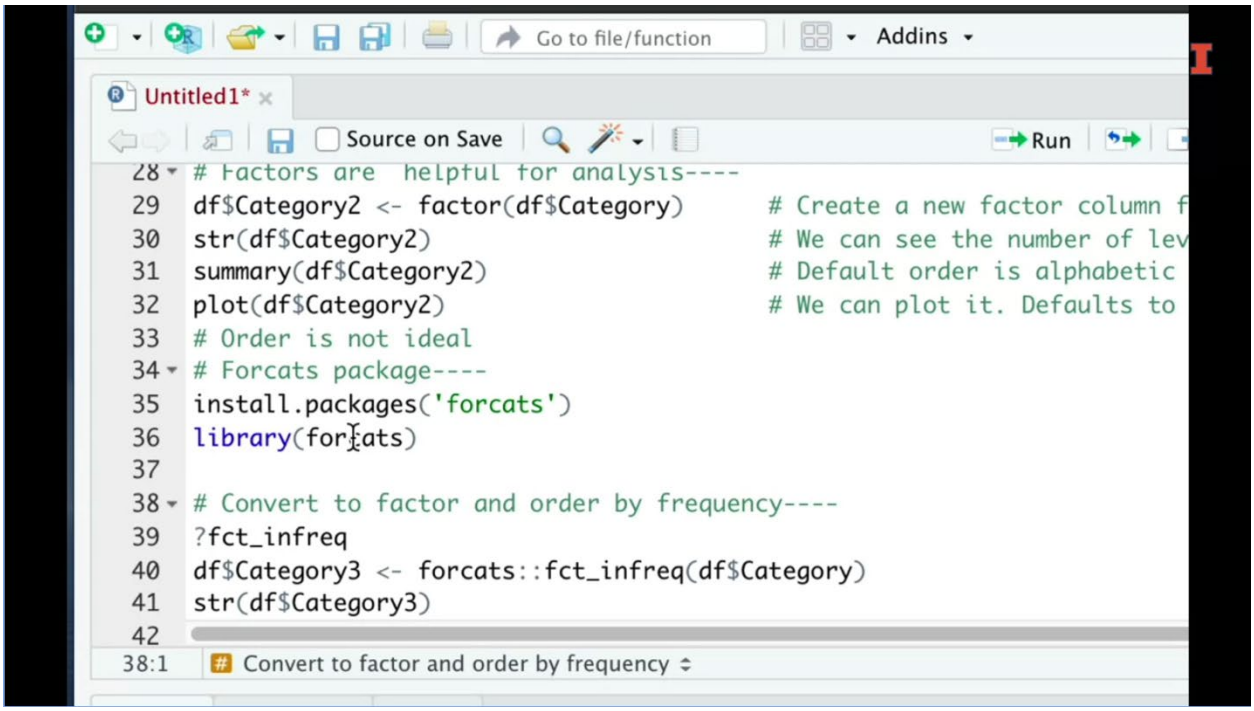
So let's go ahead and create a new column category two, and it will be a factored version of the original category column. Alright, so we're will create that column. And now if I look at the structure of that column, it tells me that it's a factor with 21 levels, and it tells me one of those levels and gives me some of the integers associated with those factor levels.



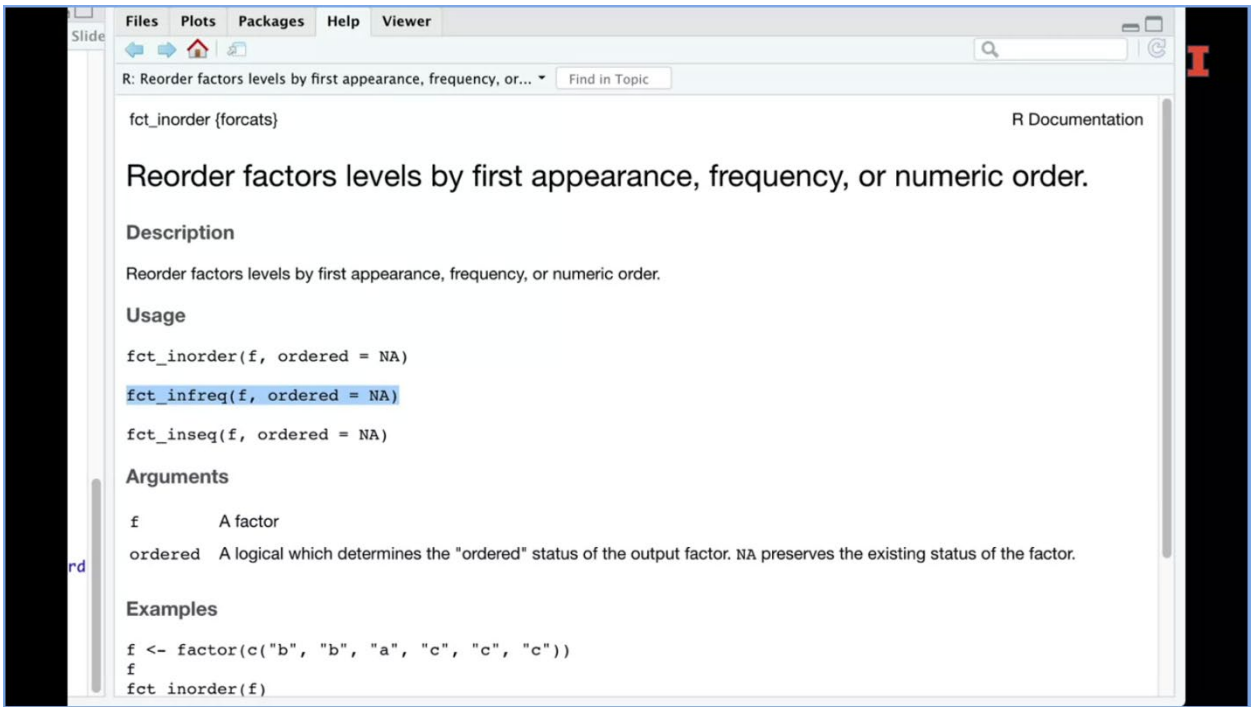
Now, if I look at the summary, it prints out a list of each level, and the order is alphabetical, so you can see it starts with a and ends with y. Now I can plot it, and once again, this is helpful. It's easy to see the most frequently occurring level of that factor, as well as what would be occurring least often. So while this is helpful, it's not ideal because the order could be improved. Typically, we want to order things in a meaningful way, ascending or descending.



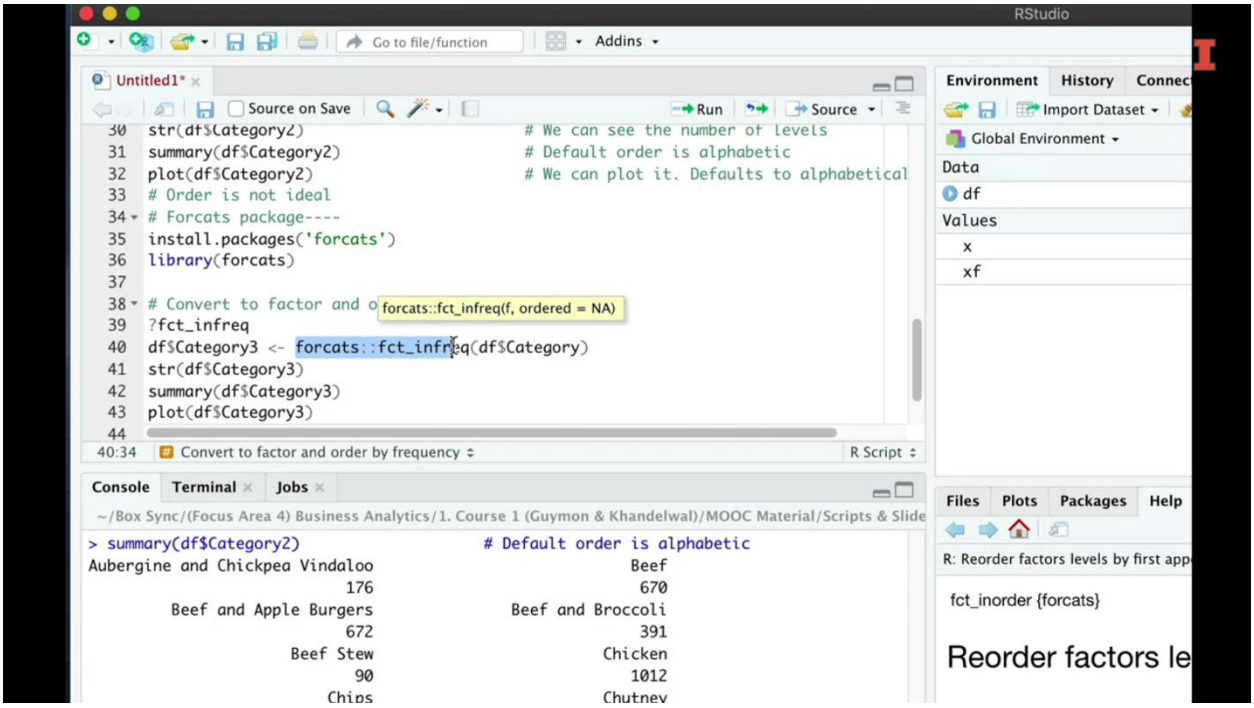
So this is where the Four Cats package is helpful now. There are some base functions that you can use for dealing with actors and ordering them correctly, but it's a lot more verbose. And so this four Cats package has some functions that make it really easy to do some common tasks with factors. And this will be really helpful when you get more into visualization, because the order of the factors will influence the orders of the bars and the history. Graham, for instance.



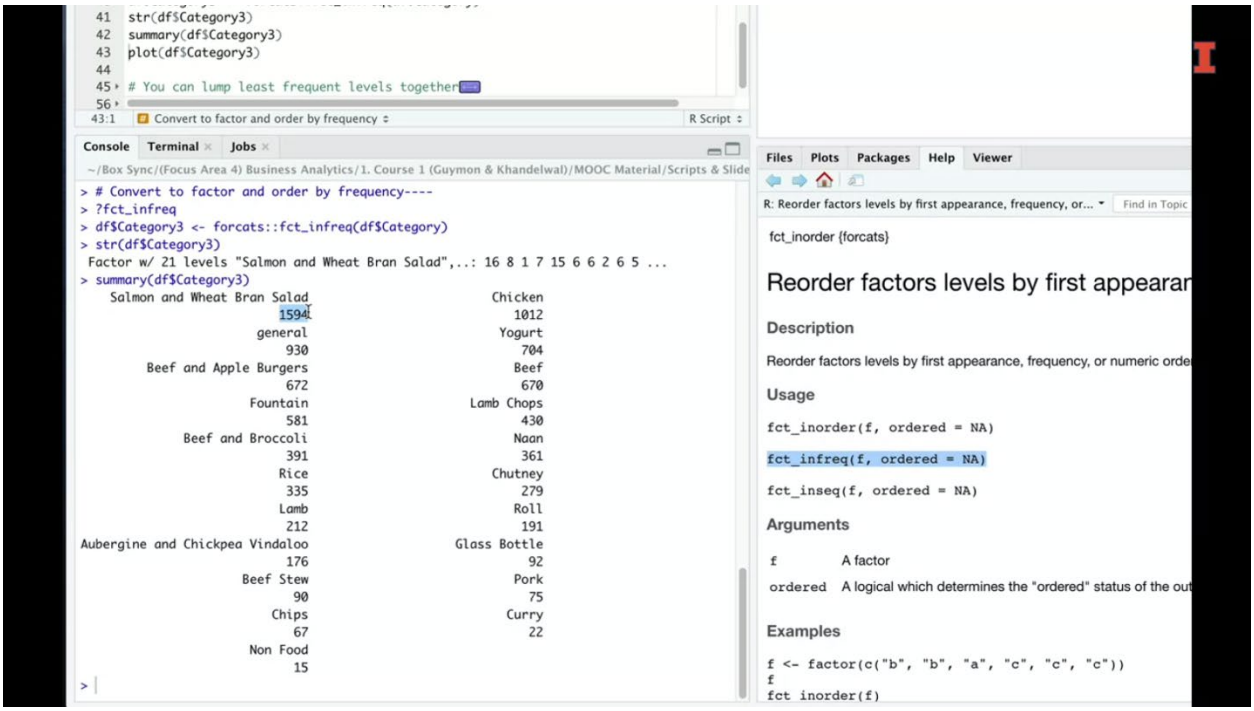
All right, so if you haven't installed the four Cats package. Go ahead and install that. You can do that using the installed packages or by going to the package pain and searching for four cats and installing that once you've installed it. Go ahead and load it by calling the library function and entering four cats in there and now you'll have access to all of the functions in that package. Let's start by talking about how to convert a character string vector into a factor vector and ordering it by frequency.



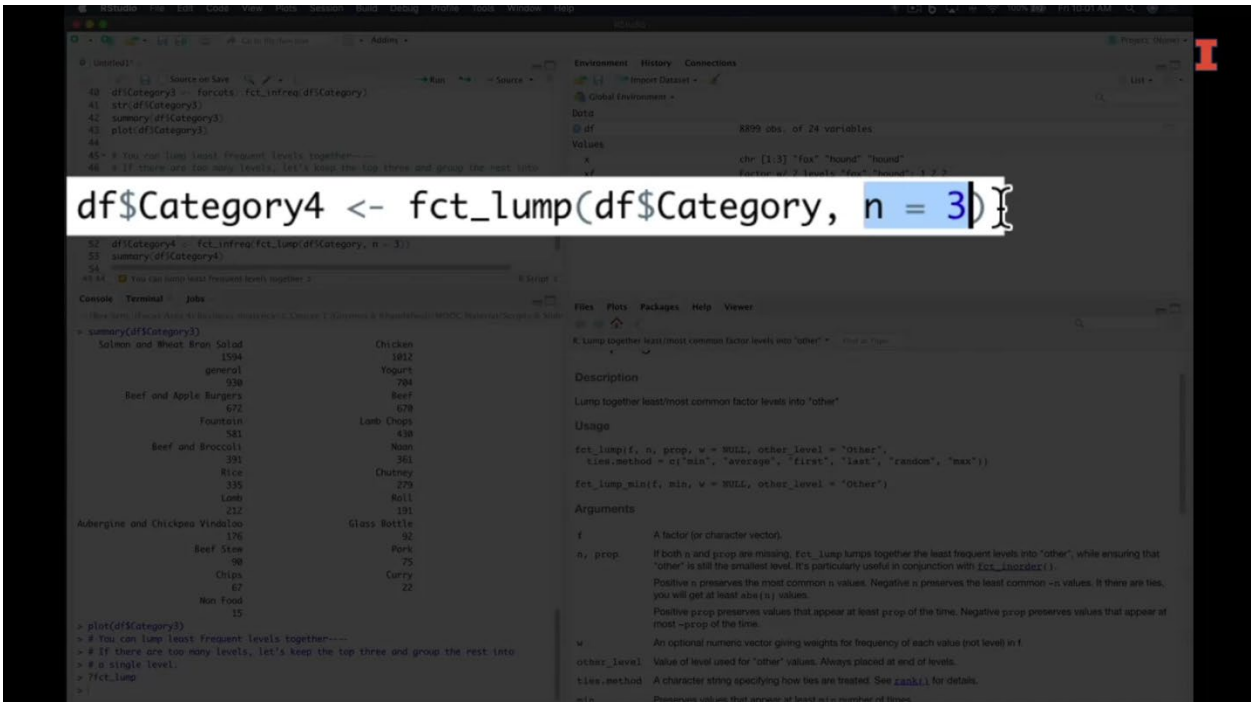
So the function that you should read about if you want to read in more detail is the f c t underscore i n f R E Q. And you can see that this is to reorder factor levels by first appearance, frequency or numeric order. So there are a variety of ways in which you could order the factors. We're going to use this in freak version of it, and so will simply create a new column Category three for comparison so we can compare it to Category two into the original category, and we'll call this function.



And just as a reminder if this library isn't loaded or if there happens to be a name space issue, meaning there's another package that has this function name in it. Then this is telling it that we want to use the function from the Four Cats package. So we're simply going to call that function and then enter the column name that we're converting to a factor, which is just that category column. So I'll do that.

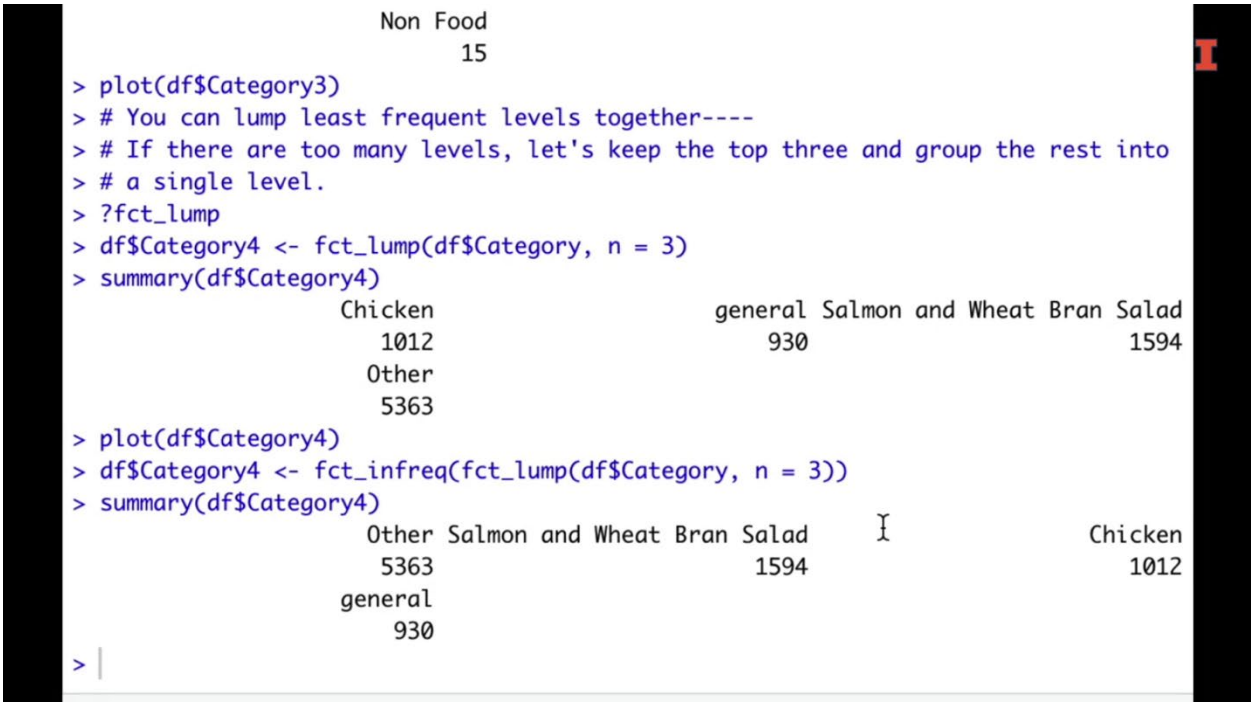


And now let's look at the structure of this category, and it tells me again it's a factor with 21 levels. If I summarize it now, I can see that it starts with salmon and ends with non food. And we've got the the other categories in here, like Aberdeen and Chickpea Vindaloo, which is in the middle somewhere. So you can see what it's doing is it's ordering it in descending order based on the frequency of appearance. And so that's really helpful, especially if we want to plot it quick, create a quick hissed a gram of it, and we can see that Sam and Wheat bran salad is the highest, and then something over here at the bottom. So this this helps us see the relationship very quickly.

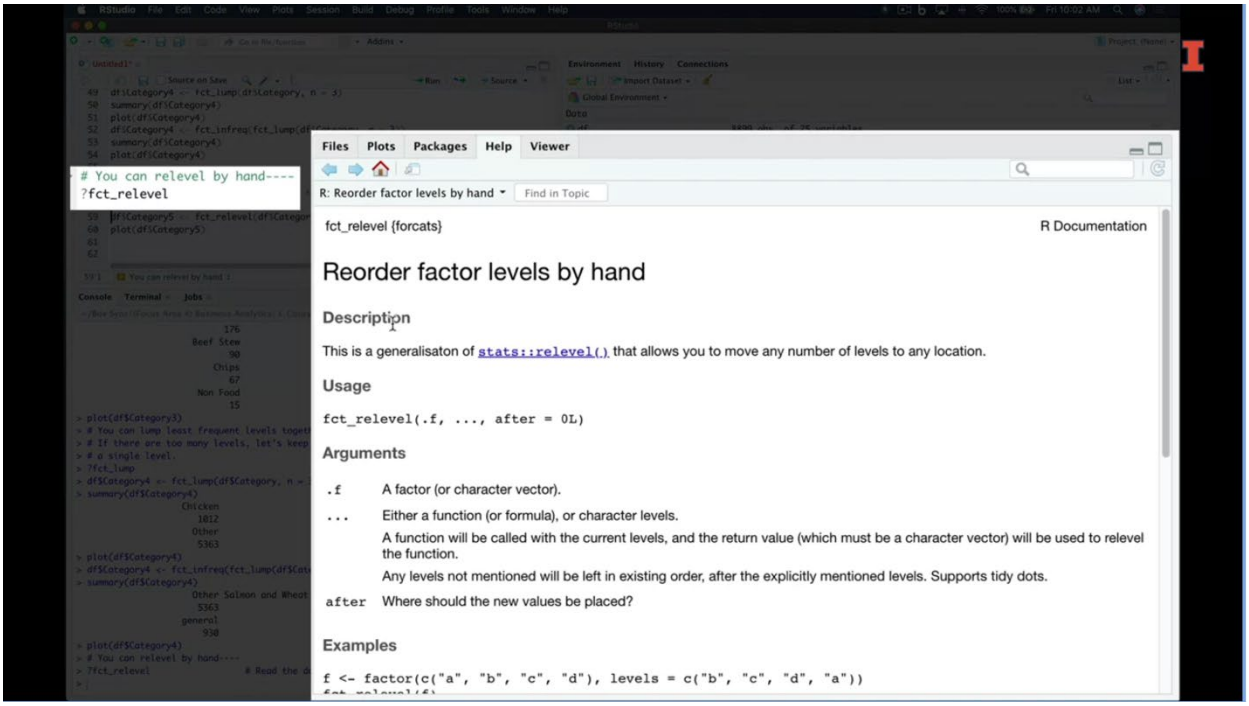


Now, another thing that you may want to do is recognize that Hey, there's a lot of different categories. I'm really not interested in all these other ones. I'm just interested in maybe the three most frequently occurring categories so we can use this other function in the four Cats package fact lump. And if you read the documentation on that, you can read all the details about this. But, what I'm going to do is I'm going to create another category column, Category four, and I'll use this fact lump function. And I'll indicate the category or sorry, the column that I'm using to create this new column from this lump factored column, and I'm going to indicate the number of unique levels in it. So I'm basically saying, I want to just see the top three levels, then lump everything else together in its own category, so I'll go ahead and run that.

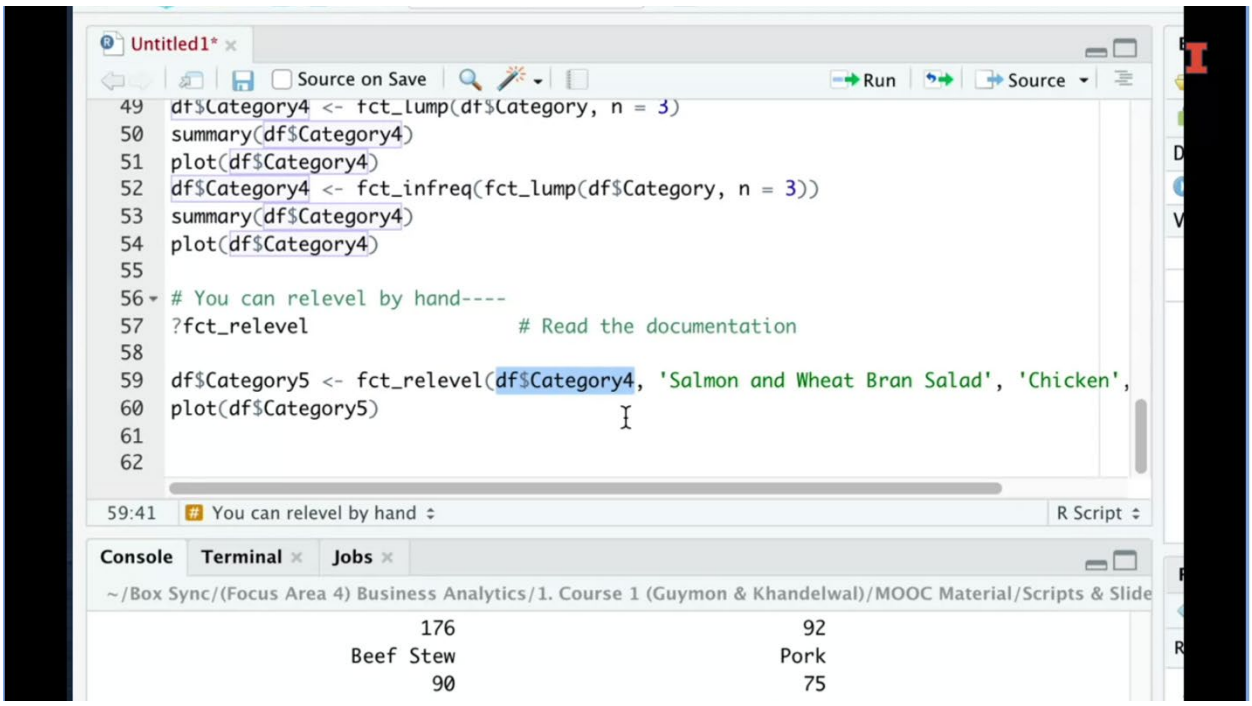
Then I'll create the summary, and you can see that I've got chicken general salmon, wheat brand salad and then the other category. And if I plot it mhm, there's the chicken general sound, wheat bread, salad and other. And so that's nice, because it narrows it down to just my top three categories, but the order is not real helpful, so let's try ordering it in descending order.



I'm going to run this fact lump again, but I'm going to wrap it in the fact and freak function. I'll run that and then look at the summary. And so now it's starting with other at the top, then salmon and white brand salad, then chicken and in general. And you can see that in the if we plot it in the history Graham that it's in descending order, which is often a great order.



All right, so that's helpful. But this may not be exactly what I want, because other is kind of everything else. That's a miscellaneous category I'm not real interested in. I may actually want to start with the highest specific category salmon, wheat bran salad, then go down and then put other at the far right because it's not as important. So we can use this factory level function and you can read more about the details of that. But simply what I'm going to do is create a new column, Category five. Then I'll run this function.

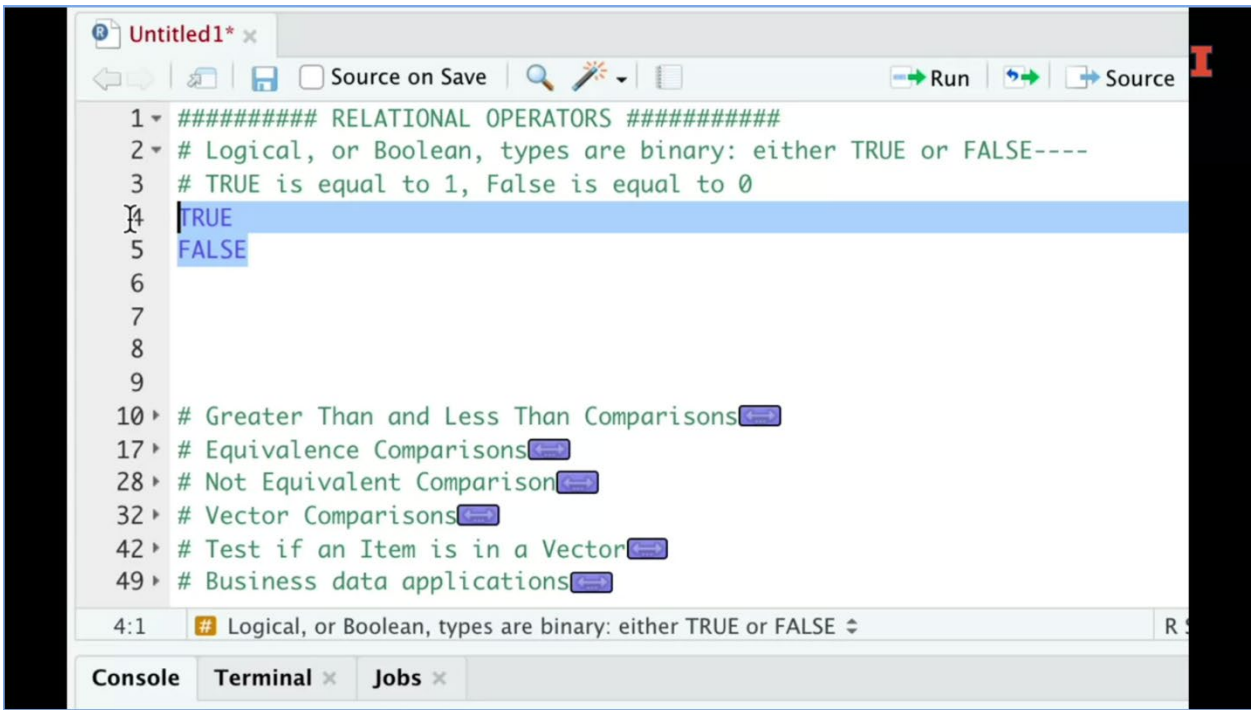


I'll base it off of Category four so a column that I've already created this lumped column and now I'm going to tell it the order. I'll manually indicate the order so you can see that I'm going to say, I want to start with salmon, wheat bran salad, then chicken and then general, I won't even put in other, and it will know what to do with it. All right, so I ran. That created Category five. Now let's plot it, and you can see that now I've got this meaningful, at least in perhaps for my context, it's a meaningful ordering of the factor levels.

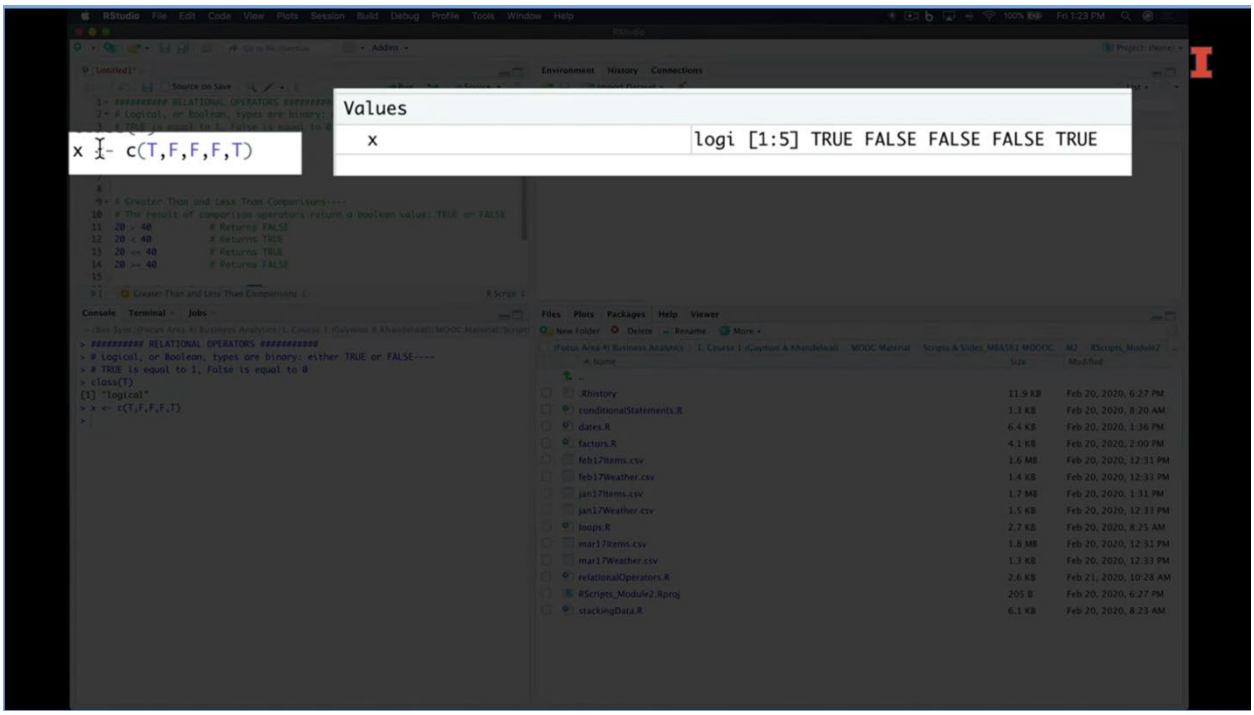
It starts with the highest specific in the second highest third highest, and everything else is together. All right, so there's a quick overview of factors and how you can use the four Cats package to help you deal with factors

Lesson 3-9 Logical Type and Relational Operators

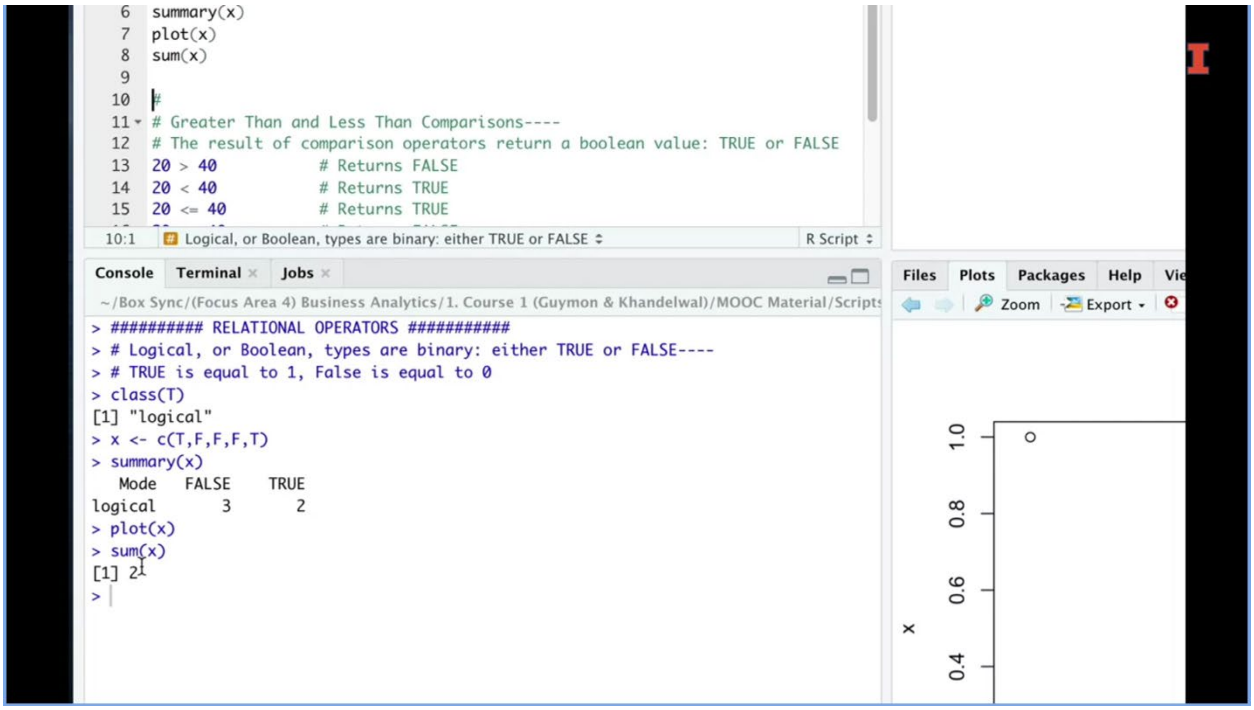
Lesson 3-9.1 Logical Type and Relational Operators



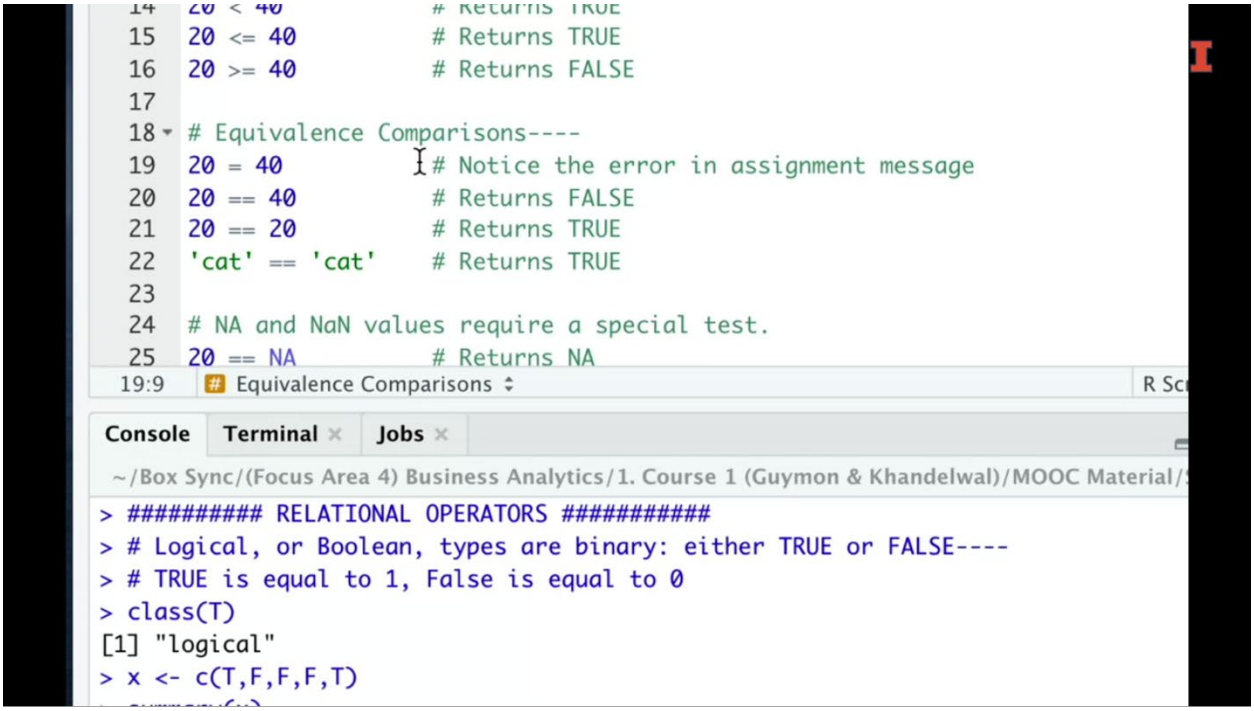
In this lesson we're going to talk about logical data types or Boolean data types. And then we'll talk about how those are used for returning the results of relational operators such as equivalent, or greater than, or less than. So let's start by identifying what is a logical or a Boolean data type. It's basically a data type that can have only two values, true or false. So if you type out TRUE or FALSE, in all caps, you'll notice that they turn blue in all because those are reserved words with special meaning. We can abbreviate those as TNF, and you'll see a capital T is blue. So is a capital F, and let's look at the class of just T. If we do that, it tells us it's a logical value.



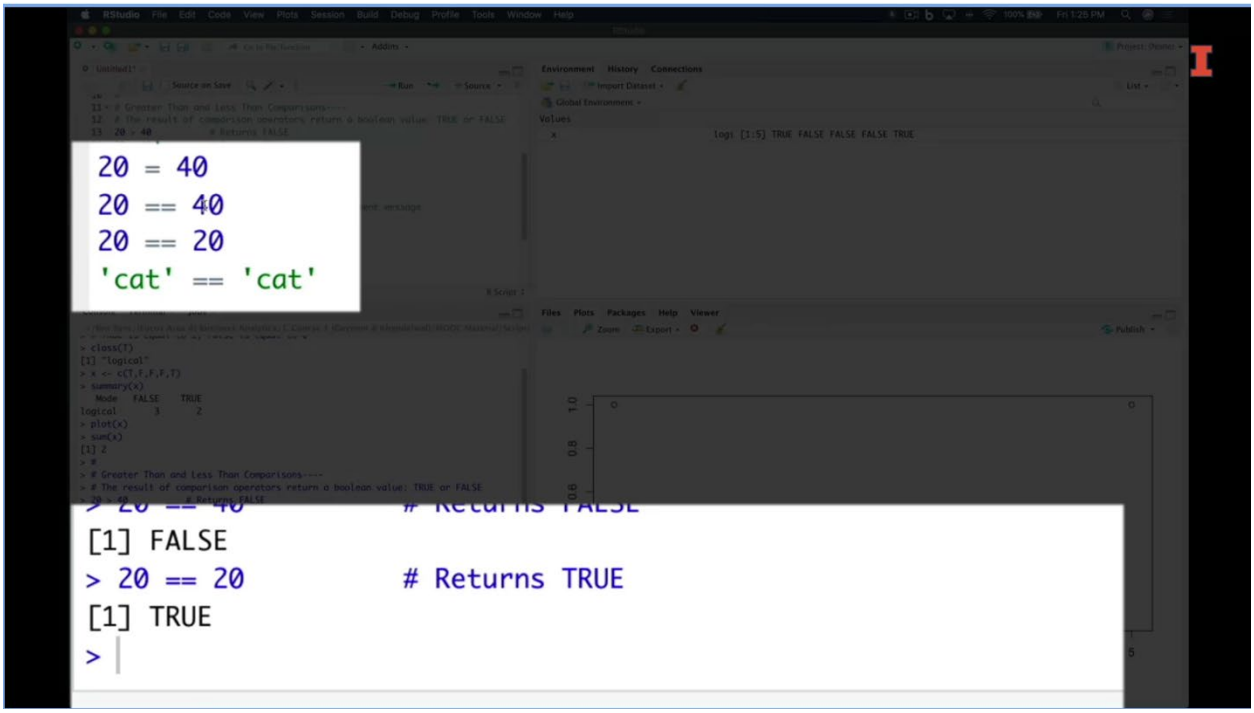
Let's create a vector of logical values. Do true, false, false, false, true. So we'll create that vector. And let's look at how vectors are somewhat like factors in the sense that you can plot them and you can get a summary of them.



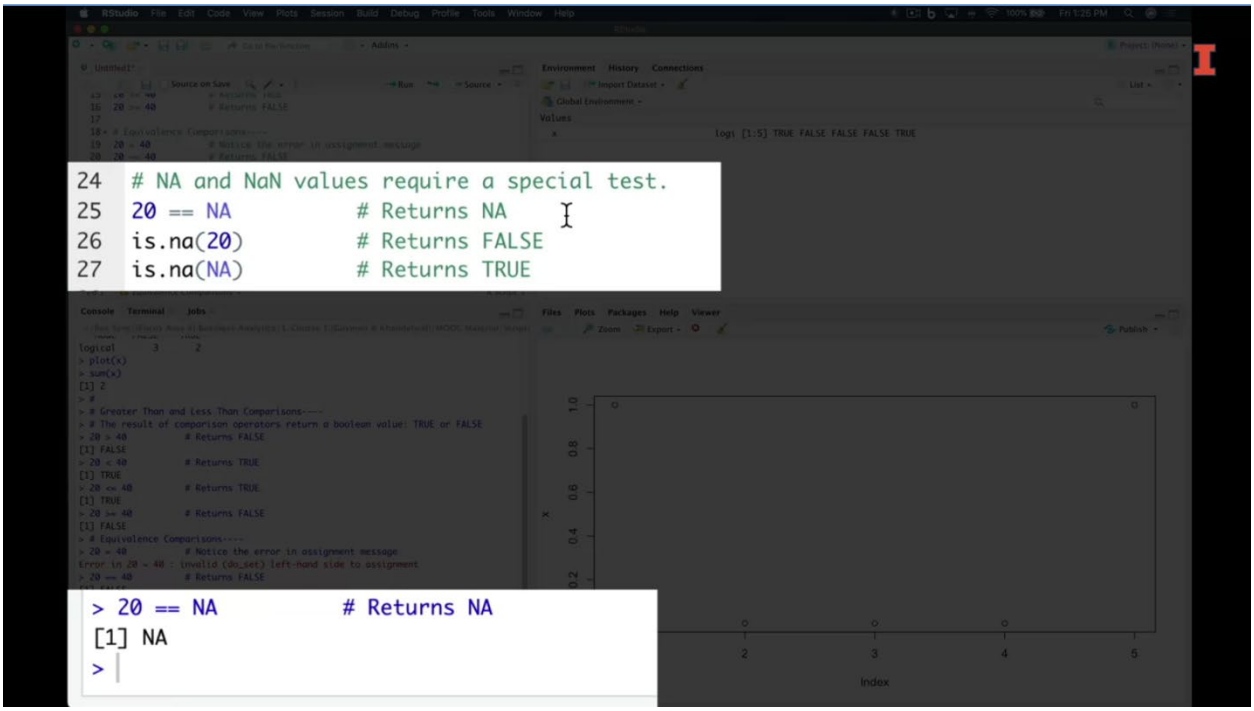
So if we use the summary function on this vector x that we created, it tells us that there are 3 FALSE and 2 TRUE and that it's a logical vector. And if we plot it, it returns a plot that shows us really the way logical values are treated. TRUEs are ones, and FALSEs are our zeros. So we can see 10001. We can also see that they're treated like zero or ones if we use a sum function on x, and it will return 2 because there are two TRUEs. So there's a quick summary of how logical data types work and are. Now that we know how logical data types work, let's perform some comparison operations.



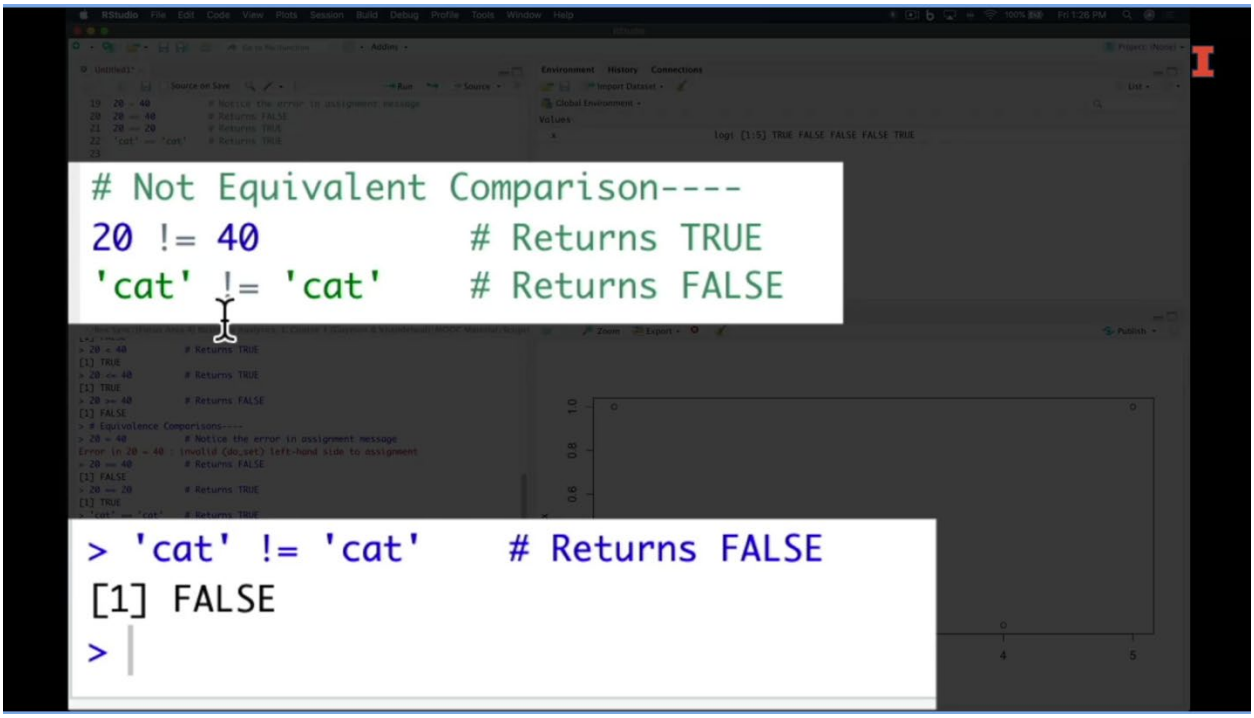
Let's perform some simple comparisons. Is 20 > 40? That's what this will be evaluated as it will return FALSE. A logical value FALSE. 20 < 40 returns TRUE, 20 < = 40 so you can use the less than or equal to sign, that is also TRUE or greater than or equal to sign. You can use as well, and that returns FALSE. So those are pretty straightforward, and I just wanted to highlight that they returned those logical values. What if you want to test if some object is equal to another object. Well, we can't use just a single equal sign because that is interpreted as assignment.



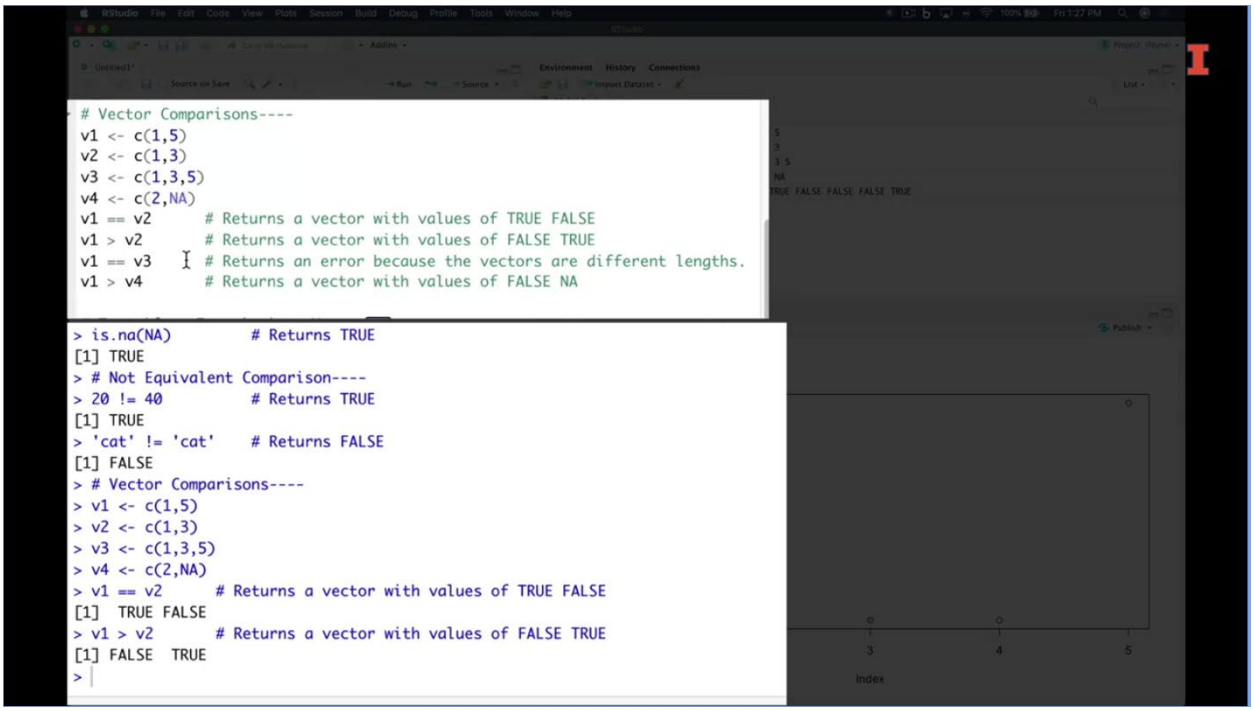
And we can't assign 40 to the value of 20, that will give us an error. So if we want to test, if a value is equal to another value, we use a double equal sign. So that's the equivalence comparison there. So it's 20 == 40? It returns FAULSE. How about 20 == 20? That returns TRUE. And we can perform this with strings as well, as the character string 'cat' == 'cat'. TRUE, that is all, right?



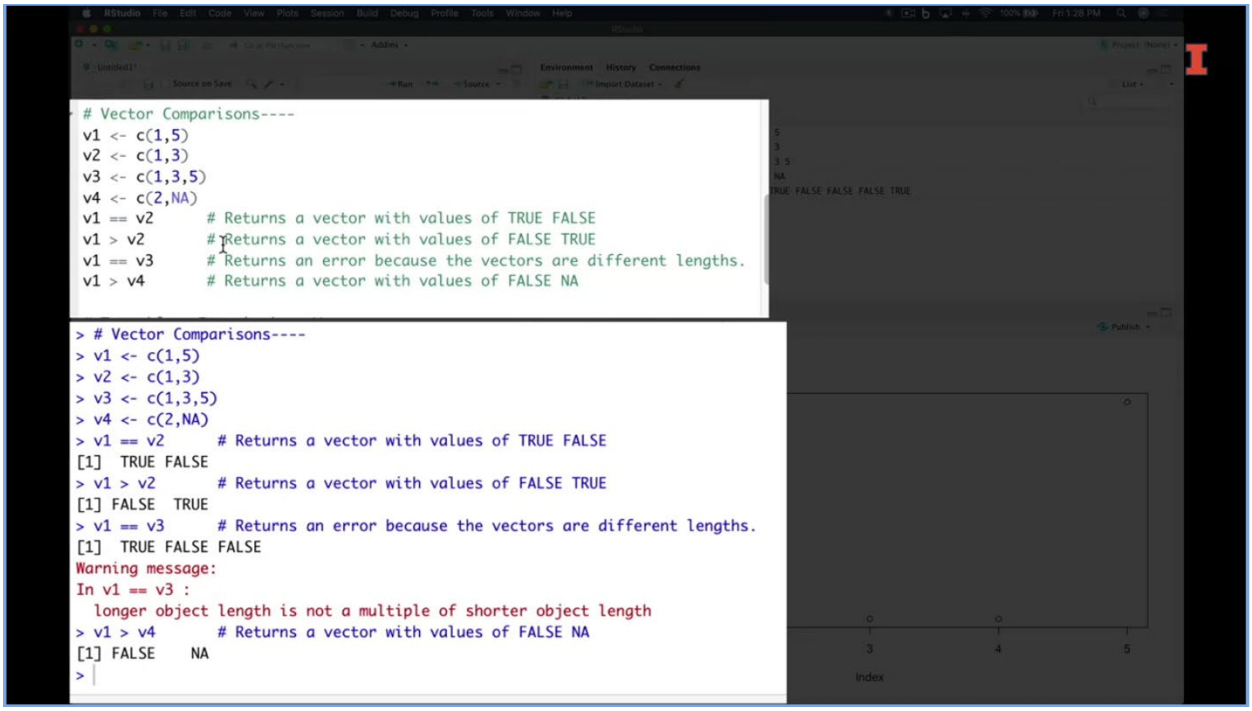
What if we want to compare something with NA or NaN? Those null values. We can't use the same equivalence comparison. The double equal sign, because it is basically saying, is 20 equivalent to an unknown value and it'll say, well, I don't know. So instead, we need to use the special equivalent operator for NAs, and that is simply is.na. So is.na 20 meaning is 20 not applicable and a non existent value? FALSE, because it has a value. Is na an NA? And that returns TRUE. All right, so those are equivalent operators.



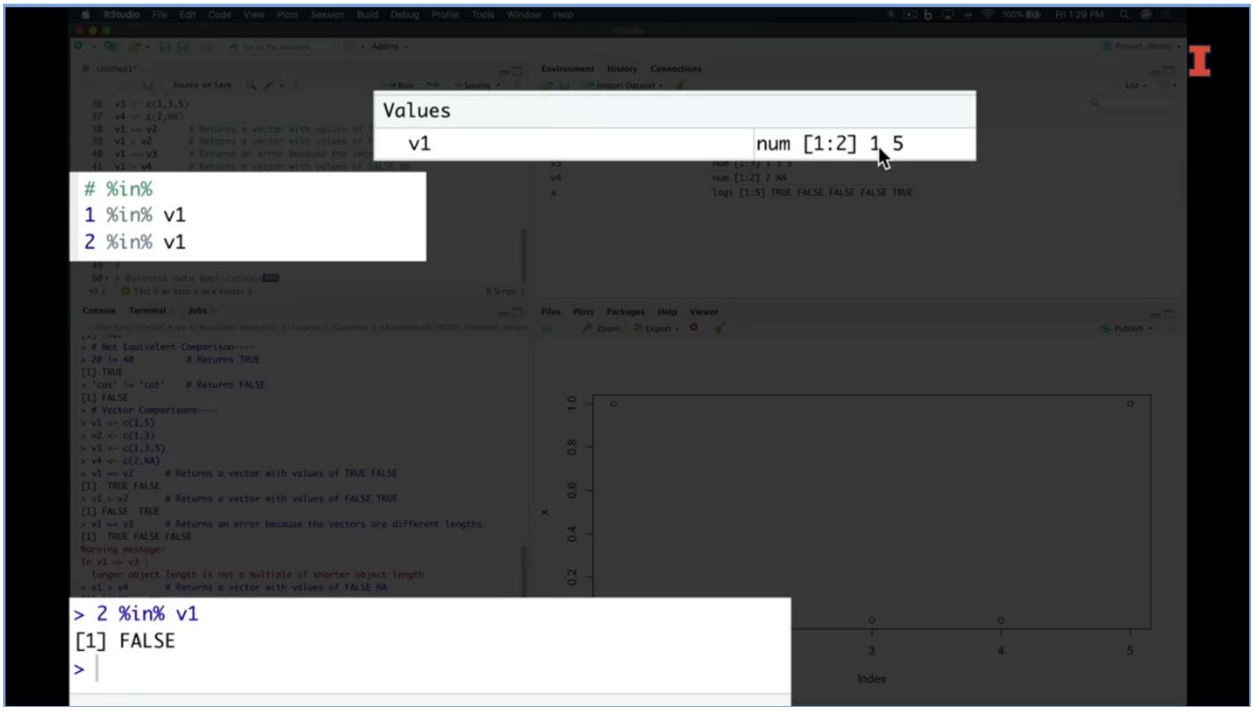
What about if we want to see if something is not equivalent to something else? Then we use an exclamation point and an equal sign. So it's 20 != 40? That is TRUE. And that's a little backwards, I realized that can be easy to be confused about that. But there definitely times when you want to see if something is not equal to something. So that returns TRUE. And if we compare things that are equivalent to each other and ask if they are not equivalent to each other, then it will return FALSE.



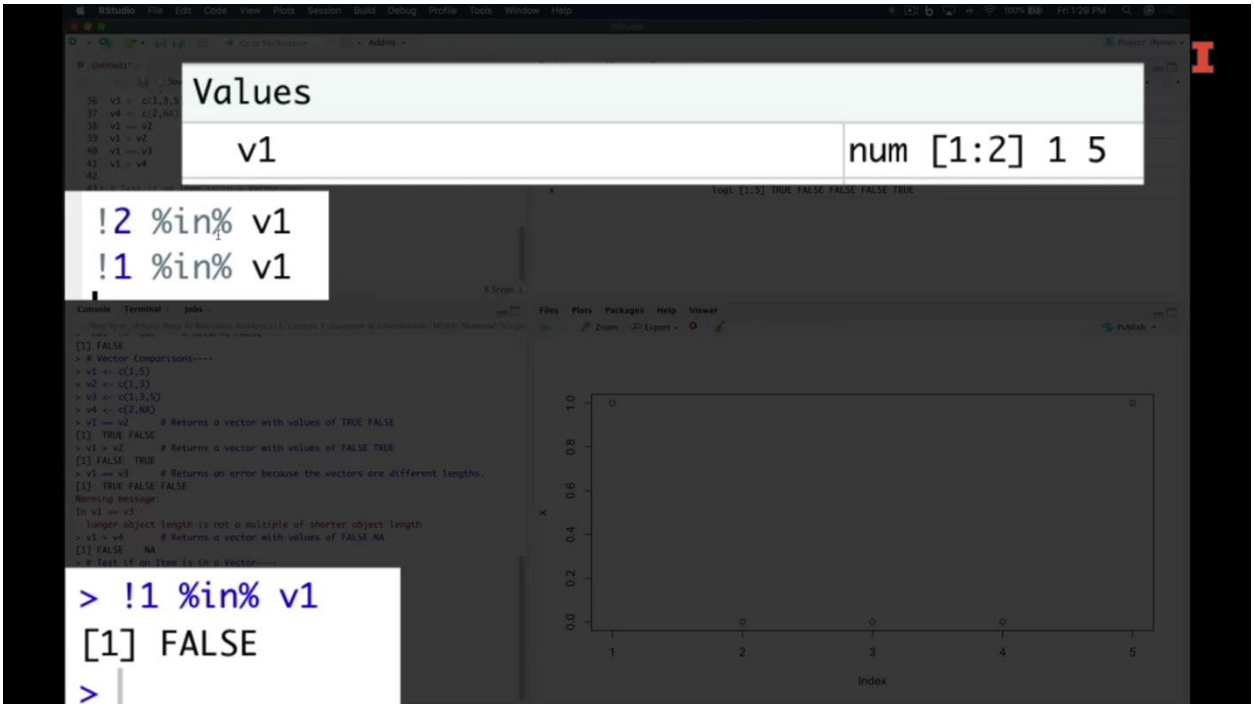
All right, and we can perform these comparisons on vectors as well. So let's create a vector1, that is 1 has the values of 1 and 5. v2 has the values 1 and 3. v3 has values 1, 3, 5 and v4 has values 2 and NA. So what happens if we compare vector1 to vector2 and see if they are equivalent to each other? Notice what happens is that it returns a vector of logical values TRUE and FALSE. So it does an element by element comparison in there. So in comparing vector1 vector2 to each other, the first element is equivalent, but the second elements are not equivalent. What if we use the greater then and perform that comparison? Is v1 > v2? Well, again, it returns a vector of logical values FALSE, because 1 is equal to 1, it's not greater than 1. And then TRUE, because 5 is greater than 3.



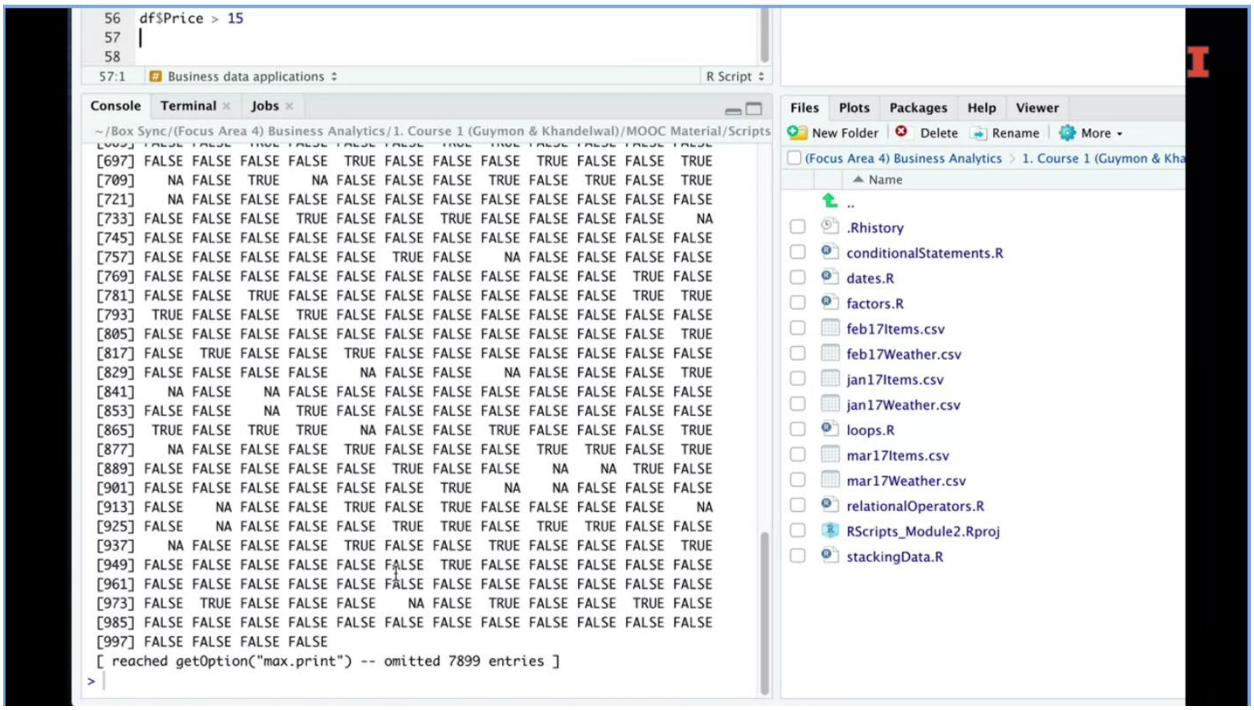
Now, just like when we perform Math with vectors, the vectors need to be the same length. If we try to compare vector1 to vector3, which are not the same length, then we'll get a warning message telling us that they're not the same length. And then finally, what if we compare values that have NaN in a vector? While just like an individual item, it will return values of FALSE and then NA. Because you can't really compare a number to a NA value. All right, so in other words, so the short of it is that you can use comparison operators on vectors, and it will do an element by element comparison.



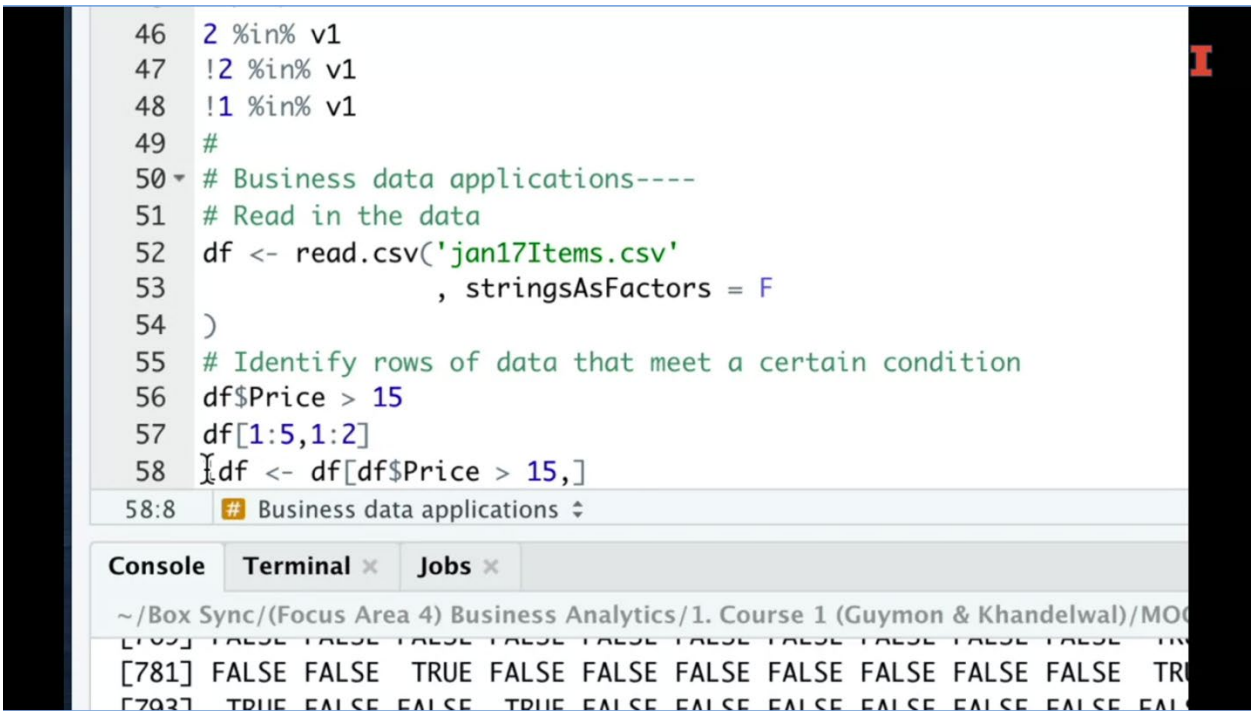
One other operator that's really helpful that I want to present to you, is this in operator? So %in%. And what this will do is it will test if a value exists somewhere in as a vector or a list of value. So let's see if 1 is in the vector of 1 in 5. We know it is, but let's use this %in and see if 1 is in v1. It returns a single value TRUE, because 1 is in v1. And if it were in their multiple times, it would still just return the single value of TRUE. In contrast, if we compared 2 in v1, then it returns FALSE because 2 is not in the vector of 1 and 5.



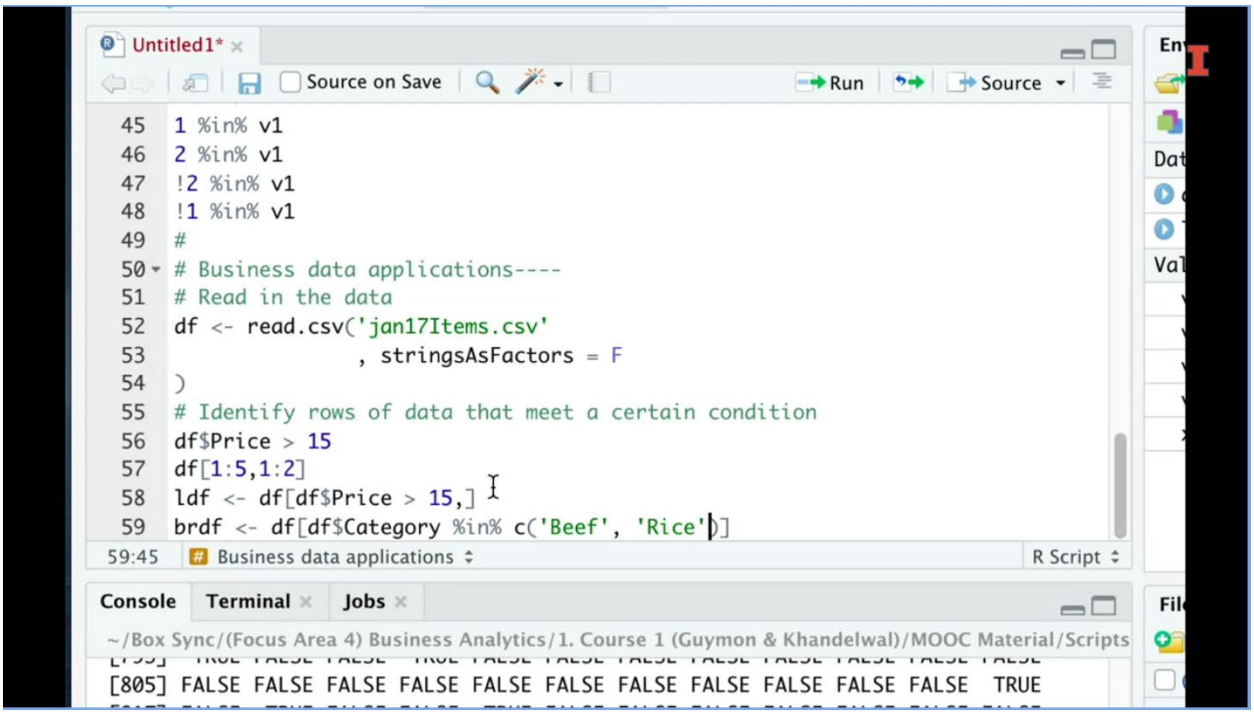
So to evaluate if something is not in a vector or a group, it's not as straightforward. But it's somewhat similar to using the not equivalent comparison So let's start with 2%in%v1. And if we want to test if 2 is not in v1 instead of if it is in v1, then we need to put the exclamation point in front of the 2 and run that. That returns TRUE because 2 indeed is not in v1. So it's not super intuitive, but that's how you do. And just for completeness, let's test if !1 %in% v1. And that is FALSE because 1 is in v1.



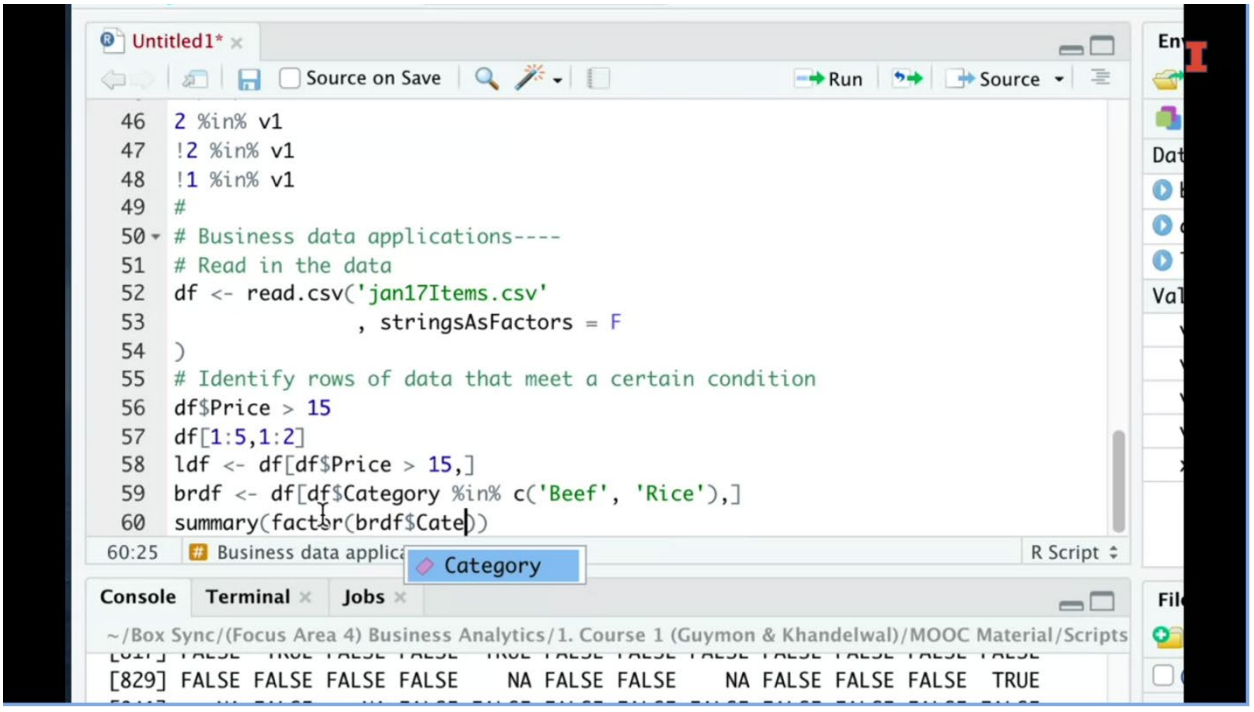
Let's now talk about how we can apply relational operators and logical values to a business data application. So I'm going to go ahead and read in the jan17items.csv file and convert it to a data frame. One way that we can use these relational operators is to filter the data down to certain rows based on a condition that we are interested in. Let's just start by looking at and making sure that if we compare this vector, let's say Price and we'll see if Price > 15. Think of Price just a vector and we're going to element by element go down and make that comparison. So what we get is a vector that is 8899 elements long. And it has logical values, TRUE, FALSE or NA, based on whether or not price in that row is greater than 15.



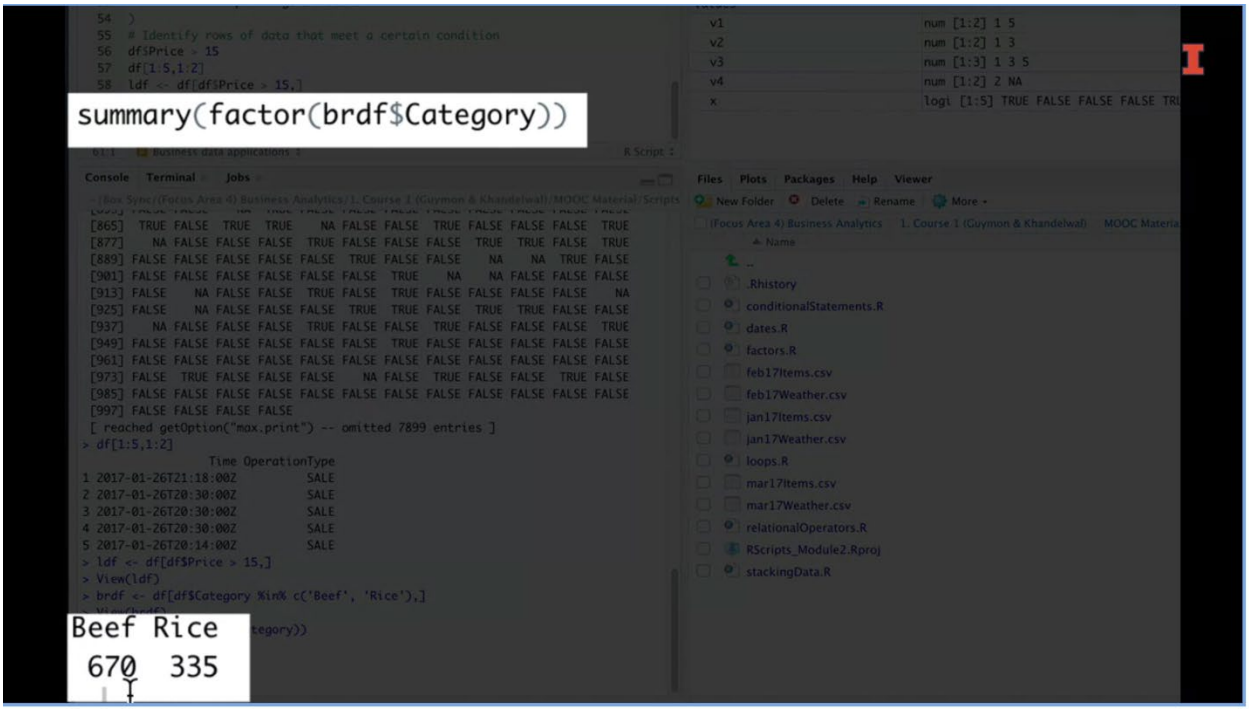
Now, that's not helpful in and of itself. But here's how it can be. So remember how we can identify certain rows of data based on location. Let's say we want to just return the first five rows of data and maybe just the first two columns of data from the DF data frame. We can do it like that. So let's take that idea and combine it with this idea, performing an element by element comparison. And let's return only the rows of data of df, for which the Price value > 15. All right, so it's going to return in this bracket here. It will basically say, return only those rows that have a value of TRUE in it or NA actually. So let's assign this to a new data frame, we'll call ldf.



And if we look at this data frame, it is a much smaller subset 1942 obs. And if we go to the price column, we can see that all the observations are either greater than 15 or RNA. Right, so that's not the most convenient way to filter data, but this is how it could be applied, at least under the hood in some instances. Now, another way in which we might want to filter the data is, let's see if we can narrow down this data frame to only the rows for which the category is beef or rice. So let's name this new data frame brdf.



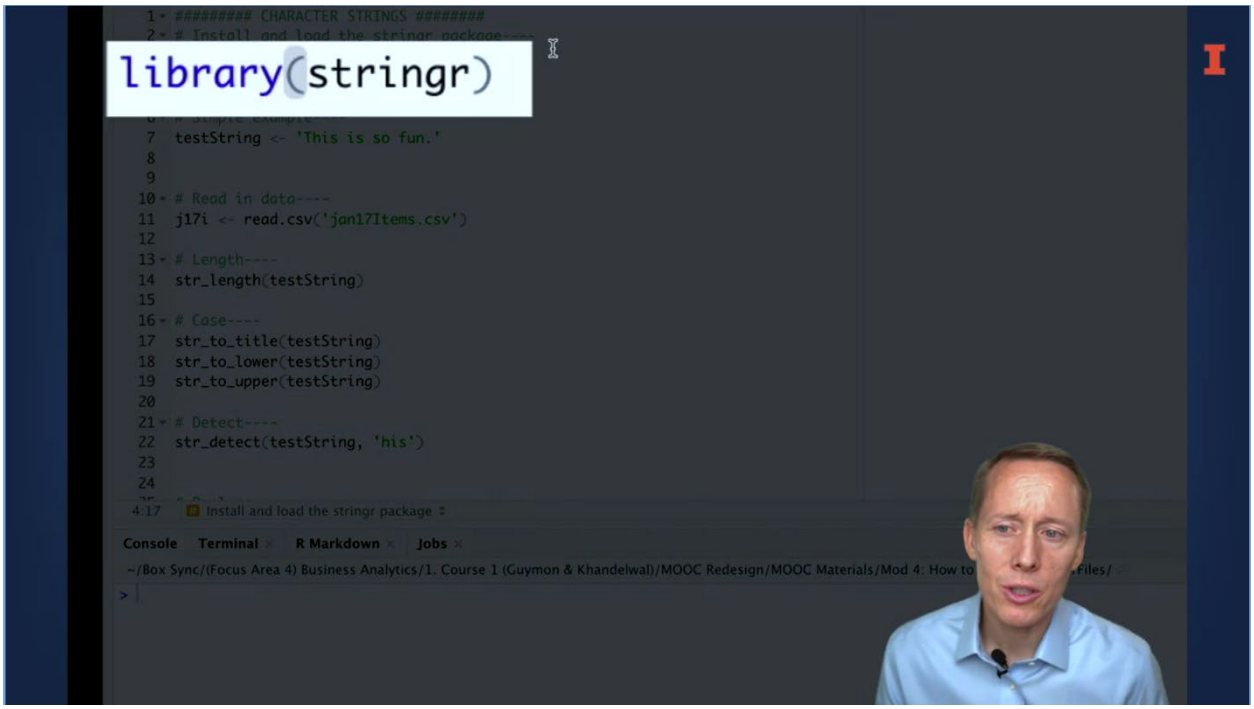
And we'll say, return the rows of df for which the df category value is in. And then we can create a vector as a character string vector, that contains Beef and Rice in it. And then we'll put a comma to indicate we want all the columns of data. And we'll run, that and we get this brdf data frame. It's 1005 observations long. And if we just visually explore that, we can look at the category and see that there's beef and rice in there. And we can even do a summary of the values in this be brdf data frame, and it currently has br category as a character string.



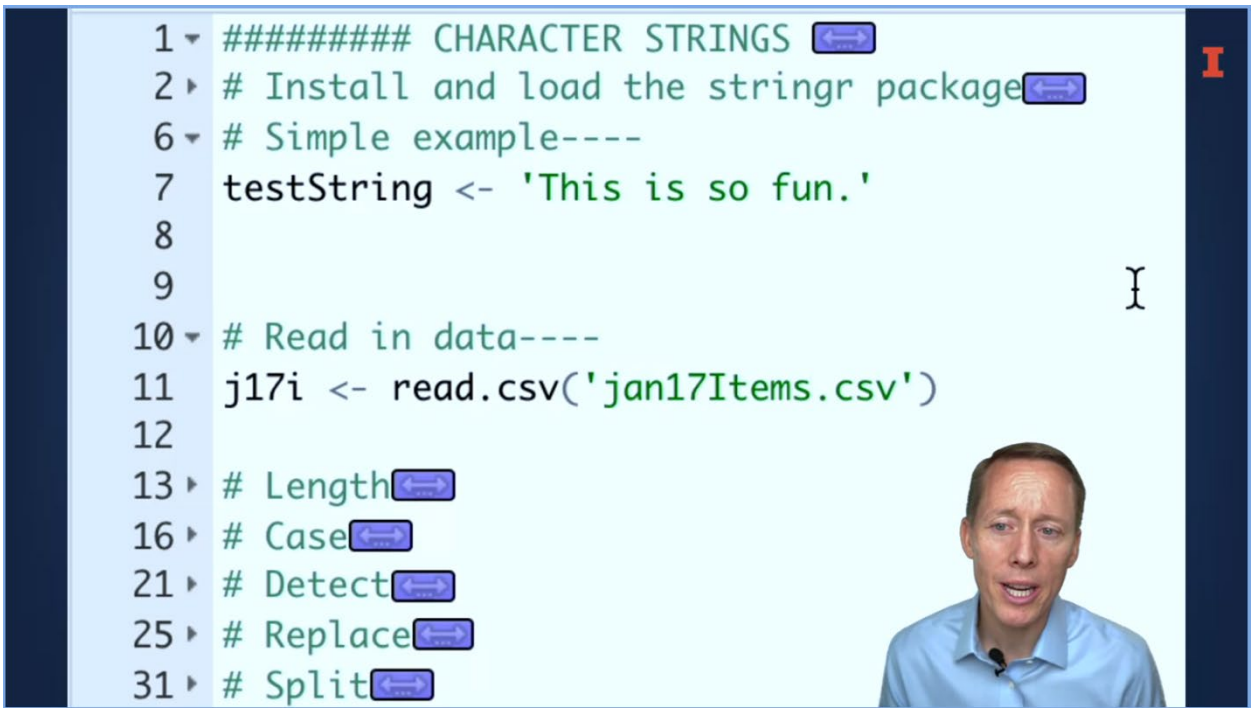
So let's change it to a factor within the summary calculation so that we can see the different values were on that. And there we've got 670 rose for which the categories beef and 335 to which the category is rice. So that's an introduction to logical data types and how they are used in relational operators, as well as how they could be applied in a business data setting. We'll talk about simpler ways to do this filtering, but hopefully that's helpful to get an idea of why they're important.

Lesson 3-10 Character Strings

Lesson 3-10.1 Character Strings

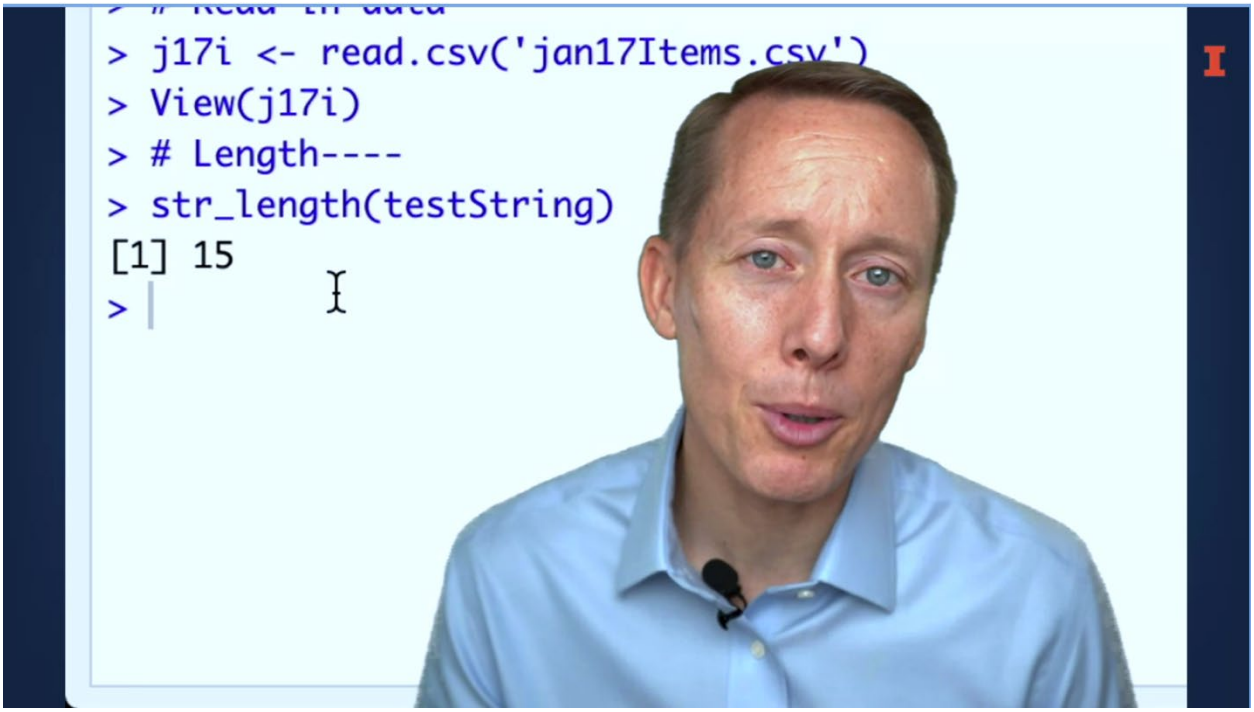


In business analytics, there are a variety of tasks that are often performed using character strings. In this lesson, I want to show you how to use functions from the stringer package to perform some of those tasks. The first thing is to make sure that you've already installed the stringer function on your machine. If not, then go ahead and run the installed packages function or install the package using the packages. Pain. Yeah, yeah. Once it's installed, make sure and load the functions from the Stringer library by using the library function.



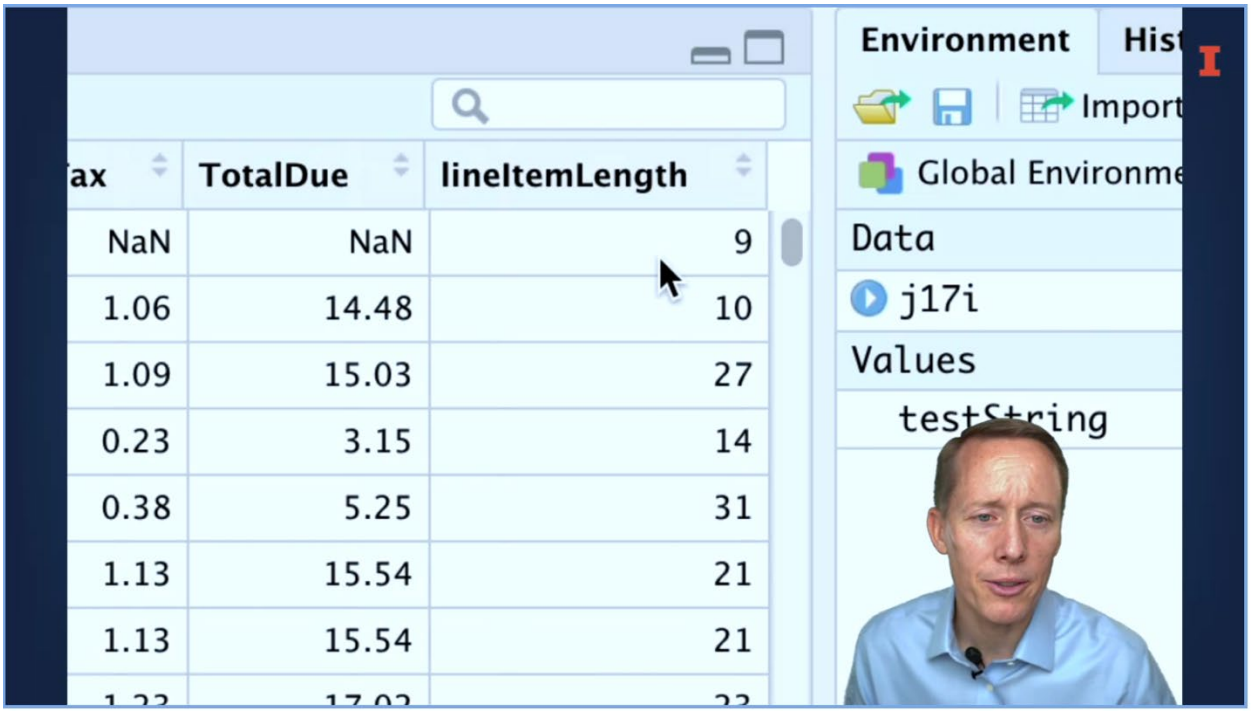
Now, before we start using functions from the stringer package, let's go ahead and look at what some of those functions are, so you can see that there are many functions that start with str underscore. We are going to review some of those functions in this lesson. Let's go ahead and start with a simple character string, and it will be a sentence. This is so fun, and we'll store in an object test string so we don't have to keep typing out. This is so fun. I also want to show you some examples using business data, so I'm going to read in the Jan. 17 items dot c. S v file and store it as a data frame called J 17 I.

Let's go ahead and quickly browse this data frame. Mhm. We can see that about the first 15 columns contain character string values. The remainder are numeric values. What are some of the business analytic functions that we'd want to perform on character string columns? Well, perhaps we'd want to separate out the date from the time and the Time Column. Perhaps we want to create some new categories based on certain key names in the menu. Items in the line Item column. Maybe we would want to analyze the cardholder names. There's a variety of other analytics that we may want to perform. That's just a sample, all right.

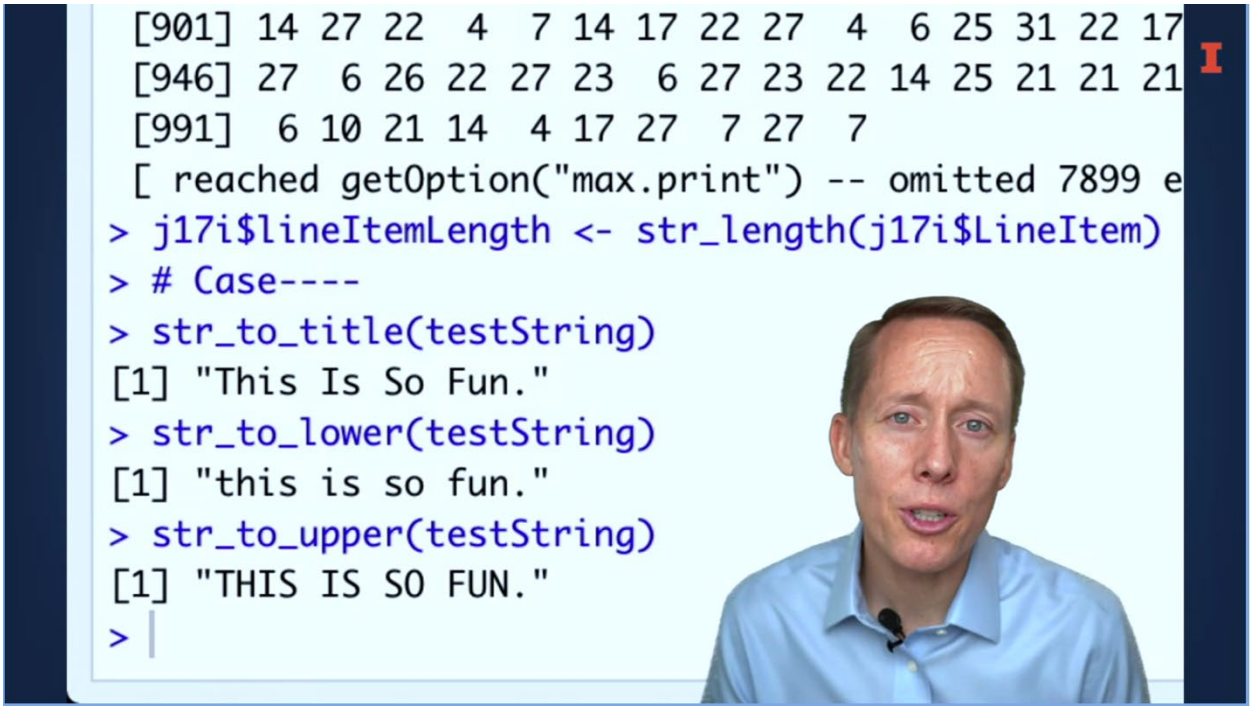


The first function that I want to show you is the string length function, and this will tell us the number of characters in a string. So I'll go ahead and run the string length function on the test string object and notice that it tells us that this character stream contains 15 characters in it. Now, all of the functions in the stringer package are vectorized, meaning it will perform that same operation on every observation of a vector. So if we want to calculate the number of characters in every line item for every observation, we can easily do that.

And here's how we would do that. You can see that what has returned is a whole bunch of numbers. Now. This is really a vector. So if we want to create a new column so that we can analyze it, let's say we want to evaluate. If the length of the line item name influences the amount of sales, we could easily do that.



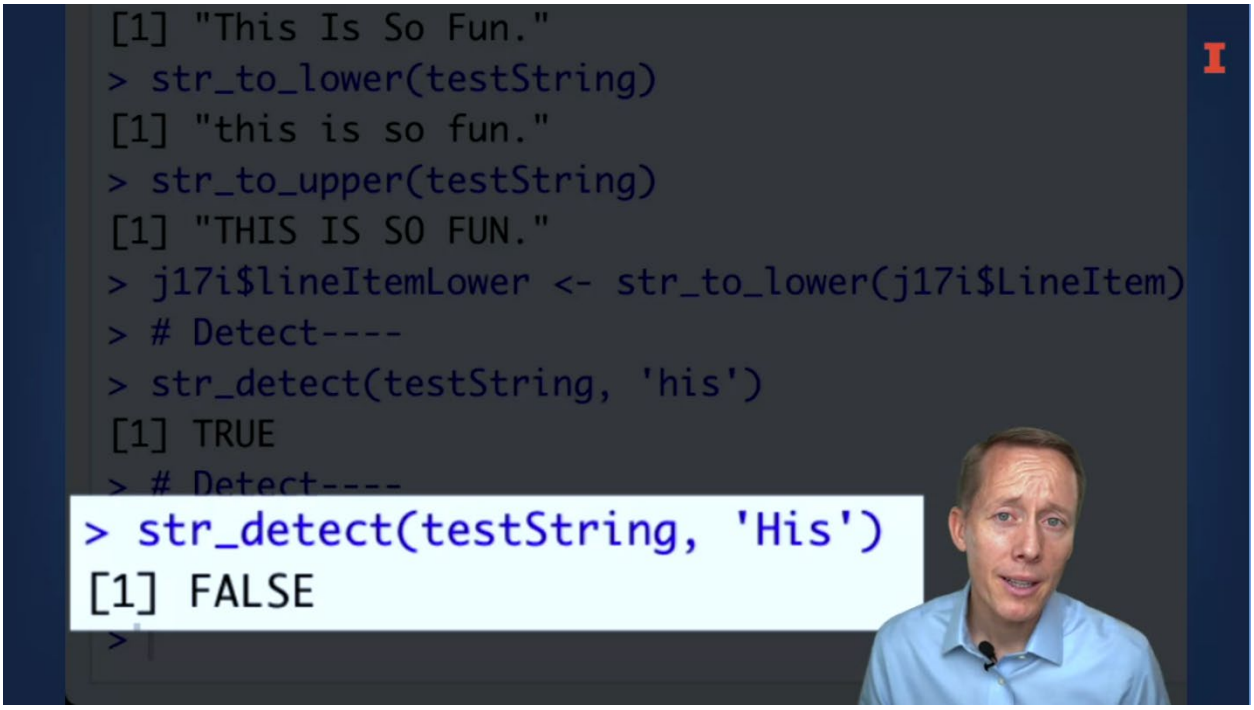
So let's create a new column and let's call it line item length and let's go ahead and look at our data frame here and scroll to the far right and we can see the line item length is created, and sure enough, it takes the length for the line item and stores it as an American value. Very good.



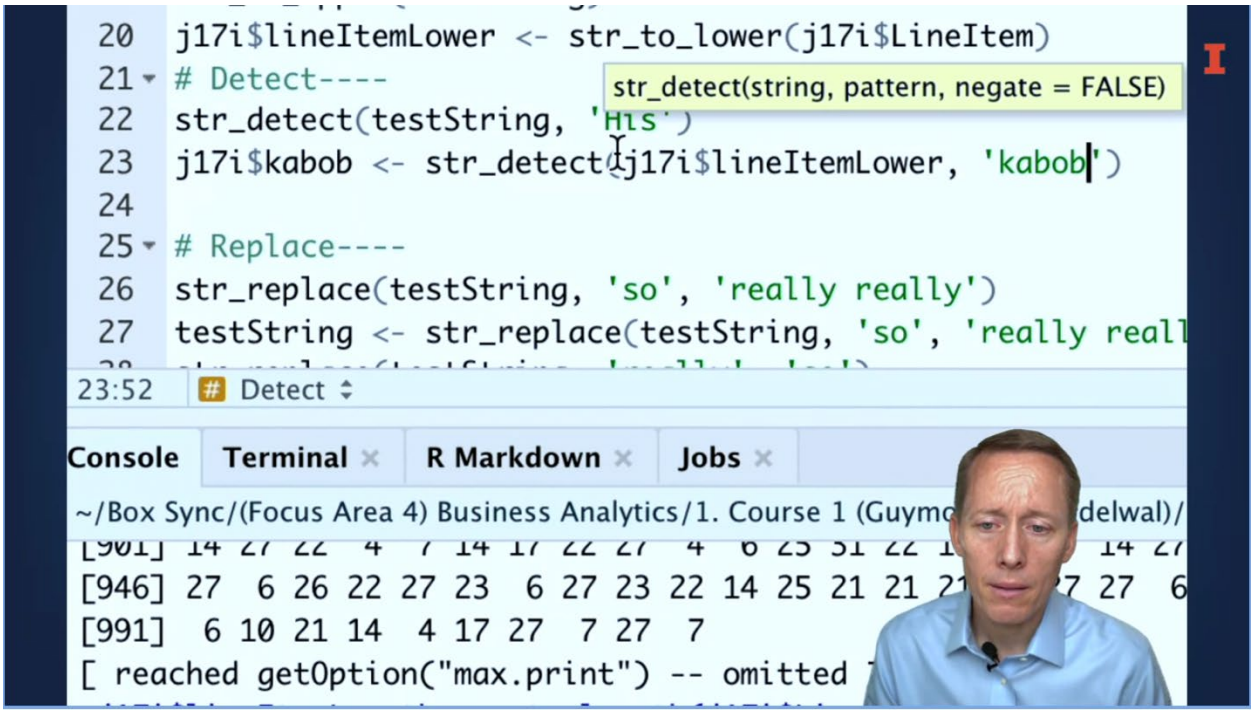
Let's go ahead and move on to case functions, so case refers to whether you're using upper case or lower case values for character strings. There are three case functions. There's title lower and upper. Here's the title. The strength to title function you can see what it did. Is it capitalize? The first letter of each word. The string to lower just converts every character to lowercase string to upper. It converts every character to uppercase. Case conversion is important In business analytics, for instance, let's say we want to create a new column in our data frame that tells us whether or not the line item name has the word kebab in it.

Well, the first thing we want to do is explore the line item names and find out if Kabob is spelled using uppercase, sometimes lower case sometimes or title case. If we think that might be the case, then we want to convert everything to either lower or upper case. So let's go ahead and do that.

Let's convert everything in the line item column to a lower case value and will create a new column. We'll call it line item lower. We'll use a string to lower function on the line Item column. Let's go ahead and take a quick look at our data frame now, and sure enough, we can verify that the line items are all lower case.

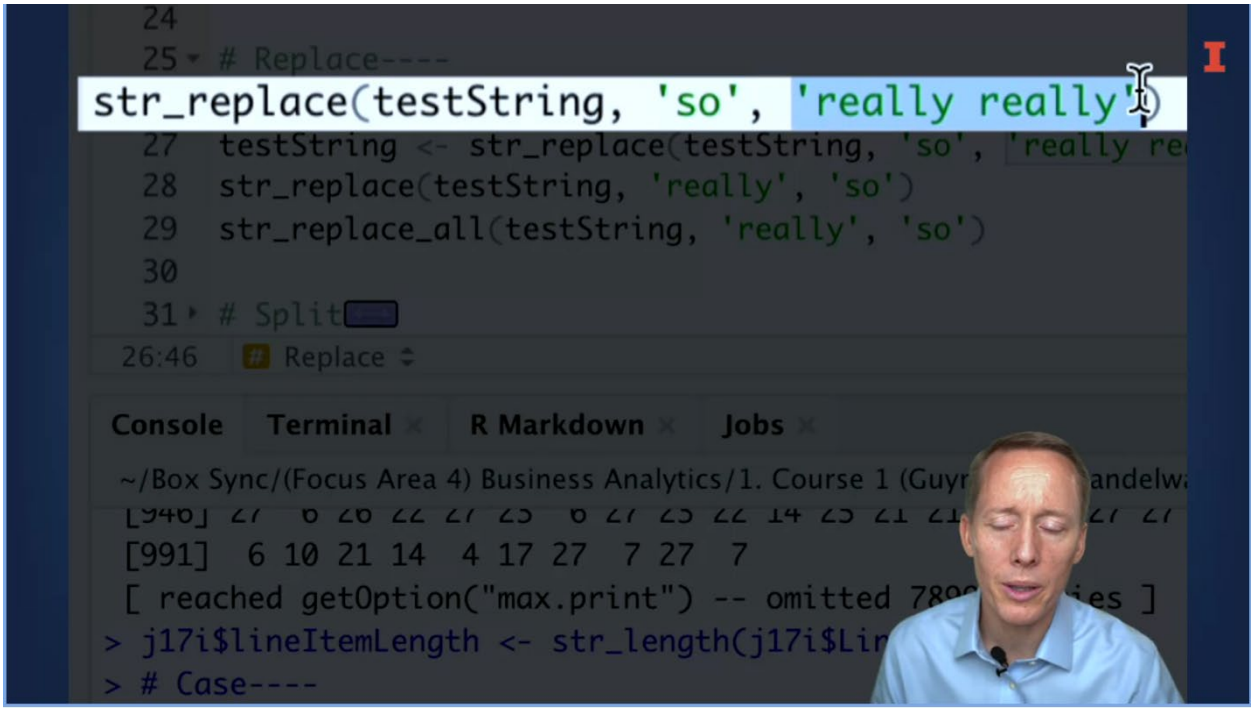


Let's talk about the detect function. The detect function allows us to detect if a certain sub string exists. So a sub string is just a sequence of characters that may appear anywhere in a character string. In this case, we're going to detect if h I s appears in our testing object. Notice that it returns a value of true. The only other value will return is false. And why does it return true? Well, our testing object is a character string that says this is so fun and h i s is in there. Now let's see what happens if we run this with a capital H i s. It returns the value of false. That's important. So the detect function is case sensitive.

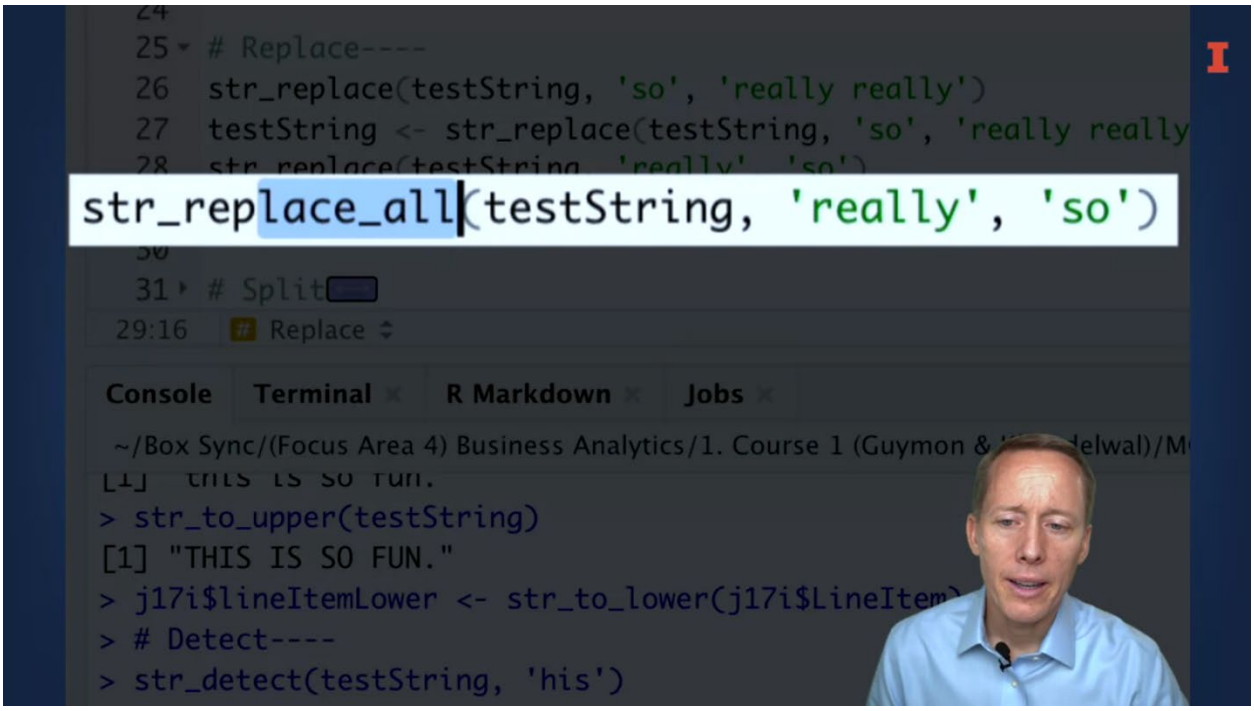


Let's go ahead and use this now. And let's create a new column in our data frame called kebab that indicates whether or not the line item menu contains kabob in its name. Somewhere I'm going to use the line item lower column and I'll go ahead and run this. Let's look at the results.

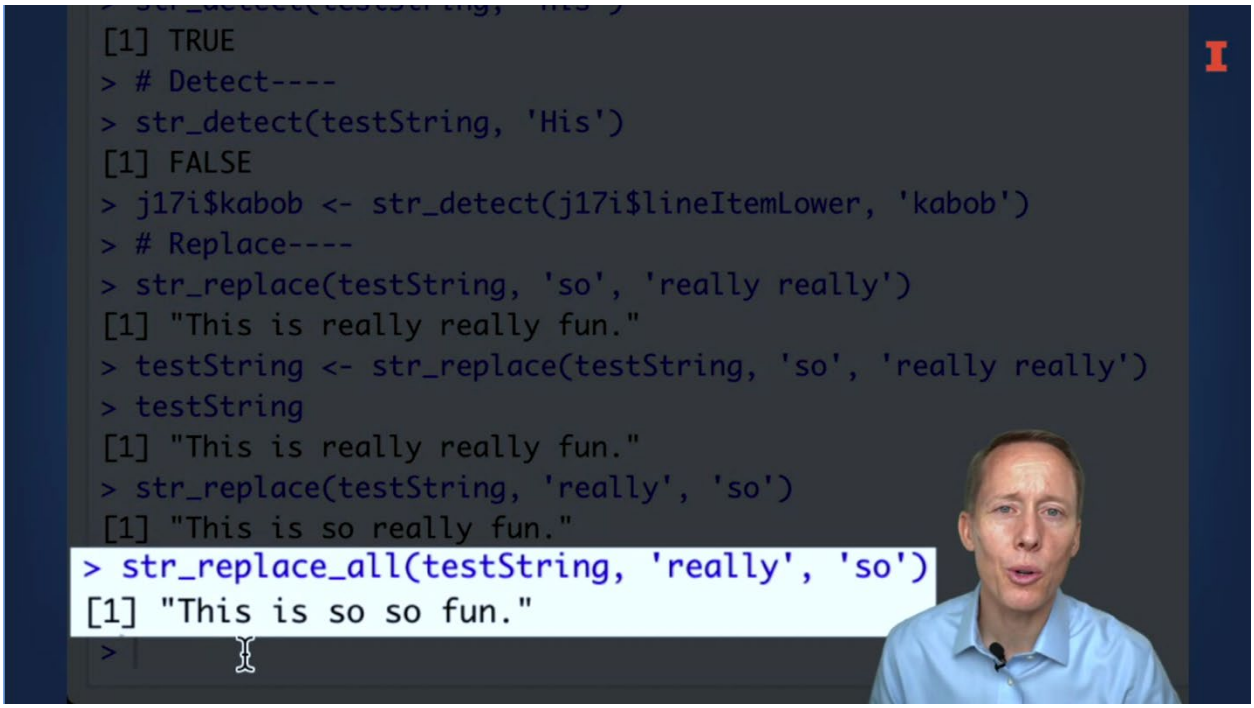
Yeah, and as we look at this kabob column here, we can see false, false, false, true, and I think it's always a good idea to just do a quick visual inspection and we can see that for these true values here they do have kebab in it for the false values they do not. So it appears to be working.



The next function I want to show you is replaced. Replace is like the find and replace so it will detect a sub string and replace it with whatever characters during you identify. So let's go ahead and run string. Replace on the testing object and we will identify any instance of Esso and replace it with really, really so you can see it printed out. This is really, really fun. Now I'm going to go ahead and replace what's in the testing object with this updated values so that now that test ring does say this is really, really fun.

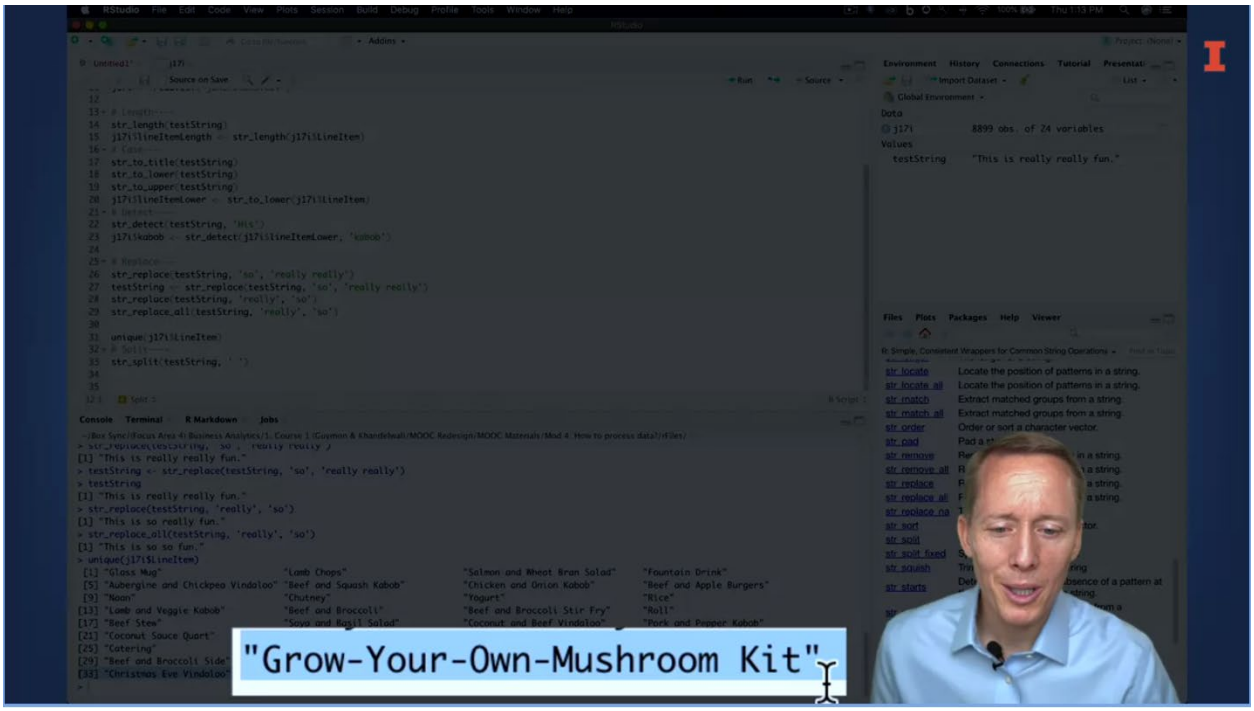


Now I want to go back and replace every instance of the word really with. So I'll run the string, replace function again on the test string object and will replace really with. So I notice what happens when I ran this. It only replace the first instance of really with. So if we want to replace every occurrence of a sub string, then we need to use the string, replace all function, and so I'll try this again.

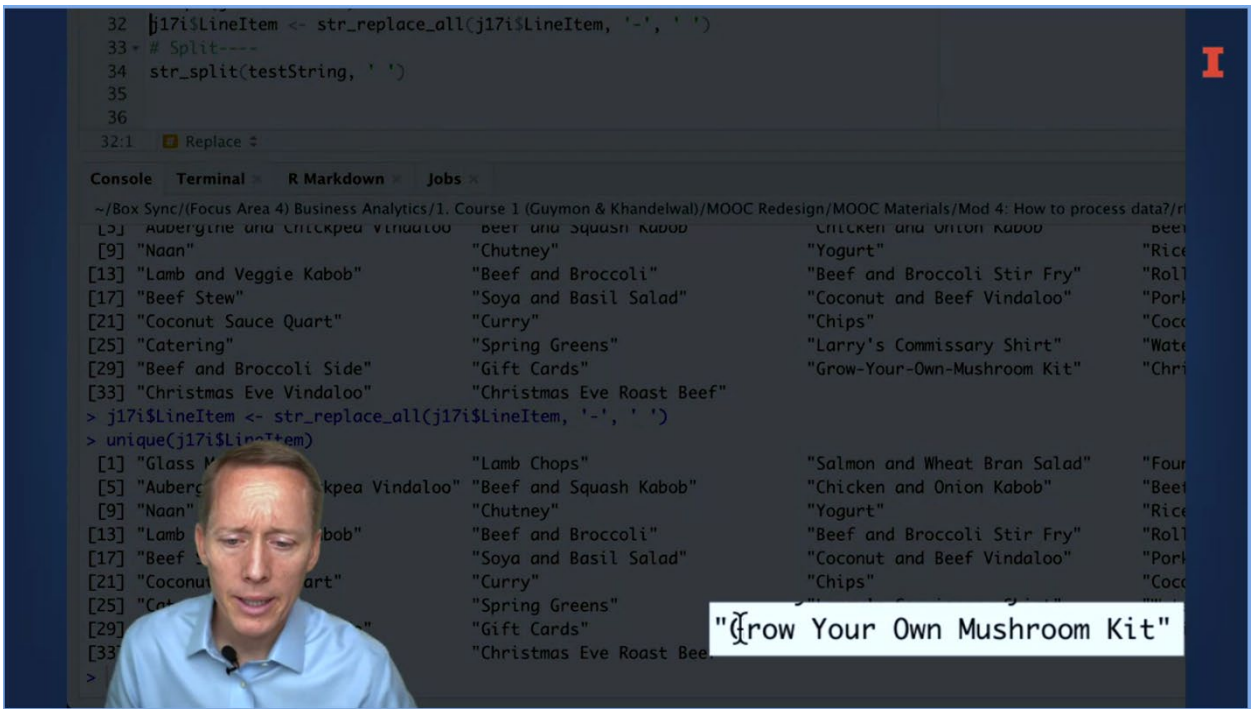


This time it worked. It says this is so, so fun. So let's talk about a business analytic application of this. I'm going to introduce a new function to that you may not have heard of.

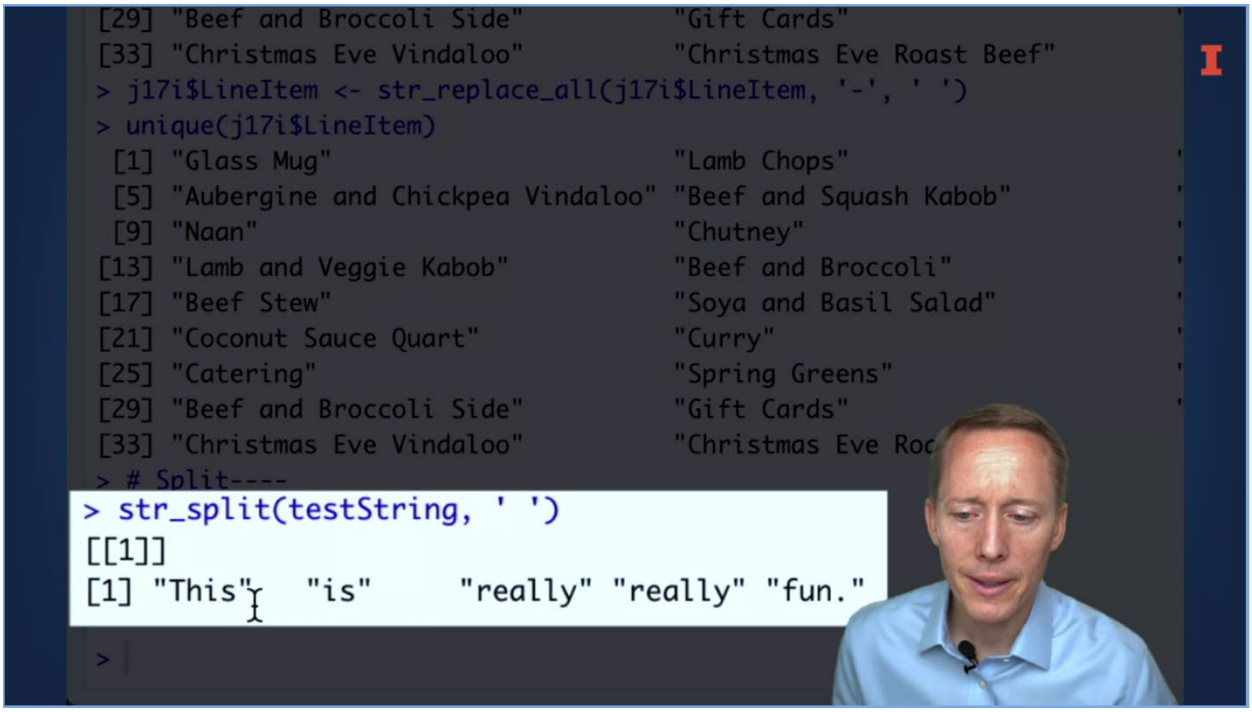
And that is the unique function. Unique function will take only the unique instances of certain values and report them. So I want to look at the unique values of the line Item column. If we go down here in the council, we can see that there are 33, 34 unique line items. And let's pay attention to this.



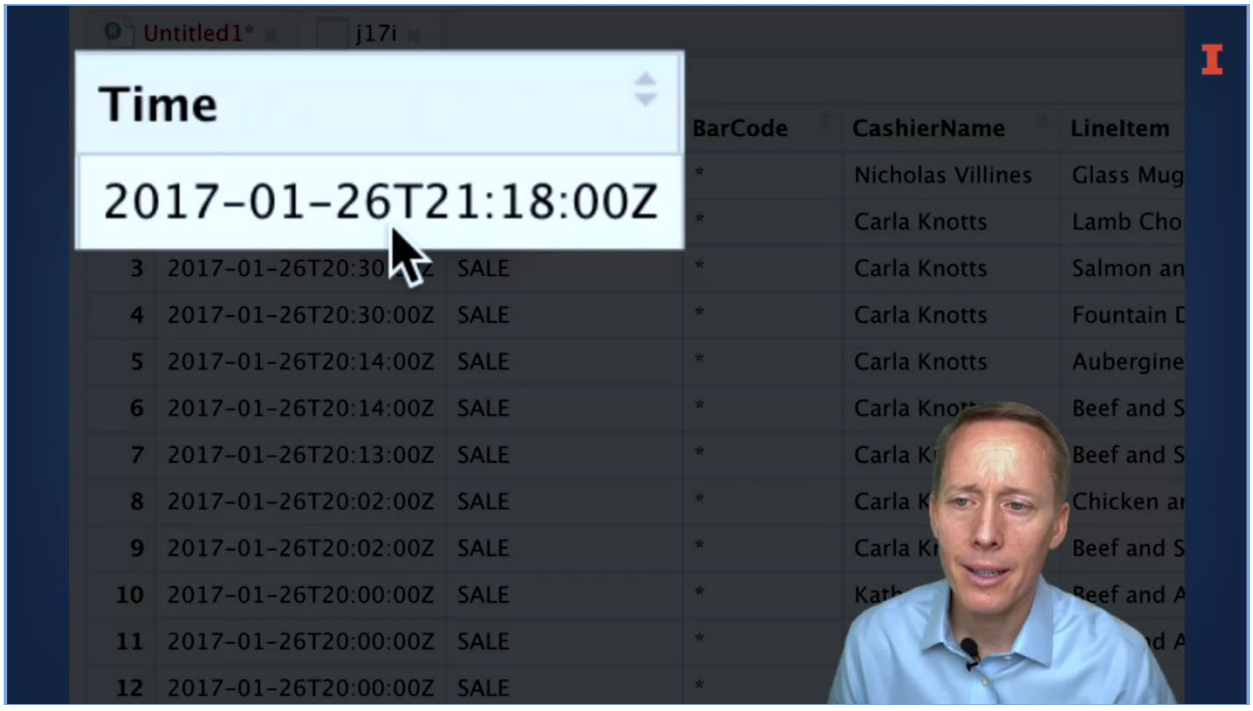
Grow your own mushroom kit, which is kind of a funny item anyway. But notice that it has three in three different hyphens in it. Let's say we want to replace those for some reason, with just spaces. We can easily do that and let's go ahead and store it back into the line item column.



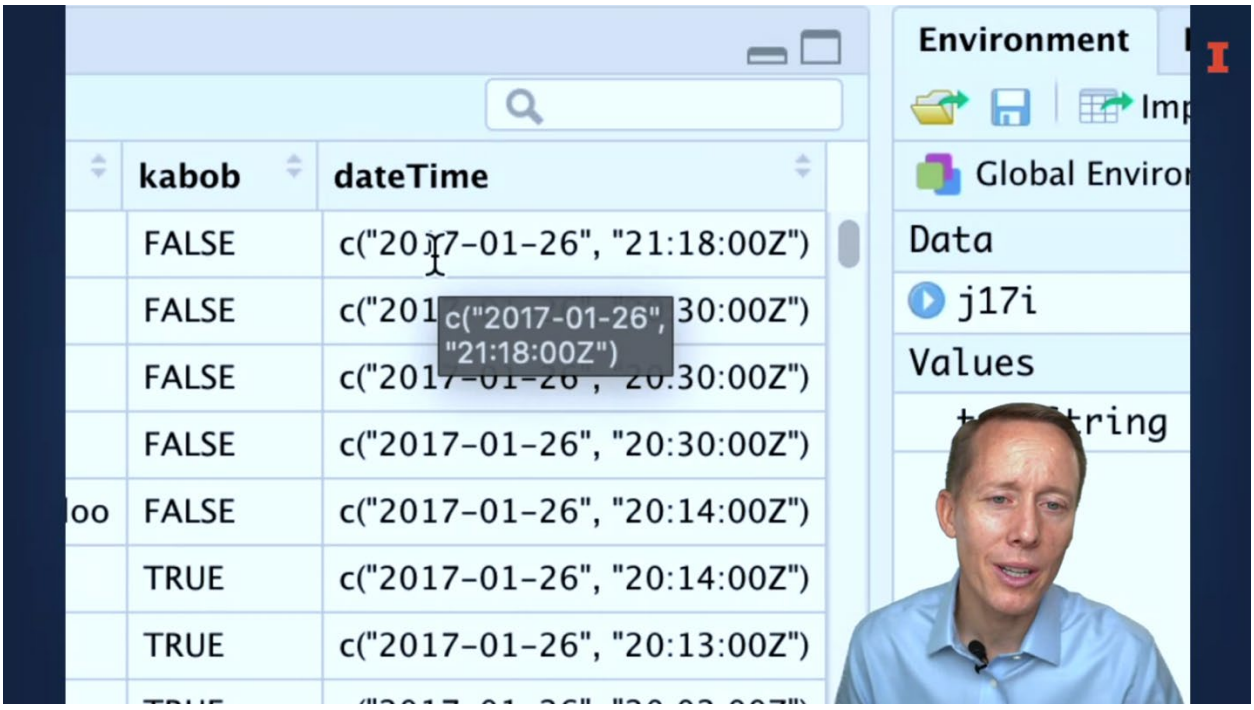
So I need to use the string replace all function, and I will replace the minus with or the hyphen with a space. Now go ahead and run that unique J 17. I line item row and go down and look at the girl. Your own mushroom kit looks like it did what we expected it to do. Very good.



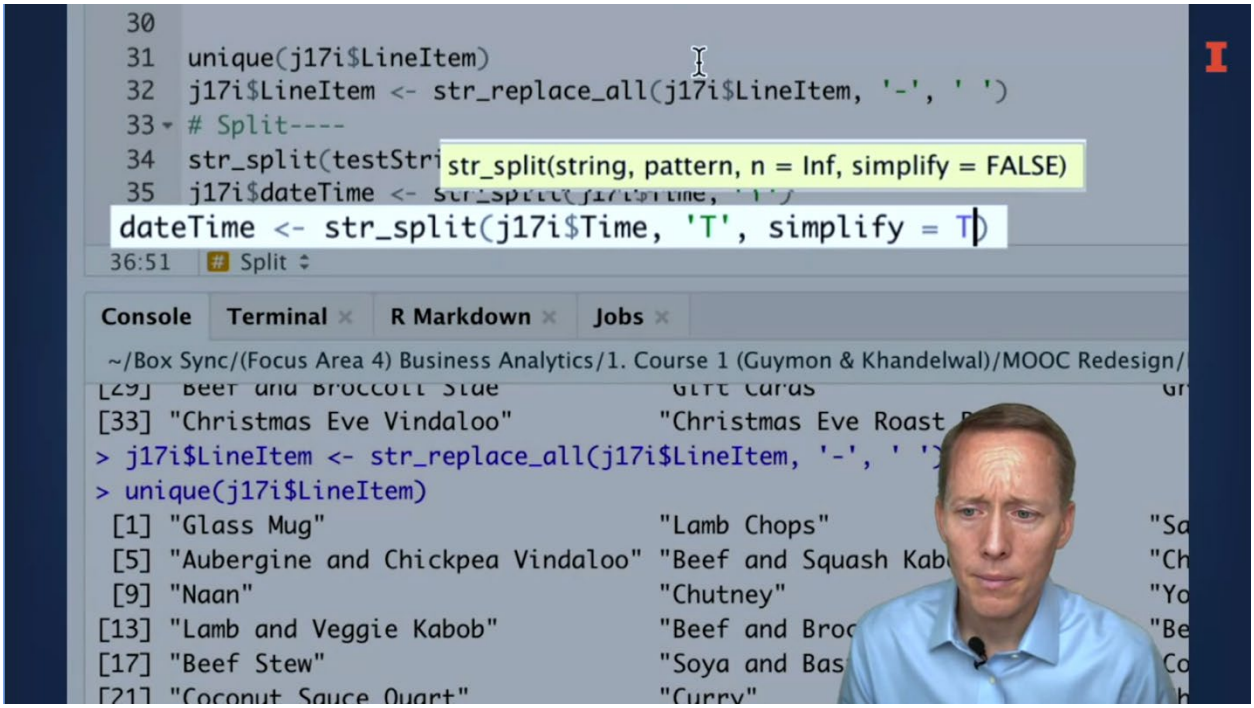
The last function I want to demonstrate is the split function. The split function takes a character string and splits it into separate elements. Let's go ahead and test this out on the test during object, and we're going to use a space as a D limiter. So as I run this notice, what it prints out is a vector with five different elements. This is really, really fun. So why would you want to use this function on business data?



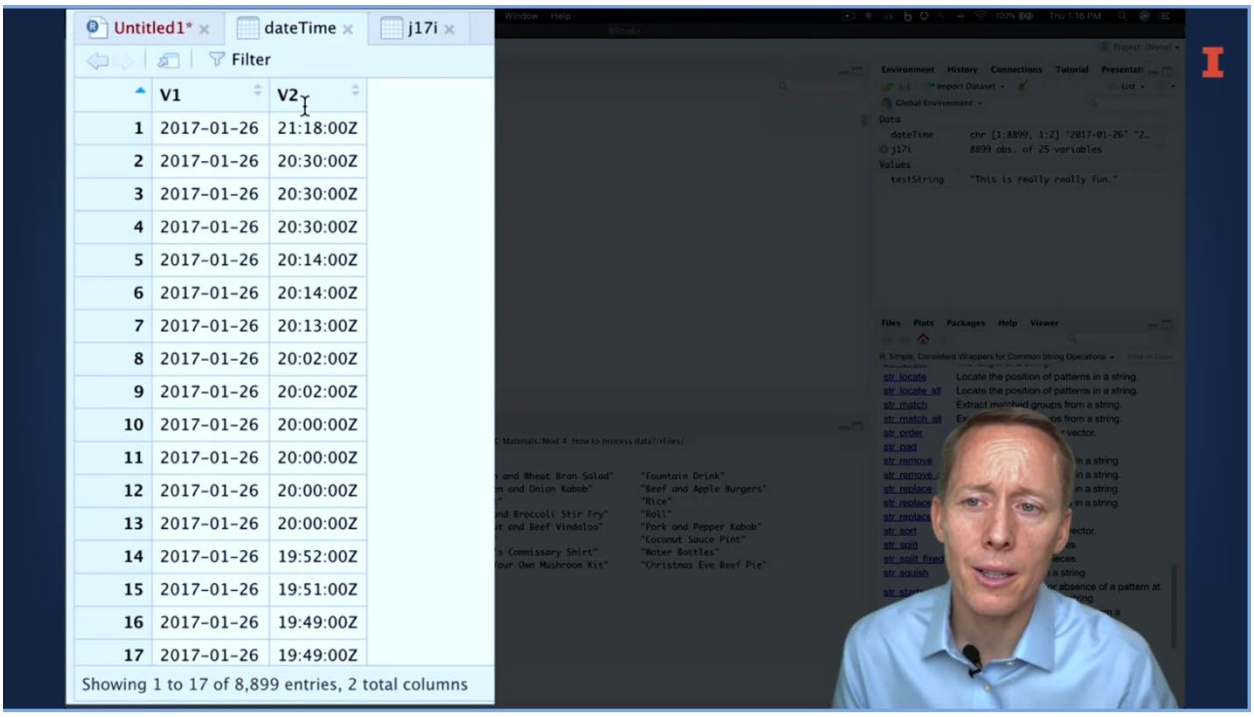
There's all sorts of reasons why one example here is if we want to look at the time column and extract something out of that so we can see that in this column here we've got the date and then a capital T, and then the time. Let's say we want to separate out the date from the time and create two different columns in this data frame.



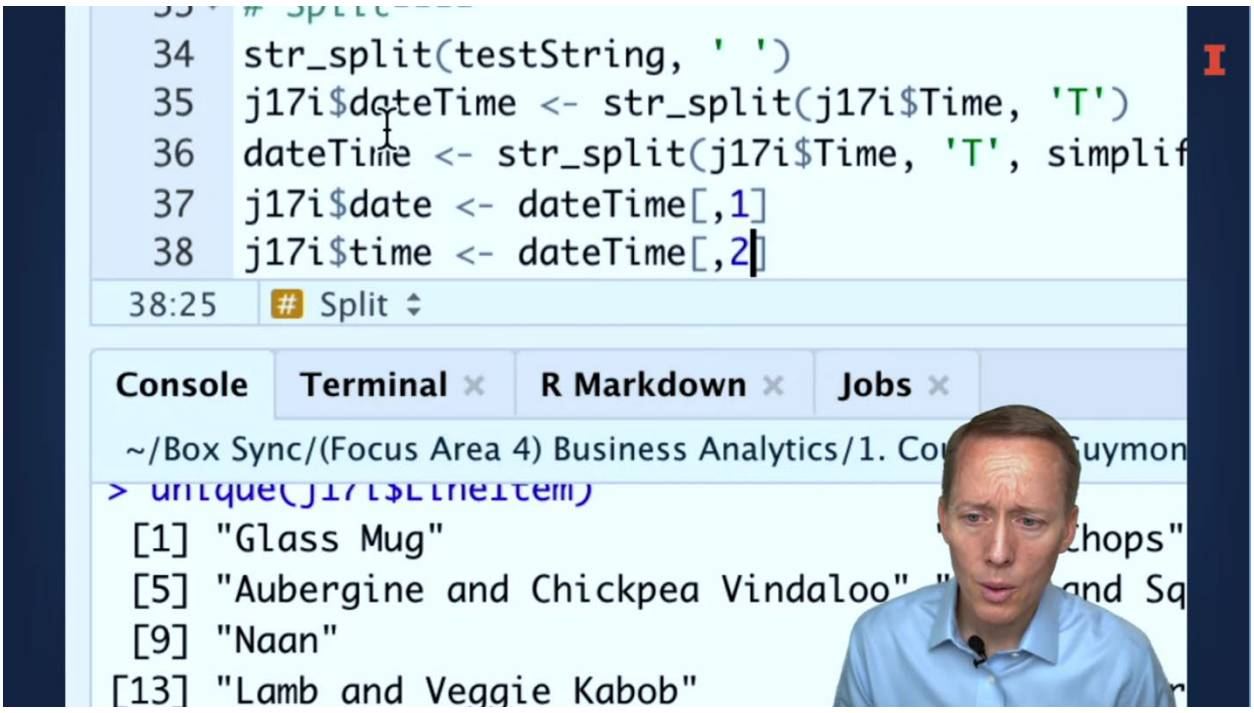
Let's test it out. First, we'll create a new column date. Time were based on the time column, and we identified that there is a T in between the date and the time. So that's what we will use as our pattern or R D limiter. And I'll run this. Yeah, and the scroll to the last column. We can see that the date time column doesn't look like the other columns there, so it actually has a list within each cell here. That's not good. That won't help us. So we want to separate this out into two separate columns, so we'll need to do this in two steps.



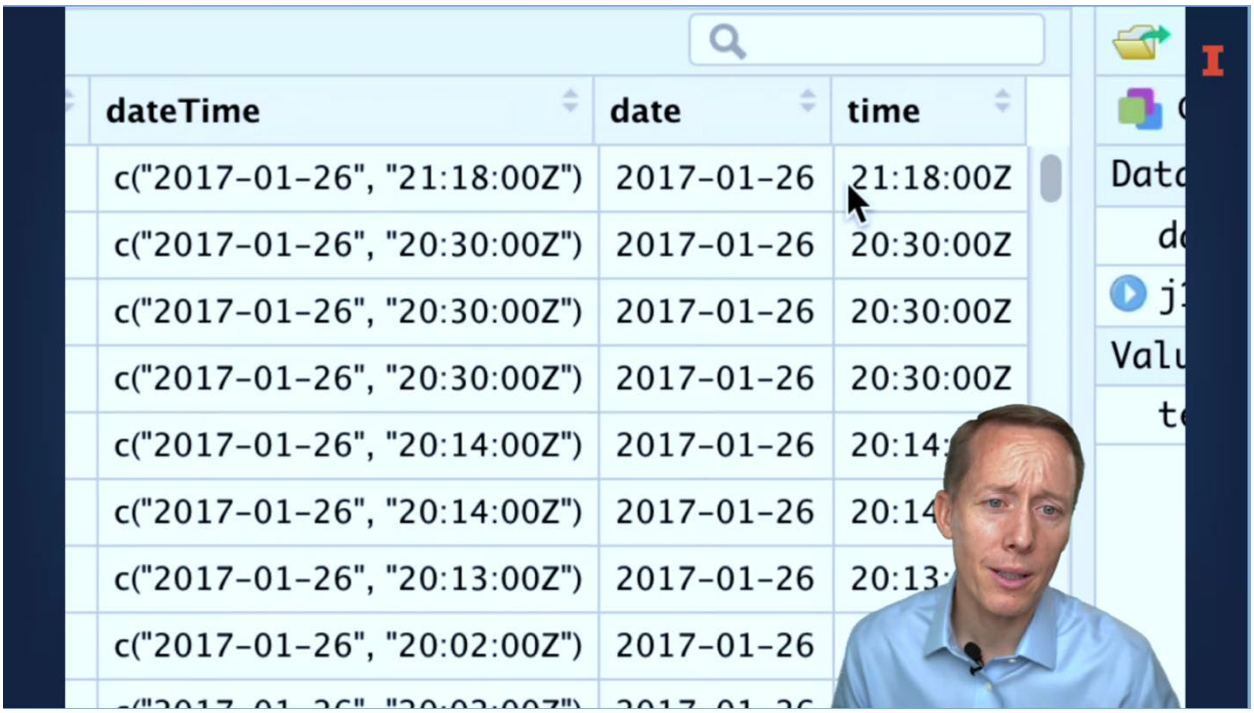
Really, the first step is to create a new date, time data frame or matrix. And so we'll use the string split function. And we also need to use this other argument to simplify argument and set it equal to true. So now, if I look at this date time Matrix here, it has two columns.



First column with the dates second column with the time that's perfect. We want that.



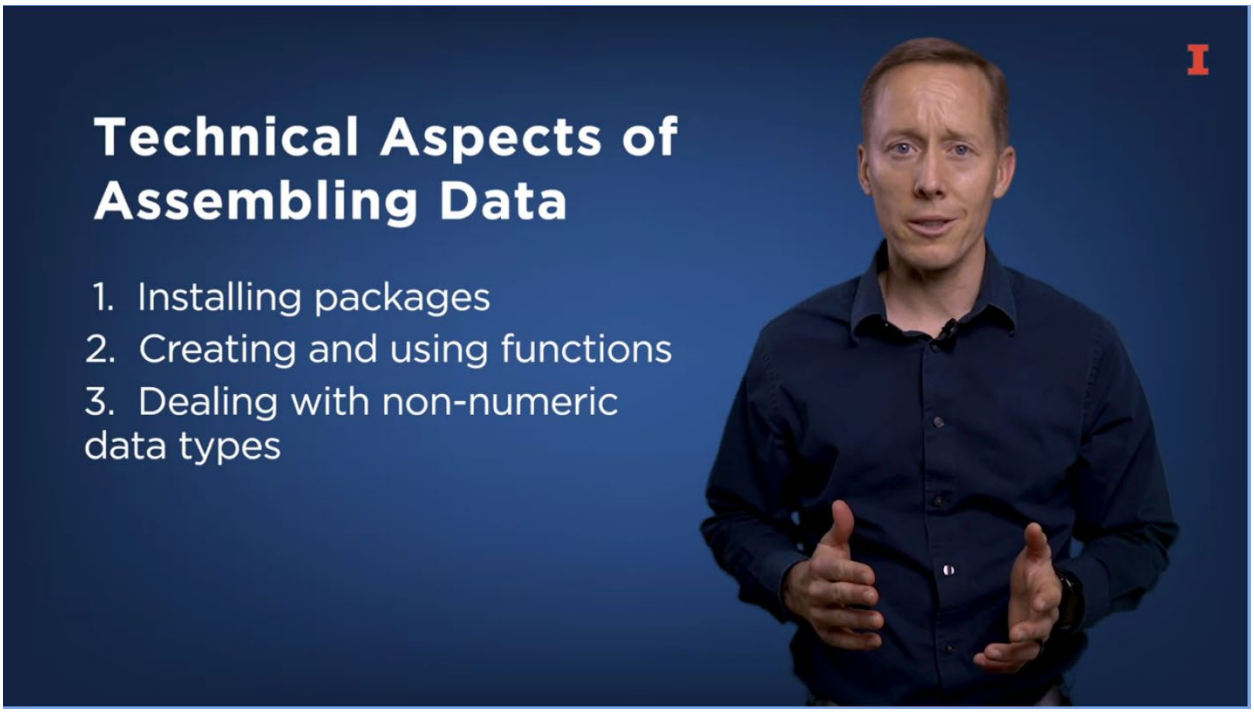
So let's go ahead and store this each each one of those columns in a new column in the J 79 data frame, the first one will be the date, so J 79 date will get the value from the daytime matrix that is in the first column j 79 time.



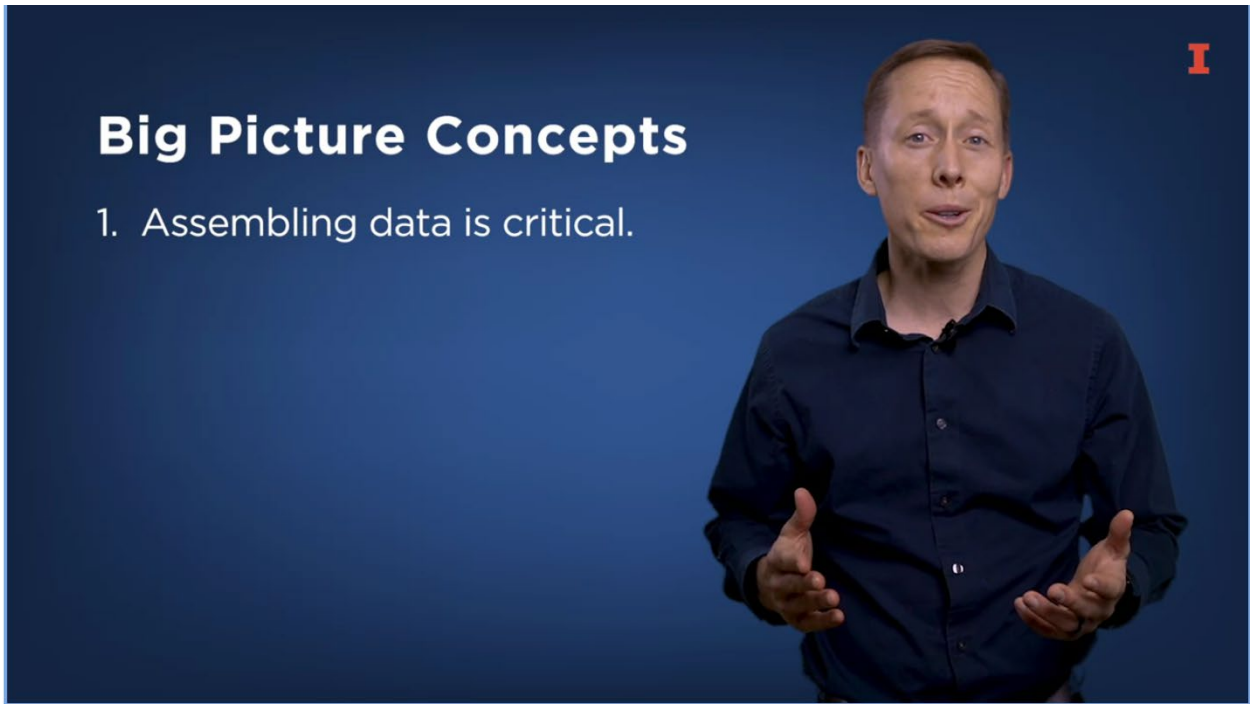
We'll get the second column and let's go ahead and look at our data frame here. And sure enough, we've got the date in one column in the time in another column so you can explore many other functions in the stringer package. It's very powerful. I hope this is a good introduction that gives you an idea of how to get started manipulating and wrangling character strings in our

Module 3 Conclusion

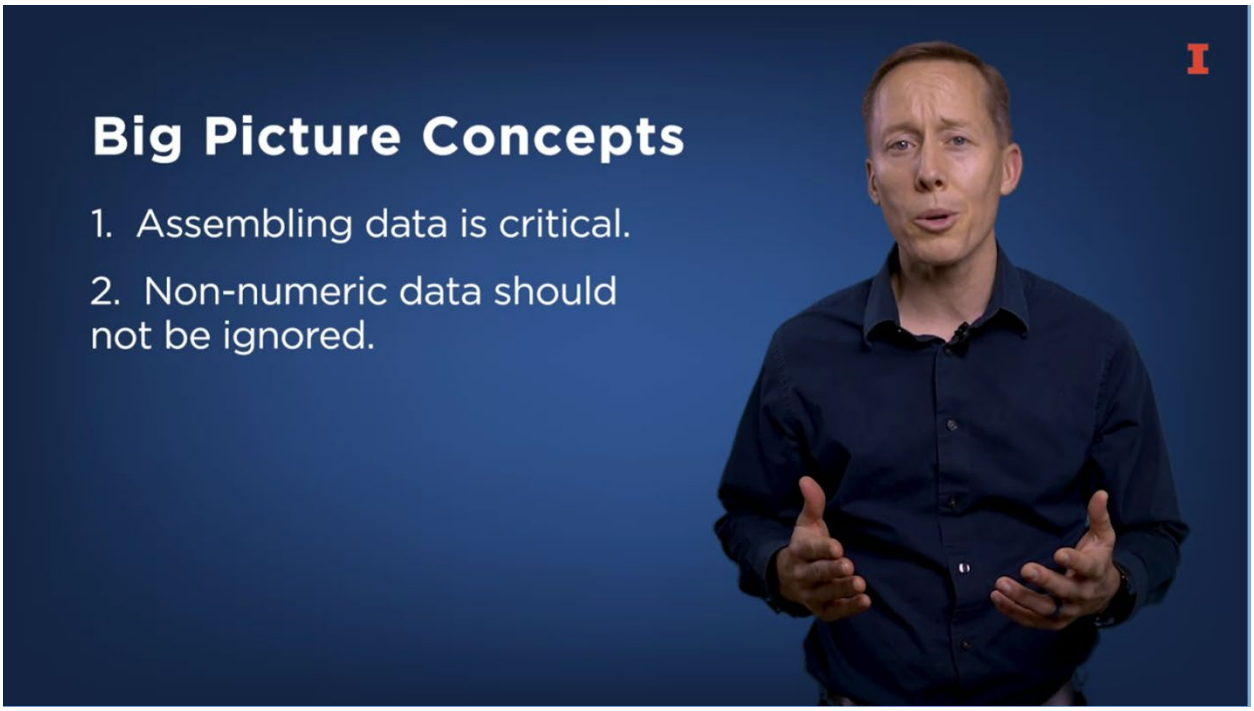
Module 3: Conclusion



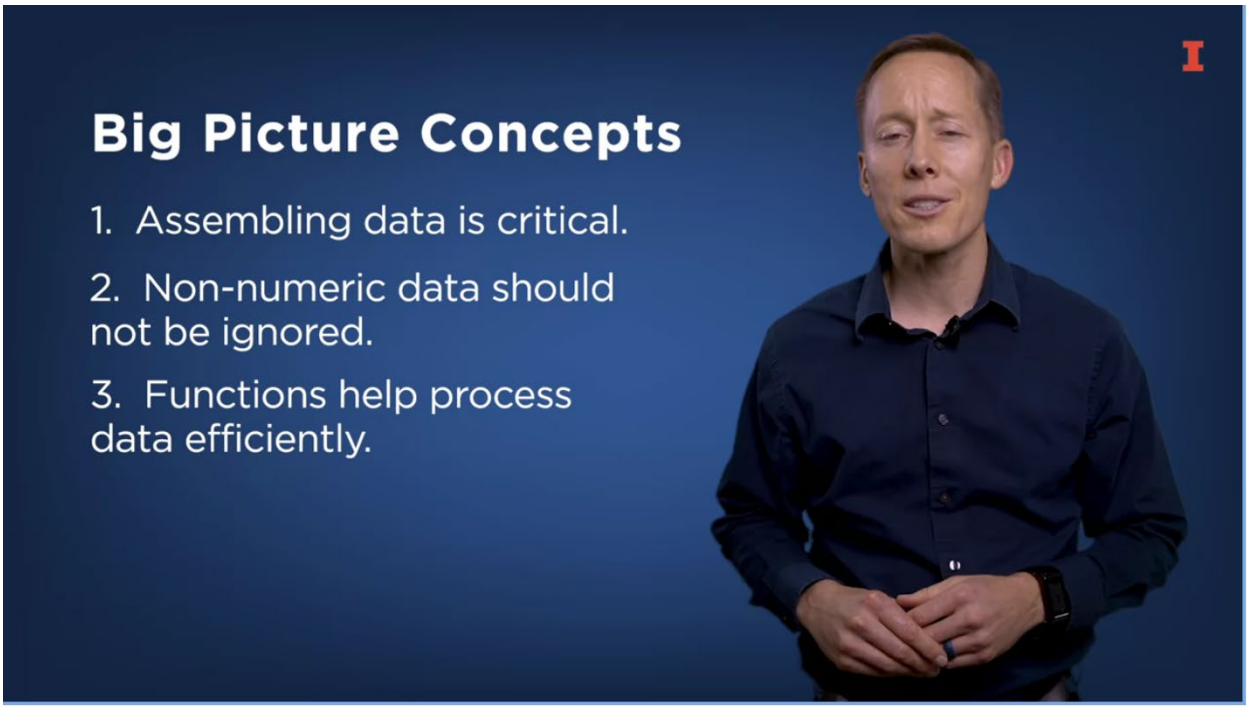
[MUSIC] As we conclude this module, your mind is probably focused on technical aspects of assembling data, installing packages, creating and using functions, and dealing with non numeric data types. Being focused on these details is both a blessing and a curse. It's a blessing because you are now better equipped to identify a number of small potential issues that could prevent a business analytic project from moving forward.



A favorite saying of mine is, it's not the Lions and Tigers, it's the gnats and skeeters that get you. Knowing how to deal with the gnats and Skeeters, or at least being aware that they exist, is really important, knowing that these little things can be a curse if it prevents you from being able to step back and see the bigger picture. So here are some key insights, first, assembling data may not be the most glamorous part of business analytics, but it's a critical part of the process. Just as photographer spend time setting up the lighting before taking a picture in a studio, business analysts need to spend a good amount of time assembling data into meaningful rows and columns.



Another big picture concept is data type, numeric data is probably the most important because algorithms depend on converting data to numbers. However, non numeric data types should not be ignored. I hope the gaining experience working with dates and times, character strings and factors allows you to work more efficiently with data as well as manage those who work with it.



Finally, functions are essential to business analytics because they're what makes it possible to assemble and process data very efficiently. Specifically, functions make it possible for us to

perform common tasks with numeric and non numeric data types without having to re-type large amounts of code and without even knowing the low level code. Perhaps the most important lesson about functions is knowing how to get.